Seamless Cross-Edge Service Migration for Real-Time Rendering Applications

Yuanzhe Li, Shangguang Wang, *Senior Member, IEEE*, Yuanchun Li, *Member, IEEE*, Ao Zhou, *Member, IEEE*, Mengwei Xu, *Member, IEEE*, Xiao Ma, *Member, IEEE*, and Yunxin Liu, *Senior Member, IEEE*

***Abstract*—Seamless cross-edge migration for real-time rendering applications is challenging. The strong interactive nature of real-time rendering applications demands a downtime lower than 15 ms to achieve an imperceptible migration. Existing methods based on virtual machine migration and container migration suffer from unpleasant downtime brought by dirty page retransmission-induced repeated memory data copy and the shared storage failure-induced extensive disk data copy. In this article, we propose Cloud-assisted Service Migration (CSM) which leverages cloud-edge collaboration to achieve seamless service migration for real-time rendering applications. CSM improves service migration user experience in three folds: First, it introduces a dual rendering mechanism to bypass the peer-to-peer data copy and compresses the freezing stage. Second, a user equipment-centric session switch mechanism is proposed to save time by well coordinating application session switches and 5G user plane session switches. Third, a smooth switching mechanism is leveraged to prevent unpleasant frame flickers during session switching. We implement CSM in edge-rendering multiplayer games and deploy it on a 5G test bed with a full-stack user plane protocol stack. The evaluation results show that CSM can reduce downtime to $<$ 14 ms and the service migration process is user imperceptible.**

***Index Terms*—5G core network, downtime, mobile edge computing, service migration.**

I. Introduction

REAL-TIME rendering applications, including cloud gaming, cloud virtual reality, cloud augmented reality, etc., are thriving in the 5G era. They are expected to become a bridge between the real world and the digital twin world [1]. However, limited by the weak computing capability of wearable display equipment, real-time rendering applications must connect to a powerful yet expensive local host. This makes it neither lightweight nor affordable for most people. Moving the rendering module to the cloud and streaming the rendered images to User Equipment (UE) seems to be a cost-friendly scheme, but the long distance between the UE and the cloud brings extra high latency, resulting in an unpleasant user experience [2], [3]. To tackle this problem, edge rendering is proposed. This design provides low latency rendering service at the network edge and frees UE from the computation-intensive rendering operation. As such, users can enjoy real-time rendering applications on a thin client at an acceptable price.

However, real-time rendering applications rely on real-time streaming protocols such as Real Time Streaming Protocol (RTSP) [4] and Real Time Messaging Protocol (RTMP) [5] to deliver high-resolution video streams to UE [6]. The low-latency requirement limits the coverage area of an edge cloud that provides edge rendering. In addition, to prevent extra latency brought by buffers, some streaming systems designed for real-time rendering applications even adopt a design that buffers no video frames at the UE side [6]. The above designs make edge rendering vulnerable to latency increases brought by user movement. If the user moves out of the connected edge cloud's coverage area, the Quality of Service (QOS) deteriorates as a result of the rising latency. To tackle this problem, *follow me cloud* [7] is proposed, which takes advantage of service migration among edge clouds to dynamically move a service instance to a more adjacent edge cloud that has lower latency. However, service migration is a costly operation. It involves large amounts of data transference and introduces service downtime. Downtime is a period of time during service migration when the service is frozen and unavailable. Users will suffer from unpleasant experiences such as motion sickness, screen freezes and black edges [3] during downtime. As a result, reducing downtime to less than 15 ms is a basic premise [8] to prevent unpleasant experiences and make the service migration imperceptible.

The mainstream methods used to conduct service migration are Virtual Machine (VM) live migration [9] and container live migration [10]. VM live migration is a mature and widely-used technology in cloud data centers. However, VM live migration brings excessive unnecessary data transference because it migrates the whole operating system. To make migration more lightweight, container live migration becomes a hot research topic in mobile edge computing [11], [12]. With the help of Checkpoint/Restore In Userspace (CRIU) [13], container live migration achieves a process-level migration. During container

Manuscript received 1 December 2022; revised 20 October 2023; accepted 25 October 2023. Date of publication 10 November 2023; date of current version 7 May 2024. This work was supported in part by NSFC under Grants 62302262, 62272261, U21B2016, and 62032003, and in part by Xiaomi Foundation. Recommended for acceptance by A.-C. Pang. *(Corresponding author: Yunxin Liu.)*

Yuanzhe Li, Yuanchun Li, and Yunxin Liu are with the Institute for AI Industry Research(AIR), Tsinghua University, Beijing 100190, China (e-mail: liyuanzhe@air.tsinghua.edu.cn; liyuanchun@air.tsinghua.edu.cn; liuyunxin@air.tsinghua.edu.cn).

Shangguang Wang, Ao Zhou, Mengwei Xu, and Xiao Ma are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: sgwang@bupt.edu.cn; aozhou@bupt.edu.cn; mwx@bupt.edu.cn; maxiao18@bupt.edu.cn).

Digital Object Identifier 10.1109/TMC.2023.3331773

live migration, only relevant files and memory states are transferred, making it much more lightweight than VM live migration.

However, VM migration and container migration are not ideal methods to guarantee the service continuity of real-time rendering applications in cross-edge scenarios. First of all, their downtime is not acceptable. Although massive methods such as pre-copy [9], post-copy [14] and layered-structure [12] have been taken to reduce downtime, their downtime ranges from tens of milliseconds to several seconds [10], [11], [15]. The performance of these methods fails to meet the 15 ms standard. Besides, our measurements in Section II-C reveal that the downtime of VM migration and container migration increases sharply if a high-frequency memory writing operation is conducted during the migration. This feature prevents them from migrating computation-intensive real-time rendering applications. In addition, VM migration and container migration both have other inherent shortcomings. VM live migration relies heavily on shared storage provided in intra-cloud scenarios such as Ceph [16] and NFS [17] to prevent heavy file system transference, which is not applicable because of the runtime performance degradation brought by Wide Area Network (WAN) environment. Container live migration fails to work for real-time rendering applications because checkpoint of graphical applications is not supported by CRIU [18].

To address this issue, we propose CSM, a cloud-edge collaboration-based method that can achieve application-level service migration for real-time rendering applications. Our insight is that directly copying all the state data from the source edge cloud is too costly. Instead, the complicated states of real-time rendering applications can be recovered from logic data flow exchanged between edge and cloud. One example is that mobile game clients can resume the game and catch up with other players' progress after disconnection. As a result, we use cloud-assisted dual rendering instead of direct peer-to-peer cross-edge state copy to achieve state synchronization. It leverages state information from the logic flow to create a twin application instance at the destination edge cloud and get synchronized with the original instance. Such a design bypasses the performance bottleneck brought by dirty page retransmission during memory copy and makes it possible to further reduce downtime.

However, realizing such an architecture should overcome the following two key challenges. 1) How to achieve a well-coordinated simultaneous switch of application session and 5G user plane session with the downtime limitation. Both of the above two switch procedures should finish in time to prevent introducing extra downtime. However, the two systems have individual controllers, which makes quick and coordinated session switches much more difficult. 2) How to maintain the continuity of the UE screen during session switching to guarantee an imperceptible migration. CSM simplifies the migration process into a switch from the source edge video stream to the destination edge video stream. Because the source edge and destination edge render images individually, the frames reach UE from both ends could not be ideally identical. As a result, even if the downtime could be compressed short enough, users may still suffer from unpleasant visually abrupt change, i.e., frame flickers. With such flickers, the migration is still perceptible.

To tackle the first challenge, we propose a UE-centric session management mechanism. In the pre-migration stage of CSM, the migration preparations are dominated by the computation controller and 5G core network individually. For computation state migration, the two edge clouds get computation states synchronized with the assistance of cloud logic servers. For the communication network, a new Protocol Data Unit (PDU) session from the base station to the destination edge cloud is set up and work in parallel with the previous PDU session. Thanks to the dual rendering mechanism, the UE receives rendering streams from the two edge clouds simultaneously at the end of the pre-migration stage. On this basis, the physical switches of computation and network sessions are turned into a one-time soft switch in the UE. The migration stage is simplified as a stream session switch at the UE side, which is achieved by exchanging the configurations in the UE streaming selector.

To address the second challenge, we introduce a smooth switching mechanism involving Structural Similarity (SSIM) [19] threshold, frame sliding and asynchronous one-frame buffer. SSIM threshold determines when to trigger stream switching. Only when video frames streamed from both edges are similar enough will it allows a stream switch. Frame sliding enables the UE end to present a weighted average of two video stream frames after the switch is triggered, making the transition between two video streams smoother. The asynchronous one-frame buffer mechanism defines the data exchange mode between the streaming client thread and the streaming selector thread in UE, which guarantees that frames with significant differences are flushed in time.

To summarize, the main contributions are listed as follows.

