Nervion: A Cloud Native RAN Emulator for Scalable and Flexible Mobile Core Evaluation

Jon Larrea
The University of Edinburgh
UK

Mahesh K. Marina
The University of Edinburgh
UK

Jacobus Van der Merwe
University of Utah
USA

ABSTRACT

Given the wide interest on mobile core systems and their pivotal role in the operations of current and future mobile network services, we focus on the issue of their effective evaluation, considering the radio access network (RAN) emulation methodology. While there exist a number of different RAN emulators, following different paradigms, they are limited in their scalability and flexibility, and moreover there is no one commonly accepted RAN emulator. Motivated by this, we present Nervion, a scalable and flexible RAN emulator for mobile core system evaluation that takes a novel cloud-native approach. Nervion embeds innovations to enable scalability via abstractions and RAN element containerization, and additionally supports an even more scalable control-plane only mode. It also offers ample flexibility in terms of realizing arbitrary RAN emulation scenarios, mapping them to compute clusters, and evaluating diverse core system designs. We develop a prototype implementation of Nervion that supports 4G and 5G standard compliant RAN emulation and integrate it into the Powder platform to benefit the research community. Our experimental evaluations validate its correctness and demonstrate its scalability relative to representative set of existing RAN emulators. We also present multiple case studies using Nervion that highlight its flexibility to support diverse types of mobile core evaluations.

CCS CONCEPTS

• **Networks** → **Mobile networks**; *Network performance evaluation*; *Cloud computing*.

KEYWORDS

Mobile Core Systems, Evaluation, RAN Emulation, Cloud Native

ACM Reference Format:

Jon Larrea, Mahesh K. Marina, and Jacobus Van der Merwe. 2022. Nervion: A Cloud Native RAN Emulator for Scalable and Flexible Mobile Core Evaluation. In *The 27th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '21), January 31-February 4, 2022, New Orleans, LA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3447993.3483248

ACM MobiCom '21, January 31-February 4, 2022, New Orleans, LA, USA

ACM ISBN 978-1-4503-8342-4/22/01…$15.00
https://doi.org/10.1145/3447993.3483248

1 INTRODUCTION

The core is a key mobile network component that handles the essential control functionality such as mobility, session and security management, and also bridges data communication between end devices and external networks like the Internet via the radio access network (RAN). Driven by the limitations of the traditional mobile network architecture and the wider trends in the mobile network industry, such as embracing network softwarization and supporting diverse use cases, mobile core networks have received a great deal of attention from the standardization bodies [1–4] as well as research [5–21] and open source [22–26] communities.

In this paper, we focus on enabling the evaluation of mobile core systems in terms of their scalability, responsiveness, support for diverse services, robustness and so on. This is considering the ever more critical role of mobile networks play in society, including their use in public safety, and given the significant part the mobile core plays in their operations. An effective evaluation methodology to this end should be able to realistically model the RAN in terms of scale and control/data plane workload patterns.

RAN emulation has become the commonly used mobile core system evaluation methodology, as reflected by [6–21], compared to other alternatives like testbed-based experimental evaluation, simulation and mathematical analysis. The general idea behind RAN emulation is to emulate the RAN elements – base stations and end devices (UEs) – from the perspective of the core system under test. RAN emulation based core evaluation can also allow seamless transition to evaluation over a testbed or production network.

As discussed in §5, existing RAN emulators fall into four categories: full stack emulation, commercial RAN emulators, trace based emulation and ad-hoc RAN emulators. The full stack emulation approach represented by OAI [27] and srsLTE [23] has severe scalability limitation and so has not been used for core system evaluation. Commercial RAN emulators, owing to their limited access within the research community, are also rarely used in the literature, except for openEPC [28] which is accessible via Powder [29] and earlier Phantomnet [30] testbed infrastructures and used in [9, 13, 14]. The trace based emulation approach used in [7, 8, 11, 12, 17, 21] has the inherent limitation of being inflexible as the RAN workload patterns cannot generalize beyond the underlying trace. The last category of ad-hoc emulators [6, 15, 18–20, 31–33] are commonly used [6, 10, 15, 18–20, 34, 35] but, as the name suggests, are tied to the target core system they are designed for. These ad-hoc emulators are also limited in their scalability (e.g., due to being single process applications) and flexibility (e.g., support only control plane or data plane but not both). Crucially, there is no one commonly accepted RAN emulator that can enable comparative evaluation of different mobile core systems.

Motivated by the above, we present Nervion as a versatile RAN emulator that takes a novel cloud-native approach to overcome the

aforementioned limitations of existing solutions while maintaining fidelity of RAN emulation from a mobile core system evaluation perspective. The Nervion system architecture is designed with scalability and flexibility in mind. To achieve scalability, Nervion employs three techniques: (1) *abstraction*: Nervion abstracts away RAN internals and protocol layers that are unimportant from the core system evaluation perspective and this allows for lightweight realization of RAN elements (UEs and base stations); (2) *refactoring*: based on the insight that UE control plane functionality is mediated through its associated base station (eNodeB in 4G and gNodeB in 5G) while the source/sink for data plane traffic resides in the UE, UE functionality in Nervion is refactored so that emulated UEs (called nUEs in Nervion) only handle the data plane whereas control plane behavior of a UE is proxied by the corresponding emulated base station in Nervion (called nBS) along with base station's own control plane functionality; and (3) *containerization*: the above refactoring allows for nUEs and nBSs to be realized as largely separate elements with minimal interaction via the Nervion controller. In Nervion, we containerize nUE and nBS so that desired number of their instances (as per the experiment needs) can be created across multiple virtual or physical machines, and optionally allowing multiple nUEs per container as separate threads for control-plane focused evaluations.

Nervion provides flexibility in multiple forms. Firstly, it uses the notion of an *emulation scenario* to provide a detailed "specification" of the evaluation scenario, including the number and configuration of the RAN elements as well as RAN control and data plane workload patterns. Control plane workload in a scenario is defined as a user-defined sequence of control plane events with specified intervals between them. The containerization of nUE and nBS allows for a TUN device [36] to be attached to each nUE instance, thus enabling the use of any application to generate data plane traffic in the UE. Secondly, Nervion relies on Kubernetes [37], the widely used open-source container orchestration system, to flexibly and automatically realize the configuration in the scenario with the required number of containers and their interconnections over the underlying compute infrastructure. This allows for easily porting the emulation scenario between different compute clusters so long as they are orchestrated by Kubernetes. Thirdly, Nervion also provides flexibility with respect to the interface between emulated RAN and core system by modularizing the interface; replacing the module and its associated events allows switching between different (even non standard compliant) interfaces.

We develop a prototype implementation of Nervion that supports both 4G and 5G standard compliant RAN emulation from control as well as data plane perspectives, including a control-plane only mode. Our choice of standard compliant RAN-core interfaces for our implementation is considering that the majority of core system designs in the literature are based on 3GPP compliant interface to the RAN for modularity and deployability reasons [7–9, 13–15, 17]. To enable repeatable experimentation and to allow other researchers to build on our work, we have created a profile on the Powder platform [29] with the Nervion implementation.

Using the above outlined implementation, we first validate the correctness of Nervion and then extensively evaluate it against a representative set of existing RAN emulators following alternative approaches [23, 27, 28, 33]. Our evaluations show that Nervion is significantly more efficient and lightweight compared to OAI [27] and srsLTE [23] RAN emulators. They also indicate that 4G and 5G versions of Nervion efficiently emulate the RAN compared to OpenEPC [28] and UERANSIM [33], respectively. These results highlight the scalability of Nervion relative to these other existing RAN emulators.

We present several use cases of mobile core evaluation with Nervion that span both control and data planes. One of them compares the scalability of multiple different mobile core designs and implementations: OpenAirInterface [22] (OAI) Core-network, srsEPC [23], NextEPC [24], MobileStream [14], Free5GC [25], and Open5GS [26]. This evaluation also demonstrates how Nervion can elastically use the underlying compute infrastructure (Powder in our case) to realize the RAN of desired size and configuration, as well as potential scalability improvement with its control-plane only mode. We also study the control plane latency with some of the above mentioned core systems using Nervion. Finally we also present a data plane focused use case that demonstrates how Nervion can flexibly create data plane traffic for different types of services (e.g., mobile broadband and IoT).

To the best of our knowledge, Nervion is by far the most comprehensive and versatile RAN emulator supporting scalable and flexible mobile core evaluation at high fidelity. We make Nervion publicly available via the Powder facility [38] to benefit the research community. We believe going forward Nervion will serve as a common RAN emulation tool to evaluate newer mobile core systems designs. In summary, we make the following key contributions:

- We present a new RAN emulator design that takes a unique cloud-native approach to enable scalable and flexible mobile core system evaluation (§3.2);
- We develop a prototype implementation of Nervion that supports 4G and 5G standard compliant RAN emulation across control and data planes; this implementation is also integrated into the Powder platform through a profile that already includes six different mobile core systems to experiment with (§3.3);
- We validate the correctness of Nervion, evaluate it in comparison with representative set of existing 4G/5G RAN emulators to demonstrate its scalability and flexibility advantages, and also shows its value for mobile core system evaluation through multiple diverse use cases (§4).

