Computer Communications 213 (2024) 33–48

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Enabling simulation services for digital twins of 5G/B5G mobile networks☆

Giovanni Nardini [a,b,*], Giovanni Stea [a]

[a] Dipartimento di Ingegneria dell'Informazione, University of Pisa, Largo L.Lazzarino, 1, 56122, Pisa, Italy
[b] Center for Logistic Systems, University of Pisa, Via dei Pensieri 60, 57142, Livorno, Italy

ARTICLE INFO

Keywords:
Digital twin network
Simu5G
Emulation
Simulation
Mobile networks

ABSTRACT

Digital Twins (DTs) have been proposed as digital replicas of physical entities (e.g., manufacturing equipment), which one can observe in real-time and interact with. Digital Twins of Networks (DTNs) are increasingly being discussed in the literature, as an enabler for efficient data-driven network management and performance-driven network optimization (e.g., to support dynamic reconfiguration, or anticipate the effects of faults). A DTN includes service *mapping models*, i.e. models that can be fed with acquired data to produce insight on the network itself - e.g., to run what-if scenarios, based on multiple underlying technologies, from Machine Learning to analytical models, e.g. Markov Chains. In this paper we examine the case of DTNs of *mobile networks*, DTMNs, tailored to 5G and beyond, where issues of dynamic reconfiguration and fault anticipation are critical. We argue that *simulation services* should be offered by the DTMN in order to allow performance-driven network optimization, and that discrete-event network simulators are ideal instruments to be employed for this purpose. We discuss the challenges that need be addressed to make this happen, e.g., centralized vs. distributed implementation, gathering input from the physical network, security issues and hosting, and we review the possibilities offered by network simulation in terms of what-if analysis, defining the concepts of *lockstep* and *branching analysis*. We present a framework to endow a DTMN with simulation services and we exemplify it using Simu5G, a popular 5G/B5G simulation library for OMNeT++, as a reference case study.

1. Introduction

Digital Twins (DTs) model physical objects in the digital world and allow users to interact with them without the cost or consequences of interacting with the physical object itself. DTs have been proposed in a variety of contexts, most notably in smart manufacturing within the Industry 4.0 paradigm, for applications such as new product design, monitoring of production processes and diagnostics [1]. A DT is realized by using data from the physical object to build a virtual replica of it. However, different categorization of DTs can be envisaged, based on the levels of automation and data integration between the digital and physical worlds [2]: a Digital Model is a virtual representation of the physical object, with no automatic data exchange between the digital and physical counterpart; a Digital Shadow enhances the Digital Model by allowing the physical object to feed the digital object with real data, such that a change in the former is reflected in the latter; a Digital Twin adds to the Digital Shadow a feedback mechanism, in a way that allows the DT to be synchronized with the PT and, in turn, to act as a controlling entity for it. In the following, we refer to a DT as in this last definition. In Ref. [3], other fields such as augmented/virtual reality and virtualization have been regarded as first attempts to create virtual replicas of an object or system. Based on the knowledge coming from those technologies and tools, the authors propose their view about the foundational properties of DTs. Therein, DTs are considered as composed of two entities, namely the physical and logical objects, and a 1-to-N relationship may exist among them: this means that a DT is composed of a data aggregation that multiple logical objects can use to represent the status of the physical object at a given time (in the past, present or future).

By inheriting the concepts from the above visions, this paper considers a DT as a software entity including a repository of data (both historical and real-time, possibly) obtained from the physical object (also called Physical Twin, PT), which can be used to create different models of the PT for different, application-specific purposes. The user of the DT can exploit one (or more) model(s) to perform operations and, possibly, apply decisions on the PT. For example, a distinctive advantage of such DT vision is the ability to *run what-if scenarios*, e.g., try

☆ A preliminary version of this paper has appeared as [26].

* Corresponding author. Dipartimento di Ingegneria dell'Informazione, University of Pisa, Largo L.Lazzarino, 1, 56122, Pisa, Italy.
E-mail address: giovanni.nardini@unipi.it (G. Nardini).

https://doi.org/10.1016/j.comcom.2023.10.017
Received 1 February 2023; Received in revised form 26 June 2023; Accepted 20 October 2023
Available online 26 October 2023

alternative settings of the PT (e.g., different models), possibly using real-world, real-time data, without actually modifying its state.

The networking community has recently pushed forward the idea of DT Networks (DTNs[1]), in both scientific works [1,4,5–9] and standardization bodies such as the Internet Engineering Task Force (IETF) [10–12]. DTNs are digital representations of computer networks, that can be fed real network data in real time, and can be used for several purposes: troubleshooting, network planning, anomaly detection by observing divergent behavior of the physical and digital copies, and – notably – performance optimization. In Ref. [10], a reference architecture is proposed, that defines functions and interfaces at a high level. In the latter, *Service Mapping Models* (SMMs) are a crucial part of a DTN, providing models of (parts of) the physical network that can be fed with real-time data and queried by an application layer to gain insight on – among other things – performance aspects in hypothetical what-if scenarios. Work [12] discusses using DTNs for performance optimization in IP-based packet networks and optical networks, highlighting the specific challenges of these technologies.

To the best of our knowledge, few works so far have discussed the inherent challenges of *new generation mobile networks*, i.e. 5G/B5G/6G networks, and discussed employing DTNs for these. These networks offer differentiated services, bound by Service Level Agreements, which cannot be jeopardized by a careless network reconfiguration. Today's mobile networks are complex systems, with several layers, protocols, functions, planes, whose interactions are hard to predict a priori, and where reconfigurations (e.g., changing the scheduling policy at a particular base station) may have performance repercussions in other parts of the network (e.g., increasing the interference level in nearby cells). Nonetheless, the very nature of the services that these networks do and will support entails dynamic reconfiguration to be a frequent occurrence. In this context, being able to experiment with Digital Twins of Mobile Networks (DTMNs) may drastically improve network management and performance optimization. The only works - that we know of - discussing DTMNs are [13–16], which mostly advocate leveraging Machine Learning (ML) techniques for what-if analysis. ML models complex systems in an abstract way, with a reduced computational load at runtime, which warrants better spatial scalability. However, ML only works *after* extensive training, is often poorly explainable (unless designed to be explainable *ex ante*), and tends to react unpredictably to situations outside the training coverage – which are exactly those that happen when one reconfigures a running network or when faults occur. This is not to say that ML is unhelpful: rather, that it should probably not be the *only* port of call when it comes to SMMs, lest one incurs in the above drawbacks.

In this paper, we discuss the case of using *network simulators* to provide additional intelligence within DTMNs. Network simulation is a mature field of research and the design and engineering of network simulators are well-established practice. Notable examples of network simulation libraries are the ns3 family [17], which includes 5G-LENA for 5G simulation [18], and the OMNeT++ framework [19], which includes the Simu5G model library for 5G/B5G networks [20]. Some of these (Simu5G, for instance) allow real-time emulation [21], i.e. can take input data from a live network (say, copies of arriving IP packets) and perform in real time the same actions that the live network would – updating its own state (e.g., protocol buffers and timers) in parallel to, and coherently with, the network's. On another note, given enough hardware resources, a reasonable network scale, and an appropriate level of modeling detail of the network elements and functions, simulators can run even *faster* than real time, which allows one to predict the evolution of the live network and - possibly - take proactive countermeasures to avoid undesired conditions. In other words, we argue the case that network simulators – being software replicas of a network – need to be *incorporated into* DTMNs, exactly because they can offer a wide variety of services, such as performing what-if analysis or providing interpretable results without the need for training, even in unlikely or untested scenarios. To the best of our knowledge, this observation has not been developed enough so far. Note that our proposal is not antithetic to using ML models within DTMNs. We instead believe that ML and simulation should not only coexist, because they address different needs, but should *interoperate* to make the most of the strength of each one, which are largely complementary. For instance, a network simulator can be used to provide *training data* for ML models that the PT *cannot* provide by definition, e.g., data related to hypothetical working conditions after reconfiguration. Conversely, using ML models within a network simulator – e.g., to model large-scale subsystems – could make simulators considerably more scalable, given the challenging response time requirements of DTMNs [12]. There have been few papers that discuss simulation in the context of DT technology (e.g. Refs. [22–24]), whereas [25] presents a prototype of a co-simulation framework for autonomous vehicles in 5G networks.

The main contributions of this paper can be summarized as follows:

- We discuss the challenges towards the realization of DTMNs, which are many, non-trivial, and specific to the mobile network context.
- We envisage the use of simulation in the context of DTMNs, by discussing the benefits it may bring and the use cases that it may address. This is done by taking into account the most recent standardization efforts about DTNs.
- We propose a possible architecture – compliant with the above standards – to integrate simulation services within a DTMN, also specifying a RESTful Application Programming Interface (API) to allow applications and other components of the DTMN to exploit such simulation services.
- We describe the realization of a proof-of-concept implementation of the discussed simulation services using the popular Simu5G system-level simulator, and exemplify the usage of the proposed API in two relevant use cases, namely the what-if analysis by a network operator and the generation of synthetic datasets for training ML models.

With respect to our previous work [26], this paper includes an up-to-date overview on the state of the art in the DTN and DTMN contexts, which also considers the activities carried out by standardization bodies. Furthermore, while [26] focused on using network simulators for what-if analysis only, this paper proposes a more generic framework to endow a DTMN with simulation services, which is compatible with the proposed standards and can be used for multiple use cases. Finally, the description of the proof-of-concept implementation has been largely improved, by including more details and considering the generation of ML datasets as a use case, in addition to the what-if analysis scenario presented in Ref. [26].

The rest of the paper is organized as follows. Section 2 reviews the related works and standardization activities on the topic of DTNs and DTMNs. Section 3 discusses the peculiar challenges of realizing a DTMN, whereas Section 4 motivates the need for simulation services within DTMN and proposes a framework to integrate them. Section 5 describes our proof-of-concept implementation of the above framework and presents two possible use cases. Section 6 draws the conclusions and provide insights about future work.

2. Related work

Research efforts addressing the relationships between DTs and communication networks quickly ramped-up in the recent years. On one hand, networking is seen as a key enabler for realizing DTs and networks of DTs. In this sense [5], reviews the existing literature about the networking-related issues that may be encountered when designing a DT, such as its placement and real-time synchronization. On the other

[1] Not to be confused with the similarly named Digital Twin Networks [39,40], whose focus is to define a *network of DTs* (of anything), interacting through well-defined interfaces.

hand, many works focused their attention on DTNs, i.e., DTs of networks [6–9]. In Ref. [6], authors propose to use a DTN to automate Software Defined Network (SDN) operations. Besides creating a DT for each element of the SDN-based network, a DT for the SDN Network Manager is included as well, which learns the management actions carried out by its physical counterpart (i.e., the human operator) and applies them autonomously, in order to reduce the manual effort by the human operator. A possible implementation based on microservices is presented and tested in a scenario to handle network flows in OpenFlow switches. The work in Ref. [7] proposes an algorithm for a DTN to estimate behavioral patterns of network flows. Authors devise a method to identify possible actions performed on network packets (e.g., forwarding rules at switches), and design an algorithm for the DTN to learn the behavior of a network flow based on the above method. The approach is evaluated for caching and routing operations in a wired network. Authors of [8] conceptualize the use of DTs in SDN-based vehicular networks, where an Artificial Intelligence (AI) model of the vehicular network can be used to manage control operations for the vehicular networks. The paper highlights the challenges and the future research directions related to this concept. Instead, in Ref. [9] the DTN is applied to the Industrial Internet of Things (IIoT) use case. In this case, too, SDN is considered the key element towards the realization of the DTN, which in turn enables features like predictive maintenance, energy optimization and resource allocation for sensors, actuators and communication infrastructure of an IIoT architecture. The authors identify the required architectural elements and presents a prototype implementation for a SDN-based Wireless Sensor Network.

