5G-CT: Automated Deployment and Over-the-Air Testing of End-to-End Open Radio Access Networks

Leonardo Bonati, Michele Polese, Salvatore D'Oro, Pietro Brach del Prever, Tommaso Melodia

The authors introduce 5G-CT, an automation framework based on OpenShift and the GitOps workflow, capable of deploying and testing a softwarized end-to-end 5G and O-RAN-compliant system in a matter of seconds without the need for any human intervention.

Abstract

Deploying and testing cellular networks is a complex task due to the multitude of components involved from the core to the radio access network (RAN) and user equipment (UE), all of which requires integration and constant monitoring. Additional challenges are posed by the nature of the wireless channel, whose inherent randomness hinders the repeatability and consistency of the testing process. Consequently, existing solutions for both private and public cellular systems still rely heavily on human intervention for operations such as network reconfiguration, performance monitoring, and end-to-end testing. This reliance significantly slows the pace of innovation in cellular systems. To address these challenges, we introduce 5G-CT, an automation framework based on OpenShift and the GitOps workflow, capable of deploying a softwarized end-to-end 5G and O-RAN-compliant system in a matter of seconds without the need for any human intervention. We have deployed 5G-CT to test the integration and performance of open-source cellular stacks, including OpenAirInterface, and have collected months of automated over-the-air testing results involving software-defined radios. 5G-CT brings cloud-native continuous integration and delivery to the RAN, effectively addressing the complexities associated with managing spectrum, radios, heterogeneous devices, and distributed components. Moreover, it endows cellular networks with much needed automation and continuous testing capabilities, providing a platform to evaluate the robustness and resiliency of Open RAN software.

Introduction

The Open Radio Access Network (RAN) architecture developed by the O-RAN Alliance, as well as the evolution in 3GPP LTE and NR designs, are moving cellular networks toward disaggregated, softwarized, programmable intelligent systems [1]. RAN disaggregation and softwarization can break the current vendor lock-in and open the cellular ecosystem to a larger number of players, facilitating innovation in the cellular market.

However, network disaggregation and multi-vendor deployments are a double-edged sword: they provide much needed pathways to market diversity and supply chain robustness, but they also introduce *interoperability, performance, and security challenges*. Interoperability testing can decrease the risks associated with Open RAN deployments and help to achieve feature and performance parity with traditional, inflexible systems. Cellular deployment and testing are challenging as they are mostly manual and require significant human intervention to install, configure, update, monitor, and evaluate network components.

This issue affects both public and private networks. Public networks become inflexible, costly, and difficult to update or upgrade. Consequently, they often lag behind the rapid pace of innovation in the wireless domain and may harbor unpatched vulnerabilities. In the case of private networks, the complexity of end-to-end cellular systems necessitates specialized skills that are not typically found within the enterprise workforce. This makes the deployment of high-performance, robust cellular systems more costly compared to unlicensed technologies [2]. Therefore, it is clear that integrating automation in wireless networks has to go beyond software and needs to also span into radio components, distributed and heterogeneous devices, and spectrum bands.

DevOps, a portmanteau of the words software *dev*elopment and infrastructure *op*erations, can be used to streamline the integration, deployment, and testing of code on the compute infrastructure [3]. This paradigm, also called *infrastructure as code*, provides automation and tracking to ensure reliable and rapid delivery of new code and functionalities, while maintaining an authoritative source for the system and infrastructure configuration.

In this article, we take a fundamental step toward cloud-native automation for the deployment and testing of open, programmable, multivendor end-to-end cellular networks. Specifically, we design, prototype, and evaluate 5G-CT, a set of automated pipelines, microservices, and infrastructure that can deploy a complete end-to-end 5G and O-RAN-compliant cellular network in a few tens of seconds. 5G-CT leverages Red Hat OpenShift orchestration on a compute cluster which supports multiple core networks, edge, RAN (including the radio frontends), and RAN Intelligent Controllers (RICs) deployed as microservices. Additional microservices support the automation and the management of radios and spectrum. GitOps [4] pipelines — a specific class of DevOps based on git repositories — implement Continuous Integration (CI) and Continuous Delivery (CD) mechanisms to track

Digital Object Identifier: 10.1109/MCOM.001.2300675

The authors are with the Institute for the Wireless Internet of Things, Northeastern University, USA.