- We show that massive data transference resulting from the dirty page retransmission-induced repeated memory data copy is the performance bottleneck of VM live migration and container live migration (Section II).
  Then, we propose CSM, a cloud-edge collaboration-based method designed for real-time rendering applications (Section III-A and III-B). CSM replaces peer-to-peer state copy with cloud-assisted dual rendering to compress the freezing stage. As far as we know, this work is the first to consider not only computation state transference, but also 5G user plane PDU session switch in service downtime reducing.
- We develop UE-centric session management and smooth switching mechanism (Section III-C). The former is in charge of coordinating simultaneous switches for both application session and 5G user plane session. The latter makes the video stream transition from the source edge to the destination edge imperceptible and guarantees the user experience.
- The proposed architecture is implemented based on a 5G core network testbed that contains full stack 5G core network user plane (Section IV). We conduct experiments on edge rendering mode (Section V). Experiment results show that our method can further reduce the average downtime under 14 ms, which is low enough for an imperceptible seamless migration.

II. Background and Motivation

In this section, we will first talk about the features of real-time rendering applications. Then the whole process of service migration is analyzed to show where the downtime comes from. Finally, we will demonstrate how VM migration and container migration work and discuss why using these methods cannot further reduce downtime.

A. Understanding Real-Time Rendering Applications

Real-time rendering applications refer to applications that interact with the real world in a real-time manner. The "real-time" here comes in two folds. First, the application should respond to user inputs in a very short time. Second, the responsive content is generated after user inputs instead of pre-recorded in advance. As a result, these applications are usually delay-sensitive and computation-intensive. One typical example is the multiplayer online game, in which multiple players collaborate or compete in one game environment. Note that real-time rendering applications are not limited to multiplayer online games. With the rapid development of virtual reality and augmented reality technologies, real-time rendering applications are supposed to thrive in various online cooperating scenarios.

Generally, real-time rendering applications consist of three parts, the global logical module, the rendering module and the front-end module. The global logical module serves as the central server in the client-server architecture. It runs the main logic and provides a shared virtual world for users. Besides, it maintains global states and provides state information exchange by collecting UE-side states and notifying shared state changes to each UE. The rendering module translates signal-level information into 2D/3D video in real time. The front-end module has two roles. As the output interface, it decodes and displays videos generated by the rendering module. As the input source, it collects user operations and sends the encoded events to the rendering module. In most cases, the rendering module and front-end module work in one-on-one mode. Each of their instances serves only one user.

According to the deployment location of rendering modules, the architectures of real-time rendering applications can be classified into local rendering, cloud rendering and edge rendering (shown in Fig. 1). Local rendering is the most commonly used architecture. It deploys the rendering module together with the displaying module either in one UE or in the same Local Area Network (LAN). This architecture can guarantee a low latency between the front-end module and the rendering module. However, because rendering operations are computation-intensive, the UE is either heavy and battery-exhausting or limited within the same LAN with a local rendering server. To tackle this problem, cloud rendering is proposed in which rendering modules are deployed into cloud data centers together with logical modules. Cloud rendering takes advantage of the abundant computing resource in the cloud data center to support rendering operations, but its QOS is still far from satisfactory. Because the rendered videos are streamed to UE, the long latency and the competitive bandwidth of the WAN pose great challenges.

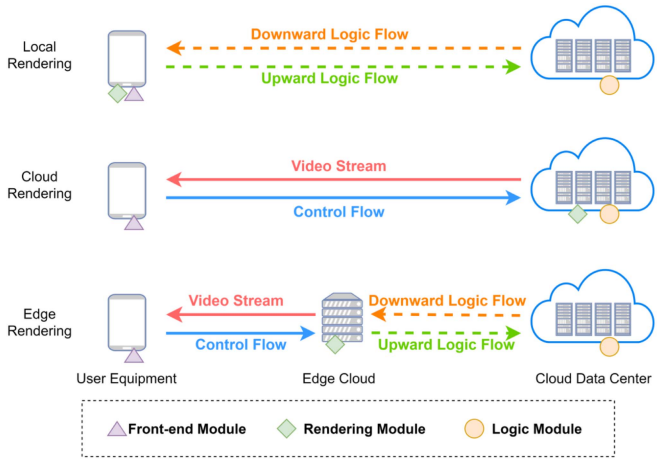


Fig. 1. System architecture for local rendering, cloud rendering and edge rendering.

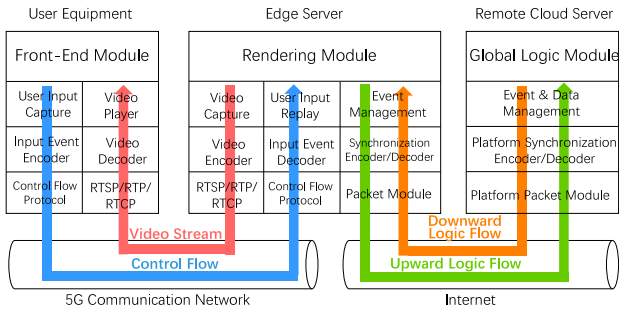


Fig. 2. Protocol stack of edge rendering architecture.

The emergence of mobile edge computing provides another solution to this problem. As shown in Fig. 2, the rendering module is deployed in the edge cloud, i.e., edge rendering. It has two roles in this architecture. In the cloud-edge connection, it serves as an application client which maintains the local states of the user's digital character and synchronizes states with the global logic module through the Internet. In the edge-UE connection, the rendering module serves as a video server. The real-time rendered views are captured, encoded and streamed to the front-end module in UE. At the same time, the front-end module uploads the captured user operations to the rendering module through a control flow. Both the video stream and the control flow are carried by the 5G communication network which provides network access with mobility support.

B. Understanding Service Migration Downtime

If a user moves out of the coverage area of the previous edge server, the latency between UE and the edge server will rise, and a service migration will be triggered. The service migration includes two processes: network hand-off and computation state migration. Network hand-off involves UPF re-selection and PDU session setup. UPF is a network function of the 5G network. It serves as an interconnect point between (3GPP) network and the external data network. As shown in Fig. 3, the 5G network

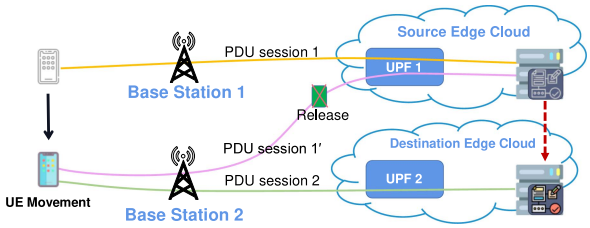


Fig. 3. Network switch during service migration.

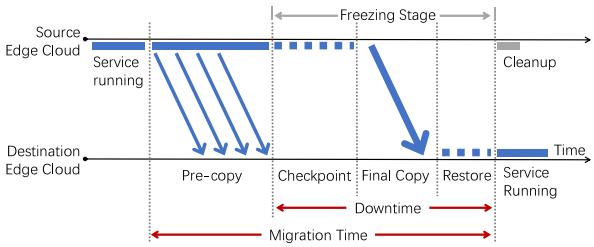


Fig. 4. Computation state migration procedure.

has to set up a PDU session between UE and edge cloud to allow user-edge communication (PDU session 1). As the user moves, the PDU session's latency increases because of the path length increment (PDU session 1′). If a service migration is triggered, a new PDU session (PDU session 2) connecting to the new edge cloud should be established. Such a process may also introduce downtime if it is not finished in time. Note that UPF re-selection is the process of choosing a destination edge cloud and selecting a UPF that can route user packets to the new edge cloud. This is a prediction and decision-making problem [7], [20], [21], which is not discussed in this article.

The computation state migration is to move service instances from the source edge cloud to the destination edge cloud together with the instance running state and runtime context. The procedure of a computation state migration using pre-copy is shown in Fig. 4. It consists of four steps. The first step is pre-copy, which includes disk data pre-copy and memory pre-copy. In this process, the service is still running and responds to the requests. The second step is the checkpoint operation, in which the application is frozen, and the remained memory data will be dumped. Then, the dump files are copied to the new edge cloud. Finally, the application will be restored and continue running based on the files transferred in the previous steps.

There are two key metrics in service migration, i.e., downtime and migration time. Downtime is defined as the total freezing time that the application has no response. Let T_{net} denote the downtime brought by network hand-off and T_{comp} denote the downtime brought by computation state migration. We assume that the two processes are triggered simultaneously. Then the total downtime T_{down} can be calculated as follows:

$$T_{\text{down}} = \max\{T_{\text{net}}, T_{\text{comp}}\} \tag{1}$$

$$T_{\text{comp}} = T_{\text{chkp}} + T_{\text{copy}} + T_{\text{restore}} \tag{2}$$

where $T_{\text{chkp}}, T_{\text{copy}}, T_{\text{restore}}$ represent time used for checkpoint, final copy and restoration, respectively. Migration time is the total time used from the start of pre-copy to the end of restoration. Therefore, it can be calculated as follows:

$$T_{\text{mig}} = T_{\text{pre}} + T_{\text{down}} \tag{3}$$

T_{pre} represents the time used in pre-copy.