2 BACKGROUND

4G (LTE) mobile network architecture (Fig. 1) is made up of two components: the RAN and the Core Network. The RAN comprises base stations, called eNodeBs (eNBs) in LTE, that connects each user device, called user equipment (UE), to the core network via the air interface. The core network, called Evolved Packet Core (EPC) in LTE, handles both control operations (such as mobility management and authentication) and data transport to/from the Internet. Specifically, the EPC consists of four main entities: (1) the Home Subscriber Server (HSS), the central database containing relevant subscriber-related information (including the authentication keys); (2) the Mobility Management Entity (MME) that is responsible for control functions including user authentication, session establishment and mobility management; (3) the Serving Gateway (SGW)

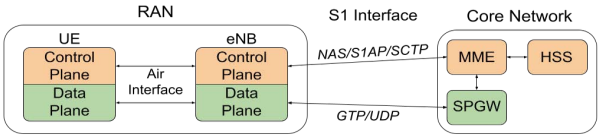

Figure 1: 4G (LTE) system architecture

which handles the user data traffic to/from the RAN; and (4) the PDN Gateway (PGW), which connects the EPC with external IP networks such as the Internet. The SGW and the PGW are typically merged into a single entity called SPGW.

The S1 interface is used to communicate between RAN and the EPC, and it has two parts: (i) S1-C for the control-plane to carry the control signaling traffic between UE/eNB and the MME in EPC; and (ii) S1-U for the data or user plane (S1-U) to transport user data traffic between the RAN and SPGW. The S1-C uses S1AP and Non-Access Stratum (NAS) protocols, respectively, for eNB-EPC and UE-EPC communication. While S1AP itself runs over SCTP, NAS messages are encapsulated within S1AP messages en route to EPC. The S1-U runs over GPRS Tunneling Protocol (GTP), which is in turn based on UDP.

5G system architecture. Mobile networks are transitioning towards a newer 5G system architecture [1–4] that has been going through standardization from 3GPP release 15 onwards. Fig. 2 shows a schematic of this 5G system architecture from the perspective of 5G core network, referred to as Next Generation Core (NGC), that reflects its key aspects: control-data plane separation, greater disaggregation of core functions and also adopts a service-based architecture. Several of the NGC functions map to their coarse-grained counterparts in 4G EPC. For example, the control and data plane parts of SPGW map, respectively, to Session Management Function (SMF) and User Plane Function (UPF) in NGC. The Access and Mobility management Function (AMF) corresponds to MME, whereas Unified Data Management (UDM) represents the HSS functionality. NGC also includes additional new functions such as Network Slice Selection Function (NSSF), Network Exposure Function (NEF) and Network Repository Function (NRF) to enable a service based architecture. Moreover, functions in NGC interact with each other as different services that can each be independently scaled and flexibly evolved.

The control plane interaction between NGC and 5G RAN is via the N2 interface, that is based on the NGAP protocol (5G counterpart of S1AP) over SCTP. As in 4G, the NAS protocol is used for control plane interaction between UE and NGC via the 5G base station (gNB), and NAS messages are encapsulated in NGAP messages. The data plane interaction between RAN and core in 5G, on the other hand, is over the N3 interface that is based on GTP/UDP (as in 4G).

Alternative Core Designs. Limitations of the traditional mobile network architecture, especially that of the 4G EPC, have come to the fore in recent years due to the growing scale and diversity of end devices, their varied access patterns, new service requirements, and the broader softwarization trend in the mobile network industry to shift towards a virtualized and cloud infrastructure based on commodity hardware. This has resulted in several proposals for alternative core designs in the research literature, including ones that seek to address the scalability, latency and reliability issues with

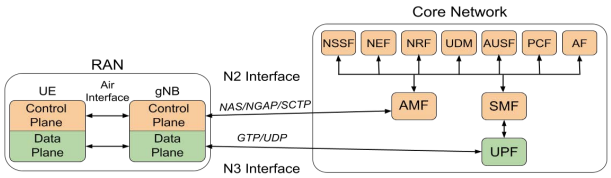

Figure 2: 5G system architecture

the EPC (e.g., [13, 14, 17, 18, 20, 21]); to leverage software-defined networking (SDN) and network function virtualization (NFV) (e.g., [5–12, 19]); and to enable deployment of the core over cloud environments (e.g., [13]) or at the edge (e.g., [15, 16]).

Containers are a form of operating system (OS) virtualization in which the application and its dependencies are isolated from other processes in the OS. As containers run as isolated processes sharing the OS, they avoid the overhead of starting and managing whole virtual machines (VMs). One of the most popular containerization technologies is Docker [39]. Docker uses the Linux resource isolation features (namespaces and cgroups) to package the application and its dependencies, allowing it to run in any location.

Kubernetes [37] is the widely used open-source container orchestration system to manage containerized applications. It allows automatic deployment, scaling and management of an application over a compute cluster. In Kubernetes, the basic work unit is the *Pod*, which is a high-level abstraction grouping containerized components. Kubernetes orchestrates the pods to create even higher level abstractions, such as *Services* (set of Pods that work together), *ReplicaSets* (maintains a stable set of replica Pods running at any given time) or *Namespaces* (partitions of the resources into non-overlapping sets). Kubernetes' architecture is based on a master-slave model, in which the master runs all the orchestration logic, resource-control and scheduling functionality, whereas the slaves are the working units on which the containerized parts of applications are executed. Each slave corresponds with a physical cluster node, as well as the master node.

3 NERVION

Our overarching objective in this paper is to design a scalable and flexible RAN emulator for realistic evaluation of mobile core system designs and implementations. In this section, we first elaborate on the scalability and flexibility goals along with the challenges they pose. We then present our design – Nervion– that addresses these goals. Finally we describe our current prototype implementation of Nervion.

3.1 Design Goals and Challenges

3.1.1 Scalability. An effective core mobile network RAN emulator should be able to support the realistic emulation of large-scale RAN scenarios. It should specifically be able to emulate large numbers of UEs as well as scalable (and variable) workloads for both control and data planes. However there is an inherent trade-off between the scale of the RAN that can be realized for a given compute resource and its realism. Here the key challenge is to decide how to realize the RAN scenario in a way that is scalable yet preserves the realism from the core evaluation viewpoint. Also note that RAN scalability that can be achieved is limited by certain aspects important for

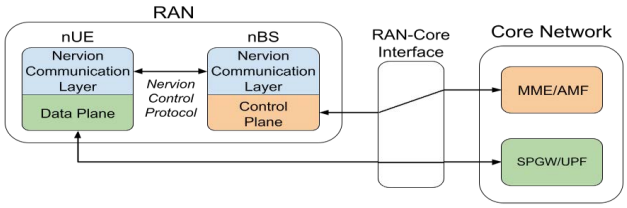

Figure 3: Schematic representation of the emulated RAN architecture in Nervion and its interaction with core network via the RAN-Core Interface module.

realistic and flexible RAN emulation such as supporting arbitrary applications for data plane traffic from emulated UEs. Another key design issue for a RAN emulator to achieve scalability is how to leverage more than a single machine or process as the underlying compute resource.

3.1.2 Flexibility. The utility of a RAN emulator depends on being able to realize RAN scenarios of desired scale and arbitrary workloads across both control and data planes. The question is how this can be achieved automatically through a programmatic interface rather than having to rely heavily on repeated manual input from user of the emulator. Another design aspect to be addressed is how to flexibly and automatically map a desired RAN configuration for an experiment onto the available compute resources. Also, the RAN emulator should be flexible enough to accommodate a variety of core system designs differing in their interfaces to the RAN while being agnostic to the internals of the core systems. The question is how to support such flexibility through a modular design of the RAN emulator.

3.2 Nervion Design

We now present the Nervion system design that addresses the aforementioned challenges. Nervion not only faithfully emulates the RAN control plane functionality but also flexibly allows experimenting with diverse types of real traffic loads through the data plane. Although Nervion design is generic, for the sake of concreteness, we present it in the context of 4G/5G RAN with 3GPP standard compliant interfaces to the mobile core network, as outlined in §2.

3.2.1 Abstraction of RAN Internals. In Nervion, the first step towards realizing a scalable RAN emulator is to abstract away internal aspects of the RAN that are *inconsequential* from the core network perspective. For example, considering the 4G system architecture (Fig. 1), the core network only sees the S1 interface – S1-C in the control plane and S1-U in the data plane. Everything beyond the S1 interface is effectively invisible to the core and so is not relevant to emulate. We use this observation to internally abstract the RAN. Expanding on the above example, instead of using the standard protocol stack for communication between UE and eNB with the different protocol layers (RRC, PDCP, RLC, MAC and PHY), Nervion uses a TCP connection between the emulated UE and emulated eNB. This results in lightweight realizations of the emulated UEs and base stations (BSs) that are referred to in Nervion as nUEs and nBSs, respectively.

3.2.2 UE Function Refactoring. To further enhance scalability, we reduce the coupling that exists between a UE and its associated

```
<nBS List>:
    - <nBS 1>:
      - nbs_id: 1
      - nbs_mnc: mnc
      - nbs_mcc: mcc

<nUE List>:
    - <nUE 1>:
      - ue_mnc: mnc
      - ue_mcc: mcc
      - ue_msin: msin1
      - ue_key: key
      - ue_op_key: op_key
      - control_plane: init-10-detach-5-attach-25
      - traffic_command: ping -I {TUN} 8.8.8.8
      - nBS: 1
```

Figure 4: A Nervion configuration file specifying an emulation scenario with one nBS and one nUE.

