Enabling 5G QoS configuration capabilities for IoT applications on container orchestration platform

1st Yu Liu Ericsson Research Stockholm, Sweden yu.a.liu@ericsson.com 2nd Aitor Hernandez Herranz *Ericsson Research* Stockholm, Sweden aitor.hernandez.herranz@ericsson.com

Abstract—Container orchestration platform is the foundation of modern cloud infrastructure. In recent years, container orchestration platform has been evolving to cross the boundary of device, edge, and cloud. More and more Internet of Things (IoT) applications such as robotics and eXtended Reality (XR) have been deployed across the device-cloud continuum through the container orchestration platform, e.g., the Kubernetes (K8s) framework. Meanwhile, the rapid expansion of advanced communication technologies like 5G has endorsed the revolution in IoT applications as more network resource is available for critical IoT use cases. This paper aims to integrate network configuration capabilities provided by a 5G Network Exposure Function (NEF) into the K8s framework which is used to simplify application deployment in an orchestration in the devicecloud continuum. Specifically, a Linux fwmark-based network Quality of Service (QoS) configuration method is proposed to expose the QoS information from an overlay network that is used by the container orchestration platform to the underlay network. A Container Networking Interface (CNI) plugin-based implementation is demonstrated to perform QoS configuration for the 5G network. The proposed solution is validated with an existing localization and mapping application to verify the feasibility. The proposed solution has the following benefits: (1) The solution is a Kubernetes-native approach which adopts the CNI plugin mechanism. (2) The solution can expose the QoS information from an overlay network to an underlay network in a non-intrusive manner. (3) No packet manipulation is required to greatly reduce the overhead for packet processing. (4) It extends the K8s bandwidth limit feature from on-node to the access network. (5) It is compatible with the 5G infrastructure without any alteration or adding extra complexity.

Index Terms—5G, QoS, container orchestration platform, container network interface, device-cloud continuum

I. INTRODUCTION

A container orchestration platform such as Kubernetes (K8s) or K3s usually manages a cluster of nodes that span different networks. Considering the network heterogeneity and to simplify communication within the cluster, an overlay network is usually established to enable communication among application pods. Typical networking addons such as flannel and calico can create overlay networks using Virtual Extensible LAN (VXLAN), IP in IP protocols or secured channels using IPSec or WireGuard.

The integration of cloud computing and 5G technology has given rise to the transformative concept of Multi-access Edge Computing (MEC). The merge promises higher computing capabilities and lower network latency at the network edge, which largely enables applications such as Augmented Reality (AR)/Virtual Reality (VR) and leads to new deployment paradigm such as cloud robotics [1]. One of the key enablers lies in that an access network (e.g., 5G, and WiFi, etc.) can provide different Quality of Service (QoS) to applications catering to different network traffic priority demands to satisfy application performance requirements while distributing network resource in an efficient manner. For instance, a User Equipment (UE) can establish multiple QoS flows in the same Protocol Data Unit (PDU) session to a 5G network.

The integration of 5G QoS configuration capabilities to the existing K8s ecosystem becomes a fundamental and demanding feature that can benefit Internet of Things (IoT) applications but remains challenging. Due to the use of an overlay network in K8s, application-specific QoS information that are hidden in the overlay network packets can hardly be extracted in the underlay network without extra overhead. Therefore, the goals for this study is to investigate options of integrating network exposure Application Programming Interface (API)s provided by a 5G network (e.g. Network Exposure Function (NEF) or Service Enabler Architecture Layer (SEAL)) into the lightweight Kubernetes which is used to simplify application deployment in an orchestration in the device-cloud continuum, to implement a bridge between container-orchestration and network QoS configuration. The platform-network interaction is validated with an existing IoT application as well as the potential impact on the application.

This paper is organized as follows. In Section II the background of QoS configuration in cellular networks and in the container orchestration platform is introduced and the pitfalls in existing solutions are pointed out. In Section III, we introduce the Container Networking Interface (CNI) pluginbased solution to enable QoS configuration for container orchestration platform as well as theoretical fundamentals. In Section IV, an experimental implementation of the traffic priority CNI plugin is detailed and verified by applying to an Simultaneous Localization and Mapping (SLAM) application. Section V concludes the paper.

