Towards A Fast Service Migration in 5G

Rami Akrem Addad[1], Diego Leonel Cadette Dutra[2], Miloud Bagaa[1], Tarik Taleb[1,4]
and Hannu Flinck[3]
[1] Aalto University, Espoo, Finland
[2] Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
[3] Nokia Bell Labs, Espoo, Finland
[4] Oulu University, Oulu, Finland
Emails:{firstname.lastname}@aalto.fi[1]; diegodutra@lcp.coppe.ufrj.br[2]; hannu.flinck@nokia-bell-labs.com[3]

***Abstract*—The development of the 5G technology has been driven by the need for faster and higher-capacity networks that would be able to sustain modern, high-demanding applications and low-latency communication. For achieving such requirements, the services should be shifted towards the vicinity of the users as much as possible. In this scope, Multi-access Edge Cloud (MEC) will play a tremendous role in 5G technology by hosting various services close to the end-users. Thanks to the MEC technology, different services could be placed near the User Equipment (UEs) for enabling very low latency and high bandwidth communication, which is required by real-time applications, such as mobile broadband in vehicles that are characterized by high UEs mobility. In order to ensure the service requirement in terms of low-latency communication, those services should follow (i.e., service migration) the user mobility by placing them always at the closest MEC. Motivated by the evolution of real time applications, we propose and evaluate three different mechanisms to improve the end user experience by using container-based live migration paradigm. In these approaches, we leverage the follow-me edge concept for enabling lightweight live migration. While the two first solutions take into consideration the mobile users' paths (e.g., cars), the third one is oblivious to the users' paths. The obtained results demonstrate the efficiency of these solutions compared to prior works.**

I. Introduction

The mobile network traffic keeps increasing at a very fast pace, and that is due to the emerging mobile applications, such as high-resolution video streaming, cloud gaming and augmented reality applications [1]. Within few years, $4G$ systems definitely will not catch up with the pace of the traffic increase, as well as the expected requirements of the new emerging applications, such as autonomous cars, Unmanned Aerial Vehicles (UAVs) and augmented reality. For this reason, many efforts, by both academia and industrial researchers, have been carried out in order to make the $5G$ system a reality in the nearest future.

There is a consensus among academia and industrials that the $5G$ system will leverage emerging technology, such as Network Function Virtualization (NFV) and Software Defined Networking (SDN), for achieving its objectives [2]. While SDN gives more flexibility for connecting different components and enables the network softwarization, NFV allows running Virtual Network Functions (VNF) as software components on top of a virtualization system (i.e., Virtual Machines - VMs - or Containers) hosted in various clouds; allowing high flexibility and elasticity to deploy network services and functions. These VNFs will run on top of cloud nodes that are sparsely distributed over the globe. The use of NFV on top of cloud nodes in the $5G$ system will reduce dramatically both capital expenditures (CAPEX) and operational expenses (OPEX). However, it can be against the requirements of $5G$ systems in terms of high data rates and low latency. In fact, instantiating different VNFs at faraway clouds would have a negative impact on both the data rates and end-to-end delay. In order to overcome this limitation, the concept of Multi-access Edge Computing (MEC) has been introduced [3]. It allows instantiating various VNFs (i.e, containers) in the vicinity of users. Indeed, the MEC technology allows the deployment of a subset of cloud resources at the edge of the cellular network (eNodeB), thus network congestion is reduced and better QoS/QoE can be ensured for different customers.

In this context, MEC has been proven to be beneficial in reducing latency and offering an overall better experience [4]. As mentioned earlier, a user in $5G$ can be characterized by high mobility that may take him far away from the original MEC node where his service runs. In order to overcome such problem, a new concept, dubbed Follow Me Cloud (FMC) [5], has been introduced. In fact, the FMC concept allows the mobility of services between different edges for placing them closest to end users, which ensures low latency ($1ms - 10ms$) and high capacity (more than 100 Mbps). However, the main challenging problem of FMC is the service interruption during the migration of services from an edge cloud to another, which dramatically affects the requirements of industrial verticals.