As a special case of DTNs, some works addressed DTMNs, i.e., DTs of mobile networks [13–16]. One of the first attempts to conceptualize DTs, at least at a high level, in the context of 5G networks was done in Ref. [13]. Authors discusses the potential benefits that the DT technology could bring into the development of 5G (and beyond) systems, and present AI as the main technological enabler to realize the DTMN due to its ability to evolve autonomously the model of the network, as opposed to network simulators. As we will discuss later in this paper, we instead believe that AI and simulations are not in contrast: network simulators can be used alongside AI tools (or even supporting each other) to offer a richer set of features to the users of a DTMN. Furthermore, the paper does not dig into the details of the implementation of the DTMN. A more practical DTMN architecture is presented in Ref. [14], which proposes an architecture to implement the DT of Beyond-5G (B5G) networks, called B5GEMINI. The main modules that compose B5GEMINI are briefly described, and the role of AI into such framework is discussed. Still, the work appears to be in its early stages. AI is the focus of [15] as well, which proposes to use Graph Neural Networks (GNNs) as a mean to predict network key performance indicators, which can then be used within DTMNs. In particular, authors propose a method to generalize a GNN-based model trained on small networks to larger scale ones. The survey in Ref. [16] highlights the importance of DT technology in the design of 6G networks, and provides an overview of the potential use cases of DT at different levels of such systems, such as the radio access, the core network and data centers. Although the paper discusses the efforts from major standardization bodies, it mostly focuses on applications of DTs in 6G networks, rather than the design the DTMN itself.

The above works, especially those related to mobile networks, mostly consider AI and ML as the only approach towards the realization of the DTN concept, while simulations are hardly considered. In this paper, our aim is to consider simulations as a technique that complements models and services provided by a DTMN, such as ML-based model, as we believe that both AI and simulation can be useful within a DT at different times – e.g., depending on the application using the DTMN – or by making them interact between each other – e.g., allowing simulations to provide new datasets for ML training. In Refs. [22–24], simulation-based DTs are envisaged to support industrial applications, instead of networks (either mobile ones or not). One attempt to consider simulations in DTMN has been made in Ref. [25], where the authors present a prototype of a co-simulation framework for twinning 5G-enabled autonomous vehicles. However, the proposed framework is used to monitor and assist autonomous vehicles in the very specific scenario of lane changing maneuvers, rather than to represent the 5G network itself, which is instead the target of our paper.

As far as standardization efforts are concerned, Internet Drafts on DTNs [10–12] have recently started to appear within the IETF Network Management Research Group (NMRG). Draft [10] lays out the basic definitions of a DTN and proposes a reference architecture, pictured in Fig. 1. The latter envisages a DTN as laying in between a physical network layer and an application layer. Possible applications are Operations, Administration and Maintenance (OAM) procedures, possibly relying on Intent-Based Networking (IBN). The DTN includes: a Data Repository subsystem, which stores real-time and non-real-time data about the physical network; a Network Management subsystem, which manages the DTN (e.g., monitors its performance and resource consumption), and a Service Mapping Models (SMMs) subsystem, that provide the models used by the applications for, e.g., network analysis, diagnosis, prediction, etc. Throughout this document, we will use this architecture as a reference. Draft [11] discusses issues related to data collection in DTNs. Its main point is that collection of "raw", elementary data (e.g., packet events) in real time from a complex, large-scale physical network is unfeasible, hence an architecture that moves part of the burden of data aggregation/synthesis to the physical network is required. The draft discusses the roles of entities in such an architecture, envisaging a module that can send instructions (possibly in the form of executable code) from the DTN to the physical network to pre-process raw data (e.g., compute a given percentile in a time series) before sending information to the DTN. Finally, draft [12] envisages Performance-oriented Digital Twins (PODTs) for packet and optical networks. These are specific SMMs whose job is to predict the network performance in particular what-if scenarios, thus helping anticipating failures, training operators, and – crucially – optimizing network performance, possibly in conjunction with a performance optimizer (running as an application in the above-mentioned architecture). Draft [12] advocates using AI and ML to devise PODTs, moving from the observation that ML models provide responses faster, unlike simulation or emulation, and tend to scale better. While this is certainly correct, we stress that ML models tend to react unpredictably to situations outside the training coverage, which makes them less trustworthy when experimenting with hypothetical, unforeseen scenarios.

3. Designing the DT of a mobile network

The challenges of designing a DTMN are quite specific to the application domain. Except where noted, we are not aware of previous works discussing them. For the sake of clarity, we refer to the architecture and terminology of Fig. 1, commented in the previous section. According to the terminology set forth in Ref. [3], DTMNs are in *strong entanglement* with their PT, since the communications between the DT and the PT is bidirectional (including a control interface to modify the status of the

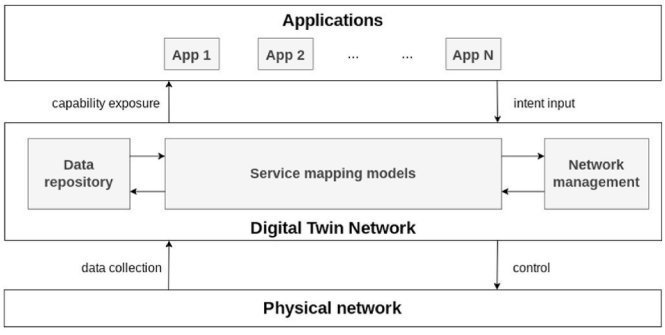


Fig. 1. DTN reference architecture (taken and re-elaborated from Ref. [10]).

PT) and near real time.

A first issue to be addressed is that of *spatial scale*. A mobile network (MN) is composed – by definition – of several interacting nodes. It is not clear whether a DTMN should be a single digital entity, or there should be *one* DT for each MN element, with DTs interacting with each other. In a 5G/B5G mobile network, this question translates to whether a DT should mirror a single base station or – possibly – a cluster thereof. The larger the spatial scale, the more complex the DTMN becomes – think, for example, in terms of input data to be recorded per unit of time at a single point. On the other hand, reducing the spatial scale (e.g., to a single base station) makes DTs simpler, but a DTMN can only be realized by networking DTs of individual network elements, according to some paradigm that – to the best of our knowledge – has not been specified yet. Moreover, there are several salient data that refer to different entities in a MN, but need to be collated to be meaningful – for instance, data related to a single user roaming across several cells. The above two possibilities are represented in Fig. 2. Regardless of whether the relationship between physical and digital MN twins is $1:1$ or $n:1$, network management should obviously have a *network-wide perspective*. MNs include functions and algorithms whose reach extends beyond a single cell's (e.g., frequency reuse, carrier aggregation, handover, etc.), hence the application layer above the DTMN should be able to address as large as spatial scale as allowed by physical or administrative constraints. This means that provisioning "small-scale" DTs will place additional burden on the application layer, which will need (e.g.) to interrogate several DTs and collate their response in order to be able to obtain the required insight. This, in turn, raises issues such as state misalignment and/or time synchronization among DTs of the same network. Moreover, collating data from different SMMs and making sense of it in a wide-scale perspective will complicate the algorithmic part of the application, possibly making it similar to a uber-SMM itself. On the other hand, having a "large-scale" DT will allow for simpler network management applications. The only constraint with this approach is the demand for computational power, storage and PT-DT interface bandwidth will rapidly grow up to a point where scale cannot be increased.

A possible middle ground between the two above extremes can be to allow DTs to form a *hierarchy*. More specifically, one could envisage *lower-level DTs*, for single network elements (e.g., base stations), and a *higher-level DT*, which implements the concept of DTMN by collating all the lower-level DTs. This view is pictured in Fig. 3. To the best of our knowledge, hierarchical DTs are a new concept in the literature, and there seems to be hardly any literature on this topic, which means that

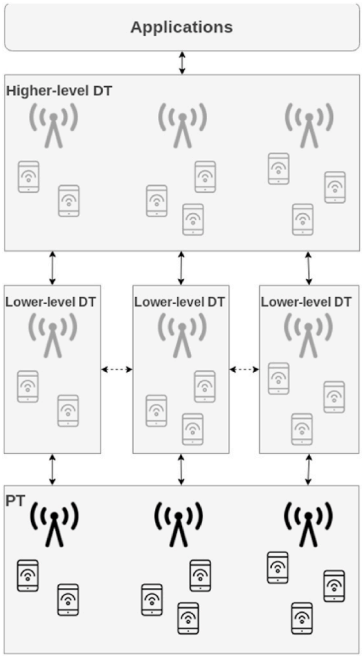
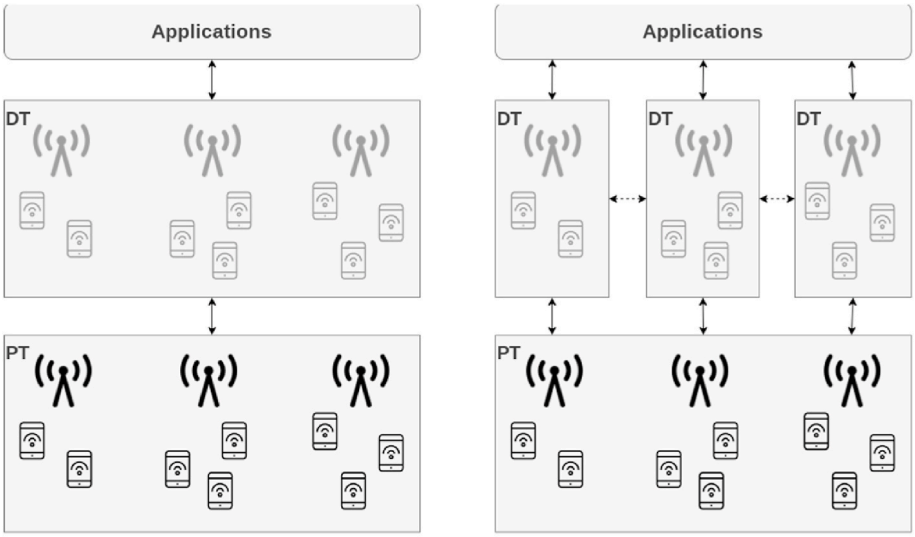


Fig. 3. Connecting applications with a hierarchical DT architecture.

the pros and cons of such an approach would need to be carefully evaluated before committing to such an architecture. We therefore limit ourselves to mentioning some of the most relevant points with respect to such a view. We exemplify this on a two-level hierarchy, for ease of exposition.

While SMMs and – possibly – network management policies can easily be envisaged to be hierarchical (e.g., a model of a network consisting of models of the network entities and so on), it would probably be unadvisable to replicate data repositories at the various levels of the hierarchy. Rather, raw PT data should reside in the lowest level of the hierarchy only, to avoid replication or misalignment of copies. A top-level DT should contain enough information to access the data repositories of the lower-level DTs, or possibly maintain higher-level or historical data views, but should refrain from caching real-time data. It is also debatable whether an application should interface with the top-level DT *only* or should instead be able to access the lower-level ones.



a) DT of a cluster of base stations

b) DTs of single base stations

Fig. 2. Connecting applications with one DT (a) or a cluster of DTs (b).

The second option would favor flexibility at the application level, but may require additional burden to maintain the network status synchronized. Think, for instance, of an application sending intent directly to a lower-layer DT, which implies reconfiguration of the network, without the top-level DT being in the loop (or, worse yet, sending contrasting intent to the top-level DT near-simultaneously). We also note that, in the current version of the IETF specification (see Fig. 1), the southbound interface is meant for the PT only, and the northbound interface is meant for an application. In a two-layer hierarchy, a DT-to-DT interface should be envisaged to allow meaningful communications between the two layers.

