社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

マルチアクセスエッジコンピューティングにおける リアルタイム処理を伴う映像の低遅延ライブストリーミングのための サービス機能再配置手法

金田　純一[†]　荒川　伸一[†]　村田　正幸[†]

[†] 大阪大学 大学院情報科学研究科　〒565–0871 大阪府吹田市山田丘 1–5
E-mail: †{j-kaneda,arakawa,murata}@ist.osaka-u.ac.jp

あらまし　近年、遠隔地でデータを処理する新しいサービスが数多く登場しており、ユーザが体感するエンドツーエンドの遅延であるアプリケーションレベルの遅延を低減することがユーザの体感品質を向上させる上で重要となっている。そこで、マルチアクセスエッジコンピューティング (MEC) によって遅延を低減することが期待される。しかし、一般にエッジサーバの処理能力はデータセンタよりも小さく、遅延低減効果の大きいサービス機能を選択して展開する必要がある。そこで将来の MEC の展開に向け、MEC 環境で発生するアプリケーションレベルの遅延と発生要因を解明し、機能再配置によってアプリケーションレベルの遅延がいかに低減されるのか調査が必要である。本稿では、OpenStack および Amazon Web Service を使用して MEC 環境を構築し、リアルタイム処理を伴う映像のライブストリーミングサービスを動作させる。アプリケーションレベルの遅延を計測し詳細に分析した結果、映像の低遅延ライブストリーミングのためには、伝搬遅延が小さいことに加えジッターの発生を抑えることが重要だと明らかになった。さらに本稿では、この分析結果に基づいてサービス機能再配置手法を考案する。実機実験の結果、考案した手法によって、アプリケーションレベルの遅延を 400 ms 未満に維持し、ビデオの品質を良好に維持することができた。

キーワード　マルチアクセスエッジコンピューティング、映像のライブストリーミング、リアルタイム処理、アプリケーションレベルの遅延、仮想マシンのライブマイグレーション

Service Function Reallocation Method for Low-latency Video Live Streaming with Real-time Processing in Multi-access Edge Computing

Junichi KANEDA[†], Shin'ichi ARAKAWA[†], and Masayuki MURATA[†]

[†] Graduate School of Information Science and Technology, Osaka University
Yamadaoka 1–5, Suita, Osaka, 565–0871 Japan
E-mail: †{j-kaneda,arakawa,murata}@ist.osaka-u.ac.jp

Abstract　In recent years, many new services that process data at remote base have appeared. In such new services, it is important to reduce application-level delays, i.e. end-to-end delays experienced by the users, in order to improve the users' quality of experience. The concept of Multi-access Edge Computing (MEC) is expected that the delays are reduced. However, the processing capability of edge servers is generally lower than that of data centers. Service functions capable of effectively reducing application-level delays should be deployed on edge servers. For future deployment of MEC, it is therefore necessary to clarify application-level delays and its occurrence factors in a MEC environment, and to investigate how to make the application-level delay lower by reallocations of the functions. In this paper, we construct an MEC environment using OpenStack and Amazon Web Service, then operate a video live streaming service with real-time processing. Results of our measurement and analysis of the application-level delay shows that, for low-latency video live streaming, it is naturally important that the average of propagation delays is small, but it is also important to suppress the jitter caused by the network side. Based on this findings, we devise and implement a simple service reallocation method for low-latency video live streaming. Our experimental results show the application-level delay is kept under 400 ms and video quality is comfortable by the method.

Key words　Multi-access Edge Computing, Video Live Streaming, Real-time Processing, Application-level Delay, Live Migration of Virtual Machine

1. Introduction

In recent years, with the progress of IoT (Internet of Things), many new applications and services that process data at remote base have appeared [1]. Telexistence is one of such new service, which uses a remote robot, a VR headset and a haptic feedback device as end devices. In such new services, it is important to reduce application-level delays, i.e. end-to-end delays experienced by the users, in order to improve the users' quality of experience (QoE). However, processing at a data center can lead to penalties in the form of large application-level delay due to geographical factors and load concentration.