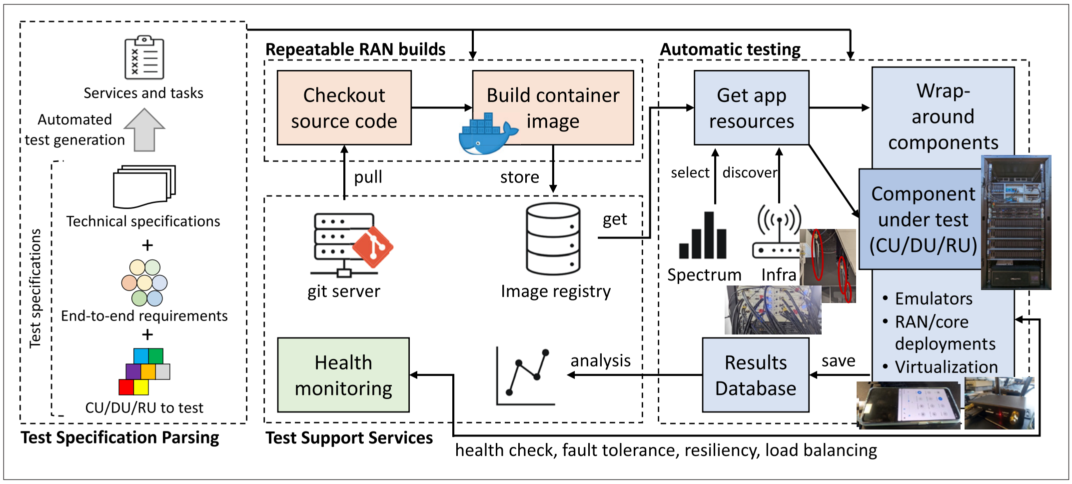

FIGURE 1. Intent-based CI/CD/CT pipeline for Open RAN.

authoritative sources for the codebase and the infrastructure configuration, and keep the deployment environment synchronized and updated.

5G-CT is deployed and operated over-the-air in the Northeastern University FCC Innovation Zone. For months, it has been collecting test results (discussed later in this article) to evaluate the performance of an end-to-end cellular network that includes the OpenAirInterface (OAI) 5G stack [5], commercial 5G User Equipments (UEs), and multiple core networks, that is, the open-source Open5GS and a commercial core from A5G Networks. By leveraging such automation, 5G-CT can effectively guarantee that heterogeneous, multivendor components can interoperate in a disaggregated and virtualized Open RAN system, and that code updates do not introduce regressions, while monitoring the system performance.

Related Work

DevOps techniques are widely used in the cloud computing domain. For instance, [6] showcases a cloud-based CI/CD workflow to build Docker images from code on remote repositories. However, this work is neither concerned with the automated over-the-air testing of RAN components, nor with evaluating how updated builds affect RAN Key Performance Indicators (KPIs).

Adopting DevOps techniques in cellular systems is challenging because of the heterogeneity of infrastructure, code, and functionalities in wireless networks. Moreover, DevOps for cellular need to manage spectrum resources and guarantee predetermined Quality of Service (QoS) levels to the end users. So far, the literature has mostly focused on the challenges to efficiently transition RAN workloads into microservices [7–10]. Even though these solutions provide enhanced and automated network control, they either do not consider the challenges involving the automated instantiation of the RAN, or the automated testing of RAN code and functionalities over the air, as we do in this work. Technologies to enable CI/CD and automated instantiation of RAN components are discussed in [2], which also provides insights on how to fine-tune the compute machines where RAN functions are deployed. Differently from our article, this work does not focus on the actual prototyping of the described CI/CD and is not concerned with testing automation.

The OAI project has developed and maintains a CI framework to run integration and testing for the RAN and core network components of the project, including 4G and 5G versions. While this toolchain contributes to the quality assurance process for OAI, it is not focused on deploying an end-to-end network on a production environment, and the radio testing is performed within a confined environment (i.e., a small Faraday cage) [11]. The authors of [12] integrate OAI for a Kubernetes (k8s)-based CD framework, which however does not embed Continuous Testing (CT) capabilities. Other works leverage DevOps for RAN slicing [13] or core network management [4], but do not consider the end-to-end RAN, core, and edge services (e.g., the RIC) deployment, as we do in this article.

Automated Open RAN Pipelines

In this section, we provide an overview of pipelines for Open RAN automation, and showcase an example of pipeline for the CI, CD, and CT of Open RAN Next Generation Node Bases (gNBs).