Hereafter, we will use the term "DTMN" in a broad sense, encompassing both a single DT and a network of DTs, assuming no hierarchy.

Another key issue is what a DTMN should consider as an *input* on its *data collection interface* (see Fig. 1). In order to be able to mirror the state of the PT, the DTMN must receive the same input as the network. Input data for a MN include:

- Packets destined to the User Equipments (Ues), at the IP layer, at the entrance of the network (e.g., at the Packet Gateway or the base station). For these, IP headers and payload length should be known – the actual payload may not be necessary. These packets must be timestamped with the time of arrival at the entry point.
- Traffic sent by the Ues, via the uplink. This enters the mobile network at the MAC layer, but includes information that will travel through the NewRadio stack up to the IP layer. MAC-layer Protocol Data Units arriving from the mobiles must be received by the DTMN as well. All the MAC, Radio Link Control (RLC), Packet Data Convergence Protocol (PDCP) and IP *headers* must be included, together with the length of the IP payload.
- Control information arriving from the mobile themselves, e.g., Channel Quality Indicator (CQI) reporting.
- Management information, e.g., which Ues has which capabilities, is attached to which base station, etc.

We need to observe that the *state* of a PT is not determined by the above quantities only, i.e., its evolution over time cannot be accurately predicted using the above information only. This is because PT elements (notably, base stations and Ues) run protocols at each (sub)layers, and each of them is – by definition – a stateful entity. Therefore, two alternatives can be envisaged, representing the two ends of a spectrum of possibilities: at one extreme, one may think of conveying the *entire* state of *all* the state machines included in the network elements as an input data to the DTMN. This has the obvious benefit of making the DTMN completely aligned with the PT – with a lag given by PT-DTMN communication delays, and processing delays at the DTMN itself. However, it is likely to require an unfeasibly large communication overhead, not to mention space occupancy for data logging. Moreover, it would require a complete re-engineering of existing PTs, whose software should be enhanced with countless probes just for the purpose of sending information to the DTMN. At the opposite end of the spectrum, we can envisage a design whereby the DTMN is only given the *external* input (as per the above bulleted list), and then is able to internally generate the missing data by running *models* of the protocol state machines as components of its SMMs. This minimizes the communication overhead, and requires minimal intervention on the physical network (tapping packets at network entry points is easily doable). The drawback is that the DTMN's own models may not behave exactly like the corresponding PT elements, leading to misalignment (or *randomness* [3]). Another possible drawback is that modeling protocol state machines is not an easy task, which increases the cost of the DTMN. Moreover, it is often the case that some of the technology within a MN is proprietary, hence confidentiality issues may arise in endowing a DTMN with models of intellectual property. In between the above two extremes lies a spectrum of possibilities. For instance, checkpointing techniques may be used to align the DTMN to the PT periodically, whereas internal models make do with the lack of external input between checkpoints. The frequency of checkpoints should be selected so that the worst-case accumulation of randomness between them is bounded to an acceptable level, given the applications that need to use the DTMN.

It is also advisable that the DTMN control interface be able to configure, as well as turn on/off certain data reporting at the PT. This allows a DTMN to reduce the data reporting generation, communication and processing load to the minimum amount required by the type of applications that will use the DTMN. Flexibility is required here, since the possible uses of a DTMN cannot be decided a priori.

A related issue is *how* to convey input to the DTMN. There is a strong need for standardized interfaces between the physical and the digital, in the interest of interoperability and to facilitate the decoupling between network manufacturers and DTMN designers. Some existing interfaces may be leveraged, e.g., Network Data Analytics Functions (NWDAFs) [27] or Open RAN (O-RAN) probes [28]. Depending on the adopted trade-off point between autonomy and accuracy of the DTMN, conveying digital copies of the network input to the DTMN may entail different overhead, and place a burden on the network link between the physical and the digital copies.

There are also *security* challenges to be considered with DTs [29]. Most notable among them is the fact that DTs can – in general – be able to send command to their PT via its *control* interface, which should be secured lest it offers a backdoor for attacks to the PT itself. The same security challenges can be found in DTMNs as well, along with new issues that are specific to the mobile network field. A DTMN can in fact be used to acquire information on the (physical) mobile network, hence could clearly be exploited by rogue users to *prepare attacks* to it, even when the control interface between the DTMN and the physical network is secured. Moreover, attacking a DTMN itself, and especially its SMS component – e.g., by succeeding in altering its behavior in a what-if analysis – could be a way to trick network administrators into making adversary network management decisions, in the false hope of achieving fabricated performance targets.

Finally, the issue of *where* a DTMN should be hosted requires some considerations. It is quite evident that the latency between the PT and its DTMN should be small enough, in the interest of accurate synchronization or *promptness* [3]. An ideal candidate for this appears to be Multi-Access Edge Computing (MEC [30]), which could also be run by telecom operators themselves, minimizing the security risks. The hosting should in fact have enough computational power to run CPU- and memory-intensive computations, as well as provide low-latency interactions between the twins, which MEC natively does. Moreover, MEC is able to query some network parameters via standardized interfaces: for instance, it can acquire user location [31]. This, in turn, would allow a SMM within a DTMN to infer more realistic estimates of the effects of some management decisions (e.g., a cell-user association policy) and to increase the fidelity of the results.

4. Simulation services within a DTMN

In this section, we discuss the motivations and use cases for using network simulation services within the SMMs of a DTMN. We then discuss the different types of what-if analysis that can be carried out with a simulator, and we present an architecture, compliant with the one proposed in Ref. [10], that allows a simulator to be used as an SMM.

4.1. Motivations and use-cases

The term "network simulator" has different meanings for different research communities. Those working on signal processing and physical-layer design use it to denote a software that allows one to predict a waveform at a receiver, given one at the sender and ambient conditions (e.g., interfering transmissions). What is above the physical layer (notably, the rest of the protocol stack) is of comparatively minor interest, and is often abstracted as a traffic generator. Networking

researchers call a "network simulator" a software that allows one to understand what happens at different network layers (e.g., transport, network, MAC, etc.), and want to be able to change blocks - e.g., TCP congestion control algorithm, or NewRadio scheduling policy - to see how this affects network performance or applications. For them, fidelity in the signal propagation is of comparatively minor interest, and they often employ statistical methods to compute the reception SINR. In this paper, we stick to the second meaning. Simu5G and 5G-LENA are two such network simulators. The reason why simulators of this type are useful for a DTMN is that they model the behavior of network intelligence (i.e., algorithms), and several DTMN applications hinge on modifying network intelligence, such as what-if analysis. Note that the term "simulator" does not necessarily imply lack of fidelity in the replica of physical processes (e.g., as opposed to "emulator" [13]). A simulator can be as detailed a replica of a network element as one wishes it to, with the above caveats – unlike a ML-based model.

A network simulator includes models of the protocols and functions run in the network elements. A point of strength of simulators is that they can have *different* models of the same element, marking different trade-off points between accuracy and efficiency, and switch between these models dynamically, depending on the requirements. For instance, at one extreme, a network element can be modeled as a queueing system, with a given service time distribution; at the other extreme, the entire finite-state machine of that element could be modeled, using the same codebase as in the real network. More detailed network models often warrant better accuracy, and – while they are costlier to run, resource-wise – they may also allow one to reduce the communication between the PT and the DTMN, by generating internally copies of data that should otherwise arrive as inputs from the PT. This capability can also be leveraged to implement checkpoint-based synchronization, where the more internal generation is done by the simulator, the larger the checkpoint period can be.

A network simulator can be used as an SMM in a DTMN, for several purposes. The most common one, already mentioned several times, is *what-if analysis*, related to any of the aspects modeled in the simulator itself: the amount of traffic from/to one or more UE, the (de)activation of microcells, changes in the scheduling policies, etc. This is made easier by the fact that a simulator has *repeatability*, i.e., the capacity to replicate network conditions on-demand, and *reproducibility*, i.e., the ability to replay successions of events, possibly under controlled variations. Network optimization can leverage a simulator's what-if analysis. A *network optimizer* can interface with a simulator in a closed loop, to realize *simulation-based network optimization*, as shown in Fig. 4: at every iteration of the loop, the optimizer configures a simulation scenario, the network simulator runs it and returns selected performance measures, and the optimizer alters the scenario to maximize a target performance metric.

While what-if analysis looks forward in time, simulators can be used to play back past network traffic traces in order to perform *troubleshooting*, i.e., to understand the root causes behind a malfunctioning or underperforming network. Work [12] also mentions *training* of management staff, which can use the simulator as a playground to understand the implications of their management policies and configuration decisions.

To these three, we add a non-trivial use case, which has recently been gaining attention in the research community [32]. A simulator can be used as a *training dataset generator* for an ML model. Recall that ML models are expected to be employed as SMMs, due to their scalability and fast response times. A known problem with ML models is that their reliability depends on their training, which is done offline. It is therefore important that ML models are exposed to the right set of conditions before being used. This implies that ML training cannot rely on data generated by the PT *alone*. In fact, the PT only generates data pertaining to its current operational conditions, whereas data *outside* these conditions are required to understand the behavior of the network in hypothetical conditions (e.g., following a fault of some equipment or modified traffic scenarios). These data can instead be reliably generated by a simulator, possibly fed with information from the PT itself, exactly *because* it contains the necessary intelligence. Data generated from a simulator may thus make training sets for ML models more reliable, increasing their coverage, in a safe and relatively inexpensive way.

4.2. Two types of what-if analysis

In this subsection, we take a more in-depth look into the what-if analysis use case, since many considerations are not trivial and may affect the type of interactions between the application carrying out the analysis and the simulator running as SMM within the DTMN. Two types of what-if analysis can be carried out in a simulator-based SMM. We call them *Lockstep* and *Branching* what-if (LWI and BWI, respectively), and they have different requirements and benefits. For the sake of concreteness – and without loss of generality – we exemplify them in the context of evaluating alternative MAC-level scheduling policies A and B. With reference to Fig. 5 (top), LWI consists in running a single simulation that has *simultaneously* both policies A and B. On each scheduling period, the functions implementing A and B are called, and two different schedules are computed. These schedules can be evaluated – e.g., as per amount of resources occupied – and the results of this evaluation can be stored for later. For instance, given a scheduling metric (say, resource occupancy), one may use LWI to understand that policy A occupies fewer resources than B 65 % of the time (possibly because it exploits channel knowledge better), and is therefore preferable on that account, given the current conditions of the PT. Note that the PT may have been using A, B, or neither, during this analysis – this is irrelevant. LWI can obviously been extended to any number of competing network functions. An LWI analysis can be run in real time, with data generated from the PT(s) which is simultaneously stored at the DTMN and used for the analysis. The downside of LWI is that the outcome of the evaluated alternatives cannot influence the state of the simulation over time. For instance, a connection may have a different backlog under different policies A and B after a scheduling period n in the simulation, call it $b_A(n), b_B(n)$, while the PT is running policy C and leaves that connection with a backlog $b_C(n)$. At scheduling period $n+1$, the DTMN will receive $b_C(n)$ as an input datum from the PT for the connection being discussed. Therefore, the next *simulated* schedules will try to allocate resources for $b_C(n)$'s worth of data, using simulated policies A or B. This has an

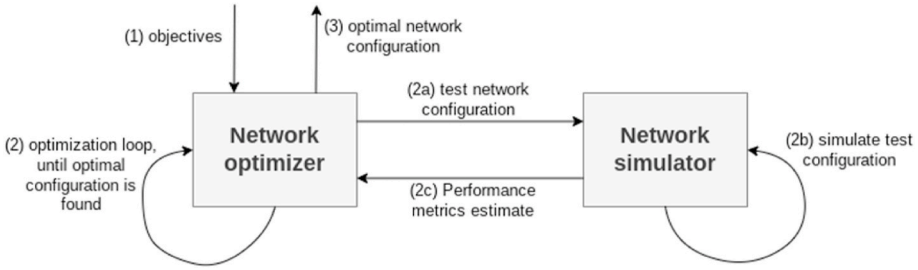


Fig. 4. Simulation-based network optimization.