BS in the standard RAN architecture. Each RAN element (UE and BS) in the standard RAN has both control and data planes from the core perspective but the BS is merely an intermediate relay in the data plane for traffic between UE and the core. This makes the BS a bottleneck as it needs to handle all traffic to/from its associated UEs. In an emulator setting, however, we can deviate from this standard RAN architecture model, without compromising the realism from a core system evaluation perspective, as UEs and BSs can be realized as separate processes that can each communicate separately with the core. Given this and considering that control plane messages from each UE are expected to come from its associated BS from the core perspective, we refactor the UE functionality in Nervion so that the UE data plane resides in the nUE and its control plane operations are executed at the corresponding nBS, along with control plane signalling originating/terminating at the BS. For coordination and exchange of necessary state information between nUE and nBS (e.g., IP address assigned to UE by the core in 4G/5G), Nervion uses an internal communication protocol termed *Nervion Control Protocol*. The control plane actions from the nUE side are passed to its corresponding nBS as requests over this control protocol; the latter completes those actions and returns the responses to the nUE. The data plane activity from a nUE bypasses its nBS altogether. Fig. 3 illustrates the resulting emulated RAN architecture in Nervion which aids in realizing larger scale RAN scenarios without negatively affecting emulation fidelity.

3.2.3 RAN Element Containerization. The majority of RAN emulators used in the research community employ a multi-threaded approach within a single process and therefore limited to a single physical machine to emulate RAN elements (UEs/BSs). With Nervion, we distribute RAN emulation over multiple processes that can be spread across different machines in a compute cluster. Our distributed approach allows increasing the scale of the RAN that can be emulated while maintaining realism. While emulated RAN elements with our approach could in principle be realized as virtual machines (VMs), containers are lightweight and faster to set up, in line with our goal to have a scalable RAN emulator. So we containerize the emulated RAN elements in Nervion (nUE and nBS), and specifically use Docker [39] for this purpose in our implementation.

Containerization also provides the required isolation between multiple emulated UEs (nUEs) when instantiated on a single machine in terms of data plane traffic. In Nervion, we use a TUN device [36] in each nUE to route traffic between UE and the core (e.g., in the case of 4G/5G to intercept the UE application traffic and

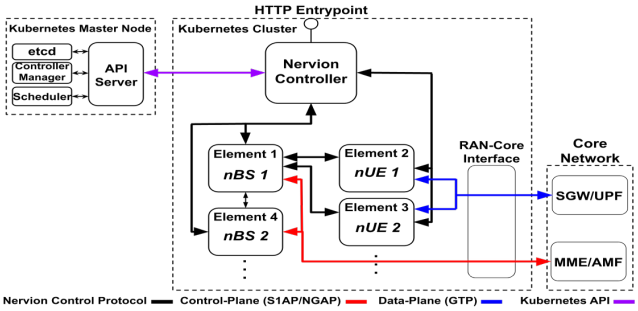

Figure 5: Nervion over a Kubernetes cluster.

encapsulate it on the corresponding GTP tunnel). This approach enables us to attach arbitrary applications to send/receive traffic in the data plane through the TUN device. And containerizing nUEs allows each nUE to have its own TUN device independent of other nUEs instantiated on the same physical machine.

When the focus of mobile core system evaluation is solely on the control plane, isolation between emulated nUEs when realized on the same machine (to be able to independently generate/receive data traffic) is not relevant. For such cases, Nervion design supports a 'Control-Plane Only' mode to further increase the scale of the RAN that can be emulated by leveraging multi-threading in addition to containerization. Specifically, with this mode, the control plane of each nUE is realized as a thread within a container, as opposed to one nUE per container in the 'full mode' of Nervion. As a result, the number of UEs that can be emulated is multiplicatively increased by the number of nUE threads per container.

3.2.4 Emulation Scenario. With the aim of providing ample flexibility to create emulated RAN scenarios of desired size and workloads in control/data planes, Nervion offers the notion of an 'emulation scenario', which is specified as a JSON configuration file (see Fig. 4 for an example) and provided by the user to Nervion via a web interface. This config file allows specifying information about each emulated RAN element (nUEs and nBSs) and their control/data plane behavior. The 'control_plane' and 'traffic_command' fields within the description of each nUE, respectively, describe its workload in the control and data planes. The control plane workload is specified as a string of tuples, each with an action (e.g., attach, detach) followed by time (in seconds) that the nUE in question will stay in the state resulting from that action. Note that some actions such as handovers need additional information (e.g., new nBS to associate with) and that is included following the action name in the control plane workload specification. The data plane workload of a nUE is specified with a command to invoke any real application or traffic generation tool (e.g., iPerf) to generate the traffic through the local TUN device. Nervion internally has an entity called *Controller* that is responsible for validating and realizing an input emulation scenario. The validation is done through a parser that checks the config file if it complies with a set of pre-coded rules (uniqueness of nBS and nUE ids, each nUE not connected to more than one nBS at any point in time, etc.)

3.2.5 Orchestration with Kubernetes. We now consider realizing a given emulation scenario in a flexible and automated manner over available underlying compute resources (ranging from a single computer to a compute cluster). This is especially an important issue when a large-scale RAN scenario needs to be emulated as manually realizing such a scenario is impractical. To this end, we use the widely used Kubernetes container orchestration system [37]. As described above, emulated RAN elements (nUE and nBS) in Nervion are containerized[1]. To simplify the deployment process, the nUE and the nBS have been merged into a single containerized entity called an 'Element'. The Nervion Controller (which itself is containerized) is the core of the system that is responsible for deploying as many containers as specified in the configuration file for an emulation scenario over the Kubernetes cluster. The Controller communicates with the Elements (which later take on the roles of nUEs/nBSs) through the Nervion Communication Layer – the same one used for communication between nUEs and nBSs. It is in the Nervion Communication Layer where the Nervion Control Protocol (NCP) is implemented. This layer is included on every Nervion component and is responsible for all internal communication.

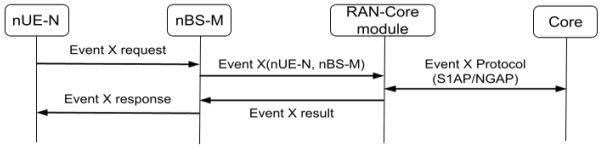

Figure 6: Generic control-plane event call flow

Fig. 5 illustrates the integration of Nervion with Kubernetes. To start with, a Kubernetes cluster is created with the provided compute resources in terms of different worker nodes and their CPU and memory settings. Then Nervion is deployed over the Kubernetes cluster, which results in the Controller pod getting instantiated with an external web service as well as an internal service, the latter for interaction with the Nervion Elements. When the configuration file for the emulation scenario is input to the Controller through the web service, the Controller uses the Kubernetes API to instantiate the Elements needed by the config file as the rest of the pods in the Kubernetes deployment. Each of the Elements, upon starting up, connects with the Controller over NCP to get their role (nBS/nUE) as well as their interconnections as per the config file. Once the specified emulation scenario for the emulated RAN is realized over the cluster, the Controller acts as a monitor to keep track of the state of the different elements of the RAN deployment and report it to the user through the web interface.

3.2.6 RAN–Core Interface Module. Even though the specific protocol used for the interface between the RAN and the core can be different (e.g., S1AP in 4G and NGAP in 5G), these protocols generally share a common set of 'events'. An event here refers to a set of messages that produces a change of state of a UE for the core network, such as Attach or Detach. Based on this observation, Nervion uses an event based RAN-Core interface in which a set of functions that represents these events implement the necessary group of messages to produce a change of state in the core network. In Nervion, this is realized in a modular manner as the RAN-Core interface module that is situated in between Nervion and the core

[1]Note that unless specified otherwise, our description in the following refers to the full mode of Nervion. It is straightforward to adapt to the control-plane only mode by focusing only on control plane and viewing nUEs as threads within a container.