The concept of Multi-access Edge Computing (MEC) has been introduced to mitigate delays [2], [3]. MEC virtualizes service functions and deploys them on virtual machines on edge servers. An edge server is a secondary data center located at the network edge, closer to the user. Incorporating Network Function Virtualization (NFV) to MEC is expected to allow flexible changes in resources and deployment locations of virtual machines [3], [4]. In a MEC environment, service applications use functions at edge servers rather than at data centers. As a result, it is expected that responsiveness to services will be improved by relaxing load concentration and reducing the delay due to geographical factors.

However, the processing capability of edge servers is generally lower than that of data centers. Service functions capable of effectively reducing application-level delays should be deployed on edge servers, since it is impossible to deploy all service functions on an edge server. For future deployment of MEC, it is therefore necessary to clarify application-level delays and its occurrence factors in a MEC environment, and to investigate how to make the application-level delay lower by MEC configurations such as the locations and reallocations of the functions. It is also needed to clarify the effects of service function reallocation on the application-level delay. Moreover, it is important to obtain policies for service function reallocation for the purpose of low-latency service provision.

In this paper, we construct an MEC environment using OpenStack and Amazon Web Service (AWS), then operate a video live streaming service with real-time processing supposing a shopping agent service using robots. A real-time image processing function is deployed on an edge server in the constructed MEC environment. By manually changing the location of the function, we measure the application-level delay of video live streaming and analyze the delay in detail. As a result of the analysis, it is revealed that the increase of buffering time for absorbing jitter, i.e. a variation in packet arriving intervals, increases application-level delays significantly. Therefore, in order to realize low-latency video live streaming, it is naturally important that the average of propagation delays is small, but it is also important to suppress the jitter caused by the network side. Based on this finding, we devise a service function reallocation method for low-latency video live streaming. In our method, we focus on the increases of jitter and application-level delay caused by the increase of the CPU load of the edge server the function is deployed. And the timing of service function reallocation is determined based on the CPU load of edge servers. We implement a simple service reallocation method and confirm the effect of the service function reallocation method on application-level delay.

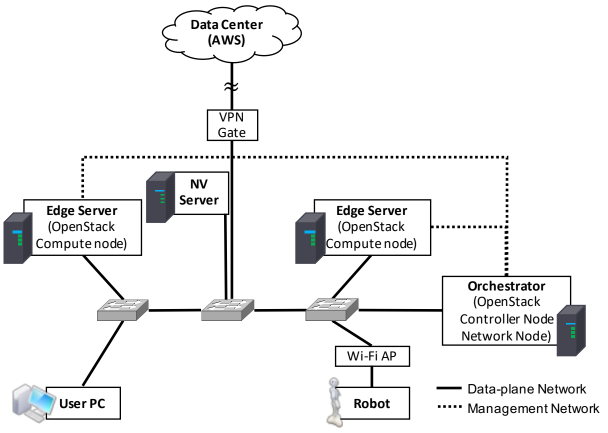

Data Center (AWS)
VPN Gate
NV Server
Edge Server (OpenStack Compute node)
Edge Server (OpenStack Compute node)
Orchestrator (OpenStack Controller Node Network Node)
Wi-Fi AP
User PC
Robot
Data-plane Network
Management Network

Figure 1 Configuration of the MEC environment

The remainder of this paper is organized as follows. Section 2. describes the implementation of a MEC environment and a video live streaming service with real-time processing. In Section 3., we measure application-level delays and analyze the factors of occurring the delays. In Section 4., we devise a service function reallocation method and confirm the effect of the method through our MEC environment. In Section 5., we present our conclusions and future works.

2. Implementation

2.1 Multi-access Edge Computing Environment

To build a MEC environment, we use OpenStack, which is open-source software for creating virtualization environments. The OpenStack development community expects OpenStack to be applied to the edge [5].