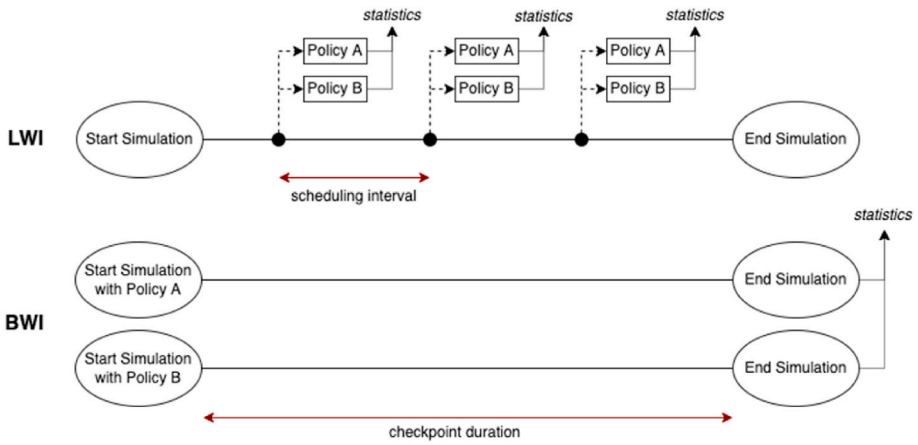


Fig. 5. LWI approach (top) vs. BWI approach (bottom).

important consequence: LWI cannot be used to evaluate metrics that have an inherent temporal span, such as fairness or mean delay, exactly because of the above problem.

The alternative approach is BWI, whereby what-if analyses are done by running *several instances* of simulations, as shown in Fig. 5 (bottom). When BWI is started, *two* copies of the current PT state are branched, one running policy A and another running policy B. The two copies do not communicate, and they evolve independently up until a checkpoint (e.g., a number of scheduling periods), where the evaluation is performed. The benefits and disadvantages are complementary to LWI's. To begin with, with BWI you need to run different instances of a simulation simultaneously, which requires more computation power than LWI. With BWI, the *outcome* of the simulated policy decisions (e.g., the resulting backlogs $b_A(n)$, $b_B(n)$) can be fed back to the simulation instance, thus enabling evaluations that go beyond the single scheduling epoch, e.g., those requiring temporal averages or integrals, such as fairness, delay, etc. An advantage of BWI is that one can run *several replicas* of the same scenario as separate simulation instances. Although initially synchronized at the onset, replicas will diverge if internal random-number generation is performed. This allows one to obtain independent and identically distributed (IID) samples, on which to compute statistical results (e.g., confidence intervals). This is not doable with LWI.

We believe that both LWI and BWI may be beneficial for a user of a DTMN, possibly at different times, hence a DTMN SMM should be designed to support both.

4.3. A framework for exploiting simulation services within a DTMN

We now discuss our proposed framework to endow the DTMN with simulation-based SMMs, that is the capability to build and run simulation models according to the data included in the DTMN and produce relevant statistics from them.

With reference to Fig. 6, we consider a DTMN compliant with the architecture being discussed within IETF [10] and composed of a data repository populated with data from the physical network, a DTMN network management entity, and several SMMs. The latter include a number of services that are useful to build data models and/or provide functionalities to external applications, such as training of ML models or network statistics visualization. Among the available SMMs, we include a DTMN Simulation Service (DTMN-SS), which exposes an interface to the applications for configuring new simulation models and running simulation campaigns, such as an application run by the MN operator to perform what-if analyses. The same interface is made available to other SMMs within the DTMN, such as an ML model that need a new training set. In the following, we generically refer to applications and SMMs using the DTMN-SS as *DTMN-SS Clients*, or *clients* for short. The DTMN-SS also needs to communicate with the data repository to obtain relevant information to configure the simulation model, by using either historical or real-time data (or both). Moreover, the DTMN-SS interacts with the DTMN network management: in fact, whenever a client requests a new simulation model to the DTMN-SS, the latter has to check whether the DTMN has enough computing resources, such as CPU and memory, to create such model and run the desired simulation instances.

Fig. 7 shows a detailed view of the DTMN-SS internal architecture

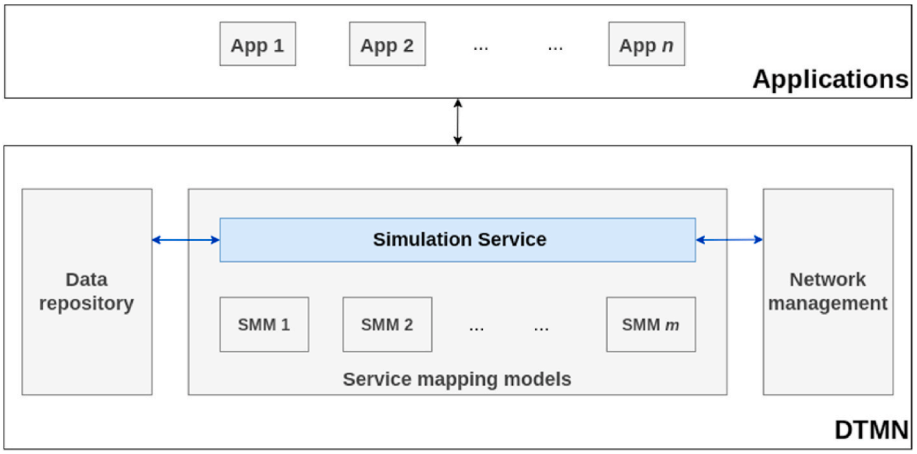


Fig. 6. High-level view of DTMN with Simulation Service.

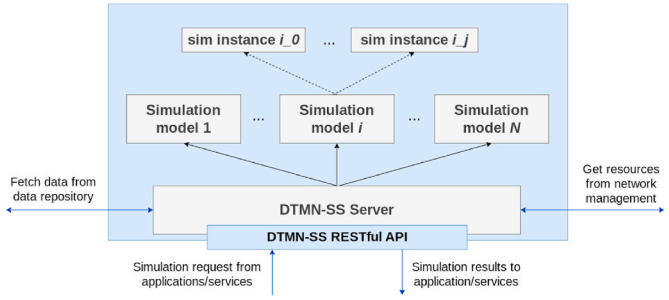


Fig. 7. Internal representation of the DTMN-SS.

and its interfaces. The core is the *DMTN-SS Server*, which is the software that receives requests from clients, builds the simulation environment, runs simulation instances and sends the results back to the clients. To do this, the DTMN-SS Server exploits one (or more) simulation library installed within the DTMN-SS itself. Examples of simulation libraries in the scope of DTMNs are Simu5G and 5G-LENA, for the OMNeT++ and ns3 simulation frameworks, respectively. Both libraries provide a large set of modules that model all the entities (e.g., UEs and gNBs) and protocols (e.g., NewRadio, TCP/IP, and so on) required to simulate a 5G network at the system level. Thus, when the DTMN-SS Server receives a request from a client, it uses the available simulation libraries to create the requested simulation model representing a specific network scenario. For example, in OMNeT++ the network scenario can be built by just customizing a file that specifies how to connect the modules; this will be explained in detail in Section 5.1. However, while the simulation library provides the generic modeling of the MN entities and protocols, they need to be parametrized in order to create the desired scenario. Depending on the target of the client requesting the simulation service, such parametrization can be made by using:

- real-time data fetched from the data repository of the DTMN. In this case the simulation model will reflect the real-time status of the physical network, which can be useful to, e.g., perform real-time monitoring/visualization;
- historical data fetched from the data repository of the DTMN. In this case the simulation model will represent the status of the network at a given time in the past, which can be useful to, e.g., reproduce some behavior or event occurred in the past for troubleshooting;
- data provided by the client itself. In this case the simulation model will represent an arbitrary configuration of the network, which can be useful to, e.g., create potential future scenarios and generate data to train ML models;
- a combination of the above. What-if analyses can be performed following this approach, since they make use of real-time data to represent the current status of the network, although its evolution can be modified by applying custom data or policies at specific modules, such as using a novel scheduling policy.

The DTMN-SS can handle multiple requests for different simulation models, from different clients in parallel. Upon the reception of a new request from a client, the DTMN-SS Server needs to contact the DTMN network management to verify the availability of enough computing resources to build the simulation model and run the intended simulations. On one hand, it is likely that the DTMN runs on a virtualized infrastructure (e.g., in the cloud or at the edge of the network) and its computing resources are not limitless. On the other hand, running simulations can be a computing-intensive task, since it can consume an arbitrary amount of CPU and memory resources, depending on the complexity of the simulation model, the scale of the network, the number of independent replicas to be run, and so on. Thus, the DTMN-SS must make sure that the DTMN has enough computing resources to instantiate and run a new simulation model, without affecting the performance of other simulation models or other DTMN services. If the network management does not grant the authorization to instantiate the new simulation model, then the DTMN-SS Server may reject the request by the application or delay it until enough resources become available.

In order to allow clients to communicate with the DTMN-SS Server, the latter exposes a RESTful API, e.g., following the HTTP protocol. For each session instantiated by a client, the DTMN-SS Server provides a set of resources that clients can access through HTTP methods. A potential list of resources exposed by the DTMN-SS are the following:

- *Simulation environment*: it includes general parameters to configure the simulation campaign, such as the duration of a simulation, the iteration variables (i.e., parameters that assume different values in different scenario configurations), the number of independent replicas for each simulation scenario, the number of parallel instances to be launched.
- *Network topology*: it includes the semi-static network configuration, i.e., parameters whose values do not change throughout a simulation, such as position of the base station, their transmit power, their interconnection with the core network, and so on. Usually, this kind of data is fetched from the data repository, as a DTMN is likely to replicate the topology of the physical network. However, the scale of the network topology could change: for example, a client might want to simulate a model of just one base station with its UEs, whereas another client might want to include also all its surrounding base stations.
- *Network parameters*: it includes the data that represents the fixed parameters for the simulation, or their initial value. It determines the parametrization of the network topology. Examples of network parameters are the number and position of UEs connected to the network, as well as their downlink and uplink traffic. Such data can be either fetched from the data repository (i.e., real-time and historical data from the physical network), or provided by the client itself.
- *Output metrics*: it specifies the output of the simulation model, i.e., what metrics the client expects to receive from the DTMN-SS after running the simulation campaign, such as the average cell throughput, the average resource utilization, etc.
- *Simulation campaign*: it represents the running instance(s) of the simulation model, as configured by the above resources. When the client invokes this resource, the DTMN-SS executes the simulation campaign and, when it terminates, it extracts the metrics included in the list of available metrics and reports their value to the client.

By using different settings of the above resources and invoking them in different order, clients can enable different use cases for the DTMN-SS, as we will exemplify in the next section.

5. A proof-of-concept implementation using Simu5G

In this section, we show a proof-of-concept implementation of the DTMN-SS described above using Simu5G as simulation library. We briefly introduce the necessary background to understand what features provided by OMNeT++ and Simu5G can be leveraged to realize the DTMN-SS, and we describe our implementation of the DTMN-SS. Then, we exemplify its usage in two use cases, namely the execution of a what-if analysis and the generation of a dataset for training ML models.