C. Case Study

We conduct case studies to evaluate the performance of current VM and container migration technologies under different bandwidths. Each evaluation is repeated five times to get the average performance. We choose Proxmox Virtual Environment (PVE) [22] and Podman [23] as representatives of VM and container systems, respectively. PVE is an open-source server management platform cloud, which provides VM support based on QEMU [24] and KVM [25]. It supports both cold migration and live migration for VMs. Besides, it also leverages Xor Based Zero Run Length Encoding (XBZRLE)[1] to speed up memory copy in the pre-copy stage. Podman is a daemonless container engine for developing, managing, and running containers on Linux. Podman has integrated CRIU and has better support for container checkpoint than Docker [26]. As for other container engines, LXD [27] supports container live migration in very limited scenarios. For example, the network interface card should be removed to enable live migration of an alpine container. Besides, containers working with systemd (e.g. Ubuntu since version 16.10) could not be migrated,[2] either. OpenVZ [28] provides system-level containers. Its container migration includes massive log files, slowing down the freezing stage. Besides, its file transference time cannot be acquired,[3] which is needed for performance evaluation. As a result, Podman is chosen for the case study.

The migrations are conducted between two Lenovo ThinkCenter M910t, which have Intel(R) Core(TM) i7-6700 CPU 3.40 GHz and 16G memory. The two servers are connected with 1 Gbps Ethernet. We leverage Linux TC [29] to change the bandwidth (from 200 Mbps to 1000 Mbps). We conduct measurement on six typical scenarios: CPU-intensive, IO-intensive, memory-intensive scenarios emulated by *stress* [30], file compressing using *tar* [31], video transcoding using *ffmpeg* [32] and real-time rendering using *Glmark2* [33] (a widely used benchmark for OpenGL). Because CRIU doesn't provide checkpoint support for Graphical User Interface (GUI) applications [18], *Glmark2* is only tested on VM migration. The VM used in the experiment is configured with 6GB memory and 20GB shared storage.

1) Virtual Machine Live Migration: From the measurement results demonstrated in Table I, Fig. 5(a) and Fig. 5(b), we have the following discoveries on VM migration. 1) High-frequency memory writing will significantly deteriorate migration performance. Compared with the *Stress-CPU* and *Stress-IO*, *Stress-Mem* witnesses at least $\times 10$ downtime increase, $\times 23$ migration time increase and $\times 30$ data transference increase. Note that, its

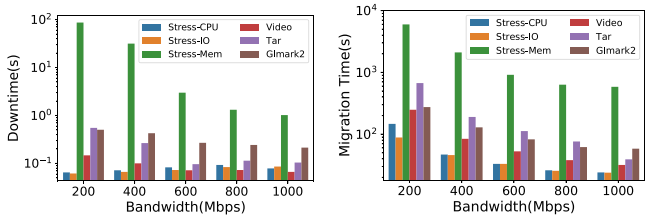
[1][Online]. Available: https://github.com/qemu/qemu/blob/master/docs/xbzrle.txt

[2][Online]. Available: https://github.com/lxc/lxd/issues/10672

[3][Online]. Available: https://github.com/checkpoint-restore/criu/issues/1966

TABLE I
DATA TRANSFERENCE AMOUNT OF VM MIGRATION (GB)

Bandwidth (Mbps)	Stress-CPU	Stress-IO	Stress-Mem	Tar	Video	Glmark2
200	3.14	1.84	130.26	11.80	5.30	5.00
400	1.82	1.80	92.22	7.16	3.44	4.80
600	1.78	1.84	60.48	6.56	3.08	4.72
800	1.76	1.72	55.54	5.78	2.82	4.70
1000	1.74	1.70	55.66	2.94	2.48	4.68

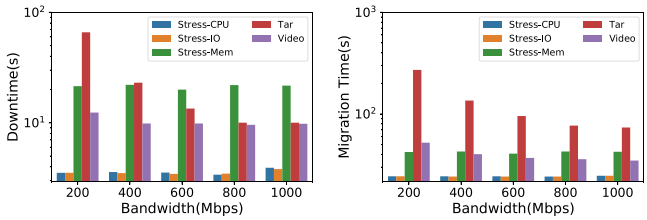


(a) Downtime. (b) Migration time.

Fig. 5. Performance of VM migration (y-axis in log scale).

TABLE II
DATA TRANSFERENCE AMOUNT OF CONTAINER MIGRATION (MB)

Bandwidth (Mbps)	Stress-CPU	Stress-IO	Stress-Mem	Tar	Video
200	0.16	0.30	17.14	5614.75	518.55
400	0.16	0.30	18.28	5070.42	504.40
600	0.16	0.30	16.72	4847.74	509.35
800	0.16	0.30	17.67	4739.45	507.04
1000	0.16	0.30	18.06	4707.36	506.42



(a) Downtime. (b) Migration time.

Fig. 6. Performance of container migration (y-axis in log scale).

data transference amount even reaches at least 9 times the size of VM memory. 2) When bandwidth decreases from 1000 Mbps to 200 Mbps, all testing applications suffer from longer migration time (at least $\times 3.7$). Meanwhile, *Stress-CPU*, *Stress-Mem* and *Video* transfer at least $\times 1.8$ data. *Tar* has the most significant data transference increment ($\times 4.0$). 3) Memory-writing-intensive applications' downtime is more sensitive to bandwidth changes. *Stress-Mem* has the most significant increment ($\times 85$), while *Tar*, *Video* and *Glmark2* rise $\times 5.3$, $\times 2.2$ and $\times 2.4$, respectively. In contrast, *Stress-CPU* and *Stress-IO* keep the downtime between 60 ms and 95 ms. This is because low bandwidth increases the nominal time for data transference. However, the memory writing operation is still working at high frequency. As a result, more dirty pages are generated during the increased time, which means more data for re-transference. Finally, it is worth noting that *Glmark2*'s downtime is at least 213 ms and the migration time is at least 55 s, indicating that the performance of migrating real-time rendering applications with VM is far from satisfactory.

2) Container Live Migration: Container live migration relies on the checkpoint and restoration functionality provided by CRIU. The migration process follows such a pipeline: 1) Pre-transfer mounted data. 2) Pull container images to the destination node. 3) Checkpoint the running container at the source node. 4) Transfer the dump files generated by CRIU during checkpoint and the intermediate files generated by the application to the destination node using *rsync*.[4] 5) Restore the container at the destination node when all files are synchronized. Note that, step 1) and step 2) can run simultaneously. But steps 3) to 5) can only run sequentially. The measurement results of total data transference amount, downtime and migration time are demonstrated in Table II, Fig. 6(a) and (b), respectively. Besides, downtime is further divided into checkpoint time, intermediate file sending time and restore time in Table III.

TABLE III
TIME USED IN DIFFERENT CONTAINER MIGRATION STAGE (S)

	Bandwidth (Mbps)	200	400	600	800	1000
Checkpoint	Stress-CPU	2.39	2.52	2.45	2.31	2.73
	Stress-IO	2.13	2.19	2.15	2.16	2.47
	Stress-Mem	15.61	16.63	15.19	16.25	16.74
	Tar	2.79	2.81	2.83	2.86	2.40
	Video	5.24	5.18	5.24	5.10	5.45
Intermediate File Sending	Stress-CPU	0.26	0.26	0.26	0.25	0.26
	Stress-IO	0.28	0.26	0.26	0.27	0.26
	Stress-Mem	1.01	0.66	0.50	0.49	0.45
	Tar	62.29	19.29	9.69	6.16	5.33
	Video	4.12	1.89	1.44	1.10	1.03
Restore	Stress-CPU	0.87	0.79	0.83	0.82	0.91
	Stress-IO	1.11	1.05	1.04	1.03	1.07
	Stress-Mem	4.79	4.83	4.29	5.16	4.53
	Tar	1.01	1.02	0.95	0.99	2.29
	Video	3.04	2.81	3.19	3.40	3.34

The tested applications can be divided into two categories according to whether it is bandwidth-sensitive or memory writing frequency-sensitive. *Tar* is the only application that is in the bandwidth-sensitive category. It witnesses $\times 6.4$ downtime increment, $\times 3.7$ migration time increment and $\times 1.19$ data transference increment when bandwidth changes from 1000 Mbps to 200 Mbps. The downtime of *Tar* mainly comes from intermediate file sending. Other applications are sensitive to memory writing and belong to the other category. Their downtime has little change (within 30%) as bandwidth decreases. Instead, their downtime mainly comes from checkpoints and restorations. These two processes take up more than 92% of downtime for

[4][Online]. Available: https://linux.die.net/man/1/rsync

Stress-CPU, *Stress-IO* and *Stress-Mem* while more than 67% for *Video*. The time used for checkpoint and restoration is mainly affected by memory writing frequency. The downtime gaps between *Stress-Mem* and the other three applications are significant, which is at least $\times 2.39$.

III. System Design

A. Overview

The key design goal is to further compress the freezing process and reduce downtime under an imperceptible level. To eliminate the service migration performance bottleneck brought by peer-to-peer memory state copy, we propose a dual rendering-based method for real-time rendering applications. Instead of directly copying memory data from the source node, our method makes full use of the global state in the cloud state server to restore the rendering module state at the destination node. This method jointly considers network switch and computation state transference. It differs from previous methods in three folds. First, it bypasses the direct memory copy process between edge clouds. Second, the system works in an asynchronous mode. The destination node gets synchronized with the source node with the help of a cloud logic server. It does not need the source node to freeze first. Third, the network switch is embedded in the whole workflow and adapted to the architecture of the 5G network.