We built the MEC environment in our laboratory by connecting four similarly configured server machines, a user's PC and a robot with switches. These devices are also connected to virtual machines on Amazon Web Service (AWS). An overview of the MEC environment is shown in Figure 1. Three of the four similarly configured machines are OpenStack nodes, one operating as an OpenStack controller and network node, and the other two as OpenStack compute nodes. The controller node operate as a control unit such as infrastructure manager and orchestrator in a virtualization environment. We call this node the orchestrator. The compute nodes operate as edge servers. In the shopping agent service, since the robot and the user are geographically separated, the two edge servers are prepared as processing bases close to each other. CentOS 7 and Kernel-based Virtual Machine (KVM) are installed on the compute nodes as a host OS and a hypervisor, respectively. Applications are executed in virtual machines with 4 GB of memory and 32 GB of storage on the compute nodes. Storage files of all virtual machines are shared by Network File System (NFS) among three OpenStack nodes. The fourth server machine is not virtualized for comparison. Hereafter, we refer to this non-virtualized server as the NV server. We also use virtual machines on AWS as data centers at the center of network.

The robot, the user PC, the virtual machines on the edge servers and AWS, and the NV server communicate using a data-plane network. The robot is wirelessly connected to the data-plane network via a Wi-Fi access point. The devices and the virtual machines on the edge servers are connected on local area network (LAN) scale. The virtual machines on AWS are connected to the data-plane network using Virtual Private Network (VPN) via the Internet. The orchestrator uses an out-of-band network to communicate with the edge servers so that they are not at all involved in the data-plane communication. Hereafter, we call this out-of-band network the management network.

2.2 Video Live Streaming Service with Real-time Processing

As a potential new service, we consider realization of a shopping agent service using robots. In this service, robots go to a physical store, and users can shop from home as if they were actually there. Using AR/VR technology, product information is superimposed on the video taken by a camera mounted on the robot and presented to the user. This process is performed on an external server, and product information is acquired from cloud. Sensing technology can also used to present tactile sensations of products and to control the robots.

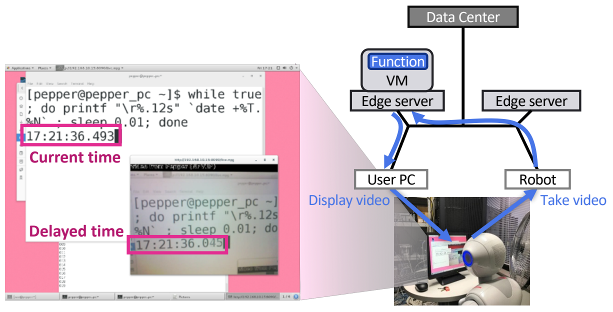
In this paper, we assume only video live streaming from the "Pepper" robot [6] to the user. In the service, the video which is taken by a camera on the robot is compressed into MPEG2 format using FFmpeg, which runs on the operating system of the robot. The video is then transferred to an edge server and text information is added using FFmpeg running on a virtual machine. Another virtual machine relays the video stream to a virtual machine hosting FFserver, a streaming server application, to stream it to the user PC for real-time user viewing using FFplay. To simplify the implementation, we do not insert the superimposition of product information, but insert a simple text. The text is inserted using drawtext filter of FFmpeg. Note that FFserver uses UDP and TCP transport protocols for reception and transmission, because of its specification.

3. Measurement and Analysis of Application-level Delay

3.1 Measurement Setting

3.1.1 Measurement Method

To measure the application-level delay of video live streaming, we use the time difference of the two clocks in the live-streamed video. First, a millisecond-precision digital clock is displayed in front of the robot. Then, the video captured by the robot is live-streamed to the user PC via real-time processing on an edge server. The digital clock



```
[pepper@pepper_pc ~]$ while true
; do printf "\r%.12s" `date +%T.
%N` ; sleep 0.01; done
17:21:36.493
```

```
[pepper@pepper_pc ~]
; do printf "\r%.12s
%N` ; sleep 0.01; do
17:21:36.045
```

Figure 2 Method for Measuring Application-level Delay

shown to the robot is displayed on the monitor of the user PC for time synchronization. The digital clock and the live-streamed video from the robot are arranged on the monitor. Next, a screenshot is taken per second for 100 seconds and the time differences of the two clocks displayed on the monitor are calculated by each screenshot. At this time, the two displayed times are converted mechanically from an image to a numerical value. Specifically, a OCR engine called Tesseract digitizes trimmed screenshots. Finally, we calculate the mean of the time differences to measure the application-level delay of video live streaming. Figure 2 shows the method for measuring the application-level delay.