5.1. OMNeT++ and Simu5G support to DTMN-SS

OMNeT++ [19] is a framework that provides a simulation engine that implements all the basic components required to run discrete-event simulations, such as handling of the event queue and generation of (pseudo-)random number generators among the others, and it allows one to build custom network simulation models by combining modules from its large (and extensible) model library. Each module is fully

described by a Network Description (NED) file and a C++ class. The former is written in NED language and defines the module parameters and its submodules, as well as the connections among them and the gates that allow the module to exchange OMNeT++ messages with other modules. The C++ class, instead, specifies the module *behavior*, i.e., the actions that the module performs to handle events, such as the reception of messages from another module or the expiration of a self-scheduled timer. Note that decoupling module structure and module behavior grants final users great flexibility when it comes to build custom network topologies. To do that, a user just needs to use the NED language to specify which modules compose the network, and how they exchange messages through connections linking their gates. Fig. 8 shows two different network topologies composed of one source and one sink, connected via two routers. OMNeT++ allows the user to simulate both networks by just writing two different NED files, and without changing any line of C++ code defining the behavior of the modules.

Although a NED file defines the topology of the custom network model, it does not define specific parameters for the network itself or the included modules. Parametrization is performed independently via an initialization (INI) file, that assign actual parameters to the modules specified in the network topology. Again, note that decoupling parameters from the network topology allows users to define multiple parameterizations for the same network topology. Among the other parameters, the INI file also contains general simulation parameters, such as the simulation duration and the path of the folder that will contain the output results. One INI file can be separated in several sections, which are called *configurations*. Each configuration refers to a specific network topology (i.e., different NED files) and contains different sets of parameters. Fig. 9 exemplifies a INI file including a *General* section (mandatory) that specifies common parameters for the two subsequent configurations, namely *Config1* and *Config2*, which in turn refer to two different network topologies with different parametrization.

OMNeT++ provides wide support to simulation automation by simply configuring the INI file appropriately. In particular, a user can specify multiple values to one or more parameters. These are called *iteration variables*. The OMNeT++ environment will automatically generate a number of simulation *experiments* which is the cross-product of all the iteration variables in the INI file. The resulting number of experiments is multiplied by the configured number of independent replicas, i.e., the repetitions of the same experiments with different seed for the pseudo-random number generator, to obtain the total number of simulation instances. The latter can then be run in parallel exploiting multi-core capabilities of the underlying hardware. An example of usage of iteration variables is reported in Fig. 10.

Once the simulation campaign is terminated, statistics are stored in output files that can be parsed using the preferred tool or language, e.g., using python. OMNeT++ users can also leverage *scavetool*, a built-in program that allows to retrieve the desired metrics from the output files based on custom filters and produce them as output in different formats, e.g., comma-separated values, that can be digested by other software.

OMNeT++ comes with a large set of libraries that model several communication domains, which are developed by both OMNeT++ developers themselves and researchers from the community. Among them, Simu5G is a model library that provides the modules for simulating the data plane of both radio access and core networks of 4G and 5G technologies [20]. By being part of the OMNeT++ ecosystem, Simu5G interoperate with many other model libraries, such as INET for TCP/IP communications [33] and Veins for vehicular mobility [34]. The main modules of Simu5G are the ones representing UEs and base stations (or gNodeBs - gNBs). They both include a NewRadio (NR) Network Interface Card (NIC) submodule, which in turn includes submodules implementing all the layers of the NR protocol stack (i.e., PDCP, RLC, MAC and PHY), and provides data-plane packet processing and advanced functionalities like carrier aggregation, multiple numerologies, frequency- and (flexible) time-division duplexing. In order to reduce simulation complexity, the physical layer of the NR NIC is abstracted via channel models that simulate the effects of packet transmissions over the wireless medium (such as packet loss, fading/shadowing, etc.), rather than modeling the transmissions of individual radio symbols. Simu5G also includes modules for simulating the ETSI MEC infrastructure [35].

To summarize, the implementation of a DTMN-SS can leverage many of the capabilities offered by OMNeT++ and Simu5G. OMNeT++ offers all the built-in functionalities required to perform simulations, hence minimizing the effort required to the developers of a simulation model. Its modular architecture allows the DTMN-SS to build custom network models in a flexible way, and simulation automation support makes the configuration of (possibly large) simulation campaigns quite simple. For example, setting up a new campaign entails i) writing a NED file describing a parametric network topology that includes modules provided by the Simu5G library, and ii) writing an INI file specifying which experiments need to be evaluated, the actual values for the parameters of the network and its modules, e.g., taken from the data repository of the DTMN.

5.2. Implementation of a DTMN-SS proof-of-concept

In this section, we describe our first proof-of-concept implementation of the DTMN-SS presented in Section 4.3, using OMNeT++ and Simu5G as simulation libraries.

We built the DTMN-SS Server as a C++ program implementing an HTTP server, which exposes a set of resources that can be accessed using the conventional HTTP methods (i.e., GET, POST, etc.). Our implementation is based on the *libhttpserver* library available on GitHub.[2] The library provides a *WebServer* class that listen to incoming connections from the clients and defines the resources that they can access to configure their simulation campaign. Note that different clients will have access to a dedicated set of resources to let them run simulation campaigns independently. To accomplish this, the DTMN-SS Server initializes a session for each client upon receiving the first request from it, and includes a cookie in the response. Subsequent requests from the client will include the cookie, in order to allow the DTMN-SS Server to associate the request coming from a client to a specific session. The class diagram representing the resources exposed by the DTMN-SS Server and their relationships is depicted in Fig. 11.

Resources are defined by extending the *HttpResource* class. The *NetworkTopologyResource* class includes the definition of the network topology, i.e., includes all the information to create the NED file representing the network simulation model. The *ParametersResource* class stores the values that parametrize the network topology, i.e., the information that will compose the INI file. Both *NetworkTopologyResource* and *ParametersResource* classes may access the data repository of the DTMN in order to retrieve data corresponding to the physical network, when requested by the client and depending on the use case the DTMN-SS is used for. For example, consider the number of UEs served by a gNB as a relevant parameter to be configured for the simulation campaign. The client may indicate to set such parameter equal to the current number of UEs in the physical network, i.e., making the DTMN-SS Server fetch the corresponding real-time value from the data repository. Alternatively, the client may request to set the number of UEs to a value in the past, i.e., using historical data from the data repository. Finally, the client may force the simulation to use a custom value indicated by the UE explicitly. The *SimConfigResource* class maintains the general information about the simulation campaign, such as the simulation duration, the number of independent replicas for each experiment, and the environment, such as the number of simulation instances to be run in parallel. The above three resources expose the POST method to let the client specify how to populate the information within the

[2] https://github.com/etr/libhttpserver, accessed June 2023.

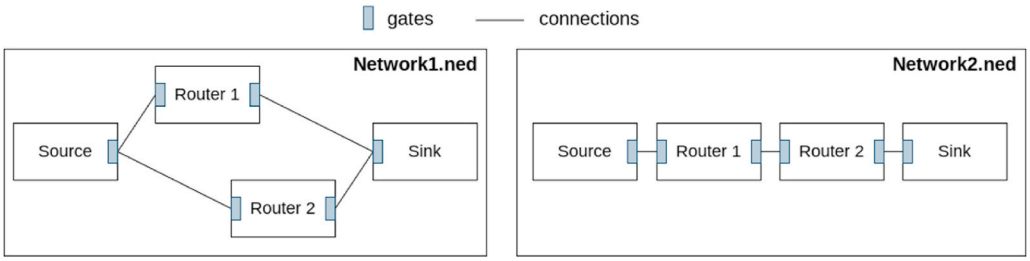


Fig. 8. Two examplary network topologies realized by composing the same modules in different ways.

```
[General]
# common parameters
sim-time-limit = 100s
...

[Config1]
network = network1.ned
# specific parameters for config1
...

[Config2]
network = network2.ned
# specific parameters for config2
...
```

Fig. 9. Example of INI file with two configurations using the two different networks.

```
[General]
# common parameters
sim-time-limit = 100s
repeat = 10
...

[Config1]
network = network1.ned
# specific parameters for config1
*.source.packetSize = {size=10, 100, 1000} B
*.source.period = {t=100, 1000} ms
...

[Config2]
network = network2.ned
# specific parameters for config2
...
```

Fig. 10. Usage of iteration variables in INI files: configuration Config1 defines iteration variables size and t, which in turn determine three and two values for packetSize and period parameters of the source module, respectively. Since each simulation is repeated 10 times, the simulation environment will produce a simulation campaign with 60 simulation instances.

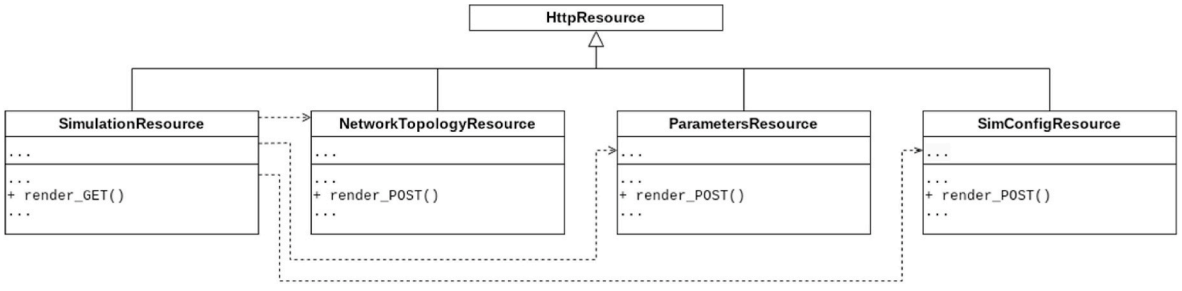


Fig. 11. Partial view of the class diagram representing the relationships among the HTTP resources exposed by our implementation of the DTMN-SS.

resource. The body of the POST message is expected to be in JSON format to facilitate parsing at the DTMN-SS Server.

The information included in the above three classes are combined together by the *SimulationResource* class, which is the core of the DTMN-SS Server implementation. When the client issues a GET request to this resource, the DTMN-SS Server invokes the *prepareSimulation()* function to retrieve all the necessary information from the other resources, and to collate them together in order to configure the simulation campaign. As a result, the network topology (obtained from *NetworkTopologyResource*) is written in the NED file, whereas its parametrization (obtained from *ParametersResource*) and the general simulation settings (obtained from *SimConfigResource*) are written in the INI file. Then, the *runSimulation()* method is executed to run the simulation campaign. This will trigger a number of simulation instances that depends on the number of experiments and independent replicas defined in the INI file. After the simulations are terminated, statistics need to be extracted from the output files. Since Simu5G (like other simulators) produces a large number of metrics, the DTMN-SS extracts only those requested by the client. The

client can give such indication directly in the query string of the GET request. The extraction of the relevant statistics is made by invoking the *scavetool* program to filter the requested metrics, and by producing a JSON file including the mean, variance, standard deviation and 95 % confidence intervals of each of them, for each configured experiment. The JSON file is sent back to the client in the body of the response message.

5.3. Use cases

In this subsection we exemplify the usage of the above proof-of-concept implementation of the DTMN-SS. First, we will show the use case of a client (e.g., an application from the MN operator) using the simulation services of the DTMN to perform a what-if analysis. Then, we will demonstrate how the DTMN-SS can be used to support network orchestration policies, such as allocation of edge-based application. Finally, we will show the use case of a client (e.g., an SMM internal to the DTMN) that needs to generate a dataset to improve the training of a ML model representing the physical network.

5.3.1. What-if analysis

Let us consider a MN operator who wants to make online decisions about whether to (de)activate carrier aggregation (CA) on one gNB is convenient or not. On one hand, CA allows the gNB to increase the overall cell throughput by using additional bandwidth, i.e., adding one or more carrier components (CCs). On the other hand, using more bandwidth may increase the gNB's power consumption. The performance of the alternative configurations cannot be done in the physical network without affecting the performance of connected UEs in an unpredictable fashion, hence the DT of the physical network may come in handy to perform a BWI analysis. Thus, we assume that the MN operator runs an application that needs to use the DTMN-SS to assess the evolution of the physical network with two different policies:

- CA-1CC: one CC is used. The gNB's scheduler allocates all the UEs in the only CC available;
- CA-2CC: two CCs are used. The gNB's scheduler allocates UEs in the CC they perceive the highest channel quality (i.e., the highest CQI).