II. STATE OF THE ART

A. QoS configuration on cellular networks

The advent of new cellular access networks has enabled a host of use cases that enhance quality of life and sustainability.

© 2024 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

These networks, particularly 5G, offer a reliable solution for interconnecting machines and humans, eliminating the need for wired connections.

In many of the use cases, it is desirable to allow the devices, i.e., UE, to be able to control the traffic and apply different QoS to different applications or flows running locally in the UE. This control allows the network to treat different types of traffic differently and provide specific resources as required. For example, in a 5G network, service data from the UE can be classified into different flows marked by different QoS Flow Identifier (QFI) [2], which allows the UE traffic to have different QoS configurations and receive different treatments when traversing the 5G network. 5G QoS Identifiers (5QIs) [2] are used to define a set of characteristics of the flow in terms of flow bit rate resource type, priority levels, packet delay budget and packet error rate, averaging window, and maximum data burst volume, among others.

There are several methods to configure QoS in a Radio Access Network (RAN) such as 5G. The most common means is to configure the QoS through NEF. NEF allows applications either running on a UE or external data networks to request a QoS for a PDU session that is to be established or request a QoS change for a given PDU session, via the APIs exposed by NEF. Alternatively, a UE can also actively initiate a QoS configuration or modification for a given PDU session through interactions with Access and Mobility Management Function (AMF). The technical details of QoS configuration in a 5G network are out of the scope of this paper and can be found in [2].

B. Network QoS on container orchestration

The K8s orchestration platform utilizes a CNI plugin mechanism to set up the network stack for containers. Existing network QoS configuration in the CNI [3] is provided by the bandwidth plugin [4], which configures a Token Bucket Filter (TBF) queuing discipline (qdisc) on both ingress and egress traffic which results in the traffic being shaped. However, the bandwidth plugin can only shape local traffic that takes place within a node.

Other mechanisms for traffic shaping and configuration based on CNI plugins and network plugins for K8s have been developed in relation to Software-Defined Networking (SDN). Examples include OpenShift SDN plugins such as Cisco ACI SDN, Flannel SDN, NSX-T SDN, Nuage SDN, and Kuryr SDN [5]. Yet, these SDN based approaches are typically enforced within nodes but not applied to the communication network.

Another project serving the traffic differentiation purpose is NBWguard [6], which is proposed by the research community. It provides network QoS in the same way CPU or memory are limited in K8s via the control groups (cgroups), a Linux kernel feature to limit and prioritize resources [7]. The three QoS modes defined in K8s, i.e., *BestEffort, Burstable and Guaranteed*, are extended to network in this work. Internally, NBWguard also uses Traffic Control (TC) and applies Hierarchical Token Bucket (HTB) queuing discipline (qdisc).



Fig. 1. Anatomy of VXLAN network packet. Source: https://projectcalico.docs.tigera.io/about/about-networking



Fig. 2. Architecture of the proposed solution. It depicts a UE application running on a K8s pod is communicating with an edge/cloud service through the 5G network. A traffic priority CNI plugin is implemented in the UE to interact with the 5G network through either AMF or NEF to configure the application pod's QoS. Note that the network interface may represent a physical interface or a 5G network modem stack or modem manager.

However, similar to the bandwidth plugin, NBWguard's scope is limited to local traffic shaping on the node.

Another approach to implement network functions is through the native device plugin feature enabled by the K8s framework. One example is the SR-IOV network device plugin [8]. This approach enables a pod to interact with physical network devices to add customized processing logic to workloads. Nevertheless, this device plugin-based solution is hardware-dependent and can hardly be regarded as a generic solution.

C. Problems with existing solutions

To enable a seamless integration of 5G QoS configuration into the container orchestration platform like K8s, the QoS demanded by the application running on the K8s platform must be delivered to the 5G network. However, many challenges remain here.

1) Overlay networks: An overlay network can simplify addressing and routing between application pods within a cluster. However, due to the nature of an overlay network that IP packets generated by an application pod are encapsulated inside an underlay network packet, which indicates the QoS or Type of Service (ToS) information that are hidden in the overlay IP packets are invisible to underlay network interface or network filters unless a decapsulation is performed on the packets, as illustrated in Figure 1. Also, this would require manual manipulation on the overlay packets including inserting QoS information to the overlay packet header or payload. This manipulation, decapsulation and re-encapsulation procedure can greatly increase the overhead and reduce the network communication performance.