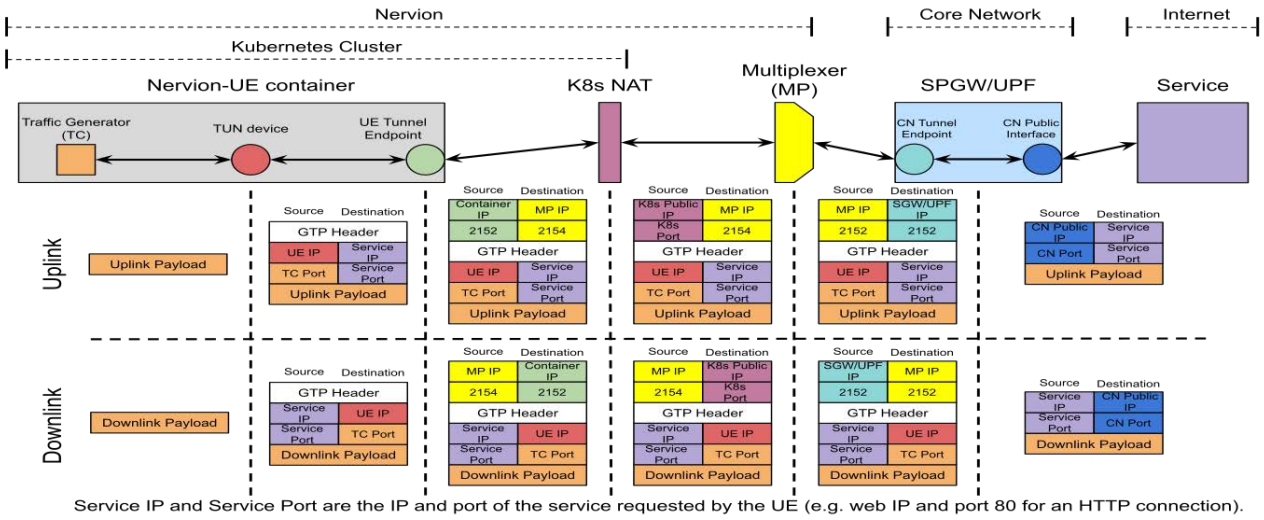

Figure 7: End-to-end dissection of the data plane at each component as implemented in Nervion.

network. This module realizes the relevant protocol to interface with the core network (e.g., S1AP or NGAP) while interacting with Nervion with an event based interface that remains the same regardless of the protocol used.

3.3 Implementation

Nervion has been implemented in two main components: the Controller and the Elements. The Controller is responsible for deploying the RAN structure based on the provided configuration and stores all the information about the different entities that make up the RAN and their status (e.g., nUE Attached, nBS Connected, etc.). On the other hand, the Element is where the interaction with the core network happens, either the control-plane executed by the nBS role or the data-plane in the nUE. The current implementation is meant to evaluate standard-compliant core networks; the flexibility and abstraction offered by Nervion, however, would enable extensions to support future standards, or indeed, non-standard implementations. The Controller has been implemented in Python, whereas the Element (that takes on nBS/nUE role) is implemented in C. Overall, Nervion implementation is more than 13K LOC.

3.3.1 Controller and Nervion Control Protocol. Once the Nervion Controller is instantiated, it instantiates two services: a web-service (linked to the Kubernetes external service), which is the access point to Nervion for the user, and the controller-service (linked to the internal Kubernetes service) accessed by the RAN elements. Internally, the Controller implements a multi-threaded architecture with a common memory area in which all the details of the experiment are stored (RAN elements, core information, interconnections, status information, etc.). This memory region is accessed by the web-service to populate it with the information provided by the user (configuration files and core parameters) and by the controller-service to pull the necessary information for each role and to update the status of each element. The Controller accesses the Kubernetes API to create the elements (pods in the K8s jargon) using, by default, the Nervion Docker base-image available at Docker Hub [40]. However, for the situations in which the application used in the data-plane traffic command of any of the nUEs is not part of the Nervion base-image, a custom Docker image that is built on top of the Nervion base-image and accessible through Docker Hub can be used.

The Nervion Control Protocol (NCP) implementation spans the Controller, the nUE and the nBS. This NCP runs over TCP between the elements (nUEs/nBSs), and over UDP between the elements and the Controller. The reason for these underlying protocols is that the connection between the nUE and the nBS has to be synchronous (because the UE has to wait until the eNB performs the requested control-plane action), whereas that is not the case with the connection with the Controller (the nUE and nBS work independently of the Controller and eventually use the NCP to request information and update their status).

3.3.2 nBS and nUE. Nervion materializes the nBS and nUE within the Nervion Element. The Nervion Element is executed by each pod just after being created. The elements start by connecting with the Controller and receiving a role, which can be nUE or nBS. The role assignation also comes with information necessary to perform that role (e.g., the nUE role comes with the IMSI, the UE key, the Operator Key, the nBS IP, etc.). The nUE role fully implements what, from the core network perspective, is the data-plane, while the nBS implements the control-plane. The control-plane actions that a nBS performs on behalf of its nUEs are triggered by those nUEs, where the UE information (UE keys, control-plane actions) is stored.

3.3.3 Control Plane. In Nervion, the nBS implements the control-plane following a service based approach. This means that the nBS receives a request from the nUE to perform different actions and returns the results generated for that action. The actions implemented by the control-plane correspond to the most frequent control events (e.g., attach, detach). The control-plane is implemented in three modules: nBS, SCTP, and RAN-Core interface.

- The nBS module provides an interface for the nUE to access the control-plane events implemented in the RAN-Core module. This module starts when the element receives the nBS role (and related information) from the Controller. The nBS module creates the necessary interfaces that facilitate connections with nUEs and with other nBSs. The first interface created is a TCP connection on which the nBS is waiting on the port 2233 for nUEs that request control-plane actions. The nBS module is also responsible for the X2 interface used in the X2 Handover procedure. Nervion nBS replaces the X2 interface by a TCP connection on the port 2234. Finally, the nBS module creates a multi-threaded server structure to concurrently attend to nUE and nBS requests on these two interfaces.
- The SCTP module is responsible for establishing the SCTP connection between the nBS and the MME/AMF to realize the S1-C interface and runs on top of the SCTP Linux Kernel module. This module connects the nBS with the port 36421 at the MME for a 4G emulation or the port 38412 at the AMF for a 5G emulation. These ports have been defined by the IANA and the 3GPP for the S1 and NG control-plane respectively [41].
- The current Nervion comes with two RAN-Core modules: S1AP module (4G) and NGAP (5G). The S1AP module and the NGAP run on top of the SCTP module using the SCTP Protocol Identifier Fields 18 and 60, respectively [42, 43] and integrate the NAS protocol. The details of all the events supported are omitted due to space constraints, but Fig. 6 depicts the functioning of a generic *Event X*.

We have augmented the Nervion implementation to support the control-plane only mode, as described earlier. This mode effectively disables the data plane and allows realization of control-plane of nUEs as threads in a container, as per the user specified configuration of number of threads per container. As with the full mode, each threaded nUE contains all the UE related information and requests control plane actions from the corresponding nBS.

3.3.4 Data Plane. The Nervion data-plane is fully implemented within the nUE and uses the GPRS Tunneling Protocol (GTP), which runs on top of UDP, between the RAN and the S-GW/UPF. To support this design, Nervion takes advantage of the S1AP/NGAP Attach procedure specification. During the Attach procedure, specifically during the Default Bearer Establishment, the nBS sends an Initial Context Setup Response message to the MME/AMF. This message contains the BS IP address from which the data-plane packets are going to be received in the S-GW/UPF. Instead of sending the BS IP, Nervion sends the nUE IP, and consequently, the S-GW/UPF creates an entry to enable traffic from/to that IP with the UE TEID. The data-plane starts in the nUE once the nUE receives the necessary information (TEID, the UE IP, and the S-GW/UPF IP) from the nBS. The data-plane has been built over a virtual-network kernel interface using a TUN device, which simulates a network layer device that operates in layer 3 carrying IP packets. In each nUE, a TUN device is created with the corresponding UE IP address assigned by the core. Then the application or traffic generator software is attached to this interface, tunnelling all the packets to the S-GW/UPF.

The nUE data-plane currently implemented in Nervion is depicted in Fig. 7. 3GPP specifies that the UDP connection used under

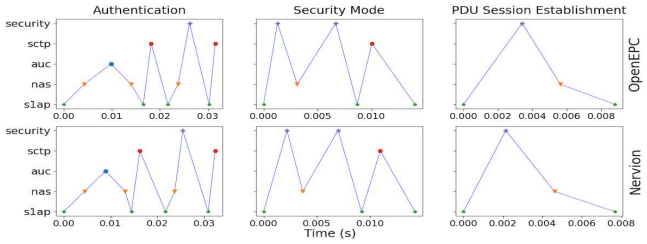

Figure 8: The effect of UE attach control-plane action with Nervion on MobileStream core network, relative to OpenEPC based RAN emulation.

the GTP Tunnel has to make use of specific UDP port, i.e., port 2152 at the S-GW/UPF and port 2152 at the eNB (or the nUE in the Nervion case). Because nUE is containerized and runs inside Kubernetes, the uplink packets reach the S-GW/UPF through the Network Address Translator (NAT) of Kubernetes. In the other direction, when a downlink packet is dispatched by the SGW/UPF function in the core network towards the RAN, UDP port 2152 is used as a destination port, as per the 3GPP standard. These downlink packets are dropped when they arrive at the Kubernetes NAT due to a missing entry for port 2152 in the NAT table (which instead contains contains a random port registered for the uplink packet).

In Nervion, we address this problem by introducing a 'Multiplexer' that acts as a reverse-NAT. Specifically, it receives uplink GTP packets, stores the source IP and source port in a hash-map, using the UE TEID as the key, and forwards it to the S-GW/UPF. Once a corresponding downlink packet is received at the multiplexer, it retrieves the stored IP and port from the hash-map using the UE TEID and forwards the packet to the Kubernetes NAT, which now is able to properly forward the packet to the correct container hosting the nUE. Note that in order to include the Multiplexer in the data-plane path, the control-plane has to register the multiplexer IP during the Attach procedure, instead of BS IP as per 3GPP standard. This multiplexer has been implemented as a multi-threaded application in C. We initially had a Python based multiplexer implementation. We also note that further improvement in the efficiency of data plane implementation over our current TUN device based solution can be achieved via kernel bypass, which we intend to address in the future.