We assume that for this BWI analysis the client will go through a number of consecutive checkpoints, with a duration of 20 s each.

With reference to Fig. 12, we discuss the interactions between the client and the DTMN-SS that realize the use case. The first action of the client is to configure the *SimConfigResource* in the DTMN-SS. This is done

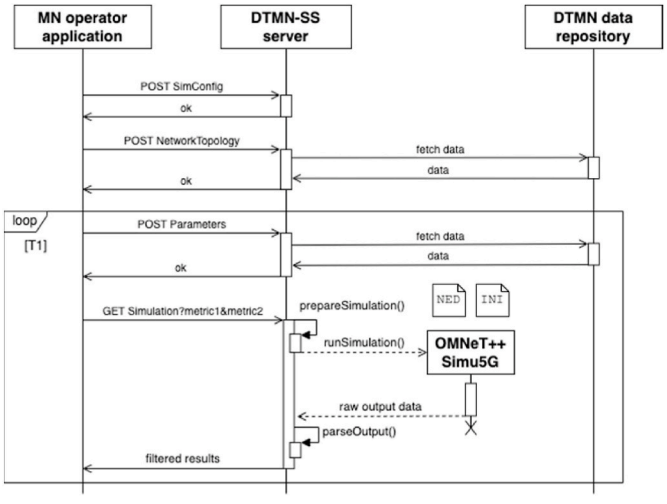


Fig. 12. Interactions between the MN operator application and the DTMN-SS Server for the BWI analysis use case.

by sending a POST request, with a body similar to the one exemplified in Fig. 13, which specifies the duration of each simulation run, the number of independent replicas for each experiment, the maximum number of simulations to run in parallel, and the name of the two policies to be compared. The latter will be used as names for the two different configurations in the INI file.

Next, the client needs to define the network topology, hence it issues a POST request to the *NetworkTopologyResource*. Since the what-if analysis is meaningful if the simulated network topology is the same as the underlying physical network, the body of the request asks the DTMN-SS to obtain information about the topology from the data repository of the DTMN. Note that this operation is performed once, since the network topology will not change from one checkpoint to the next one.

Then, the simulation model must be parametrized, and each checkpoint is likely to have different parameters, because the network conditions (e.g., the network traffic, the number of connected UEs) may vary. Thus, the client starts a loop according to which, at each checkpoint (on each period $T1 = 20s$), it configures the simulation model with new network parameters, triggers a new simulation campaign and obtain the relevant statistics. First, parameters are configured via a POST request to the *ParameterResource*. Also in this case, most of the parameters must be assigned their corresponding real-time value from the data repository. However, the parameters characterizing the two policies under evaluation needs to be indicated by the client itself in the body of the request message, in order to force the simulations to assess the alternative scenarios.

Once all the parameters are ready, the client can start the simulation campaign. This is accomplished by issuing a GET request to the *SimulationResource*, whose query string includes the metric under observation. In our case, these are the overall cell throughput and the cell power consumption. The DTMN-SS Server then creates the NED and INI file deriving from all the information configured before and launches the simulation campaign. The response to the client includes the average values of the two requested metrics, along with their variance, standard deviation and 95 % confidence interval, for both the policies under evaluation. An example of the response message is shown in Fig. 14.

We now show the results of a BWI analysis carried out according to the procedures above in a toy network scenario. Since we did not have available a real mobile network for extracting real data, we tested the above DTMN-SS prototype assuming to have a data repository that includes data for a toy network. The network is composed of the gNB under observation (i.e., the one subject to the what-if analysis) that serves a variable number of UEs and is surrounded by three interfering gNBs. UEs are receiving downlink traffic from a remote server, connected to the gNB through the mobile core network. Fig. 15 shows the resulting topology from the NED file generated by the DTMN-SS Server. In this example, we run the what-if analysis for 10 min, resulting in 30 total checkpoints. Network parameters to configure the simulations at each checkpoint are generated randomly. In particular, we assumed a scenario where a random number of UEs is added/removed at each checkpoint, hence emulating the behavior of UEs entering/leaving the coverage area of the gNB during the time frame between two checkpoints. Fig. 16 shows the number of UEs that is stored in the data repository at each checkpoint. As far as their position is concerned, the geographical coordinates for each UE are selected randomly and, from a checkpoint to the next, we assume the UEs travel a random distance between 0 and 50 m (i.e., assuming that UEs are pedestrians moving at a maximum speed of 6 km/h). As a result, the DTMN-SS will end up configuring the simulation campaign at each checkpoint according to different number and position of UEs, producing different output in terms of the metrics under evaluation.

We plot the mean and the confidence interval of the obtained results for each checkpoint. Figs. 17 and 18 report the overall cell throughput in the downlink and the total power consumed by the gNB, respectively. The power consumption is computed according to the power model

```
{
  "simDuration" : 20,
  "repetitions" : 10,
  "maxProc" : 10
  "config" : ["CA-1CC", "CA-2CC"],
}
```


Fig. 13. Body of the POST request on the *simconfig* resource, in JSON format.


```
{
  "avgCellPowerConsumption": {
    "CA-2CC": {
      "gnb": {
         "confidenceinterval": "8.35809492452258",
         "mean": "431.09884121495",
         "stddev": "12.7930024354937",
         "variance": "163.660911314549"
      }
    },
    "CA-1CC": {
      "gnb": {
        "confidenceinterval": "33.1438466022848",
        "mean": "395.106800049997",
        "stddev": "50.7303774524767",
        "variance": "2573.57119647075"
      }
    }
  },
  "macCellThroughputDl": {
    "CA-2CC": {
      "gnb": {
        "confidenceinterval": "6931.49815209419",
        "mean": "890019.800981509",
        "stddev": "10609.4359470829",
        "variance": "112560131.115256"
      }
    },
    "CA-1CC": {
      "gnb": {
      "confidenceinterval": "23919.6162153884",
      "mean": "879543.462024633",
      "stddev": "36611.6574725333",
      "variance": "1340413462.88611"
    }
  }
}
```


Fig. 14. Body of the response sent by the DTMN-SS to the client, in JSON format.

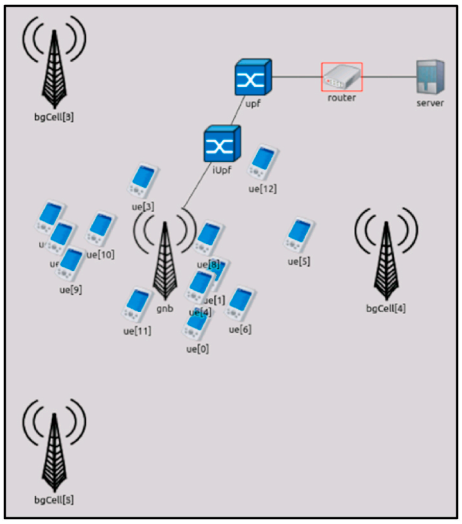


Fig. 15. Visual representation of the network topology.

taken from Ref. [36] and parametrized according to the values in Ref. [37] for a macro base station:

$$P = N_{CC} \times \left(P_0 + \Delta_p P_{max} \chi\right),$$

where N_{CC} is the number of active CCs, $P_0 = 114.5W$ is the base power, $\Delta_p = 588.1W$ is the slope of the load-dependent power consumption, $P_{max} = 30dBm = 1W$ is the maximum transmission power of the gNB,

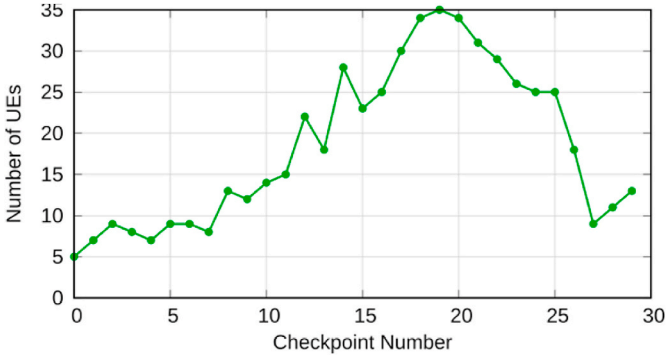


Fig. 16. Number of UEs stored in the data repository at each checkpoint.

and χ is the utilization ratio of the available bandwidth.

The charts show that in the first 12 checkpoints (i.e., 4 minutes of the real network), the two policies do not differ in terms of throughput achieved by the gNB, since the offered load is low enough to serve all the traffic with one CC only. However, activating the second CC would consume more power than the case with one CC only. Thus, the network operator would benefit from switching off the second CC, while still be able to satisfy UEs' traffic requests. When the traffic load increases after 5 min, the CA-1CC policy gets to its capacity upper limit, and the cell throughput cannot go above 1.8 Mbps. Instead, the CA-2CC policy would be able to achieve higher throughput, at the cost of slightly higher power consumption. In this situation, the network operator might choose to prioritize the quality of service for its UEs, hence it might

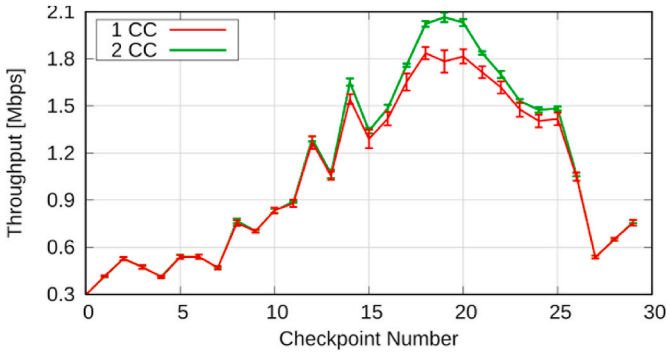


Fig. 17. Average cell throughput at the MAC layer.

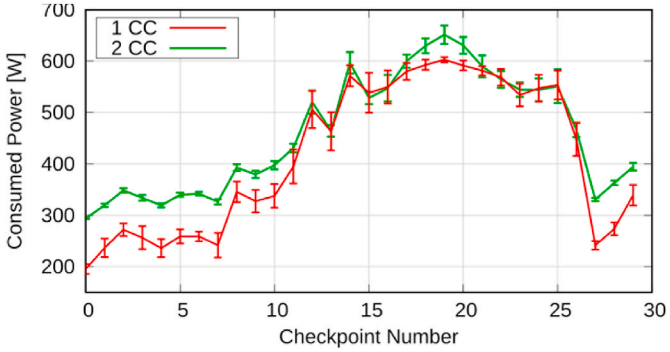


Fig. 18. Average cell power consumption.

decide to activate the second CC. Finally, when the traffic load decreases after the 27th checkpoint, the CA-1CC policy becomes preferable again.

Assuming that the client requesting the above what-if analysis is a resource management application of the MN operator, the latter can compare the results of the two scenario *on-the-fly*, i.e., as results for each checkpoint are provided by the DTMN-SS Server. Thus, at each checkpoint the management application can select the policy with the best expected results and enforce it at the base stations of the PT.

5.3.2. Support to orchestration policies

Our proposed DTMN-SS can come in handy to support decision processes in MNs. In this subsection, we exemplify this use case considering a MN assisted with a Multi-access Edge Computing (MEC) system. Despite MEC having been standardized by the European Telecommunications Standards Institute (ETSI) as an edge computing platform agnostic with respect to the network access technology, MEC finds its natural application in MNs and, in particular, in 5G networks [30]. Indeed, edge computing is expected to play a key role in beyond-5G/6G networks [38], thus it is foreseeable that it will be integral part of future MNs and their DTs.

