IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 7, 2021, accepted July 17, 2021, date of publication July 26, 2021, date of current version August 2, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3099370

Live Migration of Virtual Machine and Container Based Mobile Core Network Components: A Comprehensive Study

SHUNMUGAPRIYA RAMANATHAN[1], (Graduate Student Member, IEEE), KOTESWARARAO KONDEPU[2], (Senior Member, IEEE), MIGUEL RAZO[1], (Member, IEEE), MARCO TACCA[1], (Senior Member, IEEE), LUCA VALCARENGHI[3], (Senior Member, IEEE), AND ANDREA FUMAGALLI[1], (Senior Member, IEEE)

[1]Open Networking Advanced Research (OpNeAR) Laboratory, Erik Jonsson School of Engineering and Computer Science, The University of Texas at Dallas, Richardson, TX 75080, USA
[2]Indian Institute of Technology Dharwad, Dharwad 580011, India
[3]Scuola Superiore Sant'Anna, 56127 Pisa, Italy

Corresponding author: Shunmugapriya Ramanathan (sxr173131@utdallas.edu)

This work was supported in part by the NSF under Grant CNS-1405405, Grant CNS-1409849, Grant ACI-1541461, and Grant CNS-1531039T; and in part by the EU Commission through the 5GROWTH Project under Grant 856709.

ABSTRACT With the increasing demand for openness, flexibility, and monetization, the Network Function Virtualization (NFV) of mobile network functions has become the embracing factor for most mobile network operators. Early reported field deployments of virtualized Evolved Packet Core (EPC) — the core network (CN) component of 4G LTE and 5G non-standalone mobile networks — reflect this growing trend. To best meet the requirements of power management, load balancing, and fault tolerance in the cloud environment, the need for live migration of these virtualized components cannot be shunned. Virtualization platforms of interest include both Virtual Machines (VMs) and Containers, with the latter option offering more lightweight characteristics. This paper's first contribution is the proposal of a framework that enables migration of containerised virtual EPC components using an open-source migration solution which does not fully support the mobile network protocol stack yet. The second contribution is an experimental-based comprehensive analysis of live migration in two virtualization technologies — VM and Container — with the additional scrutinization on the container migration approach. The presented experimental comparison accounts for several system parameters and configurations: flavor (image) size, network characteristics, processor hardware architecture model, and the CPU load of the backhaul network components. The comparison reveals that the live migration completion time and also the end-user service interruption time of the virtualized EPC components is reduced approximately by 70% in the container platform when using the proposed framework.

INDEX TERMS C-RAN, virtual EPC, VM, Docker, container, live migration, CRIU, CloudLab, OAI.

I. INTRODUCTION

The 3GPP standards for the 5G mobile communication and the ETSI Network Function Virtualization (NFV) [1] are two key enablers for 5G virtualization. The Non-Standalone version of the 5G and the 4G LTE systems have the Evolved Packet Core (EPC) as the backhaul network for providing converged voice and data communication. Many ongoing 5G research activities focus on the NFV version of the EPC (referred to as virtualized EPC — vEPC) that is expected to improve system flexibility and scalability while reducing operation cost of Mobile Network Operators (MNOs) [2], [3]. Well-established virtualization orchestration platforms exist that support both the Virtual Machine (VM) and the Container-based hardware virtualization such as OpenStack, VMware vCenter, and Open Source MANO (OSM) [4]–[6]. While these virtualization platforms are widely used with commercial off-the-shelf (COTS) hardware, some open challenges remain to be addressed [7]. One of these challenges is to achieve the Quality of Service (QoS) that is required by the carrier-grade Service Level Agreement (SLA) [8]. Many believe that the vEPC compute platform must include features

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

such as live migration, snapshot, and rebirth to ensure that the SLA requirements are finally met in terms of security, reliability, and total cost of ownership. In particular, live migration is the process of migrating VNFs from one node to another one while guaranteeing zero or minimal impact to the connectivity service offered to mobile network users. Being able to live migrate the VNFs of the vEPC offers a number of significant advantages, including (i) achieving load balancing in the compute nodes [9] by timely redistributing VNFs to sparsely loaded servers, (ii) effectively performing regular maintenance — such as upgrading OS versions and changing network configurations — and (iii) effectively handling fault management. These features may also prove to be of the essence to cope with the highly fluctuating mobile traffic that is expected in real-time networks.

Realizing the importance of timely offering vEPC solutions with built-in capability for VNF live migration, this paper describes the *open-source experimental settings* designed toward this goal. Two virtualization techniques are considered, one based on VMs and the other based on Docker containers. In the former platform, live migration of VNFs running as VMs is achieved through Kernel-based Virtual Machine/Quick EMUlator (KVM/QEMU) with the *libvirt* API [10]. In the latter platform, live migration of VNFs running as Docker containers is achieved through Checkpoint and Restore In Userspace (CRIU) [11]. Even though it is highly beneficial to have containerized vEPC because of its relatively small footprint that minimizes both the storage and processing capability requirements, the main challenge is envisioning the containerized core network that comes with the inherent migration limitations discussed in Section IV-A. We leverage open software and standard solutions whenever possible, and address the container migration challenges by implementing additional custom software packages that are necessary to complete the required NFV/SDN architecture. The ultimate objective is to validate the feasibility and compare the performance of a few plausible NFV/SDN architectures, which provide live migration of vEPC functions with reduced connectivity disruption to the mobile user. Specifically, EPC as a Service (EPCaaS), for which live migration is tested, has three main core network components, namely Home Subscriber Server (HSS), Mobility Management Entity (MME), and Serving and Packet Gateway (SPGW). In our study, these vEPC components are implemented by using the OpenAirInterface (OAI) software package. To the best of our knowledge, this is the first systematic study of live migration of core network components in the containerized environment and head-to-head comparison with the VM environment.

II. LIVE CORE NETWORK MIGRATION: MOTIVATION AND BACKGROUND

This section provides a brief description of the relevant background and motivation of our work.

A. CORE NETWORK MIGRATION

The Mobile Carrier Cloud relies on a few key technologies such as NFV, SDN, and Cloud Computing which, combined, enable the implementation of vEPC as a service (EPCaaS) in the cloud. To ensure optimal connectivity to the end-users while at the same time offering flexibility and cost reduction, vEPC presents many challenges in deploying and relocating its VNF components across the underlying federated cloud. A number of studies has focused on determining the optimal number of vEPC components and their deployment relocation to best sustain QoS [12]. Also, to handle crises such as disaster recovery, periodic checkpoints of the application's state information are required, which are then used during the relocation. This relocation (live migration concept) has been studied for various user services [13], [14], but not for EPCaaS.

The telco-supported core network services are quite different from the general application's supported protocol and operating system, and require a dedicated study of core network migration under various virtualization platforms. The main challenge is maintaining a relatively short *service downtime* and *migration time* to retain the benefits of EPCaaS in the cloud. Lightweight migration is one of the promising techniques that is expected to reduce the migration time.

B. VM VS CONTAINER

VMs provide OS, kernel, and hardware-level virtualization while the containers make use of OS-level virtualization to produce VNFs. Docker-based containers [15] have been gaining traction in recent years due to their reduced resource usage and lower virtualization overhead. Container offers near-native performance to the application because of its execution within the Host-OS environment. It must be noted that CRIU with the widely-used runC based containerized software package (available in Docker, Containerd, LXC, Kubernetes Clusters) does not offer two key migration functionalities [16]: i) support for the Stream Controlled Transmission Protocol (SCTP) used in the LTE network to guarantee message delivery between MME and eNB, and ii) GPRS Tunneling Protocol (GTP) device-specific information needed by the SPGW software to provide tunneling of the user data traffic. These functionalities are required to support the telco-core network live migration while preserving the connectivity with the RAN components. To overcome these containerized core network migration limitations, two custom solutions — described in Section IV-A — are proposed and integrated into the experimental settings. Although it is well-known that containers are lightweight compared to VMs, a quantitative study of their effect on the core network migration performance does not exist in the literature. The container-supported middleware described in Section IV-A enables us to come up with this quantitative study and draw further insights.

C. MIGRATION TYPES AND DECISION OPTIMIZATION

Stateless and Stateful migrations are two categories of Service migration. Based on the migration needs, a new instance of the application is instantiated at the destination node. In Stateless migration, the current running state of the application is not moved to the destination node. Stateless migration is helpful when there is no need to store and retrieve the application states. On the other hand, the core network components must keep track of their respective state in order to guarantee continuity of their active interactions with the other Radio Access Network (RAN) components. We, therefore, consider Stateful migration of vEPC components in our study. Once the Service disruption caused by the stateful migration process is quantified experimentally, previously proposed [17]–[19] and newly designed optimization algorithms can be more effectively applied to decide when and where to migrate the VNFs.

III. RELATED WORK

EPC resiliency has already been addressed independently from the fact that EPC functions are virtualized or not. In [20], the 3rd Generation Partnership Project (3GPP) specifies different resiliency mechanisms for EPC components. It also reports how to handle failures with the help of Echo Request/Response timer messages. In addition, methodologies for recovering VNFs have been proposed. For example, in [21], multiple approaches are proposed for recovering VNF through replication and migration of network functions when outages affect compute resources. Well-known methods are available for regaining connectivity between VNFs that has been disrupted by a network failure, including those that resort to a Software Defined Network (SDN) controller [22]. More advanced solutions combine VNF replication/migration with network connection rerouting for improved resiliency [23].

More recently, methods have been proposed to specifically address failures when 5G functions are virtualized. In [24], a two-step resiliency scheme is proposed to overcome soft failures of optical circuits (lightpaths) carrying the fronthaul traffic. When not enough resources are available along the backup lightpath, the next generation eNB (gNB) functional split is dynamically reconfigured to reduce the fronthaul capacity requirements. In [13], the authors evaluated the container-based live migration of the blank and video applications to meet the Multi Access Edge (MEC) computing requirements during radio handover. The VNF migration of virtualized Central Unit/virtualized Distributed Unit (vCU/vDU) over Wavelength Division Multiplexed (WDM) network using CRIU is briefly discussed in [25]. This study focuses on the resource orchestration framework for forwarding control and VNF management, but does not provide a full description of the design and implementation choices made.

To the best of the authors' knowledge, there have been no findings addressing the implementation and providing a detailed performance evaluation of NFV-SDN systems carrying out live migration of VMs and containers supporting core network functions.

IV. LIVE MIGRATION OF VIRTUALIZED CORE NETWORK FUNCTIONS: TECHNIQUES, LIMITATIONS, AND SOLUTIONS

This section describes the migration strategies exploited, the key implementation challenges faced, and the custom software solutions developed for successfully performing core network live migration. For more details, the reader is referred to the Appendix sectionVII.

A. DOCKER CONTAINER MIGRATION WITH CRIU

Platforms for managing containerized workloads and services, such as Kubernetes, resort to runC [26] to spawn and run containers. runC is a lightweight, portable container runtime, initiated by the Open Container Initiative (OCI) and utilized by many container-based virtualization solutions, such as Docker and Containerd. To implement container live migration in Linux operating systems, runC resorts to Checkpoint Restoration In Userspace (CRIU), which is a software component offering checkpoint/restore functionality for Linux applications [11]. CRIU features several methods for container migration including StopandCopy, PreCopy, PostCopy, and HybridCopy. Unfortunately, such methods do not natively support virtualized mobile core functions. Here below, the main limitations [16] are listed along with the solutions adopted in this study to overcome them.

1) SPGW CONTAINER LIVE MIGRATION

In the LTE mobile network, IP packets are exchanged between User Equipment (UE) and SPGW component[1] through GTP tunnels. Here, the base station (i.e., the eNB) acts as the Serving GPRS Support Node (SGSN), and the SPGW acts as the Gateway GPRS Support Node (GGSN). Tunnel Endpoint Identifier (TEID) values are mutually exchanged between the base station and the SPGW to ensure correct data traffic flow. The SPGW maintains an updated list of the active GTP tunnels — TEID with UEs and base station relevant information.