2) Limitation of traffic prioritization in K8s: In K8s, the network capability is enabled by a plethora of CNI plugins which are highly environment-dependent to address the dynamic needs of networking in different scenarios. Therefore, the CNI specification is well defined for a pluggable network solution. Among existing CNI plugins, most are catering to the connectivity challenges while the traffic prioritization and characteristics are still with limited support. E.g., the native bandwidth plugin provides an approach to limit ingress and egress bandwidth, though the effect is restricted at the node. Other plugins such as Calico features network policy configurations which can be utilized to enable or disable an application pod to communicate with various network entities. However, more complicated configuration capabilities that can expose pod-specific QoS to underlay networks and enable QoS configuration in access networks are still missing. To our knowledge, no existing solutions are available to provide a similar QoS configuration capability as discussed in this paper.

III. PROPOSED SOLUTION

A. Architecture

Figure 2 illustrates the architecture of the proposed solution that can expose the 5G network QoS configuration capability to containers running on top of K8s at a UE through the standard CNI plugin approach.

In this architecture, the network QoS requirements can be inserted into the K8s control plane either by an end user or by an external orchestrator through the *kube-apiserver* component using the standard K8s API. Upon receiving the configuration, K8s will extract the network QoS information and schedule the pod creation procedure. This procedure is initiated by the *kubelet* component that is residing in every worker node, and handled by the container runtime. Among multiple tasks, the container runtime would specifically call a series of CNI plugins according to the predefined CNI network



Fig. 3. Linux packet fwmark visibility in a typical Kubernetes networking stack.

configuration, to setup the network environment for the created pod.

Our solution leverages the CNI plugin mechanism by adding a traffic prioritization plugin in the CNI plugin chain to enable K8s to interact with the 5G stack. Once invoked, this plugin would setup an IP flow for all egress traffic from the pod and then interact with the 5G stack to establish a unique 5G QoS flow that can satisfy the demanded network QoS, either via the NEF component which is through the data plane or the AMF component that is directly requested through the control plane. Upon successful establishment of the QoS flow, the plugin can create a filter applied to the physical interface to redirect the IP flow associated with the pod to the 5G QoS flow. In this way can each pod be mapped to a unique 5G QoS flow in a native manner that is supported by the K8s infrastructure.

B. Linux packet fwmark visibility and availability

To accomplish the 5G QoS configuration for a specific pod, a key step is to distinguish the traffic from the pod from all the other traffics to create a unique IP flow. Considering the nature of a typical K8s networking architecture in which VXLAN is utilized to enable pod-to-pod communication, one approach is to label the packets at the VXLAN interface. After the encapsulation of the overlay network packets, the encapsulated packets can be labelled by inserting labeling information in the packet header or the payload, which can be filtered to create an IP flow thereafter. However, this approach would greatly increase the overhead and inevitably add extra complexity to the VXLAN implementation. Therefore, a solution that can expose the pod QoS requirement that is hidden in the overlay network to the underlay network in a transparent and low footprint manner is in demand.

Among potential solutions, one promising approach is to leverage the Linux firewall mark (fwmark). The fwmark is a 32 bits field that allows to tag a packet in the Linux kernel, which doesn't need to manipulate the packet itself. The fwmark field can be set by iptables and be used to classify packets into different IP flows using a TC filter, which is a Linux-native approach and does not introduce extra overhead to packet processing.

1) Fwmark visibility: Figure 3 shows the Linux fwmark visibility in a typical K8s networking environment where flannel is used to create the VXLAN network. When a packet is generated by a pod, it is sent out through the eth0 interface and then immediately received by the veth interface that is a veth pair device. Since all veth devices are connected to a bridge device, i.e., cni0 in the example, the packet is forwarded to all devices connected to the bridge. Until now, the fwmark is not tagged. When the packet is sent from cni0 to the VXLAN interface flannel.0, a routing is needed. Before the routing procedure, a rule created in the iptables at the mangle table and prerouting chain can be applied to mark the fwmark field of the packet. When the packet is routed to flannel.0, it is encapsulated into a UDP packet as any regular VXLAN packets. After the encapsulation, the fwmark is still visible in the kernel, and the visibility is maintained till the packet is transmitted out of the host by the physical interface. This observation verifies the assumption that the fwmark can be utilized to expose QoS information from the overlay to the underlay network.