CI, CD, and CT pipelines automatically perform tasks on the Open RAN infrastructure. We leverage Tekton — a framework to create CI/CD workflows for on-premise and cloud systems — to implement these tasks and automate our pipelines. Similarly, we rely on ArgoCD — that implements the GitOps declarative philosophy in a k8s microservices cluster — to synchronize and deploy configurations for the host machines and for the tasks from a version-controlled remote repository, ensuring accountability, repeatability, and rollbacks. The tasks we designed carry out operations such as building container images and deploying them on the physical infrastructure, matching workload resources to the nodes best fit to them (e.g., low-latency nodes), discovering available radio and spectrum bands, performing automatic testing, and monitoring the overall health of system and workloads.

A high-level diagram of a CI/CD/CT pipeline for Open RAN is shown in Fig. 1. The tasks of this pipeline are divided into four main groups: test specification parsing, repeatable RAN builds, automatic testing of Open RAN components, and test support services. In the *test specification parsing* step, the details and specifications of the tests to execute are sent to 5G-CT. This is the only step that requires some form of human interaction as it involves gath-

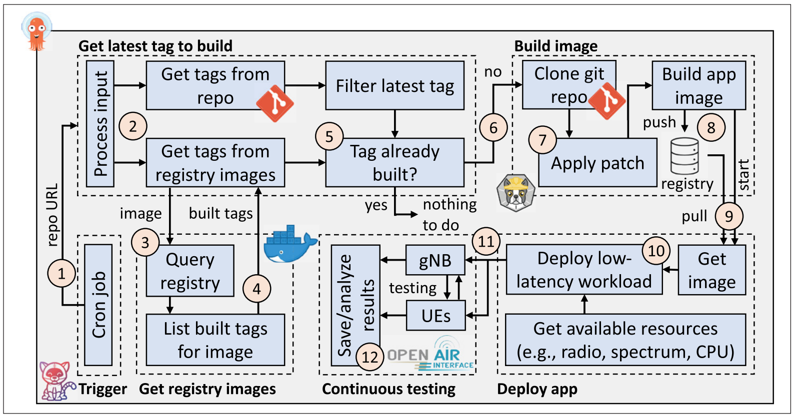

FIGURE 2. Example of pipeline for CI/CD/CT of Open RAN gNB.

ering the test details provided as input, for example, the Central Unit (CU)/Distributed Unit (DU)/Radio Unit (RU) to test, the test requirements, and the relevant technical specifications. The intent specified in this way is then automatically parsed and converted into services and tasks that can be executed by the compute nodes. After this initial setup has been carried out, the remaining steps of 5G-CT execute in a fully automated manner.

The *repeatable RAN builds* workflow packages the source code of the component to test (e.g., the gNB) into a container image to be instantiated and tested. First, the source code of the component under test is pulled from a version-controlled repository on a remote git server. Depending on specific tests, this step can also apply new developed functionalities (e.g., new schedulers or fixes to test) to the source code. Then, this workflow is used to build a container image that will be stored in an image registry and deployed as workload on the compute nodes.

The *automatic testing* workflow validates the functionalities of the built components. After being built and stored in the image registry, the new container image is deployed on one of the compute nodes, depending on resource availability and requirements. As these nodes may need to timely interface with the radio devices (e.g., in the case of a gNB), they have been optimized for low-latency operations. Resources mapped to this deployment include radio and spectrum portions — discovered through dedicated services — as well as computational, memory, and networking resources (e.g., high-speed network interfaces to communicate with the radios). After the resource mapping has been completed, a new container is *deployed* on the compute nodes, and the required resources are allocated to it. Finally, *testing support services* include services to monitor the health of the host machines and of the deployed workloads, recover them from potential failures or issues, analyze test results, interface with the git repository, and host a Docker registry for the built images.

CI, CD, and CT of Open RAN gNBs

An example of a 5G-CT pipeline to automatically build and test the functionalities of an Open RAN gNB is shown in Fig. 2. This pipeline — implemented through Tekton and synchronized with a version-controlled repository through ArgoCD — consists of six main tasks, shown in the figure with a dashed line: periodic triggers, get latest tag to build, get registry image, build image, deploy app, and continuous testing. The *periodic triggers* task runs a *cron job* that starts a *test*, whose *specification* was parsed a priori (Fig. 1), and executes the *get latest tag to build* job to check changes on a git repository (Step 1 in Fig. 2). This task receives as input the URL of the repository to be monitored and checks the latest tag published on it (Step 2). Then, during the *get registry images* task, it queries the image registry (Step 3), part of the *test support services*, for the tags built for the Docker image of the Open RAN software to test (Step 4). The list of tags is compared with the latest tag available on the git rep ository (Step 5). If no new tags have been released, the pipeline ends; otherwise, the task to *build* the new *image* is started (Step 6) to realize a *repeatable RAN build*. The goal of steps 1-6 is to decide whether or not a new container image needs to be built. These steps take 11 s to execute, on average.