The utilized OAI SPGW software implements the aforementioned data plane connectivity by using the rtNetlink socket in the Linux Kernel GTP tunneling module. This kernel module creates the GTP device interface (*gtp0*) for tunneling the user data traffic to the Packet Data Network (PDN). Unfortunately, with the current open software platforms (OAI, CRIU), the *gtp0* device and TEID information are not carried over onto the destination node during the SPGW migration. Also, the rtNetlink socket is deactivated at the source node once the checkpoint is initiated. Thus, after migration, the end-user connectivity is lost, and the entire UE connection *re-establishment* must take place again.

[1] In general, Serving Gateway (S-GW) and Packet Gateway (P-GW) are two separate components of the LTE-A core network, but in this study, they are implemented as a single entity as in Openairinterface.

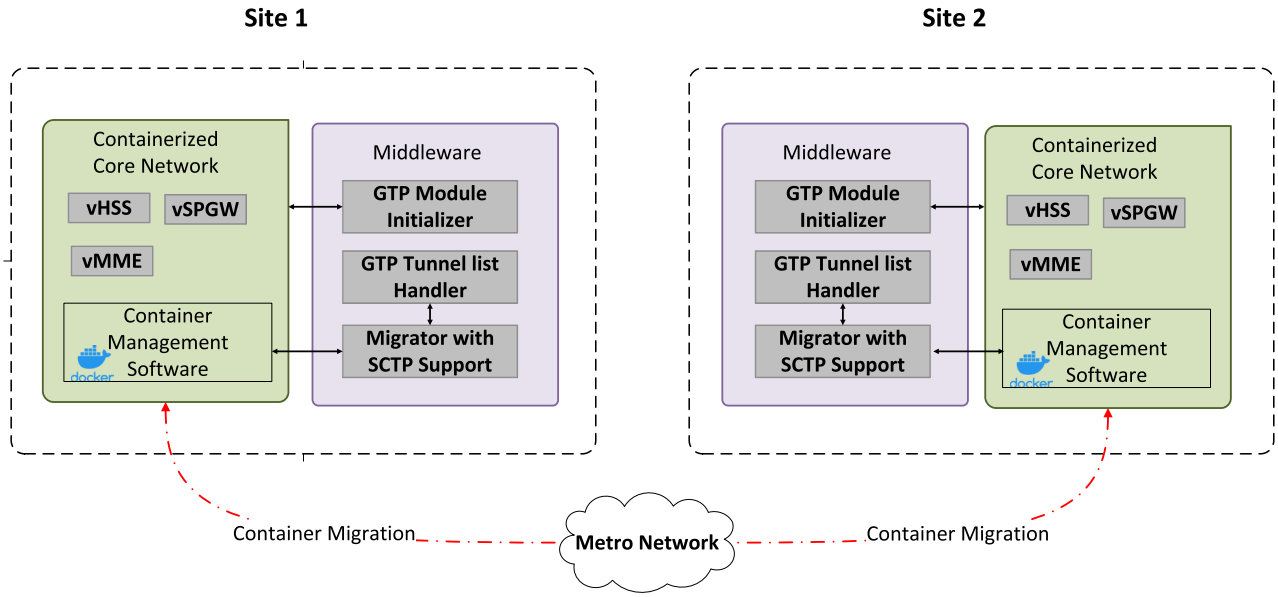


FIGURE 1. Proposed containerized core components module.

Appendix A presents a detailed description of these limitations. To overcome the above-discussed migration limitations, this study implements the solution depicted in Fig. 1. The proposed architecture introduces a new module called *middleware*, which contains *GTP Module Initializer* and *GTP Tunnel Handler*.

The GTP Module Initializer takes the prime responsibility of configuring the GTP device interface required by the SPGW application. Its high-level functions are:

1) to set the GTP kernel module and create the GTP tunnel device using rtNetlink socket;
2) to read the interface information such as name, address, mask, UE-MTU size, and SGi interface name with the masquerading option from the SPGW configuration file;
3) to send the list of system commands to configure the GTP interface as per the SPGW application's requirements; and
4) to use the created GTP interface by the SPGW application.

Since the rtNetlink socket connectivity is now associated with this GTP Module Initializer, the SPGW application checkpoint does not deactivate the associated Netlink Socket. With the proposed GTP Module Initializer, the SPGW application can be restored on the destination node with the GTP interface and active Netlink Socket. Consequently, the GTP Module Initializer resolves the GTP Interface and the rtNetlink Socket issues in the existing system. The GTP Tunnel List Handler is responsible for storing current active tunnel information of the running SPGW application. Two design strategies are possible, i.e., a global proactive-based approach and a local reactive-based approach. In the former approach the list handler is placed in a shared central node, whereas in the latter the list handler is placed in each node, where the SPGW application is running. During the SPGW checkpoint process, the tunnel list of the running SPGW container is read by the handler and added to the stored metadata file used by the restore procedure. This solution successfully takes care of the metadata concerning the tunnel list information. More details can be found in Appendix B. The user-level implementation of the modules that support the GTP tunnel migration is available in a public repository [27]. The GTP link and tunnel modules in the user-level implementation — present in the repository — run along with the vSPGW container that performs the GTP Module Initializer and Tunnel List Handler functionalities mentioned above. The GTP Tunnel List Handler implementation is based on the local reactive-based approach. In addition, the migration script file — also placed in the repository — reads the active tunnel list information from the source node, before initiating the SPGW container checkpoint process.

2) MME CONTAINER LIVE MIGRATION

The MME component makes use of the SCTP to exchange S1-MME messages with multiple eNBs. One of the main differences between TCP and SCTP is that TCP has a single association in the given socket connectivity, whereas SCTP has multiple associations with the single socket connection.

The SCTP is not supported in the current version of the container checkpoint software (i.e., CRIU). Consequently, migration of the MME component cannot be executed successfully. The additional custom software, called *migrator*, described next, has been designed and developed to overcome this limitation.

The *migrator* is the CRIU software with the addition of SCTP support (along with the required kernel changes) for achieving MME container migration. Fig. 2(a) shows

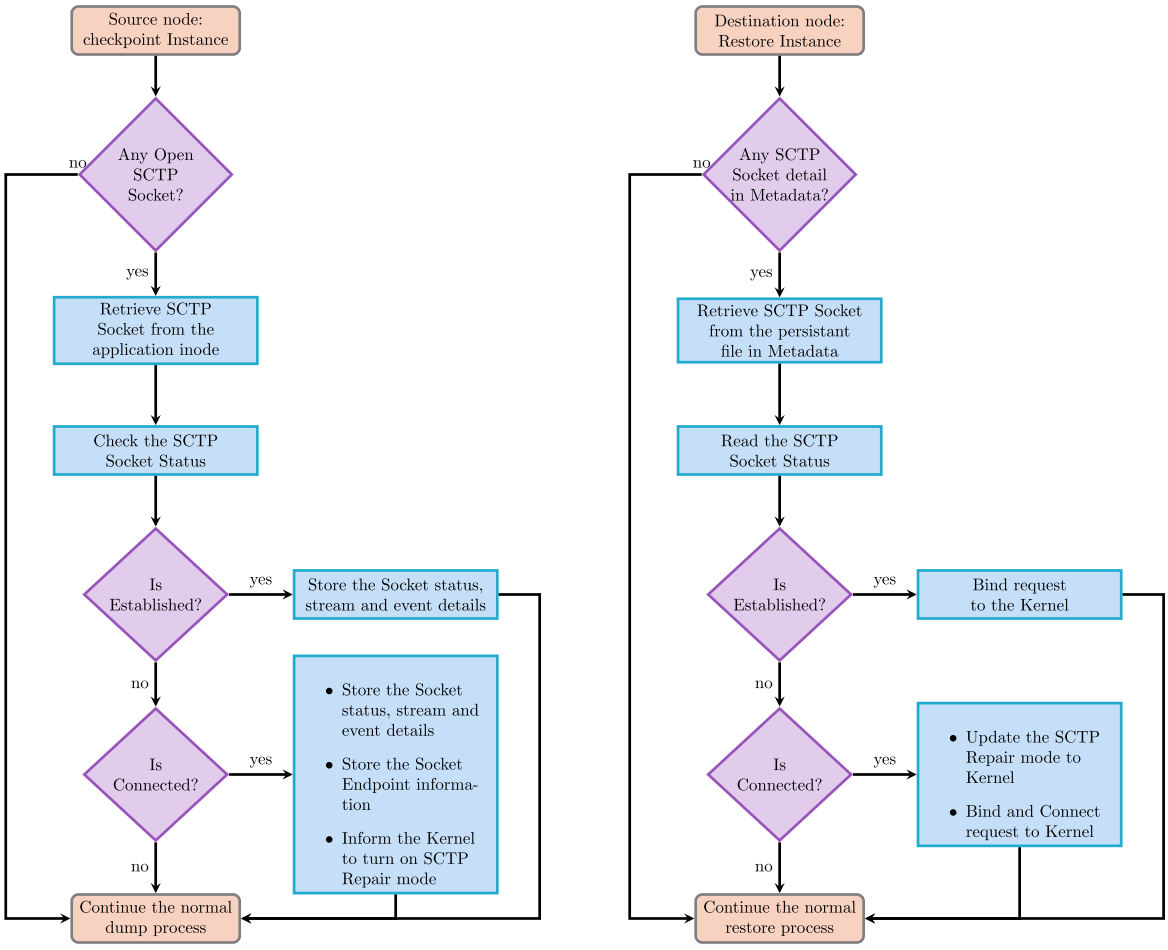


FIGURE 2. SCTP support in the migrator flowchart.

the migrator flowchart to support the SCTP migration at the source node. Once migration is initiated at the source node, if any SCTP socket connection is present, the migrator retrieves the SCTP socket information from the application inode [28]. Upon retrieving the SCTP information about the active association, if the SCTP socket status is *established*, the migrator stores the socket status, stream, and event details in a file. If the SCTP socket status is *connected*, it stores the socket end-point information (IP address and port detail) along with the socket status, stream, and event in the persistent file. Moreover, the migrator informs the Linux Kernel to turn *"ON"* the SCTP repair mode to avoid the repetition of the SCTP handshake process at the destination node for the connected SCTP sockets.

Fig. 2(b) shows the flowchart of the restore instance process at the destination node. Upon restore initiation, the migrator reads the SCTP persistent file and checks whether SCTP socket details are available in the transferred metadata. If that is the case, the migrator retrieves SCTP-associated information from the metadata and takes action based on the SCTP socket status. If the SCTP socket status from the metadata is successfully *established*, the migrator binds the request with the Linux Kernel, and continues with other restore steps. If the SCTP socket status is *connected*, the migrator updates the SCTP repair mode to the kernel module and initiates the bind and connection request with the associated endpoint. Then, the migrator continues with the normal restore process to resume application activities. The migrator support is added to the CRIU version 3.12. The user-level implementation of the SCTP migrator support-based CRIU code is available in a public repository [29]. The README document in the repository provides the list of modules added to save and restore the SCTP socket during vMME migration consistently with the flowchart in Fig. 2.

The SCTP module of the kernel code is customized with the SCTP REPAIR MODE option. If the repair mode is enabled for a socket,

- during the checkpoint process at the source node, the SCTP socket is closed at the source node without initiating the SCTP_SHUTDOWN or ABORT message;

- during the restore process at the destination node, the SCTP socket status is changed to *connected* without requiring to re-instantiate the SCTP handshake procedure.

The kernel module changes will ensure that both socket connectivity and association remain active at the other endpoint of the SCTP association during the migration. The feature changes are considered for the SCTP server scenario matching with the MME requirement. The kernel module changes are done on Ubuntu Mainline Kernel version 5.4.1 and are available in a repository [30].