2) *Fwmark availability:* Another critical aspect of the fwmark approach is the availability of the bits that can be utilized in QoS mapping.

 TABLE I

 LINUX FWMARK REGISTRY FOR BITWISE USAGE [9]

Bits	Mark mask	Software
0-12,16-31	0xFFFF1FFF	Cilium
7	0x0000080	AWS CNI
13	0x00002000	CNI Portmap
14-15	0x0000C000	Kubernetes
16-31	0xFFFF0000	Calico
17-18	0x60000	Weave Net
18-19	0xC0000	Tailscale

Table I demonstrates the bitwise usage of the fwmark by a series of software. It is noticed that in a K8s environment where popular network plugins such as CNI portmap, Calico or Weave Net are installed, most bits from bit 13 to bit 31 are occupied. In an extreme case where the Cilium plugin is used, there are only 3 bits left for other purpose. Therefore, to adopt the fwmark for QoS mapping, potential conflicts with existing software must be considered.

C. CNI plugin based QoS configuration

Figure 4 illustrates how a CNI plugin-based solution can be used to accomplish the QoS configuration for a given pod. Following the top-down order, the QoS information is passed into K8s via *kube-apiserver*, which is then forwarded to *kubecontroller-manager* and delivered to *kubelet* at the worker node where the pod is to be deployed. Through the container runtime interface, the *kubelet* component is able to interact with the container runtime to prepare necessary resource such as sandbox and container namespace for container creation. Among many tasks, the container runtime needs to create the networking environment for the pod by invoking a chain of



Fig. 4. CNI plugin based QoS configuration. A traffic priority CNI plugin is invoked by kubelet in the CNI plugin chain. It interacts with the external access network, e.g., 5G network through a daemon to configure the QoS demanded by an application pod.

CNI plugins, including the traffic priority CNI plugin that implements the QoS based traffic prioritization.

According to CNI plugin specification [10], the CNI plugins are invoked one after another by the container runtime. During each call, the parameters are passed into the CNI plugin via the STDIN and environment variables. The plugin configuration information that is stored at the CNI NetConfig file is periodically fetched by the container runtime and fed into each CNI plugin through STDIN while container-specific information such as container ID, namespace, and interface name are passed through environment variables.

The CNI plugin and CNI daemon configure QoS flows through interaction with the access network. Upon receipt of the request from the CNI plugin, the CNI daemon program will map the pod-demanded QoS requirement to the actual QoS flow characteristics such as the 5G QoS identifier (5QI) in particular. After that, the daemon program can either request the QoS flow to be established through the NEF or through the 5G modem interface, depending on implementation. Once successful, the daemon program can proceed further to generate a fwmark/QoS pair, configure the iptables to mark the packets sent from the pod, and create a TC filter to redirect the marked packets to the established QoS flow.

D. Benefits and limitations of the solution

To summarize, the proposed solution has the following advantages:

• The solution provides application pod the capability to configure traffic priority in an access network and maps the priority configuration to QoS of the access network with a CNI plugin, which is a native approach to be

integrated into existing container orchestration platform such as K8s.

- There is no need to change the packet header or payload to embed QoS information.
- It allows to transparently expose traffic prioritization tags/annotation from packets in the overlay network (pod) to the underlay network (e.g., VXLAN and IPIP), or even encrypted networks (e.g., IPSec).
- It enhances the bandwidth limit feature of K8s CNI plugin by extending the bandwidth limit scope from the network stack on the node to both the node and the access network.
- The solution focuses on mapping pod QoS requirements to QoS flows on cellular networks, between UE and Next Generation NodeB (gNB) (assuming the connectivity between gNB and the core network is perfect), which is fully compatible with the existing 5G network infrastructure without introducing any alteration or adding extra complexity to the 3GPP standard.