The image build task aims at realizing a repeatable RAN build from targeted source code, and is carried out through Buildah, an open-source CI/CD tool to build containers compliant with the Open Container Initiative (OCI) specifications — and that can, thus, be deployed on a variety of platforms including Docker, k8s, and OpenShift. We set up Buildah to execute a multistage build that first clones the git branch/tag to build, optionally applies patches to the source code (Step 7), builds application and required dependencies, and transfers the built executable to the final image, which is saved in the image registry (Step 8). Since the build process tailors the final executable to the CPU architectures of the compute nodes, this process is executed on the low-latency nodes where the workload will be deployed. For instance, building a gNB image takes 20 minutes, on average, in our setup.

The newly built image is then pulled from the registry (Step 9), matched with the available host machine resources (e.g., radio resources, spectrum availability, CPUs, RAM, physical interfaces, etc.), and *deployed as a new application container* on the low-latency nodes (Step 10, potentially stopping an older running instance) according to the *automatic testing* operations of Fig. 1. As an example, stopping a previously deployed gNB container, pulling the updated image from the registry, and deploying it on our system takes 58 s, on average, 34 s of which are required to terminate the previous gNB instance and release the resources used by it.

As part of the automatic testing, the deployed container is used for the *continuous testing* of Open RAN functionalities (Step 11). In the case of gNB testing, this is done by connecting commercial UE devices and exchanging traffic generated via benchmarking tools such as iPerf. We implemented UEs through Sierra Wireless EM9191 5G modems connected to Intel NUC computers, and tested in the CBRS band. Upon completion of the tests, relevant KPIs are stored in a database for later analysis and visualization (Step 12) performed by the *test support services*. Finally, it is worth noticing that, while we showcased the above pipeline as an end-to-end workflow, the single tasks can also be run independently, for example, to only build a novel container image, or to only test its functionalities.

5G-CT OpenShift-Based Automated Infrastructure and Microservices

The pipelines described so far can be implemented as part of a larger container platform, such as k8s, OpenShift, or OKD. These orchestrators offer a flexible virtualization environment to instantiate and manage workloads in the form of containerized applications — or *pods* —based on microservices, and manage their lifecycle, including requirements for networking, storage, and replication, among others.

We built 5G-CT on top of Red Hat OpenShift Container Platform, which abstracts the system complexity (e.g., in terms of heterogeneous nodes and compute capabilities, different CPUs, Network Interface Cards (NICs), RAM) through high-level configuration files. The architecture of 5G-CT is depicted in Fig. 3. At a high-level, it entails an OpenShift-based cluster with three control-plane nodes (Dell PowerEdge R740xd with Intel Xeon CPUs with 32 logical cores and 192 GB RAM) and two worker nodes (Supermicro AS-1023US-TR4 with AMD EPYC CPUs with 32 logical cores and 256 GB RAM), and networking infrastructure. The latter includes a 100 Gb/s Dell EMC Z9100-ON switch that connects 8 USRP X410 Software-defined Radios (SDRs) to the low-latency nodes via their Single Root Input/Output Virtualization (SR-IOV)-enabled NICs and QSFP28 cables. In addition, a 10 Gb/s Dell EMC 4048-ON switch connects to 8 USRP X310 SDRs via SFP+ cables. Finally, a link aggregation group bounds the two switches through two 40 Gb/s interfaces, thus allowing the low-latency nodes to communicate with the USRP X310 SDRs as well. Overall, these SDRs connect to antennas deployed across a 2240 square feet indoor office space representative of a private 5G deployment, and they are able to operate in the sub-7.2 GHz Radio Frequency (RF) spectrum [14].

Control-plane nodes run most of the services required to manage the cluster, as well as generic workloads. These include O-RAN Software Community (OSC) near-RT RIC, and core networks from Open5GS and A5G Networks. Worker nodes, instead, are dedicated to specialized workloads that require low-latency operations, for example, to interface with radio devices in the case of gNBs. Because of this requirement, the configuration of these nodes requires fine-tuning to enable such operations.