The dual rendering mechanism brings new challenges both in the cloud-edge state synchronization sessions and edge-UE rendering sessions. On the cloud-edge side, dual rendering requires two application instances at different edge clouds to log into the cloud logic server using the same user account. This breaks the basic account management principle that one account can only have one instance logging into the system at a time. Such a change may bring serious data conflict. In order to guarantee data consistency in the system, we introduce session priority, in which applications from edge clouds are divided into two categories. The first one is the primary session which has complete data read/write permission while the other one is the secondary session which works in read-only mode. The secondary session's connecting edge cloud has no writing permission and can only render images according to the states acquired from the cloud logic server.

On the edge-UE side, one UE may connect to different edge clouds simultaneously during the migration. Both edge clouds will render images and stream video to the UE. In this dual rendering phase, the edge cloud that has the primary session works as chief pilot while the other one works in copilot mode. Only the video from chief pilot, will be displayed on the UE. In addition, the control flow of the UE will only be sent to the chief pilot. The roles of edge clouds exchange in the stream switch phase. Because the destination edge cloud has been synchronized with the source edge cloud, the freezing stage is compressed. It remains only one operation of switching video streams. Such a simplification can greatly reduce downtime.

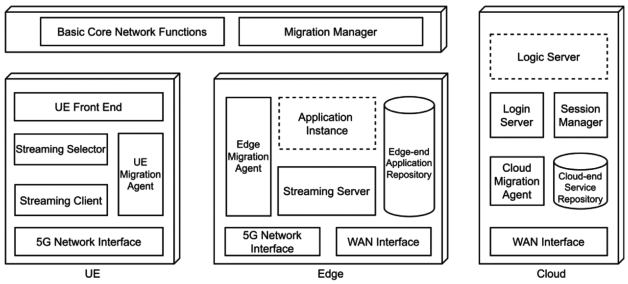


Fig. 7. General architecture of CSM.

B. System Architecture

Fig. 7 shows the general architecture of the proposed method. The solid line modules in the figure are the fixed part of the system and the dotted line modules will change according to applications.

In UE, the streaming client consists of components used for video streaming, such as user input capture, video decoder and input encoder, etc. The UE migration agent is in charge of communicating with the core network and edge clouds for migration control. The stream selector chooses video streams decoded from the streaming client and decided which one can be displayed on the UE front end.

In the edge cloud, an edge migration agent is deployed as well. It works as an application function that connects to the core network. It also serves as a migration controller in the edge cloud. The edge-end application repository stores application images. Applications that have service migration demands have to register to the edge migration agent first to enable the migration functionality. The edge migration agent invokes an application instance during the migration preparation phase. The application instance is the client end of the application deployed in the edge. Its images are captured and encoded in the streaming server. The encoded video streams will be transmitted to UE through the 5G network. On the other hand, input flows from the UE are decoded at the streaming server and sent to the application instance.

Similarly, in the cloud data center, the back-end services are stored in the cloud-end service repository. They should be registered with the cloud migration agent in advance. A session manager is added to the application back end. It works with the login server to support dual rendering during the migration. When dual-rendering is triggered, the destination edge cloud will send requests to the login server first. If the login request is valid, a new state synchronization session of the same user account is created and restored in the session manager. The session manager will maintain a session set for each user account. It will decide which session is read-only and which one has permission to write state data to the logic server.

In addition, a migration manager is deployed in the core network, which is in charge of choosing destination edge clouds and coordinating the 5G PDU sessions during the migration process.

Algorithm 1: Session Management Algorithm.

Input: A state synchronization session s from user account u_i

```
Initialize the session set 𝕊;
if s not in 𝕊 then
    if user account u_i does not login then
        Create an empty set for this user;
        Push session s into the set;
        Mark s as the primary session;
    else
        Push session s into the set;
        Mark s as the secondary session;
else
    if s is primary session then
        Transfer packets to logic server;
    else
        if s has switching signal then
            Mark the current primary session as
              secondary session;
            Mark s as the primary session;
            Transfer packets to logic server;
        else
            Drop out packets from s;
```

C. Workflow

1) Migration Initialization: If the QOS becomes unacceptable as a result of user movement, the UE will send a service migration request to the core network migration manager. Once receiving the request, the system will enter the migration initialization phase. The migration manager will choose a destination edge cloud. After the destination edge cloud is selected, the edge-end application repository will check if the image of the application exists. If the image does not exist, it will be pulled from the repository in the cloud. Then, the application will be started at the edge.

2) Session Setup: Once the destination edge cloud is decided, the migration manager will inform the edge migration agent in the destination edge cloud to start a session setup procedure. The session setup includes two operations. The first operation is setting up 5G communication between UE and the destination edge cloud. The 5G core network will trigger PDU session setup procedure and set up a GPRS Tunneling Protocol User plane (GTP-U) tunnel between the destination edge cloud and the base station that the UE links to. The IP address of the streaming server in the destination edge cloud will be sent to the UE as soon as the new PDU session is set up. Note that, the previous PDU session connecting to the source edge cloud is still maintained and working for the existing rendering stream. The second operation is setting up the state synchronization session between the destination edge cloud and the remote cloud server. The application instance will connect to the login server in the cloud and try to log in with the same user account. Once the login is permitted, a new state synchronization session will be established and transferred to the session manager. The session manager creates a session set for each user account the first time the user succeeds in logging in. The first session will be marked as the primary session. If other sessions of the same user account are created, they will be marked as secondary sessions by default and stored in the relevant session set. The details of session permission management are shown in Algorithm 1.

3) State Synchronization: When a state synchronization session between the edge-end application instance and the cloud-end logic server is set up, the application instance begins to synchronize states with the help of the cloud-end logic server. The cloud logic server will first take a whole snapshot of the current state. Then the snapshot is sent to the destination node as the start point of synchronization. Then, state changes will be sent to edge clouds. Note that, different sessions in the same user account session set share the same state update information in the downward direction. But in the upward direction, only the primary session has the writing permission to update state data in the logic server. Because the state broadcast contains all the changes, the destination edge cloud and the source edge cloud get synchronized indirectly, and the destination edge cloud renders the same scene as that of the source edge cloud.

4) Dual Video Streaming: After the edge server gets synchronized, the streaming server starts to capture the video. Then the encoded stream will be streamed to the UE. The UE rendering client will decode the video and send it to the stream selector. At this stage, the destination edge is in copilot mode. Its video stream will not be displayed until the stream configuration at the streaming selector changes.

5) Stream Switch: After the stream from the destination node is set up, UE will exchange the role of edge clouds by updating the stream selector configuration. This brings three changes. First, the destination edge becomes the chief pilot whose video stream is displayed on the UE. Meanwhile, the source edge cloud becomes the copilot. Second, UE operations are steered to the destination node. Third, the destination node will send a switching signal to the logic server. The session manager in the cloud will mark this session as the primary session. The previous primary session will be downgraded to the secondary session.

From the user's perspective, the main source of downtime is the stream switch process in the UE. Because computation state synchronization and session setup are all finished in previous phases, the freezing stage is largely simplified and the downtime can be effectively reduced. However, even if the downtime has been short enough, users may still suffer from frame flickers (a visually abrupt change) at the moment of switching if the frame difference between the two video streams is too significant. To address this issue, we introduce a triple mechanism, namely smooth switching, to optimize the continuity of the visuals during stream switching. The first is the SSIM threshold mechanism, which is mainly used to ensure the similarity of video frames. After stream switching can be performed, UE checks each frame from the two sessions and calculates the SSIM [19] of the two images. Only when the SSIM is greater than the perset threshold, that is, the two images are sufficiently similar, will it be allowed to enter the stream switching process. The second is the frame sliding mechanism, which is mainly

used to avoid abrupt visual changes during switching. Due to network latency differences, it is difficult for video frames from the two sessions to be completely identical. To avoid screen flashing during stream switching, we introduce a frame sliding mechanism, where the image presented on the UE front-end is a weighted linear combination of the two video frames after switching begins.

$$F_{\text{out}} = \alpha F_d + \beta F_s \tag{4}$$

$$\alpha = \min(\frac{n}{N}, 1) \tag{5}$$

$$\beta = \max(1 - \frac{n}{N}, 0) \tag{6}$$

where F_d and F_s denote the frames from the destination edge cloud and source edge cloud, respectively. F_{out} denotes the frame displayed on the UE front end. $n(n = 0, 1, 2, 3, \ldots)$ denotes the frame number since the frame sliding starts and $N(N = 1, 2, 3, \ldots)$ is the size of sliding window. If $N = 1$, it indicates that the video stream is switched without smooth sliding. The third is the asynchronous one-frame buffer mechanism, which flushes frames with significant differences in time to avoid frame differences accumulating in the buffer due to network jitter. UE creates a streaming client thread for each session, and each streaming client thread maintains a buffer for storing video frames obtained by it. There is only one streaming selector thread on UE, which reads video frames from each streaming client thread's buffer. We impose the following constraints on this data transfer process: 1) the buffer can only hold one frame; 2) the reading operation of the streaming selector does not clear the buffer, and the buffer's contents can only be overwritten by its relevant streaming client thread.