Meanwhile, the proposed solution is also limited to the following aspects:

- It only considers the outgoing traffic, i.e., egress traffic or uplink traffic from the pod running in the UE.
- The traffic differentiation or QoS is only considered in the RAN, between the UE and the 5G core (User Plane Function (UPF)). It does not guarantee packet QoS in the data network, e.g., the Internet, which is usually not under control.

IV. EXPERIMENTS AND VALIDATION

In this section, we demonstrate the implementation of the proposed CNI plugin and validate the applicability of the plugin.

A. Demonstrative implementation

1) Traffic priority CNI plugin: According to the CNI specification [10], four commands must be implemented for each CNI plugin, i.e., ADD, DEL, CHECK, and VERSION, which are passed into the plugin with the CNI_COMMAND environment variable. The ADD command is executed when a container is added to the network or modifications are applied. The DEL command is used to remove a container from the network or un-apply the modifications. The CHECK command verifies the container's networking is as expected while the VERSION command returns a supported CNI version list.

In the implementation, the ADD and DEL commands are emphasized to validate the feasibility. When the ADD command is called, the traffic priority CNI plugin would generate new fwmark and insert iptables rules for the added POD, request the 5G stack to create radio link, PDU session, and QoS flow using the inferred 5QI value, and then add IP filter to redirect marked packets to the QoS flow. When the DEL command is called, the plugin would delete the created iptables rules, the corresponding fwmark for the specific POD, the IP filter, as well as the 5G QoS flow.

2) Interaction with 5G emulator: A 5G network emulator built atop Linux TC is used to emulate the 5G stack. It emulates the 5G network by configuring TC queuing disciplines, classes, and filters. Clients can interact with the 5G network emulator via exposed RESTful HTTP APIs to mimic the interaction with a real 5G network, which enables clients to create radio links, PDU sessions, QoS flows and filters so as to realize traffic prioritization. Figure 5 shows an example of configured TC queuing disciplines, classes, and filters that classify traffic into different QoS flows. Specifically, three QoS flows corresponding to three different traffic priorities are highlighted, which are represented by network delays. Three filters are created to enqueue packets with unique fwmark values into the three OoS flows, respectively. In this way, the overlay network packets that are marked with a QoS fwmark can be redirected into different QoS flows in the 5G network.

In the implementation, the CNI plugin can interact with the emulated 5G network and request the creation and deletion of a QoS flow through the provided HTTP API. QoS configurations requested by the CNI plugin are translated by the emulator into TC commands which are applied to the physical network interface of the host node. In this way it emulates a UE pod to edge/cloud communication through the 5G network while different network QoS configurations can be requested.

B. Validation: network QoS configuration for SLAM

The proposed QoS configuration approach based on the CNI plugin is further validated in a real SLAM application to verify the feasibility.

1) SLAM testbed introduction: The testbed used in the validation is built on top of the K3s framework. The underlying hardware include an Nvidia Jetson NX board (arm64) acting as a device node, and other blade servers (amd64) acting as the cloud. The platform features multi-architecture support and low footprint, which enables applications to be deployed across device, edge, and cloud according to specific needs. Observability of application, platform, and hardware metrics is supported and visualized through Prometheus and Grafana that are deployed to the platform.

A SLAM application, maplab [11], has been chosen for the case study as a distributed application. Execution of maplab consists of localization, mapping, and map optimization three phases, which can be run either in a centralized mode or distributed mode. The maplab application has been containerized with multi-architecture support and deployed on the K3s platform. The machine hall dataset [12] are utilized to run the SLAM application.

2) *QoS configuration for SLAM:* Many researches have been conducted around the SLAM application, e.g., in [13] the authors investigated the network condition's impact to SLAM in distributed mode. As network latency increases, the SLAM localization error also increases proportionally, which has been observed in all experimental datasets.

In the validation, the containerized SLAM application is deployed to the aforementioned platform and configured to



Fig. 5. An example of the TC qdisc, class, and filter configurations after the QoS requirement is enforced by the 5G network emulator.

run in the high-offload distributed mode, i.e., the device node transmits both camera and inertial measurement unit data to the edge node and all computation tasks such as localization, mapping, and optimization take place at the edge.