A sample configuration for the low-latency worker nodes is shown in Listing 1. These nodes are optimized for minimal latency by disabling energy consumption optimizations in favor of maximum performance (lines 8–10), with additional kernel parameters passed in lines 11–16. Two logical CPU cores, one for each CPU socket, are *reserved* to run the OpenShift services, while the remaining 30 cores are *isolated* and only used to run user workloads (lines 17–19). Additionally, portions of the physical memory of these nodes are reserved through the use of huge pages (64 huge pages of 1 GB each) to increase the performance of the nodes (lines 20–24). Finally, worker nodes leverage dedicated NVIDIA Mellanox ConnectX-6 NICs — passed to the pods through SR-IOV for a trade-off between latency and high-availability, for example, to share the same physical interface

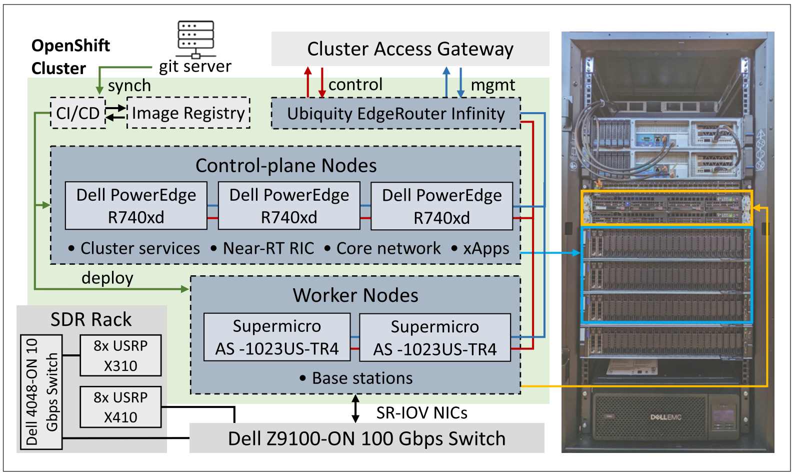

FIGURE 3. 5G-CT architecture.

```yaml
1  apiVersion: performance.openshift.io/v2
2  kind: PerformanceProfile
3  metadata:
4    name: worker-rt-performanceprofile
5  status:
6    runtimeClass: performance-network-latency
7  spec:
8    workloadHints:
9          highPowerConsumption: true
10         realTime: true
11   additionalKernelArgs:
12       - mitigations=off
13       - pci=realloc
14       - numa_balancing=enable
15       - transparent_hugepage=never
16       - skew_tick=1
17   cpu:
18       reserved: 0,1
19       isolated: 2-31
20   hugepages:
21       defaultHugepagesSize: "1G"
22       pages:
23       - size: "1G"
24         count: 64
25   nodeSelector:
26       node-role.kubernetes.io/worker-rt: ""
27   machineConfigPoolSelector:
28       machineconfiguration.openshift.io/role:
         worker-rt
```

LISTING 1. Configuration example for PerformanceProfile object.

among multiple pods — to connect to SDRs via the Z9100-ON switch (Fig. 3). OpenShift also allows clusters to integrate and manage hardware acceleration components to perform *look-aside* or *inline* layer 1 data acceleration. We plan to integrate GPU-acceleration in 5G-CT to offload layer 1 functionalities of the cellular stack onto these units.

The functionalities of the cluster can also be extended through custom microservices to integrate non-standard hardware components. An example of services auxiliary to the Open RAN ecosystem implemented by the cluster includes radio discovery functionalities that leverage Flask Application Programming Interfaces (APIs) and the `usrp_find_devices` UHD routine. Before deploying workloads that need to interface with the SDRs, 5G-CT pipelines make an API call to the Flask endpoint of this service, which returns the list of available USRP radios discovered through the `usrp_find_devices` utility. As the pod where this service runs does not need timely communication with the radio devices, it interfaces with them through the use of MacVLAN instead of SR-IOV.