4 EVALUATION

In this section, we first validate the correctness of Nervion relative to previously validated and alternative RAN emulation approaches. We then evaluate its efficiency and scalability relative to representative set of existing RAN emulators. Finally we present multiple use cases that demonstrate the value of Nervion for diverse range of mobile core system evaluation studies. All these evaluations are conducted over the Powder platform [38].

4.1 Correctness

It is important to validate the correctness of a RAN emulator before it can be relied on for mobile core system evaluations. As such, we address this issue for Nervion in this section. The correctness property we examine is keeping in mind the purpose of the RAN emulator, which is to enable mobile core system evaluations. So we

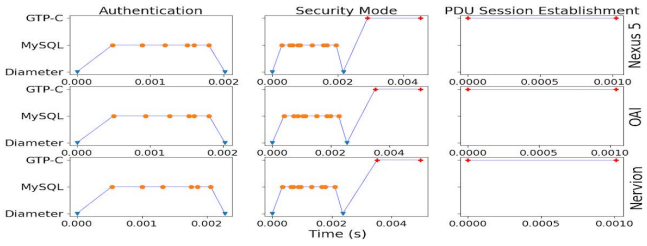

Figure 9: The effect of UE attach control-plane action with Nervion on OAI core network, relative to RAN setup with Nexus 5 phone and OAI RAN.

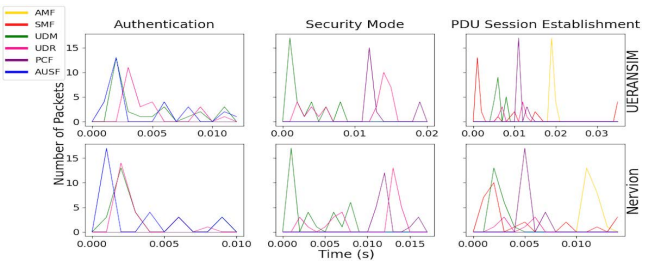

Figure 10: The effect of UE attach control-plane action with Nervion on Open5GS 5G core network implementation, relative to UERANSIM based 5G RAN emulation.

study the effect of the RAN workload generated by the emulator on the mobile core system under test. We do this relative to other previously validated and alternative approaches to RAN emulation. Our correctness evaluation spans both control and data planes.

4.1.1 Control Plane. To study Nervion correctness for control plane, we consider the UE attach procedure, a key control plane event. It also involves multiple message exchanges between the RAN and core, which makes it a good case for examining control plane correctness. Given the elaborate nature of the attach procedure, to clearly see the effect of control plane emulation in the RAN on the core network behavior, we break that down into its three main constituent phases: Authentication, Security Mode and PDU Session Establishment.

We first consider OAI as the target core network and compare the effect of the chosen emulated control plane action, the Attach event, when realized with Nervion relative to two alternatives: (i) a commercial Nexus 5 phone connected through a OAI eNB with USRP; (ii) OAI implementation of UE connected through a OAI eNB. These alternatives provide a good basis to assess Nervion correctness as they represent a real RAN or a setup that has previously been validated.

Fig. 9 shows the result of this experiment, as measured at the protocol interfaces between components of the OAI core network, which is a standard compliant 4G core network implementation. There are three main protocols: (i) Diameter protocol used between MME and HSS; (ii) HSS uses MySQL to access the DB; (iii) GTP-C is used by the MME to configure the data plane. These results show a clear alignment between the effect of attach control-plane action emulated by Nervion and that with the other two alternatives on the internal behavior in the core network for all three phases of the attach event.

We next perform a similar experiment with MobileStream [14] as the target core network and compare the effect of attach event emulated with Nervion relative to OpenEPC [28], a commercial solution for RAN emulation. Fig. 8 shows the result of this experiment. Here the y-axis shows the 5 function blocks that make up the core control-plane within MobileStream: s1ap, nas, auc, sctp and security. We see a similar effect on the interaction between these blocks due to the emulated attach action with Nervion as with OpenEPC based RAN emulation.

We now consider UE attach event in the 5G case with Open5GS [26] as the target core network and compare Nervion emulation relative to UERANSIM [33] 5G RAN emulator we have available. To our knowledge, the correctness of UERANSIM itself has not been validated. Yet this comparison is useful in assessing if the core network behavior achieved with Nervion matches with that of competing alternative in the 5G case. Results from this experiment shown in Fig. 10 indicate good alignment between the two alternatives compared. Due to the number of messages exchanged between the core network functions in Open5GS, the y-axis shows the number of packets seen by each core network function across the three phases of the attach procedure.

4.1.2 Data Plane. We now switch our attention to correctness of emulating the RAN data plane. For this, we consider a simple but illustrative 'ping' traffic pattern generated with Nervion emulated UE, relative to that by a real (Nexus 5) phone and OAI UE – the two alternatives as in the first correctness related experiment above. We measure throughput at the SPGW interface of the OAI core network used for all these three compared alternatives. Results shown in Fig. 11 show that Nervion emulation of the UE traffic pattern and the data plane is almost identical to that of the alternatives, as seen in the core network.

4.2 Nervion versus other RAN emulators

The above results clearly demonstrate that Nervion faithfully captures the RAN behavior as relevant from a core network perspective, across both control and data planes. While we have considered the full mode of Nervion for those experiments, the conclusions also apply to the control-plane only mode of Nervion when the focus is solely on the control plane – the realization of emulated UEs as threads in a container with this mode improves scalability but that does not come at the expense of correctness.

In terms of experiments that can be supported by Nervion, it can allow emulation of arbitrary RAN workload patterns in control and data planes for mobile core system testing and evaluation. Representative set of experiments supported by Nervion are featured later in §4.3. To better appreciate the capabilities offered by Nervion, we provide a qualitative comparison against other competing alternatives in Table 1. Nervion is the only solution that supports 4G and 5G RAN emulation, across both control and data planes. Another distinguishing capability of Nervion is its support for creating programmatic and automated control/data plane RAN workload patterns, a feature lacking in all other alternatives but crucial for flexible and repeatable experimentation. Overall, Nervion offers a comprehensive RAN emulation capabilities, subsuming those possible with alternative solutions.

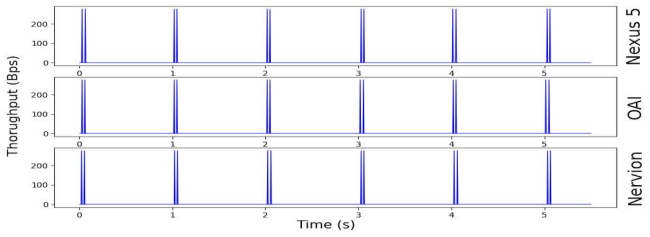

Figure 11: Emulated data plane traffic with Nervion as seen inside the OAI core network, relative to RAN setup with Nexus 5 phone and OAI RAN.

In the following, we go beyond the qualitative and flexibility benefits offered by Nervion to quantify its efficiency and scalability gains relative to representative existing solutions listed in Table 1.

4.2.1 *Comparison with full stack RAN emulators.*

S1AP Events latency comparison. An efficient RAN emulation system requires less time to execute each action, results in lower CPU usage and therefore, allows more actions to be completed within the same fraction of time. We performed an experiment to compare the control-plane performance of Nervion relative to full stack RAN emulators that realize the RAN in detail (OAI RAN and srsRAN). This experiment has been conducted with identical resources for the RANs, the resources dedicated to the EPC and the EPC implementation used (srsEPC).

Fig. 12a compares the time taken by each emulator to reach specific states in the S1AP attach process. The "Start" point represents when the eNB emulator is executed. "Connected eNB" is reached when the eNB receives the S1 Setup Response message. The UE is in the "Authenticated UE" state when it sends the Authentication Response message to the EPC. Similarly, Security Mode Complete message sets the "Security Mode Complete" state in the graph. Finally, a UE is considered to be in the "Attached" state when it sends the Attach complete message to the EPC.

The time difference between OAI/srsLTE and Nervion at Connected eNB stage is largely a software complexity issue: Nervion has been implemented with minimalism in mind, so only essential structures are initialized. On the other hand, the full stack emulators include implementations of all protocol layers between the UE and the eNB (including the physical layer with fine-grained radio condition description). The initialization of a detailed physical layer with the addition of the upper layers has a negative impact on the overall performance. The significant delay in OAI and srsLTE reaching the Authenticated UE state is also due to the initialization of a complete mobile radio stack, this time on the UE side. The Nervion design approach for realizing the RAN is also a crucial factor contributing to its high performance. In particular, with OAI and srsLTE, the NAS protocol is implemented in the UE and S1AP protocol in the eNB, whereas, with Nervion, both are implemented in the eNB. This design decision avoids the need for the eNB to forward NAS messages to/from the UE.