Towards addressing the problem of service interruption when migrating services between edge clouds, in this paper, we propose and evaluate three solutions that leverage on the container technology to reduce the migration time, and then reduce the services' interruption. The three solutions leverage LXC container tool and Checkpoint/Restore in Userspace (CRIU) for enabling stateful migration, and then ensuring the service continuity after migration. These proposed solutions enable the FMC concept for ensuring high availability and ultra-low latency for real-time applications, such as autonomous car driving and UAV traffic management. While the first two solutions assume that the trajectories of the mobile users, e.g., cars or UAVs, are well known a priori, the third

978-1-5386-4633-5/18/$31.00

solution is more general, as it is oblivious to the mobile users' trajectories.

The remaining of this paper is organized as follows. Section II presents the background of this research and some related work. In Section III, we describe the types of migration evaluated in this paper and how they are deployed in our test environment. In Section IV, we present and discuss the results of our experimental evaluation. Finally, the paper concludes in Section V.

II. Background & Related work

This section introduces several underlying concepts that ease the paper readability. A summary of past relevant research work is also included.

A. Background

1) Linux Container (LXC): LXC is a lightweight virtualization technology integrated into Linux kernel to enable the running of multiple containers on the top of the same host. LXC container is built by mixing the Linux namespaces and CGroups to ensure a soft separation without virtualizing the hardware as the legacy virtual machine does. Compared to Docker, LXC [6] is a system level container.

2) Checkpoint/Restore In Userspace (CRIU): The proposed system leverages the CRIU (Checkpoint/Restore In Userspace) tool [7], which allows to check-point/restore processes in Linux systems. It has the ability to save the state of a running application, so that its execution can later be resumed from the time of the checkpoint.

3) Live Migration: The live migration is the process that guarantees the transfer of both the disk and the current memory pages while the host is in running state.

4) Iterative Migration: The iterative migration aims at dividing the memory pages copy into several steps, each one of them (except the last one) does not stop the container and takes only the changes relative to the previous iteration. We name each intermediate step the Predump phase, while the last one is named "Dump phase". The Dump phase will provoke the stop of the instance of virtualization (i.e., container in our case). However, a small downtime will occur due to the small number of memory pages copied to the destination host.

5) Downtime: Downtime is the period during which the services provided by the migrating instance (i.e., VM or container) are not available or no longer meet user requests.

6) Total migration time: The total migration time is the period of time between the launch of the migration process till the moment when the instance is made available to the destination server.

B. Related work

Yang [8] presented a generic checkpoint/restore mechanism and evaluated its performance using Docker. Each of the checkpoint and restore phases took $2183\ ms$ and $1998\ ms$, respectively, when considering a $256MB$ container size. In contrast to Docker, LXC container gives more flexibility for running different applications, services, and protocols. Indeed, Dockers are standalone applications running in an isolated environment and do not offer any system level functionality. Moreover, the checkpoint and restore phases take a considerable amount of time for a lightweight container. For this reason, in this paper, we used the LXC container technology instead of the Docker technology.

The author of [9] evaluated and compared both OpenVZ and LXC. The comparison showed that checkpoint/restart migration on OpenVZ had poor performance when compared to LXC regarding the Downtime and Total Migration Time. In [10], the authors evaluated container migration on a MEC platform based on OpenVZ. Their results showed that the total time for migration of a blank container is considerable even when using shared storage (NFS) and shared-async mode with a range of $(10s–11s)$. Machen et al. [11] presented a multi-layer framework for migrating active applications in the MEC. While the obtained results show exceptional total migration times, the downtime was considerable with an average of $2s$ in case of a blank container. The increase of the downtime is due to the non-use of the iterative approach in the live migration process. Thanks to the use of LXC combined with our new migration strategies based on iterative migration, the proposed solutions enhance the aforementioned works in terms of migration time as well as the downtime.

III. On using lightweight containers in MEC environments for enabling mobility applications

In this section, we first present the main architecture of our proposed framework for enabling lightweight container migration. Then, we will present the three proposed solutions that will enable lightweight container migration.