3) HSS CONTAINER LIVE MIGRATION

The HSS component establishes a TCP socket at the start time and stores relevant user information in a MySQL database. A TCP connection is required between HSS and MME. Upon performing the HSS component's migration, the application needs to copy the database information into the memory page and restore the TCP connectivity at the destination node without disturbing the peer end connection state on the MME side. The *tcp-established mode* [31] must be set in the runC configuration to ensure TCP connection re-establishment at the destination node. This feature option is supported starting from version 3.5 of the Linux kernel mainline. No additional custom software is required.

4) CONTAINER MIGRATION TECHNIQUE ANALYSIS

Fig. 3 depicts the different phases of the container migration based on the StopandCopy method: checkpoint, transfer, and restore. During the checkpoint, CRIU freezes the running container at the source node (node A) and collects metadata about the CPU state, memory content, and information about the process tree [32] associated with the running container service. During the transfer phase, the collected metadata information is transferred to the destination node (node B). The restore phase resumes the container service from the frozen point with the transferred metadata at the destination node. The container migration time can be expressed as:

$$T_{cm} = T_c + T_t + T_r \tag{1}$$

where:

- T_{cm} is the total container migration time,
- T_c is the checkpoint time,
- T_t is the metadata transfer time from the source to the destination node, and
- T_r is the restore time.

For the StopandCopy method, T_{cm} also corresponds to the service downtime because the running container service is interrupted at the start of the checkpoint phase, and the service is restored only after the successful restoration.

The PreCopy method shown in Fig. 4 is another container migration approach. It consists of pre-dump, dump, pre-dump/dump data transfer, and restore phases. The application checkpoint at the source node (node A) is split into pre-dump and dump phases. The pre-dump phase collects all the container state and memory information, and the dump

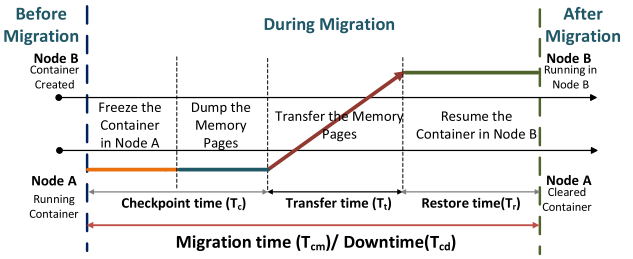


FIGURE 3. Container StopandCopy method of migration.

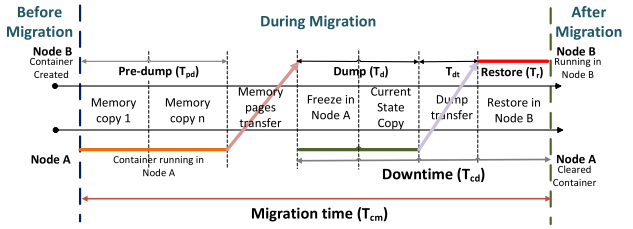


FIGURE 4. Container PreCopy method of migration.

phase collects the modified memory pages information only. In the transfer phase, the collected metadata is transferred to the destination node (node B), and in the restore phase the application service is restored.

The container migration time in the PreCopy approach can be expressed as:

$$T_{pcm} = T_{pd} + T_{pdt} + T_d + T_{dt} + T_r \tag{2}$$

where:

- T_{pcm} is the PreCopy based total container migration,
- T_{pd} is the pre-dump time,
- T_{pdt} is the pre-dump data transfer time,
- T_d is the dump time, and
- T_{dt} is the dump data transfer time.

The advantage of the PreCopy method is that the container service remains responsive during the pre-dump phase of collecting the process id, memory pages, and the execution state. As depicted in Fig. 4, the container service gets interrupted only at the start of dump phase — carrying the modified memory pages. In this case, the container PreCopy downtime T_{pcd} can be expressed as:

$$T_{pcd} = T_d + T_{dt} + T_r. \tag{3}$$

The core network elements need permanent socket connectivity with their peer elements for the control and data plane services. To avoid socket termination, the floating IP address needs to be configured to enable its seamless transfer from the source node to the destination node during migration. In our evaluation, the VPN configuration handles this floating IP requirement, adding extra time to the network setup at the destination node. The IP address dependability is explained in Appendix C. Both the PreCopy and the StopandCopy approaches are evaluated in the result analysis section to

show the impact of the migration technique chosen within the container core network components.

5) CONTAINER MIGRATION INTERRUPTION ANALYSIS

While in process, the container live migration might be interrupted due to certain common failures such as inadequate resources, root file system size mismatch, and network interruption. These most common container migration failures are simpler to correct, as discussed below.

- Inadequate resources - During the memory page transfer, file synchronization is enabled to keep track of the copied memory pages. To expedite this step only the difference between the source and destination node files is transferred. File synchronization aborts if the destination disk is full causing the live migration to fail. In this situation, the administrator must either select a different destination node or free-up the hardware resources on the current destination node before repeating the live migration.
- Root file system size mismatch - The container image is placed in the root file system that contains the system specific configuration files. At run time, the temporary log files inside the root file system might change in the source node based on the application requirement. This log changes lead to a container image size mismatch between the source and destination nodes. During such occasions, the restore phase fails resulting in an unsuccessful migration. This problem could be alleviated using the temporary file system mechanism while creating the container image. More details are available in Appendix C.
- Network interruption - Unexpected problems in the network connecting the source and destination nodes may lead to data packet loss during a live migration, which would prevent successful resumption of the application at the destination node. The file synchronization method helps to detect the mismatch of files and directories between the two locations. During such occasions, the memory copy process could be re-initiated using the multiple sub-flows for a connection to improve the performance and reliability of the migration process as discussed in [33].

For both the StopandCopy and PreCopy based container migrations, the custom shell script uses the rsync command-line utility [34] to achieve file synchronization. The advantage of this implementation is that when there is a failure in the container live migration, the migration process can roll back to the source and resume the application from the frozen point in the source node to avoid user service interruption. Live migration can be re-initiated after correcting the problem.

Table 1 summarizes the resume phases of the VNF at the source node in the three migration interruption scenarios. The different phases of the migration are explained in the Figs. 3, 4 for the Container StopandCopy and PreCopy methods, respectively. The evaluation of both the interruption time (downtime) and UE service recovery time during successful core network migration can be found in Sections VI-B and VI-C, respectively.

TABLE 1. Container migration failure scenarios.

Failure Scenarios	VNF service status at source node	
	Container StopandCopy	Container PreCopy
Inadequate resources	Service will resume after memory page transfer phase	Service will not be interrupted
Root file system size mismatch	Service will resume after the restore phase	
Network interruption	Service will resume after memory page transfer phase	Service will resume after the dump transfer phase

B. VM MIGRATION WITH THE KVM/QEMU HYPERVISOR

In the Virtual Machine (VM) migration, the entire operating system along with the application is migrated from one physical machine to another. Several virtualization frameworks are available, either open-source or commercial, such as Kernel-based Virtual Machine (KVM), VirtualBox, and VmWare. They all are based on a hypervisor that is a software-based virtualization layer between the physical machine (node) and the VM guests running on it. The hypervisor takes care of scheduling and allocating compute resources to the VM guests. KVM hypervisor is a kernel module integrated with version 2.6.20 of mainline Linux Kernel, and it is used in OpenStack [10] for providing the virtualization infrastructure. The QEMU-KVM module provides VM management such as spawning and migrating VMs using the guest execution mode.

Thanks to its maturity, the KVM/QEMU hypervisor [35] supports the live migration of VM components running core network elements. Only a few additional precautions are necessary to ensure correct migration execution in the OpenStack environment. Here, layer-2 network connectivity is provided by default using the Open Virtual Switch (OVS) integration bridge [36]. Even though the OpenStack security rule supports the SCTP protocol, the OVS firewall blocks the SCTP packets preventing them from reaching the hosts. SCTP messages must be encapsulated inside UDP frames by using Open Virtual Private Network/Virtual Extensible LAN (OpenVPN/VXLAN) connectivity to circumvent this drawback. The resulting UDP frames are not blocked by the OVS firewall. In our VM study, the SCTP protocol communication between MME and CU is achieved through an OpenVPN configuration in the S1-MME communication interface [37].

1) VM MIGRATION TECHNIQUE ANALYSIS

During VM migration, the CPU state, memory state, network interfaces, and disk image of the entire VM are migrated from the source to the destination node. During the process of copying memory pages, the dirty pages (i.e., modified memory pages) are iteratively transferred (referred to as *push* phase) while the VM is still running at the source node.

Once the maximum iteration count is reached, the VM is temporarily stopped at the source node, all the main memory pages are copied to the destination, and finally the VM is resumed at the destination node. This process of memory page coping is referred to as *PreCopy* method and illustrated in Fig. 5.

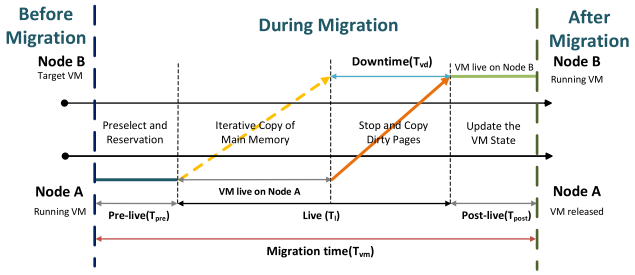


FIGURE 5. VM PreCopy method of migration.

The VM Migration time can be expressed as:

$$T_{vm} = T_{pre} + T_l + T_{post} \tag{4}$$

where:

- T_{vm} is the total VM Migration time,
- T_{pre} is the pre-live time,
- T_l is the live transfer time, and
- T_{post} is the post-live time.

As shown in Fig. 5, the *pre-live* phase considers the pre-selection of destination node and resource reservation at the destination node – such as preparing the destination node with the VNF instance details of keypair association and network interface information. In this *pre-live* phase, the hardware architecture compatibility validation takes place between the source and the destination nodes. The *live* phase does the iterative memory page copy from the source node to the destination node — referred to as the push phase — in the VM PreCopy method. The push phase is followed by the stop-and-copy phase, where the VNF is suspended at the source node and the final dirty pages and processor state information is transferred to the destination node. The downtime is observed during this stop-and-copy phase of VM migration. The *post-live* phase performs post-operation — referred as commit — after the live migration. It updates the running VM state in the MySQL and updates the Neutron database with the node information and port details. Once the migration is completed successfully, the hypervisor resumes the VNF in the destination node.

2) VM MIGRATION INTERRUPTION ANALYSIS

In real-time deployment, several possible events might cause the VM live migration to fail. In such cases, the application continues to run at the source node. Some of the most common VM live migration interruptions [38] are similar to that of container migration interruption. However, the VM migration failure handling mechanisms are quite different as discussed next.

TABLE 2. VM migration failure scenarios.

	VNF service status at source node
Failure Scenarios	**VM StopandCopy**
Incompatible processor	Detected at pre-live phase and the service is not interrupted
Insufficient resources	Detected at pre-live phase and the service is not interrupted
Network interruption	Service resumes after dirty page transfer in the stop-and-copy phase

- Incompatible processor - A successful VM migration requires that all physical host processors must belong to the same processor family. The pre-live phase ensures the compatibility before it initiates the live migration phase. For instance, the live migration cannot be carried out successfully from a host with Intel processor to a host with AMD processor.
- Insufficient resources - Destination node must have adequate physical resources to accommodate the incoming VM. The resource reservation process of the pre-live phase performs this validation and stops the live copy process if this condition is not met.
- Network interruption - VM migration is a resource intensive process which consumes significant network bandwidth while transferring disk information. It could saturate the network link when migrating VMs from the source to the destination node. To circumvent this problem shared storage type of migration would be preferred in datacenters that deploy Storage Area Network (SAN) [39].

For the Pre-copy VM migration, the VM application running in the source node is released only after the successful completion of the stop-and-copy phase. Only then the hypervisor in the destination node starts the VM and resumes normal operation. Table 2 summarizes the resume phases of the VNF at the source node under the three VM migration failure scenarios. The performance indicators during a successful VM migration are discussed in the forthcoming sections.