6) Delayed Release: As soon as the switch is finished, a countdown timer will be started. The copilot stream from the source node will not be released immediately. Instead, it will be kept alive until the countdown ends. The countdown timer duration is configured by the migration manager deployed in the core network. This delayed release mechanism is designed to deal with the ping-pong effect. The ping-pong effect refers to the phenomenon that service migration is triggered repeatedly because the latency varies near the migration threshold. The latency fluctuation could be caused by user movement or network jitters. For example, if the user moves back to the source edge cloud, it is necessary to switch back to the source edge stream. The delayed release configuration keeps the source edge working in copilot mode for a while after the stream switch. Because the application instance is kept synchronized, the previous steps from migration initiation to dual video streaming can all be skipped. The second-time migration only contains a stream switch. If another stream switch is triggered, the countdown timer will be reset. A new round of countdown starts as soon as the switch finishes. When the timer goes to zero, the copilot session will be released and the relevant edge cloud will terminate the application instance.

IV. System Implementation

The implication of the system consists of three parts: 5G network support system, streaming system and control system. We first build MiniEdgeCore, a test bed containing a simplified 5G core network system based on GTP5G project [34]. The user plane of the system has a full user plane protocol stack, which can set up PDU sessions between base station nodes and User Plane Functions (UPFs). The control plane of the core network is a simplified version. In the first stage, we've only implemented a few most important network functions concerning PDU session setup. To achieve a flexible network architecture, UEs, base stations and UPFs are deployed in Mininet [35] hosts. Mininet leverages Linux namespace to create a realistic virtual network environment. It runs real kernel, switch and application codes. By introducing Mininet, we can create different net topologies and test realistic network streams. The links between UEs and base stations deployed in Mininet are emulated by Ethernet. Because PDU session re-establishment, which pertains to the third layer of the network, is the primary concern of service migration. The status of the wireless link is not the key indicator in this problem and its management is independent of service migration. This design helps to prevent the complicity brought by wireless communication protocol stack and improve the system's flexibility.

The streaming system is implemented based on Gaminganywhere [6]. Gaminganywhere is an open-source cloud gaming platform. It provides video and audio capturing on the server side and operation capturing on the client side. In our system, the Gaminganywhere server is deployed at the edge node and serve as the streaming server. Its client is deployed in the UEs to provide streaming client functions.

The control system includes the migration manager, the UE migration agent, the edge migration agent and the streaming selector. It implements the mechanisms in Section III with python 3.7.12. The communication among the migration manager, the UE migration agent and the edge migration agent is conducted using HTTP requests. Relevant restful interfaces are provided as well.

V. Evaluation

A. Experiment Setup

We conduct experiments on six Lenovo ThinkCenter M910t servers. Each server has a 3.40 GHz Intel(R) Core(TM) i7-6700 CPU and 16G memory. Two of the servers are installed with PVE v7.0-2, while the other four servers all work on Ubuntu 18.04.6 with kernel version 5.4.0-90-generic. These servers form a three-tier system. The cloud tier has one server which runs the backend logic server. The edge node tier consists of four servers which are divided into two groups, i.e. the VM group and the CSM group. The VM group consists of the two PVE servers. The CSM group has two servers with Ubuntu 18.04.6. These two groups of servers are in charge of rendering images and audio and streaming them to UEs. The last server is deployed with MiniEdgeCore and the control functions. Its UE nodes form the UE tier of the system. UEs and edge servers are connected by the user plane of MiniEdgeCore. Their data stream is forwarded and steered by UPFs. The edge servers and cloud servers are connected by TP-LINK TL-SG1008D Gigabit switch. Linux TC is leveraged to limit bandwidth between edge servers.

Two multiplayer games are chosen to evaluate the performance of the system. This first is Minetest,[5] which is an open-source voxel game engine that provides various game mods with multiplayer support. The second is multiplayer snake implemented in Pygame [36]. Both of the games are tailored to support CSM. The modifications are in four folds. First, the original same account login restriction in the login phase is removed. Instead of rejecting concurrent logins on the same game account, the logic server will add the new login session to the session set maintained for the same game account. Second, during game playing, the real-time information about the primary session's location, perspective, inventory, and other details are sent to the logic server and synchronized with secondary sessions. This ensures that the primary and secondary sessions have consistent information and identical game views. Third, it supports role exchange of sessions during stream switch as described in Algorithm 1. Forth, the game resource recycling process is modified. The objective is to prevent sessions with the same user account are affected due to resource destruction when one of the sessions with the same account logs out.

Although VM-based, container-based and application-level methods are proposed for service migration, VM is the main tool used in real-world scenarios for real-time application migration [8]. All container migration methods fail to migrate real-time rendering applications because CRIU cannot extract state from GPU.[6] Besides, application-level methods are coupled to their target applications. Most existing methods are designed for client-server architecture applications. They do not apply to cloud-edge-UE applications. As a result, we only evaluate the performance of VM migration and CSM in this section.

B. Downtime

As is shown in Fig. 8(a), the downtime of different methods shows huge differences. CSM keeps average downtime less than 12 ms for both snake and Minetest. VM migration's downtime is no less than 64 ms, which is far away from the requirement of seamless migration. The performance gap between CSM and VM in terms of migrating snake and Minetest is at least $\times 5.5$ and $\times 15.5$, respectively. Although pre-copy is used to reduce downtime, VM migration is still limited by its sequential working pipeline in the freezing stage. In contrast, CSM finishes destination node state synchronization and network session setup before entering the freezing stage so that the downtime is greatly compressed. It is also worth noting that, unlike VM migration, the downtime difference between snake and Minetest is quite slight (within 3 ms) when using CSM. This is because CSM leverages state data in the cloud logic server instead of direct peer-to-peer memory copy, and it will not be influenced greatly by the memory writing frequency.

C. Migration Time

The main source of total migration time is different. For VM migration, most of the time is spent in pre-copy. The pre-copy

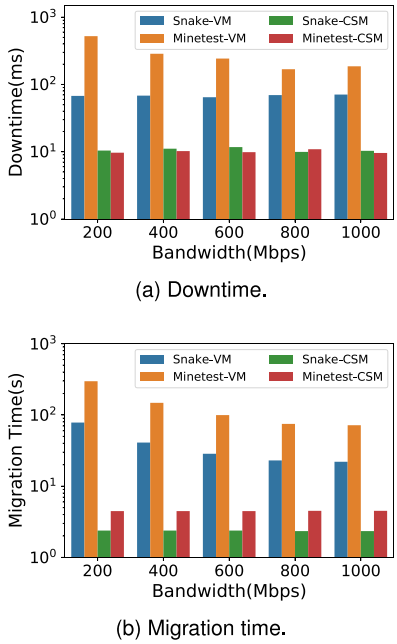


Fig. 8. Service migration performance (y-axis in log scale).

continues until the remained data is small enough for a quick service pause and migration. As for CSM, the total migration time mainly comes from informing and waiting for the destination node to start the game. Consequently, VM migration is sensitive to bandwidth. As shown in Fig. 8(b), the migration time of VM increases rapidly when bandwidth goes down, ($\times 3.5$ for snake and $\times 4.2$ for Minetest). When the bandwidth equals 200 Mbps, the migration time of migrating snake and Minetest even reach 78 s and 298 s, respectively. In contrast, regardless of bandwidth change, CSM finishes migrating snake and Minetest in less than 2.5 s and 4.6 s, respectively. This brings the following advantages: First, it brings less bandwidth pressure on the WAN between edges. Second, considering service migration is triggered when the access latency starts to deteriorate, the QOS can be improved in time through quick migration. Otherwise, users have to tolerate the bad QOS during the long memory copy.

D. Overhead

Service migration is an expensive operation. Massive CPU, memory and network resources are exploited to migrate service instances. Fig. 9 demonstrates the overhead of Minetest instance migration using VM and CSM. All the experiments are conducted with 1 Gbps network connection. In each experiment, service instances will run 10 s before and after the migration. There are three vertical dashed lines in subfigures of Fig. 9, which denote the start of migration, the end of CSM migration and the end of VM migration, respectively.