Virtual memory consumption. Another crucial factor from a scalability perspective is memory consumption. In a resource-limited environment where the available memory has to be shared among different processes, the resource consumption of a single

	Control Plane (CP)	Data Plane (DP)
Nervion	• 4G and 5G • Multiple UEs • Multiple BSs • Configurable & Automatic CP actions per UE	• Support any application • DP Traffic from multiple UEs • Configurable & Automatic traffic per UE
OAI [27]	• 4G only • Detailed RAN emulation	• Support any application
srsLTE [23]	• 4G only • Detailed RAN emulation	• Support any application
OpenEPC[28] (closed-source)	• 4G only • Multiple UEs • Multiple BSs	• DP Traffic from multiple UEs
UERANSIM [33]	• 5G only • Multiple UEs • Multiple BSs	• Support any application • DP Traffic from multiple UEs

Table 1: Nervion capabilities relative to competing RAN emulators.

emulated UE/eNB determines the number of emulated devices that can be run simultaneously. Like in the previous experiment, we have compared Nervion with the OAI RAN and srsRAN emulators. We again used srsEPC for the core network implementation for all the measurements.

Fig. 12b reports the memory consumption of different emulated elements (UE and eNB) and at different execution states of the RAN emulation systems. In particular, the figure shows virtual memory consumption. Virtual memory size (VSZ) is defined as the memory that has been allocated to a process. This includes not only the stack and heap memory but also swap memory and the memory occupied by shared libraries that are loaded (e.g., cryptographic libraries used for encryption/integrity in the NAS layer). The memory consumed by Docker and Kubernetes is also included for a fair comparison. In all three cases considered, the amount of memory consumed by the OAI (around 1.2 GB) implementation is around 4.4 times larger on average than the Nervion implementation, while srsLTE (around 2.5 GB) is around 2.2 times larger than the OAI emulator, even after including memory consumption due to Docker and Kubernetes. Again, the reason for this is that the OAI/srsRAN emulators involve a full stack RAN implementation. This includes the physical layer between the UE and the eNB, whereas Nervion only implements the essential part of the S1AP protocol to perform basic actions on the control-plane with a standard-compliant EPC.

4.2.2 *Comparison with commercial and ad-hoc RAN emulators.*

There are alternative emulators that have taken the physical layer abstraction approach, like Nervion, for lightweight realization of RAN elements while allowing multi UE support. In particular, two such most used tools are: OpenEPC (a commercial EPC implementation that comes with a RAN emulator) and UERANSIM (an open-source 5G RAN emulator). Due to limitations such as completely different emulation models or restricted access to the source code of OpenEPC and their binaries, it is difficult to evaluate the scalability properties by defining a common environment (same

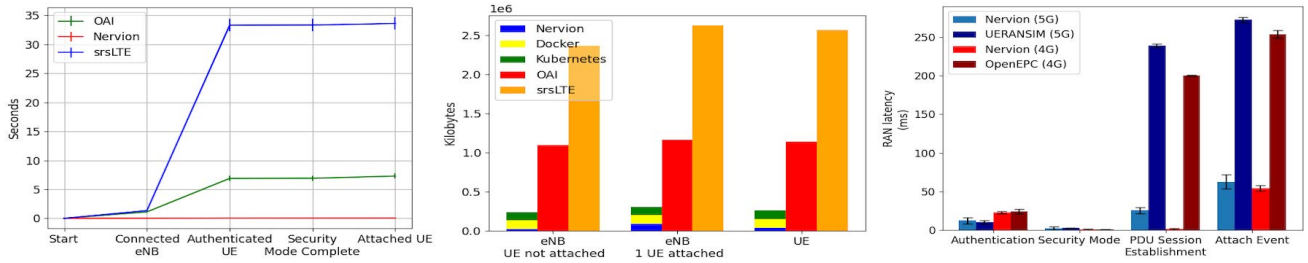

Figure 12.a) Nervion control-plane performance vs OAI and srsRAN.

Figure 12.b) Virtual memory consumption of Nervion vs OAI and srsRAN.

Figure 12.c) RAN latency of Nervion vs OpenEPC and UERANSIM.

hardware resources) to run the tools. We have instead measured the latency in the emulated RAN which, in an indirect manner, captures the efficiency of an emulator.

In Fig. 12c, different parts of the attach event are compared as well as the entire attach event for OpenEPC (4G) and UERANSIM (5G) with Nervion (4G and 5G). For the 4G measurements (red columns), the core network used is MobileStream [14], and for the 5G counterpart (blue columns), we have used Open5GS [26]. In general, Nervion outperforms both alternatives with latencies around 4.5 times smaller. Significant difference especially appears during the PDU session establishment for both UERANSIM and OpenEPC, when the data-plane is instantiated in the UE (when the UE receives its TEID, IP and the SPGW/UPF IP). Both 4G and 5G versions of Nervion instantiate the data-plane after the completion of the attach procedure. The other key factor is the way UERANSIM and OpenEPC have been implemented. UERANSIM has the control-plane implemented on the UE and uses the gNB as a forwarder to connect with the core network. OpenEPC, on the other hand, restricts access to its binaries, so we cannot affirm anything about the internal structure of the emulator, but all the packets go to the EPC through its 'client' application that acts as a relay. Nervion removes such additional latency by realizing the control plane functionality within the nBS.

4.3 Nervion use cases

Here we highlight the utility of Nervion through multiple different case studies of mobile core system evaluations.

4.3.1 Maximum number of simultaneously attached UEs. One of the more common metrics used in core network evaluations in the literature aimed at quantifying scalability improvements is the maximum number of simultaneously attached UEs a mobile core system can sustain. We used Nervion to evaluate this metric for six different core network implementations. In particular, we have evaluated four 4G core network implementations – OAI core network [22], srsEPC [23], NextEPC [24], and MobileStream [14] – and two 5G core network implementations: Free5GC [25] and Open5GS [26]. It is important to highlight that the goal of this experiment is to show the ability of Nervion to perform this type of evaluation requiring emulation of large-scale RAN scenarios and, indirectly, to show which core network is able to handle more attached UEs. The number of UEs/eNBs/gNBs supported by each implementation can potentially be increased through further optimization or reconfiguration but that is outside the scope of this experiment.

The Powder profile we created for this experiment allows deploying any of these six core network implementations over the d430 Powder node (Intel Xeon E5-2630 v3 at 2.40GHz with 64GB RAM). RAN emulation with Nervion used VMs (with 12-core CPU and 8 GB RAM) on the Powder platform for the Kubernetes cluster. The size of the cluster needed varied depending on the target core network implementation that was evaluated. As shown in Fig. 13a, the number of attached UEs with the MobileStream implementation is the highest, outperforming the second best, Open5GS with 1024, by about 10 times. They are followed by srsEPC and NextEPC with 253 and 140 UEs, respectively. Finally, OAI core network supports 16 UEs while Free5GC only reached 10 (which may be related to a bug in the current Free5GC version).

Considering MobileStream, the most scalable core network implementation from the previous experiment, as the target mobile core system, we now examine scale of the RAN (in terms of number of UEs) that Nervion can emulate for a given Kubernetes cluster size. Here we first consider the full mode of Nervion. The blue bars in Fig. 13b show the result of this experiment where the number of attached UEs for different numbers of worker nodes in the Kubernetes cluster are given. We observe a relation of 110 UEs per worker node. We have experimented with different Powder VM resource configurations (8-core/8GB, 12-core/12GB, etc.) and obtained the same results. Further investigation revealed that this is due to standard Kubernetes settings, which limit the maximum number of pods per worker node to 110 [44]. Note that in the current Nervion implementation for its full mode, each emulated UE is realized in a separate container and each pod in the Kubernetes cluster hosts one container. This explains the obtained result of 110 UEs per worker node given the maximum possible number of 110 pods per worker node.

We now repeat the same experiment with newly added control-plane only mode in Nervion to examine the potential scalability improvement it offers. Recall that with this mode, the data plane is ignored and the control plane of each emulated UE is represented by a thread in a container (pod). This adds a new dimension to increase RAN emulation scalability in terms of number of threads that can be realized per container. With the current implementation of the Nervion control-plane only mode, we were able to realize nearly 10K emulated UEs per container (pod) with the worker node resource configuration as above. With 110 pods per worker node, this result already demonstrates the potential to emulate million plus UEs on a reasonable sized Kubernetes cluster. We believe further improvements are possible by refining the implementation,

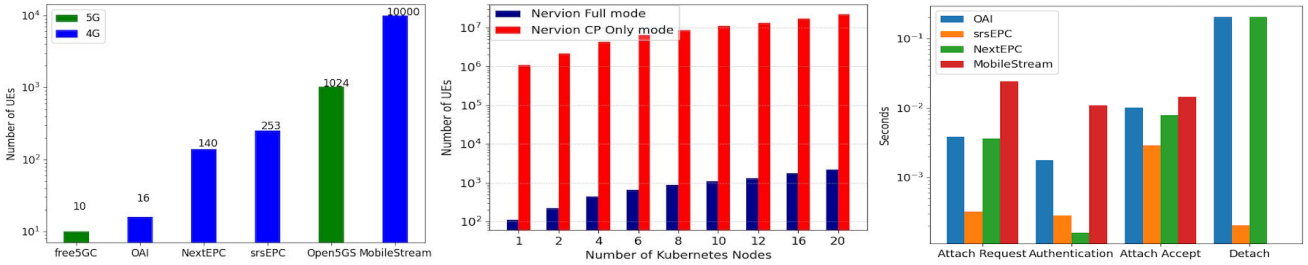

Figure 13.a) Maximum number of UEs simultaneously attached.