Other microservices we implemented on the OpenShift cluster include:

- Services to allocate workloads on nodes that best fit their requirements (e.g., compute, latency, networking)

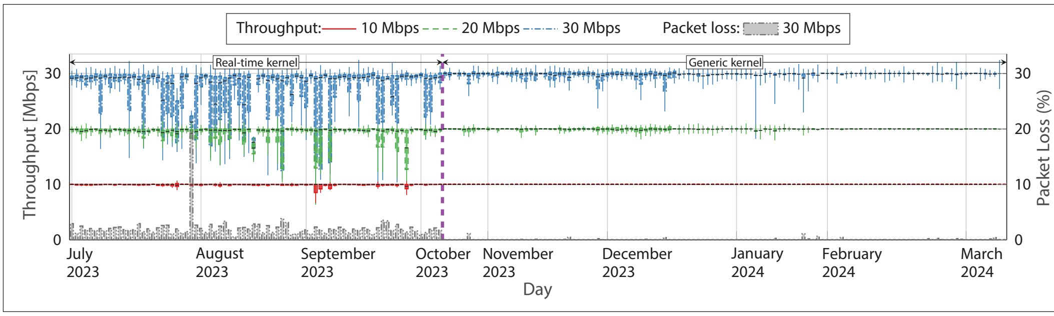

FIGURE 4. Time evolution of throughput (box plots) and packet loss (bar plot) of an OAI gNB during automated tests performed over 9 months for different data rates. The vertical line in October 2023 marks the switch from real-time to generic kernel of the host machines running the gNB.

- An image registry to store the Docker images that will be then instantiated as workloads
- A CI/CD automation for the configuration of the cluster nodes, as well as for instantiating RAN services.

Similarly to the described pipelines, also these routines are configured and performed through a CI/CD automation implemented through ArgoCD and Tekton.

Experimental Results

In this section, we show results of 5G-CT automated tests of an OAI gNB. These tests have been running every 6 hours for approximately 9 months. At a high-level, the automated testing involves:

- Instantiating the OAI gNB as an OpenShift pod on one of the low-latency worker nodes connected to the SDRs of the testbed
- Connecting as UE a commercial SierraWireless EM9191 5G modem managed by an Intel NUC Mini PC
- Exchanging downlink UDP traffic at different target data rates— for example, at 10, 20, and 30 Mb/s in this example — between the gNB and the UE
- Reporting the results to dedicated data collectors implemented as additional pods on OpenShift, where the results are also analyzed and stored.

To maintain the gNB up-to-date with the latest OAI releases, 5G-CT periodically (i.e., once a week in this example) checks for new gNB releases on the OAI code repository. If a new release (i.e., git tag) is found, 5G-CT builds a new container with the up-to-date gNB code, and uses it for all subsequent tests, until a new release is found. Tests are performed leveraging the SDRs and antenna grid of the Arena testbed [14], which allows users to run cellular experiments (among others) in an indoor environment representative of a private 5G setup. The OAI gNB uses a USRP X410 SDR as RF front-end, and connects to a commercial 5G core provided by A5G Networks. Transmissions happen in the CBRS band n48 (3.62 GHz), in Time Division Duplexing (TDD) mode with 162 Physical Resource Block (PRB) (30 MHz).

Tests are triggered through a call to the Flask APIs that we added to the gNB pod. Once this API call is received, the pod starts the OAI gNB, waits for it to be running, and makes a call to the Intel NUC connected to the Sierra Wireless 5G modem that acts as UE. At this point, the Intel NUC turns on the 5G modem, and waits for it to connect to the OAI gNB and to receive an IP address from the 5G core. Once connected, the Intel NUC defaults its network routes toward the 5G core (via the gNB), and starts an iPerf Docker container, which connects to an iPerf server deployed as a standalone pod on OpenShift, to start the performance tests at the selected rates. After the iPerf tests terminate, the results gathered by the UE are sent to a data-collector service — implemented through a replica of 3 OpenShift pods for redundancy — leveraging the same connection with the gNB used for the tests. Results are analyzed by this service, which extracts relevant metrics and stores them in a dedicated Network File System (NFS) volume on OpenShift. They are made accessible through an httpd web dashboard implemented via a replica of 3 OpenShift pods. Overall, this example of 5G-CT automated tests involves a total of 35 OpenShift pods (1 for the gNB, 1 for the iPerf server, 3 for the data-collector service, 3 for the dashboard, and 27 for the core), while an end-to-end cellular network can be instantiated in tens of seconds, as demonstrated in [15].