3.1.2 Deployment Scenarios for Real-time Processing

Seven scenarios are set to investigate the occurrence factors and the volume of application-level delays due to the differences in the locations where service functions are deployed. For each scenario, we changed the forms of service provision, such as the existence of a virtualization environment and the location of service functions. In two edge scenarios, with names beginning with "edge," the service functions are deployed on an edge server. In three DC scenarios, they are deployed on a virtual machine on AWS. The "Non-Virtualized" and "Direct" scenarios are used for comparison.

- **Edge-User-Side:** In this scenario, applications that perform text insertion, relay, and streaming are deployed on the user-side edge server and executed on virtual machines.
- **Edge-Robot-Side:** In this scenario, applications that perform text insertion, relay, and streaming are deployed on the robot-side edge server and executed on virtual machines.
- **DC-{Ohio, Singapore, Tokyo}:** In the three DC scenarios, applications that perform text insertion, relay, and streaming are deployed on AWS as a data center and executed on a virtual machine. For scenarios DC-Ohio, DC-Singapore and DC-Tokyo, we use AWS regions of Ohio, Singapore and Tokyo respectively. Ohio, Singapore and Tokyo are about 10800 km, 4900 km and 400 km away from Osaka respectively.
- **Non-Virtualized:** In this scenario, applications that perform text insertion, relay, and streaming are deployed on the NV server and executed directly on that server.
- **Direct:** In this scenario, no applications that perform text insertion, relay, or streaming are deployed. The video is directly

Table 1 Application-level Delay of Video Live Streaming in Each Scenario

Scenario	Application-level Delay [ms]		
	1st	2nd	Mean
Edge-User-Side	476.19	482.76	479.48
Edge-Robot-Side	482.76	500.99	491.88
DC-Ohio	745.18	761.13	753.15
DC-Singapore	617.68	641.64	629.66
DC-Tokyo	526.56	515.21	520.89
Non-Virtualized	467.35	477.93	472.64
Direct	419.54	430.84	425.19

sent from FFmpeg on the robot to FFplay on the user PC.

3.2 Measurement Results and Analysis

In the seven scenarios, we measure the application level delays of video live streaming twice in the way described in Section 3.1.1. The results of the two measurements are shown in Table1[1]. The results of the analysis is explained in the following subsections.

3.2.1 Delay due to Long Communication Distance

When using a data center, the increase of application-level delay cannot be explained by a simple increase of propagation delay in a large scale. For the scenarios DC-Ohio, DC-Singapore and DC-Tokyo, the differences from the application-level delay in Edge-User-Side are 273.67 ms, 150.18 ms and 41.41 ms respectively. On the other hand, the RTTs of ICMP packets from the user PC to the virtual machines on AWS are 174.94 ms, 68.32 ms and 12.83 ms for Ohio, Singapore and Tokyo regions, respectively. The RTT from the user PC to the edge server is about 0.5 ms, and the increase of application-level delay are obviously larger than the increases of RTTs. The cause of those unexpected increases of application-level delay is jitter, i.e. a variation in packet arriving intervals. Since it is difficult to make the intervals perfectly constant, the receiver buffers the packets and absorbs the variation. As the variation gets bigger, the receiver needs to buffer them for a longer time.

By newly measuring packet arriving intervals on the user PC and application-level delays, it is revealed that application-level delay increases almost in proportion to the coefficient of variation (C.V.) of packet arriving intervals in our configuration.

To summarize the above, the factor of delay increase in the case of long communication distance is divided into propagation delay and buffering time for absorbing jitter. Noted that the time taken for VPN processing is about 0.3 ms, measured by ICMP Ping, and there is almost no influence on the application-level delays.

3.2.2 Time for Real-time Processing

Comparing the the result in the scenario Non-virtualized with that of the scenario Direct, the difference in the application-level delays is 28.85 ms. Therefore, the result show that the time required for text insertion and streaming server processing is 47.45 ms. In addition, the delay of 194 ms for buffering occurs in the virtual machine on edge servers or AWS.