V. EXPERIMENTAL EVALUATION

This section describes the testbed used to experimentally evaluate the vEPC migration feasibility and performance. The considered performance parameters and influencing factors are also detailed.

A. TESTBED

The live migration solutions in Section IV are implemented in a geographically distributed testbed where the RAN components are hosted in the UTDLab (*Dallas*) and the CN components are hosted in CloudLab (*Utah*) [40]. The UTDLab and CloudLab are connected via Internet2. The CloudLab testbed provides the opportunity to validate and test the portability of the applied software configuration changes, new utility program, and CRIU code changes as detailed in Section IV-A using an open environment beside the in-house UTDLab setting.

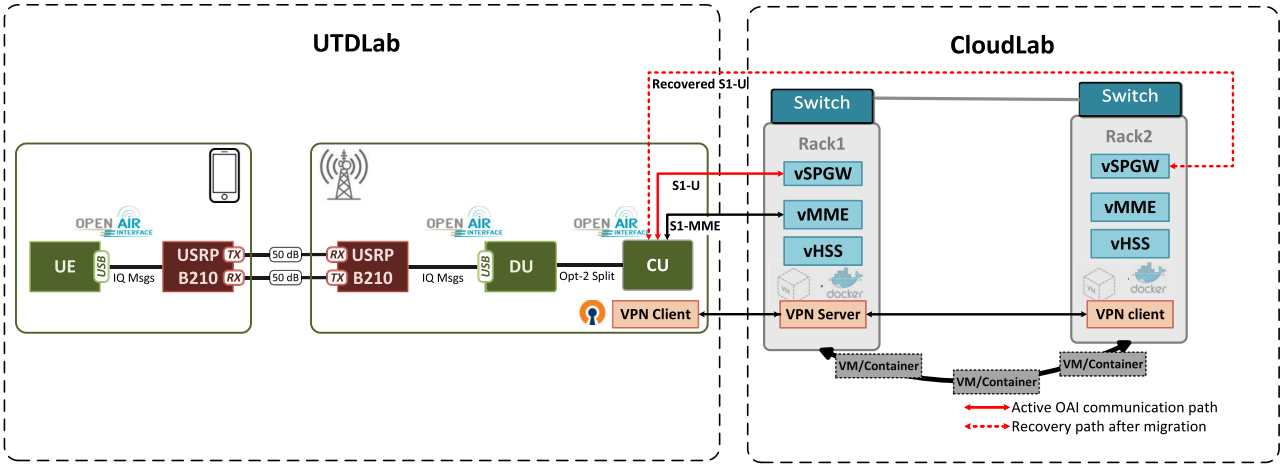


FIGURE 6. The UTDallas-CloudLab testbed.

In the UTDLab, a hydrid LTE-A/5G RAN is implemented by using OAI [41]. All the experiments make use of gNB option 2 split [42] into Distributed Unit (DU) and Central Unit (CU), according to which both Packet Data Convergence Protocol (PDCP) and Radio Resource Control (RRC) services run on the CU. The components UE, DU and CU run on dedicated physical machines. The radio units (RU) are implemented by using NI B210 radio prototyping boards [43], as shown in Fig. 6. The OAI software version for the RAN units - the UE, DU and the CU is *v2019.w25*, and the version for the core network is *v0.5.0-4-g724542d*. The interface between the radio unit (i.e., B210) and the UE/DU is USB 3.0.

Compute nodes in the CloudLab testbed are co-located to test intra-datacenter virtualized EPC component migration. The container hosts are deployed on top of Ubuntu-16 VM hosts in the OpenStack environment. The migration procedure is carried out through the OpenStack dashboard when using VMs, through Docker CLI shell script commands when using Container StopandCopy, and runC CLI shell script commands when using the Container PreCopy approach.

The CloudLab testbed provides a virtualized platform that is controlled by OpenStack [4] and is not orchestrated by an ETSI MANO [44] compliant orchestrator. As a result, the proposed solution uses custom shell scripts to migrate vEPC components while addressing the migration limitations described in Sections IV. Some ETSI MANO-compliant orchestrators such as Open Baton [45] provides a middleware solution based on TOSCA templates, which can be used to extend the proposed solution for orchestration compliance using custom scripts. Similarly, Cloudify [46] offers script plugins that can be used to run shell scripts as part of the migration orchestration process. The CloudLab system configuration details are reported in Table 3.

B. PERFORMANCE INDICATORS

The chosen key performance indicators are the *Migration time*, *Downtime*, *UE service recovery time*, and *Migration data size* [47]–[49]. These performance indicators are evaluated while migrating the CN components and keeping one UE connected. Their definitions are as follows:

- *Migration time* is defined as the required time to migrate a VNF from one node to another node. In the VM migration experiment, the considered migration time T_{vm} in Eq. (4) is evaluated from the OpenStack nova-compute log on the source node. For the Container StopandCopy and PreCopy methods the migration time — in Eqs. (1) and (5), respectively — is measured with millisecond resolution using the shell script that automates the migration process.[2]
- *Downtime* is defined as the amount of time the VNF functionality is paused and unavailable. In the VM migration experiment, the downtime is associated with the stop-and-copy phase, where the final dirty

TABLE 3. CloudLab system configuration.

Description	CloudLab
Nodes	1 control node and 4 compute nodes on the same rack
Product	HP m510
CPU	1 Intel Xeon D-1548 processors@2.00GHZ (8 cores, 2 threads/core)
Intel Architecture	Broadwell
Memory	RAM: 64GB, Disk: 256GB Flash Storage
OpenStack-Management N/W	10G (Dual-port Mellanox ConnectX-3)
OpenStack-Tenant N/W	10G (Dual-port Mellanox ConnectX-3)
QEMU version	3.1.0
Libvirt version	5.0.0
Docker version	19.03.13
Containerd version	1.2.13
CRIU version	3.12
runc version	1.0.0-rc10
CPU Utilization	< 10 %

[2]Eq. (5) is detailed in Appendix C

pages and processor state information are transferred from the source node to the destination node. In the Container StopandCopy migration experiment, migration time and downtime have the same value as defined in Eq. (1). In the Container PreCopy migration experiment, the downtime is given in Eq. (3). In all three experiments the downtime is measured using the ICMP non-responsive ping of the VNF IP with millisecond resolution.

- *UE service recovery time* is defined as the time that is required by the UE to regain mobile connectivity from the moment the UE service is temporarily paused due to the migration of one of the EPC components. The service recovery relies on the reactivation of the Control and Data services. The UE Control Plane (CP) service is momentarily interrupted during the MME migration. Similarly, during the SPGW service migration, the UE data uplink and downlink are temporarily disrupted. In all experiments, the UE service recovery time is measured using the ICMP ping from the UE *gtp* interface IP to both the MME *CP* and SPGW *gtp* interface IP, with a resolution of one hundred milliseconds.
- *Migration data size* is defined as the amount of data transferred from the source node to the destination node during the migration. In the VM migration experiment, the migration data accounts for the transfer of the CPU state, memory state, network state, and disk state. Its size is measured using the system monitoring tool named ''iftop'' [50]. In the container migration experiment, the migration data accounts for the process tree, the CPU state, memory page, namespace, and control group (cgroup) information. It is the size of metadata files taken during the application checkpoint process.

C. INFLUENCING PARAMETERS

The performance of the VNF migration is affected by a number of additional factors beside the migration method used. In OpenStack, the VM migration data flows through the management network, whose transmission data rate may affect the migration required time. Conversely, while migrating a container the OpenStack tenant network is used to transfer the metadata and its transmission data rate may affect the migration required time.

In addition, in the container migration experiment the checkpoint and restore services heavily relay on the node hardware architecture, cache size, and number of cores used. Similarly, in the VM migration experiment the hypervisor must copy the memory pages of the CN component, which also depends on the node architecture in terms of memory access speed and bandwidth. The flavors of the computing instances (compute, memory, and storage capacity) may also affect the time required to migrate both VMs and containers to the new node. Table 4 reports three flavors that are applied in this study.

In summary, the impact on migration performance is evaluated for the following parameters:

TABLE 4. OpenStack flavors used.

No	Flavor Name	vCPUs	RAM [MB]	Disk [GB]
1	Small	1	2048	20
2	Medium	2	4096	40
3	Large	4	8092	40

- Image flavor,
- Processor hardware architecture model,
- Network Interface Card (NIC) data rate, and
- CPU load.

The experiments related to the influencing factors consider the standalone vSPGW component migration to evaluate the impact of migration performance for the fault handling use-cases such as periodic checkpoint and data backup [51].

VI. RESULT ANALYSIS

This section reports the experimental data collected using the testbed. Each experiment is repeated five times totalling overall three hundred trials. The mean value along with the confidence interval at the 95% confidence level and the coefficient of variation are reported for each performance indicator, accounting for the stochastic variations due to the network, I/O, and processing delays.

A. MIGRATION TIME ANALYSIS

The migration time plays an important role in supporting ultra-reliable and low latency applications. Fig. 7 reports the migration time of the virtualized HSS, MME, and SPGW instances for the *VM PreCopy*, *Container StopandCopy* and *Container PreCopy* methods. OpenStack Flavor Small is considered in the CloudLab testbed operating with Broadwell architecture and 10G NIC rate. As expected, in terms of migration time, the container based migration is consistently better compared to that of VM. The file system and in-memory contents of the container are mainly determined by the application. For the VM, there are additional background process-related files and memory contents that require a larger image size to be migrated. When migrating containers, the PreCopy method requires considerably less time for all the core network components when compared to the StopandCopy method. Even though the metadata size remains the same for both container migration methods, the tool used to initiate the migration process affects the overall required migration time. In the StopandCopy method migration is triggered from the Docker CLI, which interacts with the Docker daemon, Containerd, and runC to carry forward the migration process. In the PreCopy method the migration process is initiated from the runC CLI.

Table 5 shows the time taken by each phase of the VM migration approach. In the live migration phase, since the VM image size is the same for all the OAI-based core network components, the memory content transfer takes approximately the same time for the HSS, MME, and SPGW components. Since the post-live phase involves the database

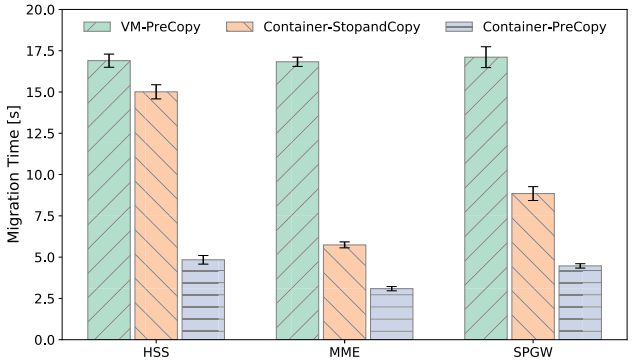


FIGURE 7. Migration time comparison for three migration methods.

TABLE 5. VM PreCopy method: Migration time breakdown.

	CloudLab - Flavor Small		
Migration time [s]	**HSS**	**MME**	**SPGW**
Pre-live	2.9	2.83	2.97
Live	10	10	10.14
Post-live	4	4	4

update, all the CN components take approximately 4 seconds in the CloudLab Broadwell environment.

For the Container StopandCopy method of migration, the checkpoint and the restore time are relatively high for the HSS component as shown in Table 6. The difference in timing is mainly due to the metadata information processing for the container application. For the HSS service, the container metadata stores the user information in the MySQL database. As the number of UE client information increases, the HSS metadata size increases as well. On the contrary, the metadata of both the MME and SPGW applications comprises mainly control-plane and data-plane socket connection information, thus requiring comparatively lesser persistent data content leading to the faster checkpoint and restore time than that of the HSS container.

Table 7 reports the migration time breakdowns of the Container PreCopy method. Accounting for the HSS application's metadata described earlier, the pre-dump, dump, and restore phases of the HSS service are marginally longer when compared to the MME and SPGW applications. The VPN update time is the time taken in setting up the OpenVPN connectivity at the destination node. The GTP tunnel update functionality applies only to the SPGW application data-plane communication.