A. Main architecture and problem formulation

Fig. 1 depicts a typical three-layer cloud-based architecture for 5G networks. The top layer (Layer 3 in Fig. 1) consists of the core network which can include data centers with powerful computing resources from different vendors (e.g., Amazon, Microsoft, and OpenStack). Orchestrated by the top layer, the edge clouds feature the Radio Access Network (RAN) with high spectral efficiency and bandwidth. This distributed computing model allows users – from the third layer – to be close to the compute capabilities according to their mobility. In our presented use-cases, we consider UAVs as well as users on board high-speed vehicles and we assume their paths can be either pre-determined/predictable (i.e., this allows the system to trigger the migration process earlier on, before reaching the edge of the cell), or random which means that they can take a random path at any given moment and the actual migration has to be done in a limited time frame (when reaching the edge of the cell). The main focus is on the implementation of the live migration itself in several aspects to ensure a seamless migration across edge clouds, without taking into account other use-case-specific aspects, such as the signal strength received by each vehicle, UE or UAV.

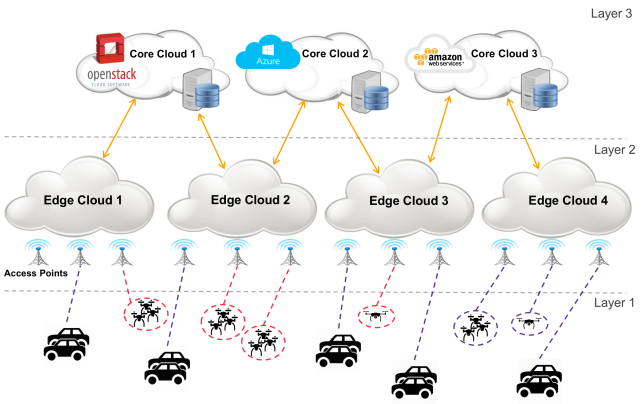


Fig. 1. A general architecture.

B. State-full service migration based on predefined path

In this section, we present two solutions that should be used for predefined paths. A common use-case for this would be a vehicle or a UAV that has a known path from departure to destination. Knowing the paths allows the system to anticipate the different source and target MECs for any migration along the way of the mobile node. This also means that the different MECs can be fully independent, e.g., no shared storage is needed, and gives enough time to copy the file system, which gives more freedom to the migration service. In the following section, we will introduce a live iterative migration based on two different approaches by using the CRIU tool [7].

1) Temporary File System based Lightweight Container Migration: This solution is also named *tmpfs migration*. The *tmpfs migration* solution starts first copying the container's file system along with the user files from the current MEC host to the destination MEC node using the **rsync** utility without service disruption. Second, the memory of the container is iteratively copied from the source MEC host to the destination MEC host. In this step, the CRIU utility will be used for iteratively dumping the container's memory - while it is running - into a **tmpfs**-mounted directory at the source MEC host. In this case, at the source MEC host, we have one reading from the memory and one writing to the **tmpfs**-mounted directory. In fact, we have one reading and one writing at the source MEC host. Each Dump is then copied to the destination host via the network into the **tmpfs**-mounted directory at the destination MEC host which will result in a writing operation at the destination MEC host. Finally, read actions will be used in order to restore the container at the destination MEC host. We will also have one reading and one writing at the destination MEC host.

2) Disk-less based Lightweight Container Migration: In the first solution, we noticed that the memory images have two readings and two writings, which could have a negative impact on the total migration time. The process can be worse if the application uses a large amount of memory and/or multiple iterations (Predumps). The disk-less migration solution overcomes this limitation. It aims to eliminate the step of copying the images to the local **tmpfs** directory in order to reduce further the total migration time. The proposed solution starts by copying the file system and preparing a **tmpfs** mount on both (source and destination) MEC hosts. Moreover, at the destination MEC host, we start a page server indicating the images directory and the port which will be used by the source MEC host to copy the files. Then on the source MEC host, using CRIU, we have adopted a new strategy by combining our iterative approach of live migration with the page server implementation. We start by dumping the memory pages directly into the target cloud using the two extra parameters: the page server's address and port while keeping the iterative concept working. Finally, we copy the rest of the images to the destination MEC host and we restore our container right after.