Figure 4 shows the overall throughput achieved by 5G-CT automated tests for the three configurations considered. We show the day in which the experiment was performed on the x-axis, and box plots of the throughput averaged over the 4 daily experiments on the y-axis for each configuration. Some boxes are missing because of unsuccessful tests (e.g., miscommunication between the gNB and the SDR, issues with the iPerf server). The figure also shows the packet loss of the 30 Mb/s configuration (bar plot).We notice that, in general, the throughput of the 10 and 20 Mb/s configurations (shown in red and green in the figure, respectively) is steady throughout all tests, with the 20 Mb/s configuration seldom having unstable behavior. The throughput of the 30 Mb/s configuration (shown in blue), instead, exhibits performance instability, which is substantially mitigated after updating the kernel configuration of the worker nodes of the cluster (vertical line in October 2023). Indeed, changing the type of Linux kernel and reducing the CPU performance spikes (i.e., passing the kernel option `skew_tick=1`), reduces the performance instability, especially in the 30 Mb/s case. Thus, tests before updating this configuration show some instability — due to a combination of issues in the OAI code (e.g., we found that some releases had issues in the communication among the gNB and the USRP SDR), interference, and kernel settings — that seems to have been resolved in tests with the updated setup. This is consistent with the packet loss, only shown for the 30 Mb/s tests (worst case) in the interest of visualization clarity.

Figure 5 shows the outcome of unsuccessful and successful tests of an OAI gNB, by comparing them with historical test data collected through 5G-CT that provide reference performance levels. The history of the throughput of previous tests (which serves as a baseline) is shown in solid lines, with the shaded areas representing the variance of the tests. The latest test to compare with the result history is shown with dashed lines. After performing the automated tests, 5G-CT compares the performance of the most recent test with the test history for the same configuration. If the performance of each test falls within the shaded areas of the test history for the same configuration, the test is marked as successful (e.g., test of Fig. 5, right). Otherwise, the test is marked as unsuccessful (Fig. 5, left). By performing such comparison, 5G-CT can flag tests as *passed* or *failed*, which will lead to automatically generated reports in future extensions.

Conclusions

We introduced 5G-CT, an automation framework based on Red Hat OpenShift that leverages the GitOps paradigm to automatically deploy and test softwarized end-to-end 5G and O-RAN-compliant systems in a matter of seconds. By extending cloud-native CI/CD pipelines to the RAN, 5G-CT effectively addresses the increasing complexity of operations such as spectrum and radio management, and allocation of heterogeneous resources, devices and distributed components, thus providing the much needed automation for the cellular ecosystem. In this way, 5G-CT has the potential to increase the reliability, robustness, and security of software for Open RAN. We integrated 5G-CT with an over-the-air SDR testbed, and demonstrated how it can be used to test open-source protocol stacks for cellular networks, including OAI, though automated CT experiments spanning several months.

Acknowledgment

This article is based upon work partially supported by the U.S. National Science Foundation under grants CNS-1925601, CNS-2112471, and CNS-1923789, by the U.S. Office of Naval Research under grant N00014-20-1-2132, by OUSD(R&E) through Army Research Laboratory Cooperative Agreement Number W911NF-19-2-0221, and by the National Telecommunications and Information Administration (NTIA)'s Public Wireless Supply Chain Innovation Fund (PWSCIF) under Award No. 25-60-IF054. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] M. Polese *et al.*, "Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges," *IEEE Commun. Surveys & Tutorials*, vol. 25, no. 2, Jan. 2023, pp. 1376–1411.

[2] S. Chenumolu, "Open RAN Deployments," *Open RAN: The Definitive Guide*, I. C. Wong, A. Chopra, S. Rajagopal, and R. Jana, Eds. New York City, NY, USA: John Wiley & Sons, Sept. 2023, ch. 10, pp. 145–71.

[3] P. Rodríguez *et al.*, "Continuous Deployment of Software Intensive Products and Services: A Systematic Mapping Study," *J. Systems and Software*, vol. 123, 2017, pp. 263–91.

[4] D. Scotece *et al.*, "5G-Kube: Complex Telco Core Infrastructure Deployment Made Low- Cost," *IEEE Commun. Mag.*, vol. 61, no. 7, July 2023, pp. 26–30.

[5] F. Kaltenberger *et al.*, "The OpenAirInterface 5G New Radio Implementation: Current Status and Roadmap," *Proc. Int'l ITG Wksp. Smart Antennas*, Vienna, Austria, Apr. 2019.