Meantime, 95 % Confidence Interval (CI), and Coefficient of Variation (CV) of the migration time of the three core network components are presented in Table 8 accounting for the three migration methods. The Container PreCopy method avoids the inter-process delay and provides fastest migration when compared to that of the Docker CLI-based StopandCopy method. All methods applied to the three components show a modest coefficient of variation.

TABLE 6. Container StopandCopy method: Migration time breakdown.

	CloudLab - Flavor Small		
Mig Metrics[s]	**HSS**	**MME**	**SPGW**
Checkpoint Time	5.98	2.19	3.581
Metadata	2.00	1.16	1.61
Restore Time	7.03	2.39	3.65

TABLE 7. Container PreCopy method: Migration time breakdown.

	CloudLab - Flavor Small		
Mig Metrics[s]	**HSS**	**MME**	**SPGW**
Pre-Dump Time	0.32	0.173	0.237
Pre-Dump Tx Time	2.35	1.053	1.723
Dump Time	0.215	0.224	0.183
Dump Tx Time	0.549	0.504	0.557
VPN Setting	0.528	0.491	0.581
GTP Tunnel	NA	NA	0.449
Restore Time	0.879	0.641	0.735

TABLE 8. Migration time comparison for different migration methods.

Migration time [s]	CloudLab - Flavor Small								
	VM PreCopy			Container StopandCopy			Container PreCopy		
	Mean	CI 95%	Coef Var	Mean	CI 95%	Coef Var	Mean	CI 95%	Coef Var
HSS	16.9	0.40	0.02	15.01	0.43	0.03	4.84	0.26	0.06
MME	16.83	0.28	0.02	5.74	0.18	0.03	3.09	0.13	0.04
SPGW	17.11	0.63	0.04	8.85	0.42	0.05	4.47	0.13	0.03

B. DOWNTIME ANALYSIS

Downtime is one of the prominent performance indicators for many mobile applications. Fig. 8 reports the downtime of the virtualized core network components for each of the three migration methods. Significant reductions of the VNF downtime is achievable when using the Container PreCopy method. Compared to the VM PreCopy method these downtime reductions are 70%, 77.86%, and 72.11% for HSS, MME, and SPGW instances, respectively. Combined, the VM dirty page transfer and setting up of the new port configuration requires more time than what is required for the final dump and restore phase of the container. First, the compact nature of the container requires a relatively smaller number of dirty pages. Second, the dump phase metadata of the Container PreCopy method is in the order of few tens of Megabytes in contrast to the Gigabytes needed in the VM page copy, therefore leading to a significant reduction of the application downtime value as reported.

Compared to the Container StopandCopy method, the application downtime caused by the Container PreCopy method is reduced by 84.6%, 67.59%, and 71.6% for the HSS, MME, and SPGW applications, respectively. During the pre-dump phase in the Container PreCopy method, the instance is fully functional compared to the StopandCopy method, which instead freezes the application once the dumping process is initiated.

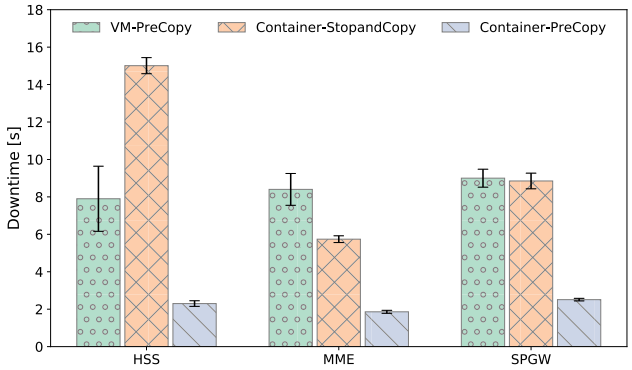


FIGURE 8. Application downtime caused by the three migration methods.

TABLE 9. Downtime comparison for different migration methods.

Downtime [s]	CloudLab - Flavor Small								
	VM PreCopy			Container StopandCopy			Container PreCopy		
	Mean	CI 95%	Coef Var	Mean	CI 95%	Coef Var	Mean	CI 95%	Coef Var
HSS	7.9	1.74	0.23	15.01	0.43	0.03	2.3	0.15	0.07
MME	8.4	0.85	0.11	5.74	0.18	0.03	1.86	0.08	0.04
SPGW	9.00	0.48	0.06	8.85	0.42	0.05	2.51	0.07	0.03

For the Container PreCopy approach, most of the metadata collection happens in the pre-dump phase totaling hundreds of Megabytes compared to only tens of Megabytes of dirty page information involved in the dump phase. Since the application service is not interrupted during both the pre-dump phase and the pre-dump transfer phase — with the values specified in the breakdown Table 7 — the application downtime is significantly shorter in the Container PreCopy method, provided that the container migration problems are addressed as mentioned in Section IV-A.

Meantime, 95 % CI, and CV of the VNF downtime are reported in Table 9 for the three network components and three migration methods. The longest downtime when migrating the HSS component is experienced by the Container StopandCopy method, mainly caused by its demanding total memory page copy procedure. Overall, the results in Table 9 clearly show that the Container PreCopy method yields the shortest downtime values thanks to its sequence of pre-dump and dump phases. The VM method presents the highest CI variation because of its dirty page size that varies from experiment to experiment.

C. UE SERVICE RECOVERY TIME

The UE service recovery time must account for both control plane and data plane connectivity. The control plane traffic considered in our study comprises the UE attach and detach requests, which are sent to the MME through both the DU and CU units. The data plane traffic consists of the UE data packet transactions, which are sent to the PDN through the SPGW GTP interface.

TABLE 10. UE service recovery time for different migration methods.

UE Serive recovery [s]	CloudLab - Flavor Small		
	VM PreCopy	Container StopandCopy	Container PreCopy
DP_GTP	9.0	2.00	1.64
DP_SGi	10.43	8.85	2.51
CP_S1-MME	8.4	5.74	1.86

Table 10 reports the UE service recovery time information for the three migration methods subject to the following definitions:

- DP_GTP - Data Plane traffic through the GTP interface connecting UE and SPGW (SPGW migration);
- DP_SGi - Data Plane traffic through the SGi interface connecting PGW and PDN (SPGW migration);
- CP_S1-MME - Control Plane traffic through the S1-MME interface connecting UE and MME (MME migration).

The UE service recovery time is less than the migration time of the core network components for the PreCopy method of migration because the communication between the UE and the network beyond the core network is still possible within part of the migration process (i.e., outside the stop and copy of the dirty pages). Thanks to the newly developed middleware modules (i.e., GTP Module initializer and the GTP tunnel handler) migration of the container running the SPGW component is now possible yielding a DP_GTP service recovery time that is 77.77% (when using StopandCopy) and 81.77% (when using PreCopy) shorter when compared to the VM PreCopy method. Similarly, migration of the container running the MME component is now possible yielding a CP_S1-MME service recovery time that is 31.67% (when using StopandCopy) and 77.86% (when using PreCopy) shorter when compared to the VM PreCopy method.

D. MIGRATION DATA SIZE

The migration data sizes for the considered migration methods reported in Table 11 help us ascertain the reason for the migration time and downtime differences across these methods. The migration data size is in the order of Gigabytes (GB) for the VMs and in the order of Megabytes (MB) for the containers. With the OAI implementation all three components (HSS, MME, and SPGW) require the same disk image size to accommodate other background processes. Conversely, the migration data size for the containers varies from component to component and is also affected by the method applied (StopandCopy or PreCopy). The HSS component presents the most significant difference between the two container-based migration methods. With the Docker CLI-based StopandCopy method the HSS database information is entirely copied, whereas only part of the modified information is copied when applying the PreCopy method.

The migration data transmission time for each of the core network components is detailed in Table 12 along with the pre-dump and dump information for the Container PreCopy method. In our experiments, the SPGW VM with its data

TABLE 11. Migration data size.

Migration Data Size	CloudLab - Flavor Small		
	VM PreCopy [GB]	Container StopandCopy [MB]	Container PreCopy [MB]
HSS	3.5	173	130.3
MME	3.5	42	41.79
SPGW	3.52	100	100.39

TABLE 12. Migration data size and transmission (Tx) time for all methods.

Mig Data Tx Time [s]	CloudLab - Flavor Small, NIC - 10GB							
	VM PreCopy		Container StopandCopy		Container PreCopy			
					Pre-Dump		Dump	
	Size [GB]	Tx_Time [s]	Size [MB]	Tx_Time [s]	Size [MB]	Tx_Time [s]	Size [MB]	Tx_Time [s]
HSS	3.5	10	173	2.00	129	2.352	1.3	0.549
MME	3.5	10	42	1.16	41	1.05	0.79	0.50
SPGW	3.52	10.14	100	1.61	100	1.72	0.39	0.56

plane communication creates more dirty pages compared to HSS and MME VMs, causing a modest increase of the migration data transmission time. When left running for an extended period of time the VM application increases its internal data storage (in terms of MB) due to the application logs. This data storage increase may cause a slight variation of the VM migration data size. As noted earlier, the pre-dump process in the Container PreCopy method dumps a significant portion of the entire metadata size. The "rsync" command is then applied to synchronize the metadata between the source and destination nodes during the following dump phase.

E. IMPACT OF IMAGE FLAVOR ON MIGRATION OF EPC COMPONENTS

Fig. 9(a) shows the migration time as a function of flavor size. The flavor size influences both the memory and the number of CPU cores allocated to each VM/Container. For the VM, increasing the number of CPU cores does not influence the migration time since the migration is performed on the compute node hosting the VM. However, increasing the storage size increases the overall VM image size thereby increasing the migration time when the flavor is upgraded. For the container, in the StopandCopy method, an increased number of CPU cores reduces the checkpoint and restoration time required. With more cores the inter-process communication time among Docker daemon, Containerd, and runC is reduced, thus accelerating the container migration performed using the Docker CLI. On the contrary, for the Container PreCopy method, increasing the CPU cores tends to increase the checkpoint and restore time due to inter-core interference, which occurs due to shared resources among the cores [52].

Fig. 9(b) shows the influence of RAM/storage size on the migration time, keeping the number of CPU cores fixed (1 vCPU core). The VM migration performance trend remains the same for both (a) and (b) scenarios since the number of CPU cores has not much influence on the VM migration. However, for the Container PreCopy method, keeping

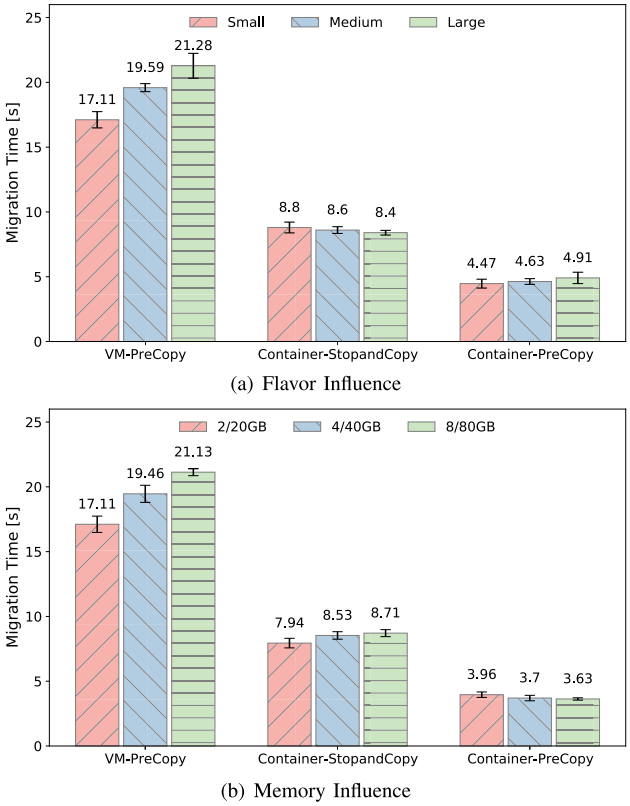


FIGURE 9. Impact of flavor and memory influence on the migration time.

the CPU core to one avoids CPU data propagation delay, and increasing the RAM size helps reduce the number of memory page copies during the pre-dump phase. Overall, the migration time decreases as the memory size increases for the Container PreCopy method. Note that this observation applies explicitly to the RAN and core network whereby the CPU inter-core interference impacts the performance.