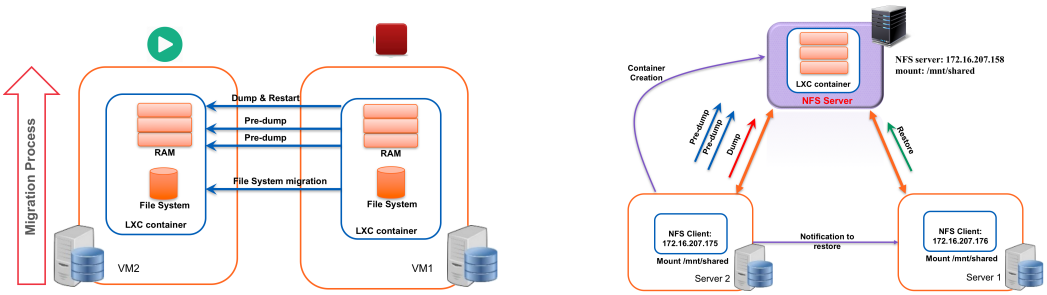
In order to test the two types of migration procedures as introduced above, we built a testbed, as shown in Fig. 2(a), to guarantee the most common architecture in a real case. The testbed consists of two VM hosts; each one representing a different Edge Cloud (i.e., an independent Infrastructure as a Service - IaaS - provider). Our container host is running on top of the first VM.

C. State-full service migration based on undefined path

In most real-world applications, the service provider (cloud service provider) does not know the movement pattern of the users. For this reason, we suggest a more general solution that considers the paths of users to be unknown a priori. In this case, the copy of the file system and memory from the source MEC host to the destination MEC host could be a challenging process. For this reason, in this solution, also named lightweight containers migration with a shared file system, we need to use an alternative, fast and efficient migration process.

To eliminate the need to copy files over the network during the migration phase, we stored the container's file system along with the system images in a shared storage pool. This means that the system needs to iteratively unload the container's memory using CRIU on the source node and then restore the container to the target node immediately after. This approach uses more network resources, while reducing the total migration time for LXC.

For evaluating the third solution, we have used the testbed, depicted in Fig. 2(b), that consists of three VMs. The first VM is the source MEC host, whereas the second one is the destination MEC host. Meanwhile, the third VM is the Network File Storage (NFS) server that is used to store the containers' file-system. We choose NFS because it represents a standard file system sharing technique, and while it is not scalable as a distributed or parallel file system, it allows the evaluation of the impact that a shared file system has on our migration procedure with containers. Furthermore, in terms of performance overhead, it represents for our setup a lower-bound as a distributed file system would impose additional overheads for a small-scale deployment. In the testbed, we have ensured that the three VMs can communicate among themselves in order to enable container migration. While the



(a) Prototype 1. (b) Prototype 2.

Fig. 2. Proposed prototypes for both known and unknown path solutions

communication between the MEC nodes and the NFS server is used for disk migration, the direct communication between MEC nodes is used for the migration of memory content.

IV. Experimental Evaluation

To evaluate the envisioned container migration scenarios, virtualized computer nodes are used. Each node uses Ubuntu 16.04 LTS with the 4.4.0-64-generic kernel and has 4 core CPU and 4GB of main memory. The interconnection among the nodes is set at 1Gb/s. The container environment was setup using LXC 2.8 and CRIU 2.6 (i.e., both are stable versions), while the NFS server used a dedicated virtual machine. For every container migration, we evaluate the total migration time and the container downtime as it directly corresponds to the application responsiveness/availability during the migration process.

TABLE I
Test video specifications.

Type	Configuration
Codec	H.264
Duration	$442s$
Bit rate	$1,560\ Kbps$
Quality	$720p$
File size	$93.5\ MB$

We conducted two sets of experiments; each was repeated ten times. The first one was a blank Linux container migration, with a file system size equal to $350\ MB$. We kept an eye on the container's network reachability throughout the migration process in order to observe the impact caused by adding persistent data. The second one was the migration of a video streaming server NGINX running on top of a container, where file-system size was $590\ MB$. The video used in our experiments has its configuration described in Table I.