With reference to Fig. 19, we consider a simple network with one gNB and two MEC hosts, namely *MEC Host 1* and *MEC Host 2*. We assume that MEC Host 2 has more computing power than MEC Host 1 but it is located farther away from the gNB, as the path between the gNB and MEC Host 2 requires traversing several User Plane Function (UPF) nodes. Specifically, we assume that MEC Host 1 and MEC Host 2 can execute 500 and 5000 Million Instructions Per Seconds (MIPS), respectively, and that the latency between *upf* and *iUpf2* in Fig. 19 introduces a random, non-negligible delay ranging from 5 to 10 ms. In this context, we consider that a number of consecutive MEC app instantiation requests are issued by UEs to the MEC orchestrator. The latter is responsible to allocate the required resources for running the MEC app (as a virtual machine or container) on either of the two MEC hosts. This is usually accomplished according to some orchestration policy within the

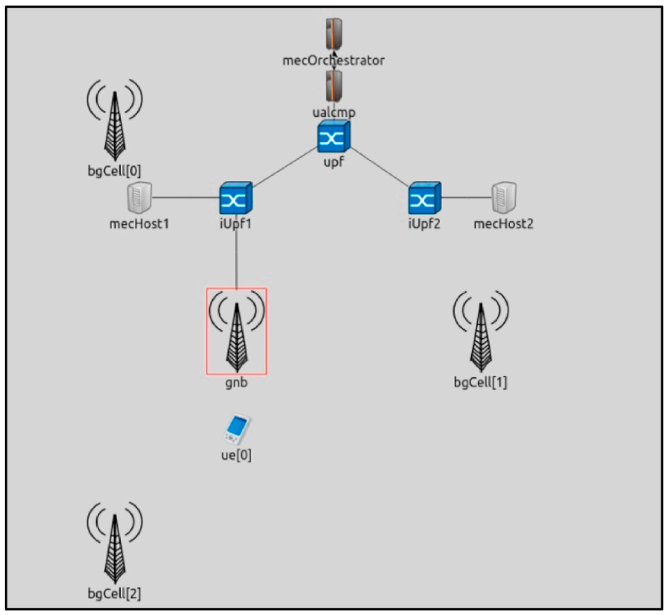


Fig. 19. Visual representation of the network topology with one gNB and two MEC Hosts.

MEC orchestrator.

As a viable orchestration option, the MEC orchestrator can leverage the capabilities of a DTMN to support such allocation decisions: the MEC orchestrator can act as a DTMN-SS Client requesting the DTMN-SS Server to conduct simulations with the objective of estimating the most suitable choice about where to instantiate new MEC apps. In more detail, one iteration of our scenario works as follow:

1. a UE requests the MEC orchestrator to instantiate a new MEC app in order to perform task offloading. This means that the MEC app will receive periodic tasks from the UE, execute it at the MEC and send back the output to the UE;
2. in order to select the most suitable MEC Host, the MEC orchestrator exploits the DTMN-SS to configure a simulation campaign that evaluates the round-trip time (RTT) that the UE would experience in case its corresponding MEC app is instantiated on either MEC Host 1 or MEC Host 2. Clearly, the RTT is affected by both the network latency required to reach the MEC Host and the processing delay at the MEC app, which in turn depends on the MEC Host capabilities and the number of MEC apps currently running on it. We assume that the simulation campaign computes the RTT in the two cases, averaged over a time window of 20s;
3. The MEC orchestrator instantiates the MEC app on the MEC Host that provides the lower RTT to the UE according to the results of the simulation campaign. This choice will contribute to increase the load on the selected MEC host, thus subsequent MEC app instantiation requests may obtain different RTT values.

For the sake of conciseness, we do not show all the interactions between the DTMN-SS Client and Server that are needed to realize this use case – they are similar to the ones discussed in the previous subsections, except for the type of parameters passed to DTMN-SS Server and the metrics returned by it (i.e., the RTT).

Fig. 20 shows, for each instantiation request coming from a UE, the average RTT of the task offloading procedure in case the MEC app is instantiated on MEC Host 1 (green bars) or MEC Host 2 (red bars). At the reception of the first request, both MEC Hosts have no running MEC apps and the RTT that the UE would experience if the MEC app is instantiated on MEC Host 1 is roughly half the RTT that it would obtain if the MEC app is instantiated on MEC Host 2. This is due to the higher

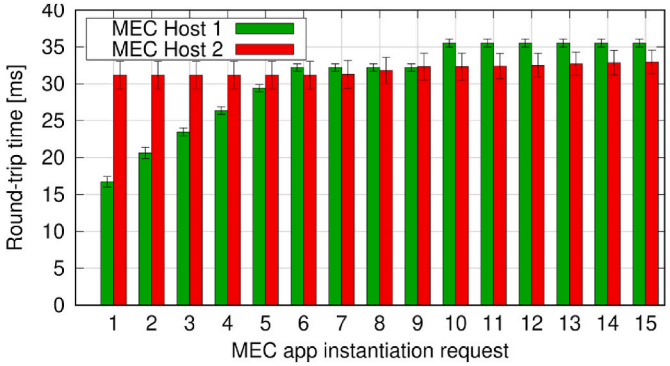


Fig. 20. Comparison of RTT for each MEC app instantiation request.

communication latency required to reach MEC Host 2 through the core network. Thus, the MEC orchestrator decides that the first MEC app is allocated on MEC Host 1. In the subsequent requests, the RTT difference gets smaller as MEC Host 1 uses more resources to accommodate an increasing number of MEC apps, as shown in Fig. 21. Upon the sixth request, the load at MEC Host 1 has grown to the extent that the RTT would be higher than the RTT in case the MEC app is instantiated on MEC Host 2. Thus, the MEC orchestrator instantiates the sixth MEC app on MEC Host 2. The same occurs until the ninth request, where MEC Host 1 becomes more beneficial again. From that point onward, MEC Host 2 will be able to provide lower RTT to all the MEC apps. Note that instantiating a new MEC app on MEC Host 2 has limited impact on the RTT with respect to the impact of a new MEC app on MEC Host 1, since the computing capabilities of MEC Host 2 are much higher than those of MEC Host 1.

To summarize, this subsection showed how the DTMN-SS can be used to support specific decisions that are based on the actual status of the MN, making it a valid alternative to decisions based on predefined algorithms. The same approach could be used, for example, to assist cell selection procedures, i.e., to select which gNB a new UE entering the MN should attach to.

5.3.3. *Dataset generation for ML models*

We now assume that the SMMs of the DTMN include a service that is responsible to train ML models representing the physical network. In the following, we refer to that as ML Training Service (MLTS). To accomplish its task, the MLTS needs data in order to train its learning algorithm, and such data can be taken from the data repository. However, such data may be not enough to build an accurate model. For example, the data found in the data repository may not have the desired time-space granularity required by the training algorithm, or the required amount of data for a specific network condition is not available, such as unusual heavy traffic due to rare events (e.g., a crowded sport match). As a result, the MLTS may be unable to provide accurate ML model of the PT, simply because of the low quality of the training dataset. In this case, simulations may be helpful to generate synthetic, yet realistic, datasets in specific network scenarios that can be used to augment those obtained by the data repository. We showed how Simu5G can be used to such purpose in Ref. [32]. In this section, we discuss how the proposed DTMN-SS can support this use case as well.

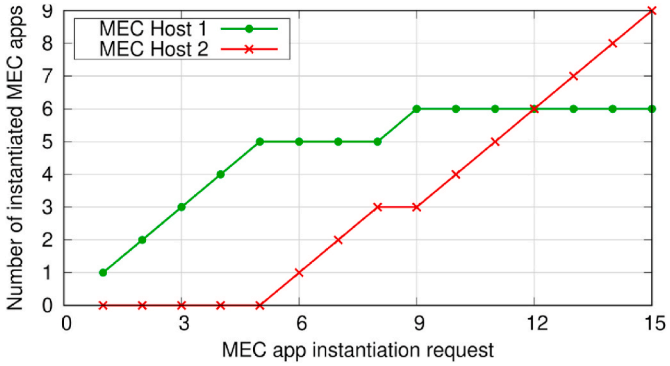


Fig. 21. Evolution over time of the load on the MEC Hosts.

We consider the same network topology as in Section 5.3.1 and illustrated in Fig. 15, where the MLTS within the DTMN wants to train a ML model to predict the quality of service experienced by UEs served by the central gNB when the latter is highly loaded, i.e., serving a large number of UEs. Since the data repository does not contain enough historical data obtained in such a situation, the MLTS leverages the DTMN-SS. Fig. 22 shows the required interactions to realize this use case. First, the MLTS configures the *SimConfigResource*, using a POST request whose body is shown in Fig. 23. Unlike the what-if use case, now the simulation duration may be longer (e.g., 86400 s in Fig. 23, corresponding to one whole day), in order to produce more data. Moreover, running one replica is enough, since the MLTS just needs a long trace of raw data and is not interested in the statistical confidence of the metrics this time. The field specifying the name of the configuration is not required since only one configuration is simulated.

As far as the *NetworkTopologyResource* is concerned, the same configuration as the what-if analysis use case can be used: the simulated network topology is set to be the same as the underlying PT by fetching relevant data from the data repository.

When it comes to configure the *ParameterResource*, the MLTS does not want to rely on either real-time or historical data from the data repository, for the reasons discussed above. Instead, it explicitly communicates the network parametrization according to the hypothetical heavily-loaded scenario it wants to reproduce. Thus, the POST request message to the *ParameterResource* contains all the required parameters. A snippet of an exemplary configuration is reported in Fig. 24: 100 UEs are deployed in the positions indicated by the uePos array, and each of them produce 200 kB/s of data traffic.

Finally, the GET request on the *SimulationResource* triggers the simulation execution. At the end, the metrics indicated in the query string are reported to the MLTS. In the example of Fig. 22, the wildcard '*' is used in the query string to obtain all the metrics whose name starts with app, i.e., to obtain all the metrics collected at the application layer. All the above metrics can be used by the MLTS to build a dataset, possibly after some post-processing (such as averaging, pruning, etc.),

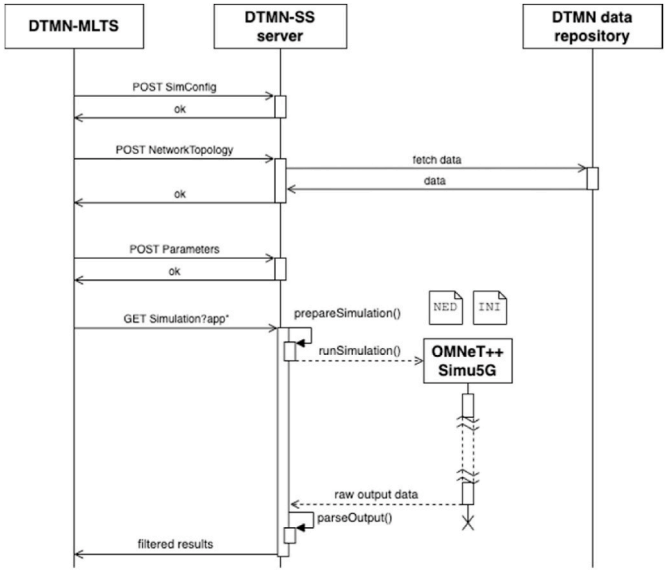


Fig. 22. Interactions between the DTMN ML training service and the DTMN-SS Server for the dataset generation use case.

```
{
  "simDuration" : 86400,
  "repetitions" : 1,
  "maxProc" : 1
}
```

Fig. 23. Body of the POST request on the *SimConfigResource*, in JSON format.

```
{
  "numUEs" : 100,
  "uePos" : [
    {
       "x" : 1074.1143243656013,
       "y" : 1154.438384022374
    }
    ...
  ],
  "ueTraffic" : [
    {
      "packetSize" : 2000,
      "sendPeriod" : 0.01
    }
    ...
  ]
  ...
}
```

Fig. 24. Body of the POST request on the *ParameterResource*, in JSON format.

that can in turn be exposed to other SMMs or applications, e.g., to perform ML-based predictions.