F. IMPACT OF PROCESSOR HARDWARE ARCHITECTURE MODEL

Fig. 10 shows the influence of the processor hardware architecture on the migration time. The NIC speed is set to 10 Gbps

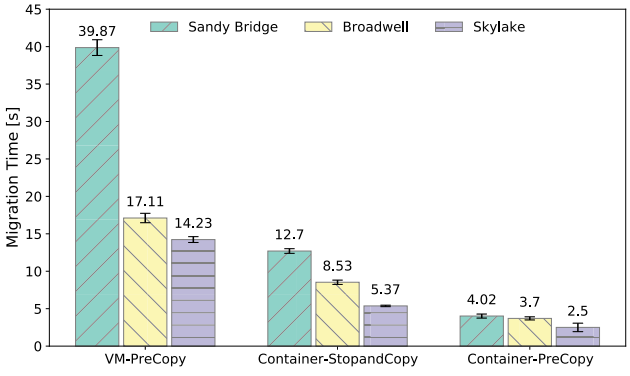


FIGURE 10. Processor hardware architecture model influence on migration.

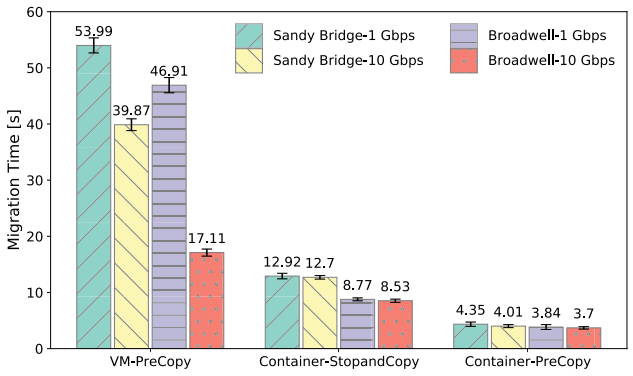
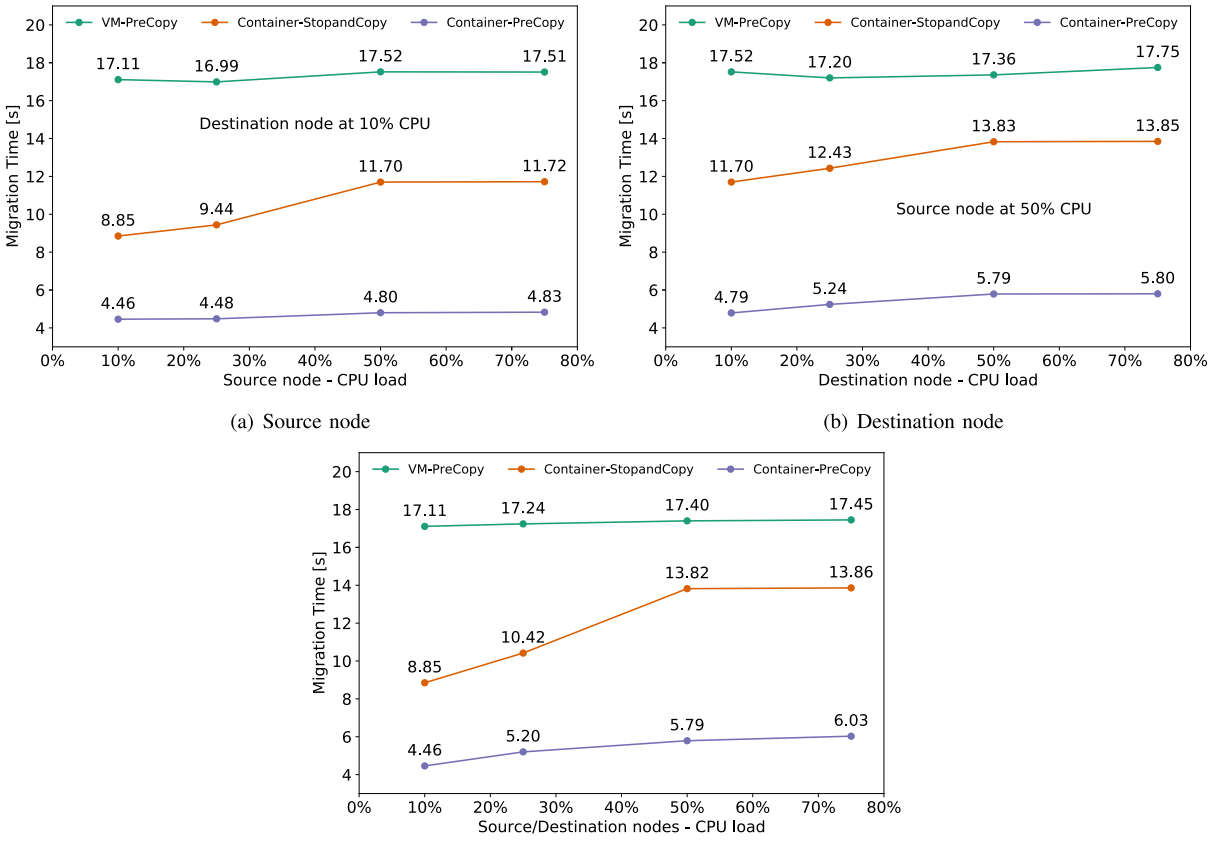


FIGURE 11. Network speed influence on migration.

for all the considered hardware models, which are Sandy Bridge E5-2450, Broadwell D-1548, and Skylake 6126. The CPU frequency is 2.1 GHz, 2.0 GHz, and 2.6 GHz for the SandyBridge, Broadwell, and Skylake model, respectively. The SandyBridge and Broadwell models have 8 cores running in the Cloudlab testbed [40], while the Skylake model has 12 cores running in POWDER testbed [53]. The RAM speed of the Broadwell processor is 800MHz faster compared to that of the Sandy Bridge model. When running the VM-PreCopy method, the migration time is highly dependent on the used hardware architecture with more than a 50% reduction between Sandy Bridge and Broadwell. Despite its slower CPU frequency, the increased RAM speed of the Broadwell processor yields superior performance. Overall, the Skylake model outperforms the other thanks to is superior CPU frequency, number of CPU cores, and RAM speed. It should also be noted that the Container PreCopy method has much less hardware dependency when compared to the VM PreCopy method.

G. IMPACT OF NIC SPEED ON MIGRATION OF EPC COMPONENTS

Fig. 11 shows the impact of the NIC speed on the migration time for two hardware architectures (i.e., Sandy Bridge and Broadwell). The NIC influence is assessed by changing the management network speed for the VM migration and the tenant network speed for the container migration, as detailed in section V-C. In the VM PreCopy method, as expected, increasing the NIC speed helps to reduce the VM migration time to a great extent (*in terms of seconds*) with the live copy phase transferring the bulk of disk images. In the two container methods, the NIC speed variation influences the stateful information transfer rate, and the information



(a) Source node

(b) Destination node

(c) Source/Destination nodes

FIGURE 12. CPU load influence on migration.

required to be transferred is in MB. Thus, for both container approaches, the NIC speed influence on the migration time is only *in the order of milliseconds*.

H. IMPACT OF CPU LOAD ON MIGRATION OF EPC COMPONENTS

Fig. 12 shows the impact of the CPU load at the source and destination node on the migration time of the SPGW component. Fig. 12(a) shows the migration time while varying the source node CPU load from 10% to 75% and keeping the destination node at 10% CPU load. This experiment is explicitly considered to provide insights into the migration behavior when performing CPU load balancing across hosts. In the VM PreCopy method, changing the CPU load has no tangible impact. Conversely, the container checkpoint time increases with the increase of CPU load in the StopandCopy method up until around 50%. After that, the migration time remains practically unchanged. For the Container PreCopy method, only a slight time rise in the pre-dump and dump phases is observed while increasing the source node CPU load. Figs. 12(b) and (c) depict the migration time in a typical fault handling or disaster recovery scenario. Fig. 12(b) shows the migration time when the source node CPU load is set to 50%, and destination node CPU load is varied from 10% to 75%. In these experiments the two container methods show little dependence on the varying destination CPU load. The restore phase of the container migration shows a modest increase in its completion time when the CPU load of the destination node is increased up to 50%. Fig. 12(c) shows the migration time when both source and destination node CPU loads are varied. In these experiments the checkpoint/dump and restore times of the two container methods are affected by the CPU load with an increase as high as 50%. The VM Precopy method continues to be unaffected by the CPU loads of both source and destination nodes being jointly varied. All three considered CPU load scenarios reveal that the container migration methods are affected up until 50% CPU load, but remain practically unaffected at higher CPU loads.

VII. CONCLUSION AND FUTURE WORK

This paper reports the first set of public experiments about the live migration of three open-source (OAI) EPC components — HSS, MME, and SPGW — virtualized through both VM and Container technologies. In all experiments the migration of any of these components is successfully completed without causing permanent loss of the UE connectivity. To successfully carry out the container live migration of the virtual EPC components, the authors designed and implemented several custom functions that overcome the current limitations of the open software packages used to implement the EPC components. The newly added software packages and upgrades are tested on the federated CloudLab testbed for full validation on a third-party and distributed platform.

More specifically, migration time and service downtime performance indicators for the two virtualization technologies (VM and Container) are discussed, while accounting for a number of system factors like flavor type of the computing instances (compute, memory, and storage capacity), processor hardware architecture model, and data rate of the network interface applied. The Container PreCopy method of migration is found to consistently outperform the other two methods (VM PreCopy and Container StopandCopy). It is however negatively affected by the dependencies in the kernel code and mobile network transport protocol support. Outside the scope of this paper and possible subject of future work, live migration of virtualized CU/DU (vCU/vDU) in the RAN and virtualized 5G Core Network (CN) in the 5G standalone architecture using both VM and Container technologies are additional critical functionalities of modern mobile networks. With the 3GPP recommended functionality split options, the RAN and 5G CN modules must cope with the transport network dependency and meet the desired mobile network service latency and throughput. The required software changes must be handled carefully to account for the network requirements (e.g., fronthaul latency) and the transport layer security functionalities. Another aspect that remains to be investigated is the possible reduction of service downtime using hybrid migration techniques and the use of artificial intelligence to optimize migration performance.

Finally, several additional open challenges in the RAN and other core network components remain to be addressed before achieving a completely flexible and fully virtualized mobile network solution that is capable of handling the live migration of all of its components. This paper's contribution takes open-source C-RAN one step closer to the ultimate goal of achieving power management, load balancing, and fault tolerance when implementing RAN NFV in the cloud and edge environments.

APPENDIX

The three sections of this Appendix A, B, C describe the SPGW container migration related limitations, GTP Tunnel List Handler design approaches, and miscellaneous issues related to IP address dependability and container root file system.