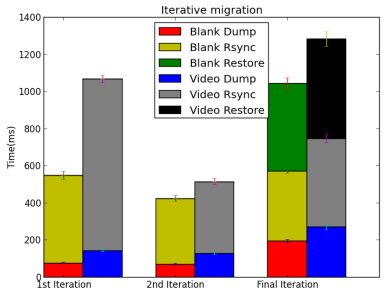
A. Migration induced downtime

The first experiment uses the tmpfs migration strategy where the blank container's downtime, standard deviation and 95% confidence interval are $1042.974ms$, $52.790ms$ and $39.806ms$, respectively. Regarding the video-streaming container, the downtime, standard deviation and $95\% C.I$ are $1282.012ms$, $76.141ms$ and $57.415ms$, respectively. As expected, the results for the video-streaming container are larger when compared to the blank container's results. The difference in these results is due to the additional copies of the network connections status and the NGINX internal control data to the target cloud. We also noticed that the addition of the NGINX HTTP server introduced more variability in our experiments, nevertheless, this represented an increase in the coefficient of variation of less than 15.686% going from 0.051 in the blank container to 0.059 in video streaming container.

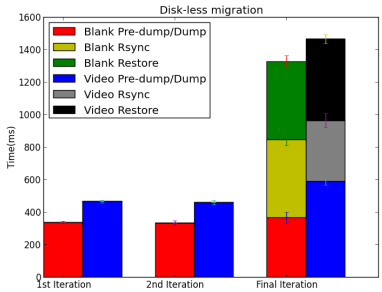
Fig. 3(a) presents a breakdown of these times, alongside the breakdown of the initial phases of the migration procedure. As previously described, the migration procedures, adopted for this paper, use two pre-copy phases – 1^{st} and 2^{nd} iteration in the figure – before actually migrating the container to the destination compute node, in the final iteration phase. In the figure, each one of these iterations can be viewed in the X-axis, while the Y-axis presents the time in milliseconds. We also see that for both pre-copy phases, the video container took longer than the blank container, as the migration procedure needs to save and copy additional memory updates during each iteration, which ends up increasing the downtime in comparison with the blank container by 22.919%.

Our second experimental scenario evaluates the Disk-less Migration described in Section III. The Disk-less Migration procedure imposed a 10.499% increase in the downtime for the blank container migration in comparison with the tmpfs Migration procedure, which represents a mean downtime of $1152.476ms$. The standard deviation for this experiment was $76.332ms$ and 95% confidence interval was $57.558ms$. As for the video container, the mean downtime was 1465.046, which represents an increase of 14.277% over the blank container. The standard deviation for the video container Disk-less Migration was $107.117ms$ and 95% confidence interval was $80.772ms$. These results represented an increase in the coefficient of variation for the downtime of 10.606% when compared with the blank container going from 0.066 to 0.073.

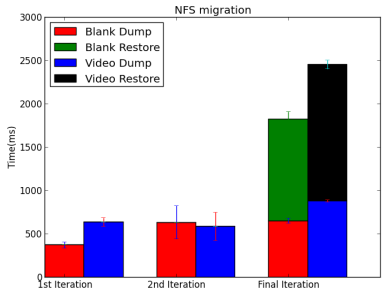
Fig. 3(b) presents a breakdown of the time for the 3 phases for the Disk-less Migration experiment scenario using the same X/Y-axes setup of the previous figure. As Fig. 3(b) shows, the increase in the downtime was caused by the increase of the Dump/PreDump sub-phase of the migration procedure. As perceived in Fig. 3(a), Fig. 3(b) reveals that the video container took longer time to Dump/PreDump in comparison with the blank container. Furthermore, comparing the results presented in Fig. 3(a) and Fig. 3(b), it can be easily



(a) Downtime comparison in case of an Iterative Migration process.



(b) Downtime comparison in case of a Disk-Less Migration.



(c) Downtime comparison in case of a NFS migration.

Fig. 3. Downtime in case of different approaches

noticed that for the blank container and in the pre-copy phases (i.e., 1^{st} iteration and 2^{nd} iteration), the combination PreDump took longer time than the same container in the Disk-less Migration, while the page server augmented PreDump takes a longer time than the normal PreDump, which reinforces our suspicions about the page server implementation.

In the last experimental scenario, we evaluate the NFS migration described in Section III. As the previous scenarios, this evaluation is carried out with both a blank container and the video container. Our NFS migration imposed a 74.946% and 58.323% increase in the downtime for the blank container migration in comparison with the tmpfs Migration and the Disk-less Migration, respectively. This represents a mean downtime of $1824.64ms$, while the standard deviation and the 95% confidence interval for this experiment were $122.657ms$ and $57.558ms$, respectively. The mean downtime for the video container was $2454.374ms$, which represents an increase of 74.342% in comparison with the blank container. The standard deviation for the video container NFS migration was $72.592ms$ and 95% confidence interval was $92.490ms$.