Comparing the operations against the ones in Section 5.3.1, we observe that in this case there are no loops. The simulation campaign is run once, although it is significantly longer than in the case of BWI analysis. Furthermore, for the BWI analysis the DTMN-SS Server needs to contact the DTMN data repository twice in order to handle the requests on both *NetworkTopologyResource* and *ParameterResource*, whereas for the ML dataset generation the *ParameterResource* is configured with custom data from the MLTS. In conclusion, we showed that the designed interface is flexible, as it allows one application or service to achieve different objectives by using the same resources in different ways.

6. Conclusions and future work

This work studied the challenges related to the development of Digital Twins of mobile networks, which span from scalability and DT-PT synchronization issues to security and deployment ones, among the others. We motivated the opportunities offered by network simulators to integrate models and services (e.g., machine learning) within DTMNs, as well as the use cases they can enable, and we proposed a framework compatible with standardization efforts by IETF to endow DTMNs with *simulation services*. Finally, we described a proof-of-concept implementation of the proposed framework using OMNeT++ and the Simu5G simulation library, showing how such framework can be leveraged to support applications and DTMNs' integrated services, such as the evaluation of what-if scenarios and the generation of synthetic datasets related to unforeseen scenarios for the training of ML models within the DTMN.

Future works include enhancing the proposed architecture to support the simulation of multi-scale network models having different requirements in terms of, e.g., speed and accuracy, as well as the orchestration of simulation instances in order to efficiently manage the computing resources of the infrastructure hosting DTMNs.

Author statement

Giovanni Nardini: Conceptualization, Methodology, Software, Data curation, Investigation, Writing – original draft, Writing – review & editing; Giovanni Stea: Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervising.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Simu5G software is open source and publicly available on GitHub

Acknowledgments

Work partially supported by the Italian Ministry of Education and Research (MUR) in the framework of the FoReLab projects (Departments of Excellence).

References

[1] F. Tao, H. Zhang, A. Liu, A.Y.C. Nee, Digital twin in industry: state-of-the-art, IEEE Trans. Ind. Inf. 15 (4) (2019) 2405–2415.
[2] W. Kritzinger, M. Karner, G. Traar, J. Henjes, W. Sihn, Digital Twin in manufacturing: a categorical literature review and classification, IFAC-PapersOnLine 51 (11) (2018) 1016–1022, https://doi.org/10.1016/j.ifacol.2018.08.474. ISSN 2405-8963.
[3] R. Minerva, G.M. Lee, N. Crespi, Digital twin in the IoT context: a survey on technical features, scenarios, and architectural models, 10, in: Proceedings of the IEEE, vol. 108, 2020, pp. 1785–1824, https://doi.org/10.1109/JPROC.2020.2998530. October.
[4] H. Ahmadi, A. Nag, Z. Khar, K. Sayrafian, S. Rahardja, Networked twins and twins of networks: an overview on the relationship between digital twins and 6G, 4, in: IEEE Communications Standards Magazine vol. 5, 2021, pp. 154–160, https://doi.org/10.1109/MCOMSTD.0001.2000041. December.
[5] M. Vaezi, et al., Digital twins from a networking perspective, 23, in: IEEE Internet of Things Journal, vol. 9, 2022, pp. 23525–23544, https://doi.org/10.1109/JIOT.2022.3200327, 1 Dec.1.
[6] A. Lombardo, G. Morabito, S. Quattropani, C. Ricci, Design, implementation, and testing of a microservices-based Digital Twins framework for network management and control, in: 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2022, pp. 590–595, https://doi.org/10.1109/WoWMoM54355.2022.00092. Belfast, United Kingdom.
[7] G. Lin, J. Gel, Y. Wu, H. Li, L. Li, Digital twin networks: learning dynamic network behaviors from network flows, in: 2022 IEEE Symposium on Computers and Communications (ISCC), 2022, pp. 1–6, https://doi.org/10.1109/ISCC55528.2022.9912864. Rhodes, Greece.
[8] L. Zhao, G. Han, Z. Li, L. Shu, Intelligent digital twin-based software-defined vehicular networks, in: IEEE Network, vol. 34, 2020, pp. 178–184, https://doi.org/10.1109/MNET.011.1900587. September/October.
[9] M. Kherbache, M. Maimour, E. Rondeau, Network digital twin for the industrial Internet of things, in: 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM), Belfast, United Kingdom, 2022, pp. 573–578, https://doi.org/10.1109/WoWMoM54355.2022.00089.

[10] C. Zhou, et al., Digital twin network: concept and reference architecture, in: Internet Draft Draft-Zhou-Nmrg-Digitaltwin-Network-Concepts-07, 2022, 5 March.
[11] C. Zhou, et al., Data collection requirements and technologies for digital twin network, in: Internet Draft Draft-Zcz-Nmrg-Digitaltwin-Data-Collection-01, 2022, 7 November.
[12] J. Paillisse, et al., Performance-Oriented Digital Twins for Packet and Optical Networks (v2), in: IETF Network Management Research Group, 2022. https://datatracker.ietf.org/doc/html/draft-paillisse-nmrg-performance-digital-twin-02. (Accessed 23 August 2023), 24 October.
[13] H.X. Nguyen, R. Trestian, D. To, M. Tatipamula, Digital twin for 5G and beyond, in: IEEE Communications Magazine, vol. 59, February 2021, pp. 10–15, https://doi.org/10.1109/MCOM.001.2000343, 2.
[14] A. Mozo, A. Karamchandani, M. Sanz, J.I. Moreno, A. Pastor, B5GEMINI: digital twin network for 5G and beyond, Budapest, Hungary, in: NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, 2022, pp. 1–6, https://doi.org/10.1109/NOMS54207.2022.9789810. April.
[15] M. Farreras, P. Soto, M. Camelo, L. Fàbrega, P. Vilà, Predicting network performance using GNNs: generalization to larger unseen networks, Budapest, Hungary, in: NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, 2022, pp. 1–6, https://doi.org/10.1109/NOMS54207.2022.9789766.
[16] N.P. Kuruvatti, M.A. Habibi, S. Partani, B. Han, A. Fellan, H.D. Schotten, Empowering 6G communication systems with digital twin technology: a comprehensive survey, in: IEEE Access, vol. 10, 2022, pp. 112158–112186, https://doi.org/10.1109/ACCESS.2022.3215493.
[17] ns3 Network Simulator" Website: https://www.nsnam.org, accessed January 2023.
[18] N. Patriciello, S. Lagen, B. Bojovic, L. Giupponi, An E2E simulator for 5G NR networks, Simulat. Model. Pract. Theor. 96 (2019), https://doi.org/10.1016/j.simpat.2019.101933.
[19] OMNeT++ Discrete Event Simulator" Website: https://omnetpp.org, accessed January 2023.
[20] G. Nardini, D. Sabella, G. Stea, P. Thakkar, A. Virdis, Simu5G – an OMNeT++ library for end-to-end performance evaluation of 5G networks, IEEE Access (2020), https://doi.org/10.1109/ACCESS.2020.3028550.
[21] G. Nardini, G. Stea, A. Virdis, Scalable real-time emulation of 5G networks with Simu5G, IEEE Access 9 (2021) 148504–148520, https://doi.org/10.1109/ACCESS.2021.3123873.
[22] Y. Gao, H. Lv, Y. Hou, J. Liu, W. Xu, Real-time modeling and simulation method of digital twin production line, in: 2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), 2019, pp. 1639–1642.
[23] M. Schluse, J. Rossmann, From simulation to experimentable digital twins: simulation-based development and operation of complex technical systems, in: IEEE International Symposium on Systems Engineering (ISSE), 2016, pp. 1–6.
[24] U. Dahmen, J. Rossmann, Experimentable digital twins for a modeling and simulation-based engineering approach, in: 2018 IEEE International Systems Engineering Symposium (ISSE), 2018, pp. 1–8.
[25] M. Barbi, A.A. Ruiz, A.M. Handzel, S. Inca, D. Garcia-Roger, J.F. Monserrat, Simulation-based digital twin for 5G connected automated and autonomous vehicles, in: 2022 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), 2022, pp. 273–278, https://doi.org/10.1109/EuCNC/6GSummit54941.2022.9815732. Grenoble, France.
[26] G. Nardini, G. Stea, Using Network Simulators as Digital Twins of 5G/B5G Mobile Networks, TwinNets2022, Belfast, UK, 2022, pp. 14–17. June.
[27] 3GPP TS 23.288 v.17.4.0, 5G; Architecture Enhancements for 5G System (5GS) to Support Network Data Analytics Services (Release 17), 2022. May.
[28] A. Arnaz, J. Lipman, M. Abolhasan, M. Hiltunen, Toward integrating intelligence and programmability in open radio access networks: a comprehensive survey, in: IEEE Access, vol. 10, 2022, pp. 67747–67770, https://doi.org/10.1109/ACCESS.2022.3183989.
[29] B.R. Barricelli, E. Casiraghi, D. Fogli, A survey on digital twin: definitions, characteristics, applications, and design implications, in: IEEE Access, vol. 7, 2019, pp. 167653–167671.
[30] S. Kekki, MEC in 5G networks, 2018. June, https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf. (Accessed 28 August 2023).
[31] ETSI GS MEC 013 v2.2.1, "Multi-access Edge Computing (MEC); Location API", January 2022.
[32] G. Nardini, A. Noferi, P. Ducange, G. Stea, Exploiting Simu5G for generating datasets for training and testing AI models for 5G/6G network applications, Elsevier SoftwareX 21 (2023), 101320, https://doi.org/10.1016/j.softx.2023.101320. ISSN 2352-7110.
[33] INET Framework" Website: https://inet.omnetpp.org, accessed January 2023.
[34] C. Sommer, R. German, F. Dressler, Bidirectionally coupled network and road traffic simulation for improved IVC analysis, 1, in: IEEE Transactions on Mobile Computing, vol. 10, 2011, pp. 3–15, https://doi.org/10.1109/TMC.2010.133. Jan.
[35] A. Noferi, G. Nardini, G. Stea, A. Virdis, Rapid prototyping and performance evaluation of ETSI MEC-based applications, Elsevier Simulat. Model. Pract. Theor. 123 (2023), 102700, https://doi.org/10.1016/j.simpat.2022.102700. February.
[36] Y.-N.R. Li, M. Chen, J. Xu, L. Tian, K. Huang, Power saving techniques for 5G and beyond, in: IEEE Access, vol. 8, 2020, pp. 108675–108690, https://doi.org/10.1109/ACCESS.2020.3001180.
[37] B. Debaillie, C. Desset, F. Louagie, A flexible and future-proof power model for cellular base stations, in: 2015 IEEE 81st Vehicular Technology Conference (VTC Spring), 2015, pp. 1–7.
[38] E. Peltonen, et al., 6G White Paper on Edge Intelligence, 2020 arXiv:2004.14850, April.
[39] P. Almasan, et al., Digital Twin Network: Opportunities and Challenges, 2022 arXiv:2201.01144v2 [cs.NI], 7 Jan.
[40] Y. Wu, K. Zhang, Y. Zhang, Digital twin networks: a survey 18, 18, IEEE Internet of Things Journal 8 (2021) 13789–13804, https://doi.org/10.1109/JIOT.2021.3079510, 15 Sept.15.