Figure 13.b) Nervion scalability based on the Kubernetes cluster size.

Figure 13.c) Comparison of control-plane latency with different cores.

allowing multiple containers per pod and relaxing the 110 pods per worker node limitation with standard Kubernetes setup. It is also important to highlight that multiple factors play a role in the scale of the emulated RAN that can be be achieved, such as the desired number of UEs, the worker resource specifications, and the control plane workload pattern of each UE. In the above experiments, we focused on the case where each emulated UE simply attached to the core.

4.3.2 Latency of control-plane events. Another key core network performance metric of interest is its control plane latency. Here we present an experiment using Nervion to measure the average control plane latency latency of four 4G core network implementations: OAI, srsLTE, NextEPC, and MobileStream. We have measured the latency of different messages that are part of the attach and detach events. The latencies shown in Fig. 13c are the Attach Request (time elapsed between the Attach request message and the Authentication request message), Authentication (time between the Authentication response message and Security mode command message), Attach Accept (time between the Security mode complete message and the Attach accept message), and Detach (time between the Detach request message and the Detach accept).

As expected, the srsEPC core implementation provides better results due to its centralized design in which MME, HSS and SGPW have been merged into one single entity. This centralized design has an even more pronounced impact on the latency of control plane events that involve responses from the HSS or the SPGW. On the other hand, MobileStream (which does not support the detach event), through its decomposed function architecture penalizes latency in favor of scalability. Finally, OAI and NextEPC obtain similar results due to their similar architectures, although the differences in their authentication latencies may be due to different cryptographic implementations.

4.3.3 Different data-plane traffic patterns. Here we focus on the data plane and show how different traffic patterns can be created in a flexible and automated manner via the Nervion configuration file. The setup for the two experiments reported below consists of the RAN part with 10 UEs and 1 eNB realized using Nervion and using OAI core network as the EPC. We consider two diverse types of data plane traffic scenarios: IoT traffic and mobile broadband traffic.

IoT devices typically tend to remain disconnected most of the time to save energy and intermittently attach to the network to exchange low-bandwidth data [45]. So in the IoT traffic scenario, the IoT devices were configured to remain disconnected for 100 seconds and then send/receive data traffic for 20 seconds using the ping tool. Fig. 14a shows the control-plane and data-plane traffic footprints for this IoT experiment. From the figure, the expected correlation between the control and data planes is apparent. The Attach event is reflected in the graph as peaks of 10 Kbps when the UEs attach to the EPC and peaks of 3 Kbps when the detach event occurs. The correlation between control and data planes is also reflected by the fact that data-plane traffic only appears to reach peaks of 20/25 Kbps after an Attach event and before a Detach event.

For the mobile broadband traffic scenario, devices remain attached most of the time with a moderate level of bandwidth consumption, mainly in the downstream direction [46]. We modelled such behavior with 10 UEs in total: five UEs remained attached throughout the experiment, while the other five detached from the EPC for 20 seconds every 100 seconds. We use the iPerf tool to generate 500Kbps of traffic per UE. Similar to the IoT test, the control-plane events in the mobile broadband traffic experiment can also be inferred from the values shown in Fig. 14b. At 50 seconds in the time series, all the UEs perform an attach procedure and start to generate traffic in the data-plane. At 100 seconds, 5 of the UEs perform the Detach event, which is reflected by the drop in control-plane traffic and the data-plane traffic also reduces from 5 Mbps to 2.5 Mbps. After 20 seconds, the five disconnected UEs again attach to the core-network and, like in the IoT experiment, these actions are performed in a loop until the end of the experiment.

4.3.4 Multiplexer evaluation. As described in §3.3.4, the current 3GPP 4G and 5G specifications require the use of specific UDP ports, which necessitate the use of a Multiplexer-like artefact in the data-plane for scenarios in which either the RAN, the core network, or both are behind a NAT. The Nervion emulator is instantiated within a Kubernetes cluster and as such data plane traffic has to traverse the Kubernetes NAT that in turn makes the use of multiplexer mandatory to realize a working, standard-compliant RAN emulation. Consequently, we consider that it is essential to provide an evaluation of the penalty introduced by this multiplexer on the data-plane performance.

In Fig. 14c, we compare the received throughput versus offered data plane traffic load with two setups: Nervion implementation (that includes the multiplexer), and an ad-hoc Nervion version that avoids the need for a multiplexer via realizing the emulated UE as a stand-alone process (not within a container and Kubernetes cluster). We considered the 10-110 Mbps range for the UE offered load based on evaluations done in previous works [15, 47] that show

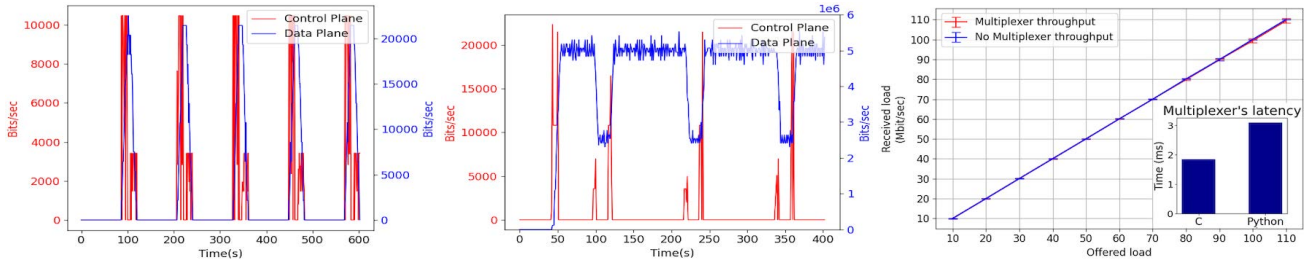

Figure 14.a) IoT control and data traffic footprints. **Figure 14.b) Mobile broadband control and data traffic footprints.** **Figure 14.c) Data-plane throughput comparison with and without multiplexer.**

the working range of different core network alternatives are less than 50 Mbps. For this range, we see from the results in Fig. 14c that the multiplexer does not pose any performance penalty on the data-plane. These results are based on a multiplexer implemented in C. In the inset figure, we compare this implementation with a Python based multiplexer implementation and find that, as expected, the C based implementation is faster in processing and forwarding GTP packets across the multiplexer by about 30%. These results are obtained with the multiplexer realized on a 2-core and 8GB VM.

5 RELATED WORK

Full stack emulators. The OpenAirInterface (OAI) [27] provides a high-fidelity implementation of the RAN part with a UE and eNB emulators that are capable of replicating the full LTE protocol stack. srsLTE [23] provides similar capability. This full stack approach has the obvious and severe scalability limitation.

Commercial RAN emulators. There exist several commercial RAN emulators [28, 48–52]. Some rely on dedicated hardware (e.g., [48]) whereas others target software solutions (e.g., [28, 49, 50]). However the main issue with these commercial emulators is their limited accessibility to the research community, due to their high cost. The only exception is OpenEPC [28] that is accessible through the PhantomNET [30] (now Powder) testbed infrastructure. So it has been used in some works [9, 13, 14].

Trace based emulation. Because of the inaccessibility issue with the above mentioned commercial tools, several works have used trace based emulation [7, 8, 11, 12, 17, 21]. Within this category, there are multiple approaches that slightly differ from one another, such as trace replay or trace-driven simulation. The straightforward way of generating traffic from traces is by simply replaying the packets contained in the captured traces. Trace-driven simulation does not, however, directly use the packets contained in the trace. Instead, this method extracts the pattern presented in the trace to model it with a distribution and then generates packets according to that distribution [7, 11]. Some works [7, 8, 21] rely on traces generated using ng4T [51], a commercial RAN emulator, to drive their core system evaluations. On the other hand, Moradi et al. [17] synthesized traces for UEs based on the real ones that were then fed to their core system prototype through a simulator. Other proposals (e.g., [12]) rely on real traffic extracted from commercial eNBs. Trace based emulation methods have one major disadvantage, which is that only the traffic patterns present within the traces can be emulated later and generalization beyond that is not possible.

Ad-hoc RAN emulators. There exist several tools that fall in this category [6, 15, 18–20, 31–33]. Jain et al. [6] present a tool that is then used to compare the performance of SDN based and NFV based mobile core paradigms. Using the same tool, Amogh et al. [10] evaluate a cloud-native and scalable MME solution. Similarly, in [18], a new MME architecture is presented and tested with UE SIM, a tool that generates attach requests on the control-plane. A control and data plane load generator is provided in [19] to evaluate their specific proposal. Focusing on the data plane, Open Mobile Evolved Core project [34] uses a data-plane packet generator which only generates uplink traffic [31]. S1APTester [32] is a module used in the Magma project [35] to validate S1-C and S1-U interfaces to an LTE EPC. The common limitation of the aforementioned tools [6, 18, 19, 31, 32] is that they are tied significantly to the specific project for which they have been designed and so are not readily usable for core network evaluation generally; adapting them to other studies is a non-trivial endeavour. The most scalable of these ad-hoc solutions [6, 18] are able to emulate multiple UEs performing different control plane events simultaneously via a threading model but still limited to a single machine (process). UERANSIM [33] is a recent open-source 5G RAN emulator that can generate both control (NGAP) and data (GTP) plane traffic. But it also has the same limitation as above outlined singe machine multi-threaded 4G RAN emulators.