1) CPU Utilization: Fig. 9(a) demonstrates the CPU overhead of both methods during migration. The CPU overhead of CSM mainly comes from the dual rendering mechanism. During the migration process, the CPU utilization of the source

[5][Online]. Available: https://www.minetest.net/
[6][Online]. Available: https://github.com/containers/podman/issues/12275

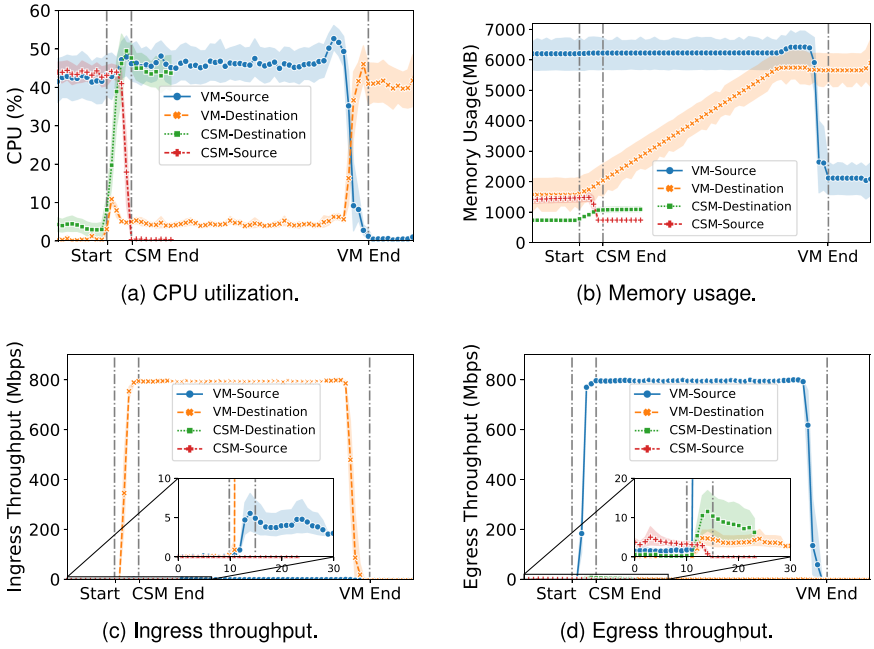


Fig. 9. Service migration overhead.

edge cloud remains almost the same, while the CPU utilization of the destination edge cloud will rapidly increase to around 50% and gradually decrease to 45%. CSM will keep such a double CPU resource utilization until the source edge cloud releases resources. In contrast, during VM migration, the CPU utilization of the destination edge cloud is at a relatively low level (on average5%). This part of CPU resources is mainly used to control the pre-copy of memory state data. When the pre-copy process is completed, the CPU utilization of the source edge cloud decreases to zero rapidly. At the same time, the CPU utilization of destination edge clouds sharply increases to34% and maintains at this level to support running Minetest and keep video streaming. Overall, during service migration, CSM has a higher CPU overhead per unit of time. However, considering that CSM has a much shorter migration time than VM, the CPU overhead of CSM is acceptable.

2) Memory Usage: There is a huge difference in memory usage between CSM and VM. In CSM, the increased memory usage of destination edge clouds only includes the cost of starting Minetest, and the average increase in memory usage throughout the entire process is 1035 MB. In contrast, the memory usage of destination edge clouds witnesses a linear growth for VM migration, from the initial 1126 MB to the final 5084 MB. The memory usage of VM is much larger than that of CSM. Due to the need for a complete point-to-point copy of the VM's memory during the VM migration process, a large amount of memory data, including the operating system state of the VM, is copied. Meanwhile, due to the need to support the operation of the operating system within the VM, the static memory usage of the VM is also higher.

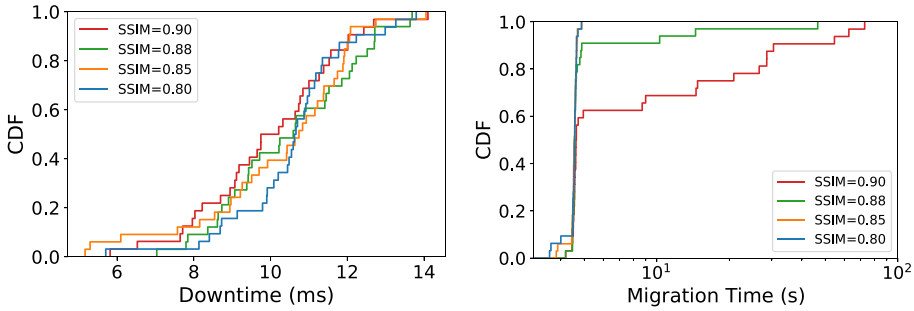
3) Network Throughput: The difference in network throughput overhead is more significant. As shown in Fig. 9(c) and (d), since the VM pre-copy is a point-to-point operation, the traffic

TABLE IV
DATA TRANSFERENCE AMOUNT OF DIFFERENT METHODS (GB)

	snake-VM	minetest-VM	snake-CSM	minetest-CSM
200	1.60	6.38	0	0
400	1.56	6.23	0	0
600	1.54	6.25	0	0
800	1.50	6.17	0	0
1000	1.50	6.22	0	0

of source edge cloud and destination edge cloud is symmetrical, that is, the ingress throughput of the destination edge cloud is almost the same as the egress throughput of the source edge cloud, both reaching over 800 Mbps. In the meanwhile, the ingress throughput of the source edge cloud and the egress throughput of the destination edge cloud is less than 5 Mbps, which is mainly used for control information exchange during the migration process. In comparison, the bandwidth consumption of CSM is much smaller than that of VM. The egress throughput of the source edge cloud and the destination edge cloud reaches 5.45 Mbps and12.80 Mbps, respectively. This traffic is mainly used for video streams. The ingress throughput of both nodes, containing user control signals, can be almost ignored.

As a result, the peer-to-peer data transference between source edge server and the destination edge server differ dramatically, as well. As shown in Table IV, although only memory data are transferred, the data amount of VM migration still reaches a level above 1 GB. In specific, the transferred data of snake ranges from 1.5 GB to 1.6 GB. As for Minetest, it needs at least 6.17 GB data transference to complete the migration. However, the peer-to-peer data transference amount of CSM keeps zero for both of the games. This is because CSM takes full advantage of the state in the cloud server. It recovers states via the normal game



(a) CDF of downtime. (b) CDF of migration time (X-axis in log scale).

Fig. 10. CDF of CSM with different SSIM threshold.

TABLE V
PERCENTAGE OF VOLUNTEERS CLAIM NOTICEABLE FLICKERS

Category	Percentage
CSM	4.35%
Static	3.49%
No-Opt	92.17%
No-Sliding	36.52%

logic interactions between the edge and the cloud. Note that, PVE leverages shared storage to prevent disk data transference. However, shared storage is not applicable between different edge nodes because of the long latency. As a result, the data amount will be much larger if disk data is also transferred.

E. Quality of User Experience

The quality of user experience is a subjective indicator. In order to verify whether CSM can ensure user imperceptible session switches, we recorded videos displayed at the UE during the service migration process. Then we invited volunteers to watch these video clips. The volunteers are unaware of the video sources. After watching each video clip, they are asked to evaluate whether there is a noticeable flicker in the video. We have received a total of 115 valid questionnaires, and the experimental results are shown in Table V. The videos are divided into 4 categories. CSM uses our smooth switching mechanism. Static is a completely static game scene. No-Opt directly switches between two video streams without optimization. No-Sliding only applies SSIM threshold and one-frame buffer mechanisms and drops frame sliding mechanism. The experimental results indicate that if switching directly between two video streams, the session switch will be noticeable (with 92.17% of volunteers reporting frame flickers). If the one-frame buffer and the SSIM threshold were introduced, this number would be reduced to $36,52\%$. With all three mechanisms being applied in CSM, only 4.35% of the volunteers reported frame flickers, which is close to that of Static. This indicates that our design can guarantee a good user experience.

F. SSIM Threshold

In theory, the higher the SSIM threshold, the better the user experience it will be. Because a high SSIM threshold can ensure that the frames from the source edge cloud and destination edge cloud are as consistent as possible during the video stream switching process. However, an excessively high SSIM threshold may hinder the migration process. As shown in Fig. 10(a), all the downtimes are between 5 ms and 14 ms after the streaming switch has been triggered. The SSIM threshold is different, but the downtime distributions are similar. However, there are significant differences in the distribution of migration time (shown in Fig. 10(b)). When SSIM is 0.80 or 0.85, migration time falls between 4 s and 5 s. Although in most cases migration can still be completed within 5 s, long-tail effects begin to appear after SSIM exceeds 0.85. In a few cases, the migration may take several tens of seconds. The migration time even reaches72.82 s in extreme cases. Furthermore, the higher the SSIM value, the more severe the situation. When SSIM is 0.90, even about 40% of migration time exceeds 10 s. This shows that an excessively high SSIM threshold will block streaming switches from occurring. Based on the above experimental results, the SSIM threshold in our CSM method is set to 0.85.

VI. DISCUSSION

Working with VM and container: CSM is not conflict with VM and containers. It only provides an alternative method for migration. The application distribution and deployment function of VM and container can be maintained. In addition to the performance improvement in terms of downtime, CSM can be regarded as a supplement of VM and container. First, it supports shared instance deployment. VM and container can only migrate exclusive instances that only serve one user. If a shared instance is migrated, the other users of the instance are affected. Since CSM is application level, it could migrate one user's states without affecting other users that share the same instance. Second, it makes up for the container's inability to migrate GUI applications and enables containers to be used for deploying real-time rendering applications. This combination makes full use of the flexible and lightweight feature of container in application deployment and high migration performance of CSM.