In Fig. 3(c), we present the 3 phases of the NFS migration using the same X/Y-axis as before. Compared to the previous figures, the downtime increases due to the restore part of the final iteration. Our preliminary investigation suspects the network side because the target host restore procedure uses a remote file system which incurs an additional latency in the restore procedure. Furthermore, comparing the results presented in Fig. 3(a) and Fig. 3(b), the restore part takes much longer time in case of NFS and also the only manner to do the restore part is to use the target host, which reinforces our suspicions about the network side.

B. Total migration time evaluation

Our previous experimental results showed that our proposed approach reduces the downtime caused by the migration procedure. However, to enable its use for ultra-short latency services, we also need to address the total migration time.

The mean total migration time for the blank container using the tmpfs migration method was $16,405.157\ ms$ and the 95% confidence interval for this experiment was $312.089\ ms$. For the video container, this migration method had a mean

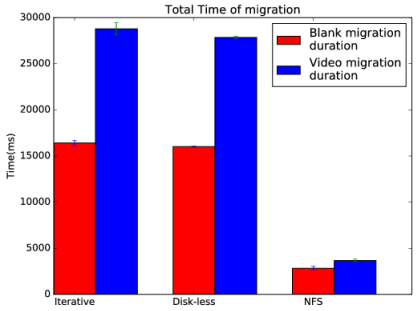


Fig. 4. Total migration time experienced in case of different approaches.

total migration time of $28,788.045\ ms$ and $658.231\ ms$ as its 95% confidence interval. The Disk-less Migration method imposed a mean total migration time of $16,015.2164\ ms$ and $27,841.798\ ms$ for both blank and video containers, respectively. Finally, for our shared storage scenario, the mean total migration time was $2,831.442\ ms$ for the blank container with 95% confidence interval of $54.738\ ms$, while in case of the migration of the video container, these values were $3,678.089\ ms$ and $155.701\ ms$, respectively.

From the results, we can observe that for tmpfs and Disk-less Migration, the long migration time was due to the file system copy, which was avoided in the NFS scenario. Furthermore, for the video streaming container, we clearly notice that the container size merely affects the total time of migration in the case of local storage because of the file system copy. Furthermore, for the NFS scenario, the longer migration time of the video container in comparison with the blank is due to the greater number of memory pages been copied.

C. Impact of the number of pages on the migration downtime

As our experimental evaluation results are influenced by the hardware technology available to us, e.g. network bandwidth, we summarized in Table II the number of memory pages copied during the last Dump step in our migration procedures. In the first column, we have the evaluated scenario, followed by the mean number of pages copied during our evaluation, the standard deviation, and the 95% confidence interval. An initial assessment of this table shows us how disparate the results for

the disk-less migration solution are in comparison with the others, which corroborates the results presented in Fig. 3(b), as it indicates that the larger downtime of this approach was caused by the interaction of the page server and the CRIU migration code which caused an increase in the number of copied pages by almost 8.84 times in our worst-case scenario.

TABLE II
SUMMARY OF PAGES COPIED DURING THE LAST DUMP.

Migration types & cases	Mean N. Pages	Std N. Pages	CI 95%
Blank-Tmpfs Migration	509.5	6.3	5.14
Blank-Disk-less Migration	1779.3	6.84	5.57
Blank-Shared file system Migration	517.8	17.56	14.30
Video-Tmpfs Migration	577.1	7.52	6.125
Video-Disk-less Migration	5100.2	12.52	10.20
Video-Shared file system Migration	551	16.8	13.68

The behavior exhibited for the tmpfs and shared file system solutions are quite similar for both the blank container and our video streaming container, albeit their disparate downtime performances. As previously discussed, the higher downtime for the shared file system solution is caused by the multiple small writes over the network. Meanwhile, the *tmpfs* solution was able to drastically reduce the downtime as it keeps the number of copied pages in the last Dump small and is able to fully utilize the network bandwidth through **rsync**. Furthermore, when we analyze the difference between the mean number of copied pages for the *tmpfs* and shared file system solutions for the blank container, and the same difference for the video streaming application, we conclude that the CRIU Dump code exhibits a similar behavior for both solutions.