Experiments are conducted in two categories namely QoSunlimited and OoS-limited to compare the performance in respect of the Absolute Position Error (APE) metric. For the QoS-unlimited case, no QoS is configured for the application pod while for the QoS-limited case, the pod running the device application is configured to use a QoS flow that has a 10 ms latency using the developed traffic priority CNI plugin. According to [13], the localization error at 10 ms latency starts to be significant. The results are shown in Table II, which shows the SLAM application's performance of the QoS-limited category is slightly degraded in both the RMSE and Mean of APE metrics compared to the QoS-unlimited category, which is in line with the expected results as shown in [13]. It also demonstrates the traffic priority CNI plugin is able to perform traffic prioritization of the network QoS for a given application deployed to the K8s platform so as to verify the feasibility.

TABLE II

COMPARISON OF ABSOLUTE POSITION ERROR (APE) BETWEEN QOS-UNLIMITED AND QOS-LIMITED SLAM EXPERIMENTS. THE HIGHER THE WORSE.

Category	RMSE (cm)	Mean (cm)
QoS-unlimited	9.24 ± 0.14	8.13 ± 0.16
QoS-limited	10.08 ± 0.12	8.85 ± 0.11

V. CONCLUDING REMARKS

As an effort to facilitate the synergy of the cloud and the telecommunication domains, this study aims to expose the cellular network QoS configuration capability to applications running on a container orchestration platform such as K8s. A low footprint QoS mapping approach is proposed and implemented by leveraging the Linux fwmark feature and the K8s CNI plugin mechanism. The experimental validation shows the proposed approach can be applied to real SLAM applications and perform prioritization of traffics that are configured with distinguished access network QoS. A quantitative performance evaluation of the solution can be conducted as a future study.

In particular, the proposed solution addresses the challenge to expose overlay network packet's QoS information to the underlay network in a non-intrusive manner, which is significant to the container orchestration environment such as K8s where an overlay network is commonly used. The core essence of the solution can also be regarded as a complement to the existing 5G IP filter set-based approach that is used to configured 5G QoS flows defined in the 3GPP standard.

REFERENCES

- M. Afrin, J. Jin, A. Rahman, A. Rahman, J. Wan, and E. Hossain, "Resource allocation and service provisioning in multi-agent cloud robotics: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 842–870, 2021.
- [2] 3GPP, 3GPP TS 23.501 System architecture for the 5G system (5GS).
- [3] "Container network interface." [Online]. Available: https://www.cni. dev/plugins/current
- [4] "Bandwidth plugin." [Online]. Available: https://www.cni.dev/plugins/ current/meta/bandwidth/
- [5] "Openshift network plugins." [Online]. Available: https://docs.openshift.com/container-platform/3.11/architecture/ networking/network_plugins.html
- [6] C. Xu, K. Rajamani, and W. Felter, "Nbwguard: Realizing network qos for kubernetes," in *Proceedings of the 19th International Middleware Conference Industry*, ser. Middleware '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 32–38. [Online]. Available: https://doi.org/10.1145/3284028.3284033
- [7] "Linux programmer's manual." [Online]. Available: https://www.man7. org/linux/man-pages/man7/cgroups.7.html
- [8] "Sr-iov network device plugin for kubernetes." [Online]. Available: https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin
 [9] "firewall mark registry," https://github.com/fwmark/registry, 2020.
- [10] "Container network interface (cni) specification." [Online]. Available: https://www.cni.dev/docs/spec/
- [11] T. Schneider, M. T. Dymczyk, M. Fehr, K. Egger, S. Lynen, I. Gilitschenski, and R. Siegwart, "maplab: An open framework for research in visual-inertial mapping and localization," *IEEE Robotics and Automation Letters*, 2018.
- [12] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016. [Online]. Available: http://ijr.sagepub.com/content/early/2016/01/ 21/0278364915620033.abstract
- [13] A. Rensfelt, A. C. Hernandez, B. P. Gerö, C. G. Blázquez, P. C. Cubero, Y. Nezami, and M. Dohler, "Network performance and the metaverse: Can 5g deliver what's needed?" https://www.ericsson.com/en/blog/2022/ 11/network-performance-metaverse-5g, 2022.