APPENDIX A
SPGW CONTAINER MIGRATION LIMITATIONS

The issues associated with the SPGW container migration failure are categorized as follows:

(i) No Active GTP Interface: The data plane connectivity of the SPGW module uses the GTP device interface (*gtp0*) for the S1-U communication between the gNB and the SPGW and also for masquerading the packet to the SGi interface to reach the PDN. Since the container checkpoint software does not include the device interface-specific information in its metadata, the GTP interface information is lost in the restored node. Without the GTP interface, the SPGW application functionality is affected in the considered scenario of stateful container migration. Fig. 13 shows the Wireshark (network traffic analyzer) capture taken at the UE side by

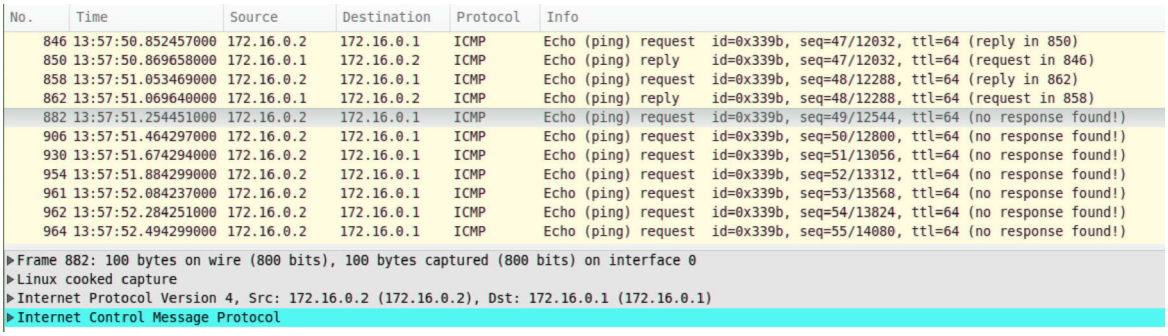


FIGURE 13. UE communication failure with the partially restored SPGW application.

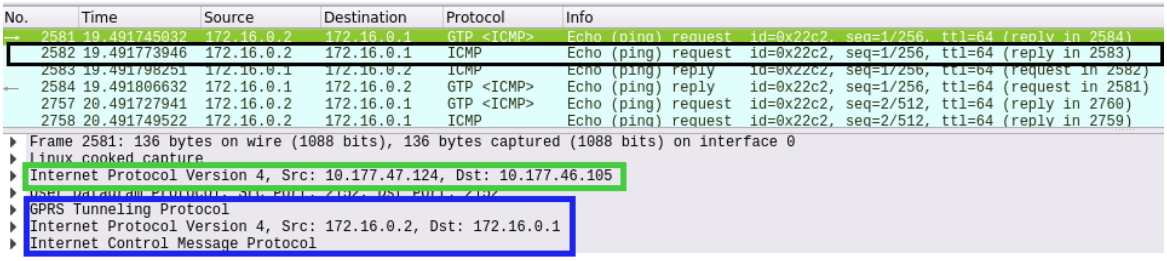


FIGURE 14. Normal GTP communication in SPGW.

using *Ping utility*, where the Internet Control Message Protocol (ICMP) packets are exchanged between the UE and the core network. Starting from packet number: 882, the capture shows the UE communication disruption due to the partial SPGW application restoration at the destination node with the given checkpoint software limitations.

(ii) Lost Tunnel list: The Tunnel list in the SPGW module stores the SGSN address, UE address, and the TEID of the base station as referred in Section IV-A1. This information is required to validate the authenticity of the received GTP message from the base station. However, since this tunnel list is associated with the GTP interface, the metadata created during the application checkpoint did not carry the GTP Tunnel list information. Thus, the SPGW application during the restoration at the destination node does not have any existing Tunnel list data to re-establish the communication with the attached UEs.

(iii) rtNetlink Socket deactivation: rtNetlink Socket belongs to the Linux Netlink Socket family that carries the inter-process communication between the Kernel and the User Space. The rtNetlink Socket is used for carrying the GTP decapsulated message from the GGSN Kernel module. Once the application is checkpointed, the associated Netlink Socket is deactivated immediately, but the GTP interface remains active (in the Source node). This socket connectivity problem is verified by restoring the SPGW container application from the frozen point (using the metadata file) in the same node. The experiment confirmed that the GTP interface is active (since running in the same node), and the UDP encapsulated GTP message from the UE is received at the SPGW node but the rtNetlink Socket was not active to forward the GGSN decapsulated IP Payload. This observation shows that the checkpointed SPGW application cannot be restored successfully on a different node and even on the same node with an active GTP interface and the tunnel list data.

Fig. 14 shows the Wireshark capture of the regular GTP communication. Here, packet number 2581 indicates the ICMP encapsulated GTP message received from the base station (with the Green box) showing the SGSN address and (with the blue box) the GTP information. The packet is decapsulated by the GGSN module and the Netlink Socket forwarding the ICMP packet request is shown in packet number 2582. Fig. 15 shows the Wireshark capture after the partial SPGW migration *in the same node*. The capture shows that the ICMP encapsulated GTP message is received from the base station. However, there is no further action taken to forward the decapsulated packet due to the rtNetlink Socket deactivation of the checkpointed application.

APPENDIX B
GTP TUNNEL LIST HANDLER DESIGN

Two design approaches — proactive and reactive-based tunnel list storage are considered in the study.

- Global approach: The nomenclatures, namely master node and GTP nodes, are considered. The SPGW application-hosted servers are considered GTP nodes with one master node in the cluster of nodes. The Tunnel

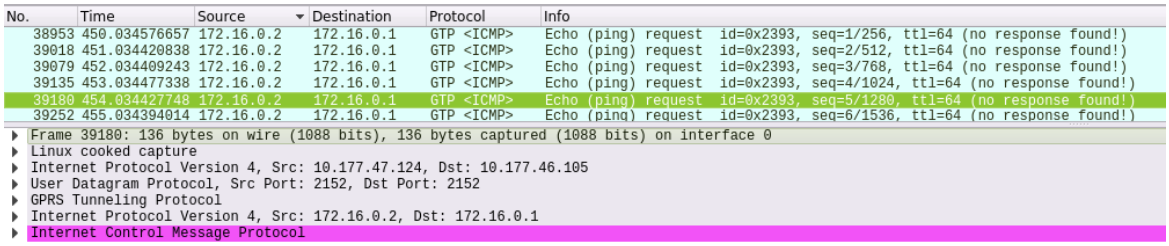


FIGURE 15. GTP communication failure in partially restored SPGW.

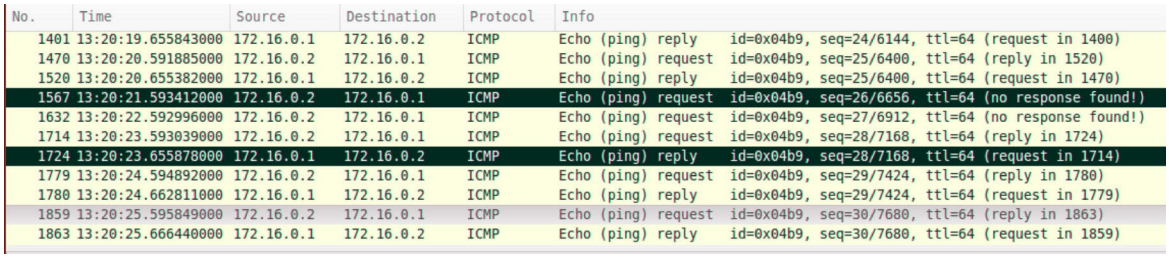


FIGURE 16. UE communication re-established with the restored SPGW application.

list handler service in the master node carries out the periodic health check of the assigned GTP nodes by reading the GTP tunnel information and stores it in its memory. Also, when the tunnel list information is modified, the GTP node will update the master. This method is a pro-active approach to store the tunnel list information to handle any type of sudden failure.

- Local approach: In this method, the handler service is started in each node where the SPGW container service is running. At the time of checkpoint, the tunnel handler opens a general socket, lookup for the GTP family, and then reads the active tunnel information associated with the GTP interface and store it in a file. This method is a reactive approach where the tunnel list information is read when the migration is initiated.

Fig. 16 shows the Wireshark capture of the UE communication during the migration process with the proposed middleware components. Packet number 1724 of the Wireshark capture confirms the successful restoration of the UE communication during the stateful live migration of the SPGW application. Some preliminary groundwork about the SPGW migration constraints alone was presented in [54]. This article provides a comprehensive discussion with the experimental analysis on the virtualization orchestrator platform.

APPENDIX C
MISCELLANEOUS ISSUES

Additional generalized challenges exist in the container environment, and appropriate deployment strategies are required for resolving them.

A. IP ADDRESS DEPENDABILITY

Docker container creation comes with the default bridge network - a software-based link layer for forwarding the traffic between network subnets. This software bridge allows the containers connected to the same bridge to communicate with each other providing better isolation from the outside network. When there is a need for the container to communicate with the external node, there is an option of assigning the node network to the container. Some applications, such as the core network elements, require permanent socket connectivity with other clients. This creates an IP Address dependability situation to be taken care of during the container migration, which, by default, does not carry the underlying node network information in its metadata.

This IP address dependability is addressed with the overlay network (namely OpenVPN) - a logical network distributed on top of the physical network. The OpenVPN helps to tie up the multiple networks in two different sites with proper encryption. Once the migration process is initiated, for the container application to maintain the same IP address, the node taking over the container application (the destination node) configures the OpenVPN address (of the Source node). This approach helps to solve the IP address dependability at the time of migration. For instance, considering the PreCopy approach, due to this overlay network addition, the overall container migration time stated in the equation (2) becomes:

$$T_{pcm} = T_{pd} + T_{pdt} + T_d + T_{dt} + T_n + T_r \tag{5}$$

where T_n is the network setup time at the destination node. T_n time constraint is needed only for applications such as core network which has the IP address dependability when being migrated. And, there are multiple ways to optimize this T_n value with several networking strategies such as Virtual Extensible LAN (VXLAN), Floating IP, and the investigation is outside the scope of this study.

B. ROOT FILE SYSTEM

The root file system is the top-level directory on top of which the other file systems are mounted during the system boot-up. The root file system contains the critical files required for system operation, including the program for system booting. In the Docker container, the root file system is to be considered as the 'golden image' [55] since it is prone to security attacks. During the restoration procedure, the container's root file system in the source node should match with the destination node's root file system. An application such as SPGW software modifies specific existing log files residing in the root file system during its execution during new tunnel creation, UE attachment. This modification eventually causes the file size mismatch in the container image of the source node and the destination node, which triggers restoration failure.

This size mismatch problem can be addressed by making the root file system read-only that solves the security attack. However, setting this read-only configuration would crash the application's requirement (such as the SPGW case above-mentioned) of updating a specific file at run time. The temporary file system (tmpfs) configuration is needed during the container's runtime to overcome this issue. The tmpfs option makes the non-persistent data write to the root file system, thereby satisfying the application requirement of modifying the file at run time. This configuration option enables successful restoration during the migration process since the root file system's persistent file size is not affected.

ACKNOWLEDGMENT

This article provides a comprehensive discussion with the experimental analysis on the virtualization orchestrator platform.

REFERENCES

[1] G. Carella, A. Edmonds, F. Dudouet, M. Corici, B. Sousa, and Z. Yousaf, "Mobile cloud networking: From cloud, through NFV and beyond," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2015, pp. 7–8.

[2] *NFV Report Series Part 1: Foundations of NFV: NFV Infrastructure and VIM*, SDN Central Producer Rep., 2017. Accessed: Jan. 2, 2021. [Online]. Available: https://sdx.io/nfvi-vim-report-2018

[3] V. G. Nguyen, A. Brunstrom, K.-J. Grinnemo, and J. Taheri, "SDN/NFV-based mobile packet core network architectures: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1567–1602, 3rd Quart., 2017.

[4] OpenStack. *OpenStack Cloud Infrastructure*. Accessed: Mar. 2, 2021. [Online]. Available: http://openstack.org

[5] VMWare. *VMWare Orchestrator*. Accessed: Mar. 2, 2021. [Online]. Available: https://www.virtualizationworks.com/datasheets/VMware-vCenter-Orchestrator-DS-EN.pdf

[6] OSM. *Open Source MANO*. Accessed: Mar. 2, 2021. [Online]. Available: https://osm.etsi.org/

[7] P. Rost, I. Berberana, A. Maeder, H. Paul, V. Suryaprakash, M. Valenti, D. Wäbben, A. Dekorsy, and G. Fettweis, "Benefits and challenges of virtualization in 5G radio access networks," *IEEE Commun. Mag.*, vol. 53, no. 12, pp. 75–82, Dec. 2015.