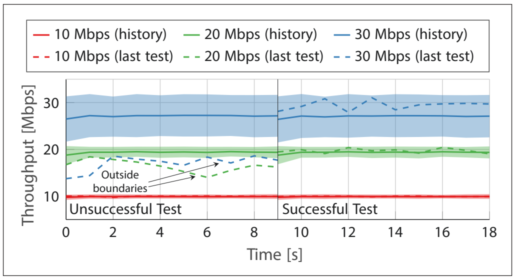

FIGURE 5. Comparison between unsuccessful and successful tests of an OAI gNB for different targeted data rates. The solid lines show the history of previous tests, with shaded areas depicting their confidence intervals. The dashed lines show the most recent test to compare.

[6] C. Singh *et al.*, "Comparison of Different CI/CD Tools Integrated with Cloud Platform," *Proc. IEEE Int'l. Conf. Cloud Computing, Data Science & Engineering*, Noida, India, Jan. 2019.

[7] R. Schmidt, C.-Y. Chang, and N. Nikaein, "FlexVRAN: A Flexible Controller for Virtualized RAN Over Heterogeneous Deployments," *Proc. IEEE ICC*, Shanghai, China, May 2019.

[8] S. K. Moorthy *et al.*, "OSWireless: Enhancing Automation for Optimizing Intent-Driven Software-Defined Wireless Networks," *Proc. IEEE MASS*, Denver, CO, USA, Oct. 2022.

[9] X. Foukas, M. K. Marina, and K. Kontovasilis, "Orion: RAN Slicing for a Flexible and Cost-Effective Multi-Service Mobile Network Architecture," *Proc. ACM MobiCom*, Snowbird, UT, USA, Oct. 2017.

[10] G. Garcia-Aviles *et al.*, "Nuberu: Reliable RAN Virtualization in Shared Platforms," *Proc. ACM MobiCom*, New Orleans, LA, USA, Oct. 2021.

[11] R. Defosseux, "OpenAirInterface Continuous Integration Training," 2019; available: https://tinyurl.com/oai-ci-slides.

[12] O. Arouk and N. Nikaein, "Kube5G: A Cloud-Native 5G Service Platform," *Proc. IEEE GLOBECOM*, Taipei, Taiwan, Dec. 2020.

[13] X. Li *et al.*, "Automated Service Provisioning and Hierarchical SLA Management in 5G Systems," *IEEE Trans. Network and Service Management*, vol. 18, no. 4, Dec. 2021, pp. 4669–84.

[14] L. Bertizzolo *et al.*, "Arena: A 64-Antenna SDR-Based Ceiling Grid Testing Platform for Sub-6 GHz 5G-and-Beyond Radio Spectrum Research," *Computer Networks*, vol. 181, Nov. 2020, pp. 1–17.

[15] L. Bonati *et al.*, "NeutRAN: An Open RAN Neutral Host Architecture for Zero-Touch RAN and Spectrum Sharing," *IEEE Trans. Mobile Computing*, Aug. 2023, pp. 1–13.

Biographies

Leonardo Bonati [M'23] is an Associate Research Scientist at the Institute for the Wireless Internet of Things, Northeastern University, Boston. He received a Ph.D. degree in Computer Engineering from Northeastern University in 2022. His research focuses on softwarized approaches for the Open RAN of next generation of cellular networks.

Michele Polese [M'20] is a Research Assistant Professor at the Institute for the Wireless Internet of Things, Northeastern University, Boston. He received his Ph.D. degree from the University of Padova in 2020. He then joined Northeastern University as a research scientist. His research interests are in the analysis and development of protocols and architectures for future generations of cellular networks.

Salvatore D'Oro [M'17] is a Research Assistant Professor at the Institute for the Wireless Internet of Things, Northeastern University, Boston. He received his Ph.D. degree from the University of Catania in 2015. His research interests include optimization, artificial intelligence, and security applied to Open RAN systems.

Pietro Brach del Prever [S'23] is a Ph.D. student at the Institute for the Wireless Internet of Things, Northeastern University, Boston. He received his M.S. degree in Mechatronic Engineering from the Polytechnic University of Turin, Italy, in 2022. His research interests focus on NextG wireless systems.

Tommaso Melodia [F'18] received a Ph.D. in Electrical and Computer Engineering from the Georgia Institute of Technology in 2007. He is the William Lincoln Smith Professor at Northeastern University, the Director of the Institute for the Wireless Internet of Things, and the Director of Research for the PAWR Project Office. His research focuses on wireless networked systems.