3.2.3 Increase of Delay due to Virtualization

Comparing the mean of the results in the two edge scenarios with the result of the scenario Non-Virtualized, the difference is 13.04 ms. Therefore, the results show that software operation in the virtualized environment increases application-level delay by 13.04 ms. We also measure the increase of server processing time due to software operation in the virtualized environment. Since it is difficult to directly measure server processing times, we use the difference in captured times between incoming and outgoing time packets of a server. As a result, the increase in server processing time due to virtualization is 4.00 ms.

3.2.4 Processing Time at End Devices

Although the scenario Direct is set to measure the processing time at the end devices, we also modified FFplay application on the user PC to output timestamps in order to investigate the details of the processing time. Analyzing the outputted timestamps, the delay occurred in FFplay is about 80 ms. Also, since the refresh rate of the display of the user PC is 60 Hz, a delay of up to 15 ms is likely to be occurred [7]. The camera and FFmpeg on the robot will generate delay of about 136 ms, that is the difference between the total processing time currently confirmed and the result of the scenario direct. FFmpeg modified to output timestamps shows that its processing time is 27.6 ms. The remaining 108 ms should be occurred in the camera of the robot, and then a delay of up to 33 ms is likely to be occurred just before capturing since the frame rate is 30 fps [7]. However, they cannot be confirmed because of the specification of the robot. Note that the measurement of the processing time by the timestamps can not completely clarify the processing time at the user PC. This is because it does not include the time until presentation timestamps (PTS), that is used for identifying data in FFplay, are extracted from arrived packets. In particular, it is difficult to measure the buffering time below the application layer.

3.3 Discussion for Low-latency Video Live Streaming

Based on the result in Section 3.2, we can obtain policies for service function reallocation aimed at reducing application-level delay of video live streaming service with real-time processing. The following three solutions are considered.

First, application-level delay can be reduced by deploying a service function of real-time processing on edge servers, that is a key idea of Multi-access Edge Computing. Processing at a data center can lead to penalties in the form of a large propagation delay due to geographical factors. In a new service with real-time processing, that delay significantly damages the user's QoE and is required to be reduced. In our experiment, a RTT of 12 to 170 ms occurred between the user PC and the data centers. Using a function on the edge can almost eliminate that delay.

Second, it is necessary to deal with jitter at network side rather

(1): The dates and times (UTC) of the measurements are 8:20 a.m. on May 31, 2018 and 5:10 a.m. on June 1, 2018 for Ohio region, 10:00 a.m. on May 31, 2018 for both measurements on Singapore region, and 4:00 a.m. on June 15, 2018 for both measurements on Tokyo region.

than at end devices. Conventionally, applications have buffered the packets and have absorbed jitter in order to ensured quality of service (QoS) and user's QoE. Contrary to that conventional circumstance, in new services which demand high real-time performance, the time for buffering will lead to degradation of QoS and user's QoE. Therefore, it is required to deal with jitter at the network side. In Section 3. 2. 1, the maximum increase of the application-level delays due to the buffering time for absorbing jitter was 144 ms. By reducing jitter, this delay is eliminated. In our experiment, the impact of jitter was clarified by the result of providing service at data center. If the cause of jitter is congestion of the route to the date center, the effect can be reduced by MEC. However, even using an edge server, there is a possibility of occurring jitter due to high load of the edge server. In Section 4., we devise a service function reallocation method based on the CPU load of edge server.

The third solution is to improve the performance of end devices. Cameras capable of taking high frame rate video and high refresh rate monitors can push the limits of application-level delay reduction. In addition, improving the processing performance of end devices leads to reduction of compression time and decompression time of MPEG videos. However, even if the performance of end devices improves, the delay reduction is on the order of a few tens of milliseconds, which is smaller than the reduction of propagation delay or buffering time for absorbing jitter.