D. Results Discussion

As addressed in Table II, the *tmpfs* and shared file system solutions exhibit a similar behavior with regards to the number of copied pages during the last Dump for both the blank container and the video streaming application. The analysis of the number of copied pages also enables us to do a qualitative assessment of the impact that the network bandwidth had over our experimental results, as Table II shows the average number of pages transferred in our best experimental scenario was 509.5 for the blank container and 577.1 for the video container. This means that on average, we transferred 2086912 bytes in the blank container case, and 2281882 bytes in case of the video application, assuming the default 4KB memory page used in our evaluation hardware. These transfers would take around $16.7\ ms$ for the blank container, and $18.9\ ms$ for the video container assuming a Fast Ethernet connection (100Mbps), while for a Gigabit Ethernet (1Gbps) the time spent copying the pages over the network was of the order of $1.67\ ms$ for the blank container and $1.89\ ms$ for the video container. This ten times millisecond improvement due to the network is still fifty times smaller than our improvements in downtime in comparison with Yang [8] and Machen et al. [11].

We have also observed that although the solutions based on predefined paths offer the best downtime results, the total migration time is the highest compared to the other ones. This is simply due to the fact that the file system, user files, and memory images are copied during the migration phase. This is not an issue since we can foresee the migration operations in advance. In contrast, the solution applied in case of unknown paths offers great overall migration time, while sacrificing a few downtime milliseconds since the storage speed is limited by the available bandwidth of the network. However, this is the best solution for this scenario since the decision to trigger the migration along with the migration process itself has to be done in a matter of a couple of seconds.

V. CONCLUSION

In this paper, we proposed and evaluated three migration approaches for the container technology to enable the Follow Me Edge concept using MEC technology, ensuring high availability and supporting ultra-low latency for real-time applications, such as autonomous car driving. The Follow Me Edge allows the system to guarantee a lower latency between the mobile user and the service provider, which is a fundamental requirement for 5G networks. We have evaluated the proposed solutions using real testbed experiments. The obtained results showed the efficiency of the proposed solutions.

ACKNOWLEDGMENT

This work was supported in part by the Academy of Finland 6Genesis Flagship (grant no. 318927). The work was also supported in part by a direct funding from Nokia Bell Labs, Espoo, Finland.

REFERENCES

[1] N. Alliance, "5G White Paper," Tech. Rep., February 2015. [Online]. Available: https://www.ngmn.org/uploads/media/NGMN_5G_White_Paper_V1_0.pdf

[2] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network Slicing; Softwarization: A Survey on Principles, Enabling Technologies; Solutions," *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, pp. 1–1, 2018.

[3] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1657–1681, thirdquarter 2017.

[4] Y. C. Hu, M. Patel, D. Sabellaa, N. Sprecher, and V. Young, "Mobile Edge Computing A key technology towards 5G," Tech. Rep., September 2015. [Online]. Available: http://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf

[5] A. Ksentini, T. Taleb, and M. Chen, "A markov decision process-based service migration procedure for follow me cloud," in *2014 IEEE International Conference on Communications (ICC)*, June 2014, pp. 1350–1354.

[6] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, Sept 2014.

[7] team CRIU, "Criu (checkpoint and restore in user space) main page," 2016. [Online]. Available: https://criu.org/Main_Page

[8] Yang Chen, "Checkpoint and Restore of Micro-service in Docker Containers," in *Proceedings of the 3rd International Conference on Mechatronics and Industrial Informatics*.

[9] P. S. V. Indukuri, "Performance comparison of Linux containers (LXC) and OpenVZ during live migration," Master's thesis, Blekinge Institute of Technology, Sweden, 2016.

[10] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile Edge Computing Potential in Making Cities Smarter," *IEEE Communications Magazine*, vol. 55, no. 3, pp. 38–43, March 2017.

[11] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live Service Migration in Mobile Edge Clouds," *IEEE Wireless Communications*, vol. PP, no. 99, pp. 2–9, 2017.