6 CONCLUSIONS

We have proposed Nervion, a scalable and flexible RAN emulator for realistic mobile core system evaluations. Nervion significantly extends the state-of-the-art on RAN emulation by allowing emulation of any sized RAN in terms of number of elements (UEs and BSs), limited only by the cluster compute resources available. It also features a control-plane only mode to further increase the scale of the RAN scenario that can be realized with a given compute resource for core system evaluations focusing on the control plane. Also, unlike most existing RAN emulators, Nervion supports generating workloads on both control and data planes in a flexible and programmatic manner, and can support evaluation of any mobile core system. This has been possible thanks to a novel cloud native RAN emulation system design that facilitates the above with a light-weight realization of RAN elements at the right level of abstraction from the core network perspective and the use of Docker and Kubernetes to containerize and orchestrate the emulated RAN elements over a computer cluster to realize large-scale RAN scenarios.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments and suggestions that greatly improved this paper.

REFERENCES

[1] J. Kim, D. Kim, and S. Choi. 3GPP SA2 architecture and functions for 5G mobile communication system. *ICT Express*, 3(1):1 – 8, 2017.
[2] Service-Based Architecture for 5G Core Networks. https://www.3g4g.co.uk/5G/5Gtech_6004_2017_11_Service-Based-Architecture-for-5G-Core-Networks_HR_Huawei.pdf.
[3] A. Sutton. 5G network architecture. *J. Inst. Telecommun. Professionals*, 12(1):9–15, 2018.
[4] 3GPP 5G System Architecture. https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/15.03.00_60/ts_123501v150300p.pdf.
[5] V. Nguyen, A. Brunstrom, K. Grinnemo, and J. Taheri. SDN/NFV-Based Mobile Packet Core Network Architectures: A Survey. *IEEE Communications Surveys Tutorials*, 19(3):1567–1602, 2017.
[6] A. Jain, Sadagopan N S, S. K. Lohani, and M. Vutukuru. A comparison of SDN and NFV for re-designing the LTE Packet Core. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 74–80, 2016.
[7] Z. Qazi, P. Penumarthi, V. Sekar, V. Gopalakrishnan, K. Joshi, and S. Das. KLEIN: A Minimally Disruptive Design for an Elastic Cellular Core. In *Proceedings of the ACM Symposium on SDN Research (SOSR)*, 2016.
[8] Z. A. Qazi, M. Walls, A. Panda, V. Sekar, S. Ratnasamy, and S. Shenker. A High Performance Packet Core for Next Generation Cellular Networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 348–361, New York, NY, USA, 2017. Association for Computing Machinery.
[9] M. T. Raza, D. Kim, K. Kim, S. Lu, and M. Gerla. Rethinking LTE network functions virtualization. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pages 1–10, 2017.
[10] P. C. Amogh, G. Veeramachaneni, A. K. Rangisetti, B. R. Tamma, and A. A. Franklin. A cloud native solution for dynamic auto scaling of MME in LTE. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–7, 2017.
[11] F. Ojala, A. Rao, H. Flinck, and S. Tarkoma. NoSQL stores for coreless mobile networks. In *2017 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 200–206, 2017.
[12] M. Pozza, A. Rao, A. Bujari, H. Flinck, C. E. Palazzi, and S. Tarkoma. A refactoring approach for optimizing mobile networks. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6, 2017.
[13] B. Nguyen, T. Zhang, B. Radunovic, R. Stutsman, T. Karagiannis, J. Kocur, and J. Van der Merwe. ECHO: A reliable distributed cellular core network for hyper-scale public clouds. In *Mobicom 2018*. ACM, October 2018.
[14] J. Cho, R. Stutsman, and J. Van der Merwe. MobileStream: A Scalable, Programmable and Evolvable Mobile Core Control Plane Platform. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, page 293–306, New York, NY, USA, 2018. Association for Computing Machinery.
[15] M. Moradi, K. Sundaresan, E. Chai, S. Rangarajan, and Z. M. Mao. SkyCore: Moving Core to the Edge for Untethered and Reliable UAV-Based LTE Networks. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, page 35–49, New York, NY, USA, 2018. Association for Computing Machinery.
[16] S. Sevilla, M. Johnson, P. Kosakanchit, J. Liang, and K. Heimerl. Experiences: Design, Implementation, and Deployment of CoLTE, a Community LTE Solution. In *The 25th Annual International Conference on Mobile Computing and Networking*, MobiCom '19, New York, NY, USA, 2019. Association for Computing Machinery.
[17] M. Moradi, Y. Lin, Z. M. Mao, S. Sen, and O. Spatscheck. SoftBox: A Customizable, Low-Latency, and Scalable 5G Core Network Architecture. *IEEE Journal on Selected Areas in Communications*, 36(3):438–456, 2018.
[18] V. Nagendra, A. Bhattacharya, A. Gandhi, and S. R. Das. MMLite: A Scalable and Resource Efficient Control Plane for Next Generation Cellular Packet Core. In *Proceedings of the 2019 ACM Symposium on SDN Research*, SOSR '19, page 69–83, New York, NY, USA, 2019. Association for Computing Machinery.
[19] R. Shah, V. Kumar, M. Vutukuru, and P. Kulkarni. TurboEPC: Leveraging Dataplane Programmability to Accelerate the Mobile Packet Core. In *Proceedings of the Symposium on SDN Research*, SOSR '20, page 83–95, New York, NY, USA, 2020. Association for Computing Machinery.
[20] A. Mohammadkhan, K. K. Ramakrishnan, and V. A. Jain. CleanG – Improving the Architecture and Protocols for Future Cellular Networks With NFV. *IEEE/ACM Transactions on Networking*, pages 1–14, 2020.
[21] M. Ahmad, S. U. Jafri, A. Ikram, W. N. A. Qasmi, M. A. Nawazish, Z. A. Uzmi, and Z. A. Qazi. A Low Latency and Consistent Cellular Control Plane. SIGCOMM '20, page 648–661, New York, NY, USA, 2020. Association for Computing Machinery.
[22] OpenAirInterface. https://www.openairinterface.org/.
[23] srsLTE. https://github.com/srsLTE/srsLTE.
[24] NextEPC. https://nextepc.org/.
[25] Free5GC. https://www.free5gc.org/.
[26] Open5GS. https://open5gs.org/.
[27] OpenAirInterface RAN (openairinterface5G). https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/home.
[28] OpenEPC. https://sites.google.com/a/corenetdynamics.com/openepc/home.
[29] J. Breen, A. Buffmire, J. Duerig, K. Dutt, E. Eide, M. Hibler, D. Johnson, S. K. Kasera, E. Lewis, D. Maas, A. Orange, N. Patwari, D. Reading, R. Ricci, D. Schurig, L. B. Stoller, J. Van der Merwe, K. Webb, and G. Wong. POWDER: Platform for Open Wireless Data-driven Experimental Research. In *Proceedings of the 14th International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH)*, September 2020.
[30] PhantomNET. https://www.phantomnet.org/.
[31] il_trafficgen. https://github.com/omec-project/il_trafficgen.
[32] S1APTester. https://github.com/facebookexperimental/S1APTester.
[33] UERANSIM. https://github.com/aligungr/UERANSIM.
[34] Open Mobile Evolved Core. https://www.opennetworking.org/omec/.
[35] Magma. https://connectivity.fb.com/magma/.
[36] TUN device. https://www.kernel.org/doc/Documentation/networking/tuntap.txt.
[37] Kubernetes. https://kubernetes.io/.
[38] Powder. https://powderwireless.net/.
[39] Docker. https://www.docker.com/.
[40] Docker Hub. https://hub.docker.com/.
[41] IANA Service Name and Transport Protocol Port Number Registry. https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt.
[42] 3GPP Specification Numbering. https://www.3gpp.org/specifications/79-specification-numbering.
[43] SCTP Payload Protocol Identifiers. https://www.iana.org/assignments/sctp-parameters/sctp-parameters.xhtml#sctp-parameters-25.
[44] Kubernetes Pod limitations. https://kubernetes.io/docs/setup/best-practices/cluster-large/.
[45] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, and J. Wang. Large-Scale Measurement and Characterization of Cellular Machine-to-Machine Traffic. *IEEE/ACM Transactions on Networking*, 21(6):1960–1973, 2013.
[46] G. Tsoukaneri, X. Foukas, and M. K. Marina. ASPIS: A Holistic and Practical Mechanism for Efficient MTC Support over Mobile Networks. In *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 284–292, 2017.
[47] G. Garcia-Aviles, M. Gramaglia, P. Serrano, and A. Banchs. POSENS: A Practical Open Source Solution for End-to-End Network Slicing. *IEEE Wireless Communications*, 25(5):30–37, 2018.
[48] Ixia Keysight. https://about.keysight.com/en/newsroom/pr/2019/21feb-nr19025.shtml.
[49] Viavi TeraVM. https://www.viavisolutions.com/en-uk/products/teravm.
[50] Spirent Landslide. https://www.spirent.com/products/core-network-test-5g-lte-ims-wifi-diameter-landslide.
[51] ng4T. https://www.ng4t.com/.
[52] dsTest. https://www.developingsolutions.com/products/about-dstest/.