4. Service Function Reallocation Method

4. 1 Service Function Reallocation Method

In this section, we focus on the increase of jitter caused by the increase of CPU load of the edge server. In our method, the orchestrator monitors the CPU utilization of the edge server as an index of CPU load and determine when start to service function reallocation in order to avoid increase of application-level delay of video live streaming. The reason why the orchestrator does not directly measure jitter is that it is difficult for the orchestrator to directly access the network statistics of the user's device in an actual MEC environment. In our method, service function reallocation is realized by live migration of virtual machine.

Our method only determines the timing that the orchestrator issues an instruction to start live migration. There are two factors to determine the beginning of live migration: CPU load at which application-level delay start to increase and the total migration time at that CPU load. We investigate the value of CPU utilization at which jitter and application-level delay start to increase. The CPU load of the edge server is given by using a stress tool called "stress-ng", and the packet arriving intervals and the application-level delays are measured when the CPU load is 0, 50, 80, 95 and 100 percent. The results show that jitter increases from when the CPU load exceeds 80% and the application-level delays start to increase from when the CPU load exceeds 95%. In addition, the total migration time is about 8 seconds when CPU load is 95%. From the results,

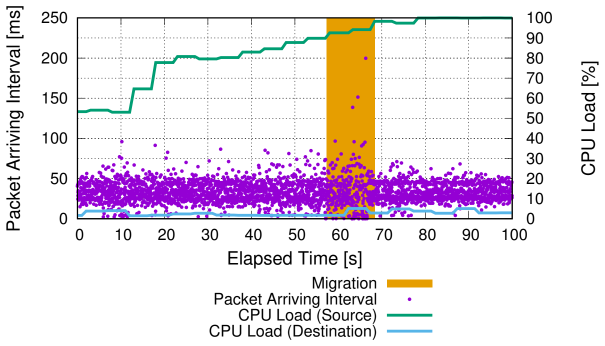


Figure 3 Packet Arriving Intervals with Live Migration

it is appropriate to start live migration at 8 seconds before the CPU load of source host reaches 95%.

4. 2 The Effect of Service Function Reallocation

To confirm the effect of the service function reallocation method on application-level delay, we develop a program running on the orchestrator. The program monitors the CPU utilization of the edge servers using Simple Network Management Protocol (SNMP). In our environment, data acquired by SNMP is updated every 5 seconds. The program also instructs KVM on the edge server to start live migration. The virtual machine that to be migrated is executing real-time text inserting processing using FFmpeg, and it is migrated from the edge server of the user side to that of the robot side. To simplify the situation, there is only that virtual machine. There are no virtual machines for relay or FFserver. As findings in Section 4. 1, the appropriate timing of start to live migration is about 8 seconds before the CPU load of source host reaches 95%. However, our method does not predict when the CPU load of edge servers reaches 95%. Therefore, it starts migration when the CPU load of the source reaches 90% under a condition that the CPU load is increased monotonically. The CPU load of the destination is fixed at 0%, since an edge server with low CPU load should be selected as the destination when reallocating a service function in an actual MEC environment.

Figure 3 and Figure 4 show the packet arriving intervals and the application-level delay when live migration is enabled. The migration is started when the CPU load reaches 90%. The packet arriving intervals are scattered before/during the migration, but they are calm after it. The application-level delay is stable overall, though it slightly increases after the CPU load reaches 80% and during the migration. Figure 5 and Figure 6 show the packet arriving intervals and the application-level delay when live migration is disabled. The packet arriving intervals are greatly scattered after the CPU load reaches 90%. The application-level delay also increases after the CPU load is over 85%. After the CPU load reaches 100%, the application-level delay increases significantly. As a result, it was possible to prevent the occurrence of large jitter and the increase of application-level delay in advance.