[8] D. Wang and J. Wu, "Carrier-grade distributed cloud computing: Demands, challenges, designs, and future perspectives," IGI Global, USA, Tech. Rep., 2014, ch. 92.

[9] J. Zhang, F. R. Yu, S. Wang, T. Huang, Z. Liu, and Y. Liu, "Load balancing in data center networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2324–2352, 3rd Quart., 2018.

[10] OpenStack. (Sep. 2015). *Hypervisor Support Matrix*. [Online]. Available: https://wiki.openstack.org/wiki/HypervisorSupportMatrix

[11] CRIU Community. (2019). *Checkpoint/Restoration In UserSpace (CRIU)*. [Online]. Available: https://criu.org/

[12] M. Bagaa, T. Taleb, A. Laghrissi, and A. Ksentini, "Efficient virtual evolved packet core deployment across multiple cloud domains," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2018, pp. 1–6.

[13] R. A. Addad, D. L. C. Dutra, M. Bagaa, T. Taleb, and H. Flinck, "Fast service migration in 5G trends and scenarios," *IEEE Netw.*, vol. 34, no. 2, pp. 92–98, Mar. 2020.

[14] A. Aissioui, A. Ksentini, A. M. Gueroui, and T. Taleb, "On enabling 5G automotive systems using follow me edge-cloud concept," *IEEE Trans. Veh. Technol.*, vol. 67, no. 6, pp. 5302–5316, Jun. 2018.

[15] *Docker*. Accessed: Jan. 21, 2021. [Online]. Available: https://docs.docker.com/get-started/overview

[16] CRIU Team. (2019). *CRIU What Cannot be Checkpointed*. Accessed: Jan. 21, 2021. [Online]. Available: https://criu.org/What_cannot_be_checkpointed#Devices

[17] A. Ceselli, M. Premoli, and S. Secci, "Cloudlet network design optimization," in *Proc. IFIP Netw. Conf.*, May 2015, pp. 1–9.

[18] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.

[19] D. Amendola, N. Cordeschi, and E. Baccarelli, "Bandwidth management VMs live migration in wireless fog computing for 5G networks," in *Proc. 5th IEEE Int. Conf. Cloud Netw. (Cloudnet)*, Oct. 2016, pp. 21–26.

[20] *Technical Specification Group Core Network and Terminals; Restoration Procedures*, document TS 23.007 16, 3GPP, Mar. 2020.

[21] F. Carpio and A. Jukan, "Improving reliability of service function chains with combined VNF migrations and replications," *CoRR*, vol. abs/1711.08965, pp. 1–6, Nov. 2017.

[22] A. Giorgetti, A. Sgambelluri, F. Paolucci, F. Cugini, and P. Castoldi, "Demonstration of dynamic restoration in segment routing multi-layer SDN networks," in *Proc. Opt. Fiber Commun. Conf.*, 2016, pp. 1–3.

[23] L. Valcarenghi, F. Cugini, F. Paolucci, and P. Castoldi, "Quality-of-service-aware fault tolerance for grid-enabled applications," *Opt. Switching Netw.*, vol. 5, nos. 2–3, pp. 150–158, Jun. 2008.

[24] K. Kondepu, A. Sgambelluri, N. Sambo, F. Giannone, P. Castoldi, and L. Valcarenghi, "Orchestrating lightpath recovery and flexible functional split to preserve virtualized ran connectivity," *J. Opt. Commun. Netw.*, vol. 10, no. 11, pp. 843–851, Nov. 2018.

[25] J. Feng, J. Zhang, Y. Xiao, and Y. Ji, "Demonstration of containerized vDU/vCU migration in wdm metro optical networks," in *Proc. Opt. Fiber Commun. Conf.*, 2020, Paper Th3A.4, pp. 1–3.

[26] *Runc Blog*. Accessed: Jan. 24, 2021. [Online]. Available: https://www.docker.com/blog/runc/

[27] *Handling GTP Interface of SPGW*. Accessed: Apr. 20, 2021. [Online]. Available: https://bitbucket.org/PriyaRamanathan/spgw_gtp/

[28] Wikipedia. *Inode-DataStructure*. Accessed: Mar. 24, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Inode

[29] *SCTP Support in CRIU*. Accessed: Apr. 20, 2021. [Online]. Available: https://bitbucket.org/PriyaRamanathan/sctp_criu/

[30] *SCTP Changes in Kernel*. Accessed: Apr. 20, 2021. [Online]. Available: https://bitbucket.org/PriyaRamanathan/sctp_kernel/

[31] (2019). *CRIU TCP Connection*. [Online]. Available: https://criu.org/TCP_connection

[32] *Linux Man*. Accessed: Jan. 26, 2021. [Online]. Available: https://linux.die.net/man/1/pstree

[33] Y. Qiu, C.-H. Lung, S. Ajila, and P. Srivastava, "Experimental evaluation of LXC container migration for cloudlets using multipath TCP," *Comput. Netw.*, vol. 164, Dec. 2019, Art. no. 106900.

[34] Phoenixnap. (2019). *Rsync Command Line*. [Online]. Available: https://phoenixnap.com/kb/rsync-command-linux-examples

[35] W. Hu, A. Hicks, L. Zhang, E. M. Dow, V. Soni, H. Jiang, R. Bull, and J. N. Matthews, "A quantitative study of virtual machine live migration," in *Proc. ACM Cloud Autonomic Comput. Conf.* New York, NY, USA: Association for Computing Machinery, 2013, pp. 1–10.

[36] OpenVSwitch. *OpenVSwitch*. Accessed: Jan. 26, 2021. [Online]. Available: https://www.openvswitch.org/

[37] S. Ramanathan, K. Kondepu, B. Mirkhanzadeh, M. Razo, M. Tacca, L. Valcarenghi, and A. Fumagalli, "Performance evaluation of two service recovery strategies in cloud-native radio access networks," in *Int. Conf. Transparent Opt. Netw. (ICTON)*, 2019, pp. 1–5.
[38] A Guide to VM Migration. (2014). *Common Causes of a Failed Hyper-V Live Migration*. [Online]. Available: https://searchservervirtualization.techtarget.com/tip/Common-causes-of-a-failed-Hyper-V-live-migration
[39] TechTarget. (2019). *Storage Area Network*. [Online]. Available: https://searchstorage.techtarget.com/definition/storage-area-network
[40] D. Duplyakin, R. Ricci, A. Maricq, and G. Wong, "The design and operation of CloudLab," in *Proc. Annu. Tech. Conf. (ATC)*, Jul. 2019, pp. 1–14.
[41] (Oct. 2014). *OpenAir Interface: A Flexible Platform for 5G Research*. [Online]. Available: https://www.openairinterface.org/
[42] *Study on New Radio Access Technology; Radio Access Architecture and Interfaces*, document vol. 2.0.0, 3rd Generation Partnership Project, 2017.
[43] *Ettus*. Accessed: Jan. 25, 2021. [Online]. Available: https://www.ettus.com/all-products/ub210-kit/
[44] ETSI GS NFV-MAN 001. (2014). *Network Functions Virtualisation (NFV); Management and Orchestration*. [Online]. Available: https://www.tacc.utexas.edu/systems/stampede
[45] *Open Baton: A Framework for Virtual Network Function Management and Orchestration for Emerging Software-Based 5G Networks*. Accessed: Jan. 18, 2021. [Online]. Available: https://openbaton.github.io/
[46] Cloudify. *Script Plugin*. Accessed: Jul. 18, 2021. [Online]. Available: https://docs.cloudify.co/latest/working_with/official_plugins/configuration/script/
[47] M. Galloway, G. Loewen, and S. Vrbsky, "Performance metrics of virtual machine live migration," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, Jun. 2015, pp. 637–644.
[48] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Proc. 21st Euromicro Int. Conf. Parallel, Distrib., Network-Based Process.*, Feb. 2013, pp. 233–240.
[49] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, F. Longo, and A. Puliafito, "Container migration in the fog: A performance evaluation," *Sensors*, vol. 19, no. 7, p. 1488, Mar. 2019.
[50] Tecmint. (2019). *Linux Network Bandwidth Monitoring Tool*. [Online]. Available: https://www.tecmint.com/iftop-linux-network-bandwidth-monitoring-tool/
[51] Cloudian. *Data Backup*. Accessed: Jul. 26, 2021. [Online]. Available: https://cloudian.com/guides/data-backup/data-backup-in-depth/
[52] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, "Bounding memory interference delay in COTS-based multi-core systems," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2014, pp. 145–154.
[53] J. Breen, A. Buffmire, J. Duerig, K. Dutt, E. Eide, M. Hibler, D. Johnson, and S. K. Kasera, "POWDER: Platform for open wireless data-driven experimental research," in *Proc. 14th Int. Workshop Wireless Netw.ork Testbeds, Experim. Eval. Characterization (WiNTECH)*, Sep. 2020, pp. 17–24.
[54] S. Ramanathan, K. Kondepu, M. Tacca, L. Valcarenghi, M. Razo, and A. Fumagalli, "Container migration of core network component in cloud-native radio access network," in *Proc. 22nd Int. Conf. Transparent Opt. Netw. (ICTON)*, Jul. 2020, pp. 1–6.
[55] Cloud Scale Monitor. *Containers Root File System as Read Only*. Accessed: Jul. 26, 2021. [Online]. Available: https://docs.datadoghq.com/security_monitoring/default_rules/cis-docker-1%.2.0-5.12/

SHUNMUGAPRIYA RAMANATHAN (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with The University of Texas at Dallas (UTD). She worked as a Technical Lead in embedded firmware development with Honeywell. Her research interests include 5G, the Internet of Things (IoT), fault management, and software design in embedded platforms.

KOTESWARARAO KONDEPU (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from the Institute for Advanced Studies Lucca (IMT), Italy, in July 2012. He is currently working as an Assistant Professor with the India Institute of Technology Dharwad, Dharwad, India. His research interests include 5G, optical networks design, energy-efficient schemes in communication networks, and sparse sensor networks.

MIGUEL RAZO (Member, IEEE) received the Ph.D. degree in computer science from The University of Texas at Dallas, in 2009. He is currently a Research Associate with the Open Networking Advanced Research (OpNeAR) Laboratory and a Senior Lecturer with the Computer Science Department, Erik Jonsson School of Engineering and Computer Science. His research interests include network planning, fault protection, telecommunication software design, protocol design and network modeling, emulation, and simulation.

MARCO TACCA (Senior Member, IEEE) received the Laurea degree from the Politecnico di Torino, in 1998, and the Ph.D. degree from The University of Texas at Dallas, in 2002. His research interests include aspects of optical networks, high-speed photonic network planning, fault protection and restoration, and performance evaluation.

LUCA VALCARENGHI (Senior Member, IEEE) has been an Associate Professor with the Scuola Superiore Sant'Anna of Pisa, Italy, since 2014. He published almost 300 articles (source Google Scholar, May 2020) in international journals and conference proceedings. His main research interests include optical networks design, analysis, and optimization, communication networks reliability, energy efficiency in communications networks, optical access networks, zero touch network and service management, experiential networked intelligence, 5G technologies, and beyond. He received a Fulbright Research Scholar Fellowship, in 2009, and a JSPS "Invitation Fellowship Program for Research in Japan (Long Term)," in 2013.

ANDREA FUMAGALLI (Senior Member, IEEE) received the Laurea and Ph.D. degrees in electrical engineering from the Politecnico di Torino, Torino, Italy, in 1987 and 1992, respectively. From 1992 to 1998, he was an Assistant Professor with the Electronics Engineering Department, Politecnico di Torino, Italy. He joined UT-Dallas, as an Associate Professor of electrical engineering, in August 1997, and was elevated to the rank of a Professor, in 2005. He is currently a Professor of electrical and computer engineering with The University of Texas at Dallas. He has published about 250 technical articles in peer-reviewed refereed journals and conferences. His research interests include aspects of wireless, optical, the Internet of Things (IoT), cloud networks, and related protocol design and performance evaluation.