Applicable application type: CSM assumes that there is a logic server containing global state information. This deployment architecture, usually used in multi-user applications, separates the global logic module and the rendering module by nature and suit CSM well. However, not all applications conform to this assumption. Some applications place partial state information on the client to reduce information exchange, e.g., user's perspective. Another exception is the single-user application, which does not have a central logic server at all. To make CSM applicable to the exceptions mentioned above, modifications to the applications should be made. For applications with partial states at clients, the source edge cloud could temporarily send local states to the global logic server during migration to enable complete state synchronization. Such temporary state sharing has been implemented in the modified Minetest used in the evaluation. The user's perspective states are sent to the logic server first and then synchronized to secondary sessions to guarantee an identical view. For single-user applications, the rendering model and the logic model could be decoupled to make them similar to multi-user applications. As a result, the logic model can play a role similar to the cloud logic server in multi-user applications. Besides, a network interface should be provided for cross-server state synchronization.

Necessary modification to applications: Since CSM is an application-level method, the applications should be modified to make CSM applicable. The good news is that the core logic of the application is not affected. The developers only need to apply the session management mechanism mentioned in Session III-C2 to support copilot session. This will allow one user account to log in simultaneously at different places during the migration and enable the dual rendering process.

VII. Related Work

VM live migration is widely used in cloud data centers first. With the emergence of mobile edge computing, massive work has been conducted to improve the VM migration performance. Kiryong Ha et al. [15] propose VM migration to solve the access latency deterioration resulted from user mobility. They focus on reducing transfer data size and total migration time leveraging difference operation between running VM at source node and base VM at the destination node. In another work, Kiryong Ha et al. [37] propose an agile edge computing system to cope with situations including sudden user increment, edge device failure and user movement. They leverage VM to conduct migration and guarantee safety, multi-tenant isolation and ease software provision. Takaaki Fukai et al. [38] point out that live migration of bare-metal services is not supported since such a functionality relies on virtualization software. They implement a thin hypervisor which is only in charge of the process of migration. Such a thin hypervisor exposes hardware directly to the guest operating system so that it can achieve the same performance as that of original bare-metal systems. Pre-copy [9], [39] and post-copy [14] are the most widely used mechanisms in VM live migration. Zhong Wang et al. propose Ada-copy [40] which adaptively select the appropriate migration method based on different dirty page rate to reduce migration time.

Research on container live migration is based on checkpoint and restoration provided by CRIU [13].

Pekka Karhula et al. [41] use container checkpoint to make it possible to apply function as a service on IoT devices and enable service migration among devices. They have shown that compared with Docker pause, container checkpoint is more memory-saving and increases the number of long-running functions running on one IoT device. Keerthana Govindaraj and Alexander Artemenko [42] focus on guaranteeing the safety and timely operation of factory automation applications. They propose redundancy migration, which leverages checkpoint, packets buffer replay to reduce downtime during seamless service migration. Lele Ma et al. [10], [12] discover that transferring the whole image brings extra file system synchronization overhead. To mitigate this problem, they propose to take full advantage of the container's layered structure and only transfer the top layer as well as runtime memory during migration, which significantly reduces migration time. Similarly, Andrew Machen et al. [43] propose a three-layer model to reduce downtime of live migration. In their model, a service consists of three layers, i.e. a base layer that includes guest OS, an application layer that contains application files and an instance layer containing running states. During a migration, only the instance layer is transferred, which significantly reduce the amount of data to be transferred. Instead of trying to reduce the data amount in the migration, Piush K Sinha et al. [44] propose to leverage memory ownership relocation to simplify the process of live migration. However, the source VM and the destination VM is required to be on the same host to make the relocation applicable.

In addition to VM-based and container-based migration methods, application-level migration methods are studied to further reduce the transferred data amount. Patrik J. Braun et al. [45] implement an application-level migration which separates state data from the main logic. Such a method can reduce migration time. R. Bruschi et al. [46] migrate virtualized network functions leveraging an SDN-based packet duplication mechanism. A lightweight heuristic algorithm is also proposed to optimize the orchestration of new service instances placement. Michael Gundall et al. [47] achieve an application-level migration containing two options. The first option leverage packet buffering to realize a parallel migration methodology. If the first option is not applicable, a CRIU-based method will be used. Tung V. Doan et al. [48] achieve seamless service migration in autonomous driving scenarios. Their framework periodically synchronizes application state among edge nodes and proactively migrates service instances to the synchronized nodes. A. Gember-Jacobson et al. [49] propose a two-phase scheme for updating network forwarding state in OpenNF. A dual-service method is used for process migration, which requires a centralized cache to support sequentially executed operations. Application-level service migration methods are usually specifically applicable to its target application. The aforementioned methods are effective but not designed for multi-user real-time rendering applications. Besides, 5G network session switches are not considered, either.

Network management in service migration is also studied. Existing work focuses on overcoming migration obstacles brought by cross-domain network communication. Yuqing Qiu et al. [50]

focus on possible network failures during migration data being transferred through WAN. They propose to leverage the multi-path TCP protocol to improve the resilience of service migration between different edge nodes. Hany Assasa et al. [51] proposed a Platform as a Service framework that supports UE context migration edge clouds. Rami Akrem Addad et al. [52] focus on the inconvenience of inter-edge migration brought by the infeasibility of ARP broadcasting over the Internet. They introduce SDN-based methods to handle network issues. Jaehee Ha et al. [53] leverage distributed hash table and live migration access router to enable the WAN to support VM and container migration between edge nodes. Compared with these methods, cross-domain network communication is not a problem in our method because peer-to-peer data copy is bypassed.

VIII. Conclusion

Reducing downtime is the key challenge of achieving imperceptible service migration. We show that because of the complicated memory copy operation and the unavoidable data transference, it is hard to reduce the downtime of VM live migration and container live migration to a degree that can satisfy the requirement of real-time rendering applications. We present CSM, an application-level service migration method that takes both computation state migration and 5G user plane PDU session switch into consideration. In order to reduce service migration downtime, CSM takes advantage of global state data in the cloud logic server and synchronize states between edge clouds via dual rendering. This bypasses the memory copy mechanism and significantly simplifies the freezing stage. A UE-centric migration switch management mechanism is proposed to achieve well-coordinated network switch and computation state migration, and a smooth switching mechanism is leveraged to guarantee user experiences during migration. CSM is implemented and evaluated in a system with a 5G network and application streaming capabilities. The experiment results show that CSM successfully achieves imperceptible migration with downtimes lower than 14 ms when migrating the tested real-time rendering applications. The cross-edge data transference is even reduced to zero.

References

[1] Q. Yu, J. Ren, Y. Fu, Y. Li, and W. Zhang, "Cybertwin: An origin of next generation network architecture," *IEEE Wireless Commun.*, vol. 26, no. 6, pp. 111–117, Dec. 2019.

[2] iLab, "Cloud VR network solution white paper," *Hua Wei iLab, Tech. Rep.*, 2019. [Online]. Available: https://www.huawei.com/minisite/pdf/ilab/cloud_vr_network_solution_white_paper_en.pdf

[3] iLab, "Cloud VR black edge and network delay relationship white paper," *Hua Wei iLab*, Tech. Rep., 2019. [Online]. Available: https://www-file.huawei.com/-/media/corporate/pdf/ilab/2019/cloud-vr-blackline-network-delay-en-v1.pdf

[4] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol (RTSP)," 1998. [Online]. Available: https://www.ietf.org/rfc/rfc2326.txt

[5] R. Mallipeddi and M. Srivastava, "Real time messaging protocol (RTMP)." [Online]. Available: https://rtmp.veriskope.com/docs/spec/

[6] C.-Y. Huang, K.-T. Chen, D.-Y. Chen, H.-J. Hsu, and C.-H. Hsu, "GamingAnywhere: The first open source cloud gaming system," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 10, no. 1s, pp. 10:1–10: 25, Jan. 2014.

[7] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2333–2345, Oct. 2018.

[8] iLab, "Cloud VR solution white paper," *Hua Wei iLab, Tech. Rep.*, 2018. [Online]. Available: https://www.huawei.com/minisite/pdf/ilab/cloud_vr_network_solution_white_paper_en.pdf

[9] C. Clark et al., "Live migration of virtual machines," *Proc. Sympos. Netw. Syst. Des. Implementation*, vol. 2, pp. 273–286, 2005.

[10] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via docker container migration," in *Proc. IEEE/ACM 2nd Symp. Edge Comput.*, 2017, pp. 11:1–11:13.

[11] S. Stoyanov and M.J. Kollingbaum, "Efficient live migration of Linux containers," in *High Performance Computing, ser. Lecture Notes in Computer Science*, R. Yokota, M.J. WeilandShalf, and S. Alam eds., Berlin, Germany: Springer, 2018, pp. 184–193.

[12] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Trans. Mobile Comput.*, vol. 18, no. 9, pp. 2020–2033, Sep. 2019.

[13] CRIU, "Main page – CRIU," 2022. [Online]. Available: https://www.criu.org

[14] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *ACM SIGOPS Operating Syst. Rev.*, vol. 43, no. 3, pp. 14–26, Jul. 2009.

[15] K. Ha et al., "Adaptive VM handoff across cloudlets," Tech. Rep. CMU-CS-15–113, 2015.

[16] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "CEPH: A scalable, high-performance distributed file system," in *Proc. Symp. Operating Syst. Des. Implementation*, 2006, pp. 307–320.