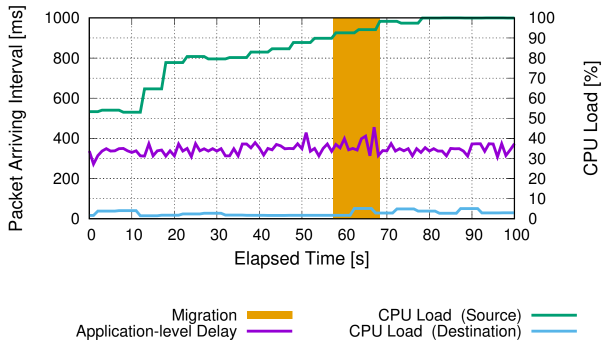


Figure 4 Application-level Delay with Live Migration

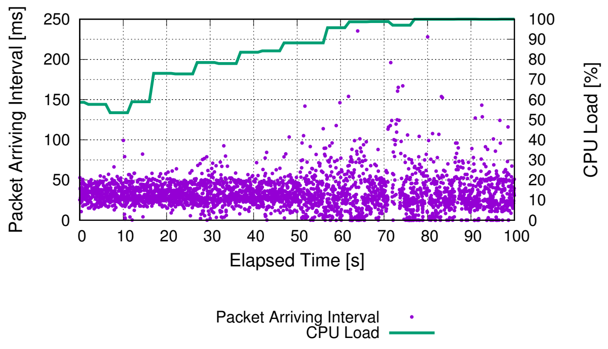


Figure 5 Packet Arriving Intervals without Live Migration

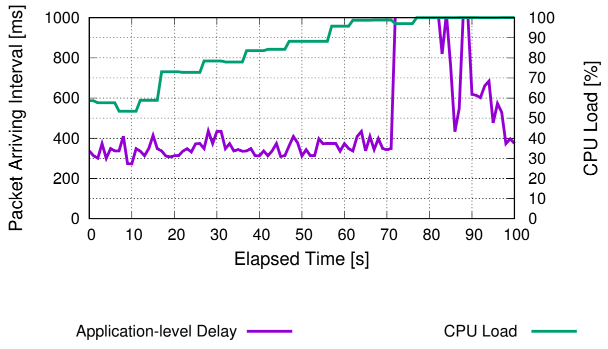


Figure 6 Application-level Delay without Live Migration

5. Conclusion

In this paper, we measured the application-level delays of video live streaming and analyze the delays in detail. As a result of the analysis, it was revealed that, for low-latency video live streaming, it is naturally important that the average of propagation delays is small, but it is also important to suppress the jitter caused by the network side. Based on this finding, we devised a service function reallocation method for low-latency video live streaming. In our method, we focused on the increases of jitter and application-level delay caused by the increase of CPU load, and the timing of service function reallocation was determined based on the CPU utilization of edge server. Also, we confirmed that our method prevent the occurrence of large jitter and the increase of application-level delay in advance. The work in our paper does not focus on QoE, but the measurement and analysis of application-level delay will contribute to understanding QoE in MEC environments, because QoE metrics include application-level delay [8]. As a future work, we will perform live migration in larger scales such as metropolitan area network and wide area network environments. As a further prospect, we evaluate the effect of application-level delay on the user's QoE in a two-way communication application. Moreover, we would like to evaluate the effect in a service providing non-visual information using tactile sensors.

Acknowledgment

This work was partially supported by Grant No. 19104 from the National Institute of Information and Communications Technology (NICT) in Japan.

References

[1] Z. Huang, W. Li, P. Hui, and C. Peylo, "CloudRidAR: A Cloud-based Architecture for Mobile Augmented Reality," in *Proceedings of ACM Workshop for Mobile Augmented Reality and Robotic Technology-based Systems*, Jun. 2014, pp. 29–34.

[2] "Mobile Edge Computing - A Key Technology Towards 5G," ETSI, Sep. 2015.

[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 Architecture & Orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, May 2017.

[4] "Mobile-Edge Computing (MEC); Framework and Reference Architecture," ETSI GS MEC 003 V1.1.1, Mar. 2016.

[5] Cloud Edge Computing: Beyond the Data Center. [Online]. Available: https://www.openstack.org/assets/edge/OpenStack-EdgeWhitepaper-v3-online.pdf

[6] SoftBank Robotics. [Online]. Available: https://www.ald.softbankrobotics.com/en/robots/pepper

[7] "Latency in Live Network Video Surveillance," AXIS Communications, 2015.

[8] S. Winkler and P. Mohandas, "The Evolution of Video Quality Measurement: FromPSNR to Hybrid Metrics," *IEEE Transactions on Broadcasting*, vol. 54, no. 3, pp. 660–668, Jun. 2008.