[17] L. NFS, "Main page – Linux NFS," 2022. [Online]. Available: https://linux-nfs.org/wiki/index.php

[18] CRIU, "What cannot be checkpointed," 2022. [Online]. Available: https://criu.org/What_cannot_be_checkpointed

[19] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.

[20] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on markov decision process," *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1272–1288, Jun. 2019.

[21] Z. Rejiba, X. Masip-Bruin, and E. Marín-Tordera, "A survey on mobility-induced service migration in the fog, edge, and related computing paradigms," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 90:1–90: 33, Sep. 2019.

[22] Proxmox VE, "Main page – proxmox virtual environment," 2022. [Online]. Available: https://pve.proxmox.com/mediawiki/index.php

[23] Podman, "Main page – podman," 2022. [Online]. Available: https://podman.io/

[24] F. Bellard, "Qemu, a fast and portable dynamic translator," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf.*, 2005, Art. no. 41.

[25] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: The linux virtual machine monitor," *Proc. Linux Sympos.*, vol. 1, no. 8, pp. 225–230, Jul. 2007.

[26] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, 2014, Art. no. 2.

[27] LXD, "Main page – LXD," 2022. [Online]. Available: https://linuxcontainers.org/lxd/

[28] OpenVZ, "Main page – OpenVZ," 2022. [Online]. Available: https://openvz.org/

[29] TC, "TC – linux manual page," 2023. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html

[30] Stress, "Stress - linux man page," 2023. [Online]. Available: https://linux.die.net/man/1/stress

[31] Tar, "Tar - linux man page," 2023. [Online]. Available: https://linux.die.net/man/1/tar

[32] S. Tomar, "Converting video formats with FFmpeg," *Linux J.*, vol. 2006, no. 146, 2006, Art. no. 10.

[33] Glmark2, "Glmark2 - an opengl 2.0 and es 2.0 benchmark," 2023. [Online]. Available: https://github.com/glmark2/glmark2

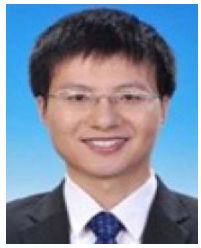
[34] PrinzOwO, "Main page – GTP5G," 2022. [Online]. Available: https://github.com/PrinzOwO/gtp5g

[35] Mininet, "Main page – Mininet," 2022. [Online]. Available: http://mininet.org/

[36] Pygame, "Main page – Pygame," 2022. [Online]. Available: https://www.pygame.org

[37] K. Ha et al., "You can teach elephants to dance: Agile VM handoff for edge computing," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2017, pp. 1–14.
[38] T. Fukai, T. Shinagawa, and K. Kato, "Live migration in bare-metal clouds," *IEEE Trans. Cloud Comput.*, vol. 9, no. 1, pp. 226–239, Jan./Mar. 2021.
[39] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf.*, 2005, Art. no. 25.
[40] Z. Wang et al., "Ada-copy: An adaptive memory copy strategy for virtual machine live migration," in *Proc. IEEE Int. Conf. Parallel Distrib. Syst.*, 2017, pp. 461–468.
[41] P. Karhula, J. Janak, and H. Schulzrinne, "Checkpointing and migration of IoT edge functions," in *Proc. Int. Workshop Edge Syst. Analytics Netw.*, 2019, pp. 60–65.
[42] K. Govindaraj and A. Artemenko, "Container live migration for latency critical industrial applications on edge computing," in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Automat.*, 2018, pp. 83–90.
[43] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 140–147, Feb. 2018.
[44] P. K. Sinha, S. S. Doddamani, H. Lu, and K. Gopalan, "mWarp: Accelerating intra-host live container migration via memory warping," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2019, pp. 508–513.
[45] P. J. Braun, S. Pandi, R.-S. Schmoll, and F. H. P. Fitzek, "On the study and deployment of mobile edge cloud for tactile internet using a 5G gaming application," in *Proc. IEEE Annu. Consum. Commun. Netw. Conf.*, 2017, pp. 154–159.
[46] R. Bruschi, F. Davoli, P. Lago, C. Lombardo, and J. F. Pajo, "Personal services placement and low-latency migration in edge computing environments," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw.*, 2018, pp. 1–6.
[47] M. Gundall, J. Stegmann, M. Reichardt, and H. D. Schotten, "Downtime optimized live migration of industrial real-time control services," in *Proc. Int. Sympos. Indus. Elec.*, 2022, pp. 253–260.
[48] T. V. Doan et al., "Seamless service migration framework for autonomous driving in mobile edge cloud," in *Proc. Annu. Consum. Commun. Netw. Conf.*, 2020, pp. 1–2.
[49] A. Gember-Jacobson et al., "OpenNF: Enabling innovation in network function control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 163–174, Aug. 2014.
[50] Y. Qiu, C.-H. Lung, S. Ajila, and P. Srivastava, "LXC container migration in cloudlets under multipath TCP," in *Proc. IEEE Annu. Comput. Softw. Appl. Conf.*, 2017, pp. 31–36.
[51] H. Assasa, S. V. Yadhav, and L. Westberg, "Service mobility in mobile networks," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2015, pp. 397–404.
[52] R. A. Addad, D. L. C. Dutra, T. Taleb, M. Bagaa, and H. Flinck, "MIRA!: An SDN-Based framework for cross-domain fast migration of ultra-low latency 5G services," in *Proc. IEEE Glob. Commun. Conf.*, 2018, pp. 1–6.
[53] J. Ha, J. Park, S. Han, and M. Kim, "Live migration of virtual machines and containers over wide area networks with distributed mobility management," in *Proc. EAI Int. Conf. Mobile Ubiquitous Syst.: Comput. Netw. Serv.*, 2018, pp. 264–273.

Yuanzhe Li received the PhD degree from the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications (BUPT), in 2022. He is a postdoc with the Institute for AI Industry Research (AIR), Tsinghua University. His work won the best paper award from CollaborateCom 2022. His research interests include mobile edge computing, cloud computing, and service computing.



Shangguang Wang (Senior Member, IEEE) is a professor with the School of Computer Science, Beijing University of Posts and Telecommunications, China. He is the founder&chief scientist of Tiansuan Constellation. He is also deputy dean with Beiyou Shenzhen Institute, and Director with Star Network and Intelligence Computing Laboratory, China. His research interests include service computing, mobile edge computing, cloud computing, and satellite computing. He is currently serving as chair of IEEE Technical Community on Services Computing (TCSVC), and vice chair of IEEE Technical Community on Cloud Computing. He also served as General Chairs or Program Chairs of 10+ IEEE conferences, advisor/associate editors of several journals. He is a Fellow of the IET. More details could be found by http://sguangwang. org/

Yuanchun Li (Member, IEEE) received the BS and PhD degrees in computer science from Peking University, and was a senior researcher with Microsoft Research Asia. He is a research Assistant Professor with the Institute for AI Industry Research (AIR), Tsinghua University. His research interests lie in the efficiency and reliability of edge AI systems. His work won the UbiComp Honorable Mention Award and ISEUD Best Paper Award, and the related systems and tools are widely used in the open-source community. He is a member of ACM.

Ao Zhou (Member, IEEE) is an associate professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, China. Her research interests include service computing, cloud computing, and mobile edge computing. She is a PI or Co-PI of several network service-involved research projects funded by National Key Research and Development Program of China, National Natural Science Foundation of China, and Key Research and Development Program of Guangdong Province. She has published more than 50 papers, and received the best paper award from IEEE SCC 2021. Her recent work can be found on premier journals such as *IEEE Transactions on Services Computing*, *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Emerging Topics in Computing*, and premier conference such as WWW, INFOCOM, ICWS. She served as TPC Chair of IEEE DATACOM 2021, IEEE SAGC 2020, CBPM 2020, and CollaborateCom 2016, etc. She was awarded the Rising Star Award of IEEE Technical Committee on Cloud Computing, and the First Prize of WU WEN JUN AI Science & Technology Award of China.

Mengwei Xu (Member, IEEE) received the BS degree in 2015 and the PhD degree in 2020 from the Department of Computer Science and Technology, Peking University. He is an assistant professor in the computer science Department with the Beijing University of Posts and Telecommunications. His research interests cover the broad areas of mobile computing, edge computing, artificial intelligence, and system software. Web page: https://xumengwei.github.io.

Xiao Ma (Member, IEEE) received the PhD degree in Department of Computer Science and Technology from Tsinghua University, Beijing, China, in 2018. She is currently a lecturer with the State Key Laboratory of Networking and Switching Technology, BUPT. From 2016 to 2017, she visited the Department of Electrical and Computer Engineering, University of Waterloo, Canada. Her research interests include edge computing and satellite computing.

Yunxin Liu (Senior Member, IEEE) received the BS and MS degrees from Tsinghua University, and University of Science and Technology of China (USTC), respectively, and the PhD degree from Shanghai Jiao Tong University (SJTU). He is a guoqiang professor with Institute for AI Industry Research (AIR), Tsinghua University. He was a principal research manager with Microsoft Research Asia (MSRA). His research interests are mobile computing and edge computing. He received MobiSys 2021 Best Paper Award, SenSys 2018 Best Paper Runner-up Award, MobiCom 2015 Best Demo Award, and PhoneSense 2011 Best Paper Award.