Design and Implementation of Integrated ICN and CDN as a Video Streaming Service

A Thesis Submitted to the Department of Computer Science and Communications Engineering,
the Graduate School of Fundamental Science and Engineering of Waseda University
in Partial Fulfillment of the Requirements for the Degree of Master of Engineering

Submission Date: July 22nd, 2019

YAN Chengkai
(5117FG18-9)
Advisor: Prof. Takuro Sato
Research Guidance: Research on Ubiquitous Communication System

Acknowledgement

First of all, I would like to express my deep gratitude to my supervisor Prof. Takuro Sato for his patient guidance and constant support through my research activities during my master's degree study. Then, I would like to give my thanks to Prof. Toshitaka Tsuda who has supported me on the study and research in the project "5G! Pagoda" and ICN.

Secondly, I would like to thank Mr. Xin Qi, Mr. Qiaozhi Hua and Dr. Zheng Wen with other members in SATO Lab, who have supported my research study and encouraged me at Waseda University; and for gave me effective advices and positive academic discussions.

Thirdly, I would like to give my sincere gratitude to all FSE faculty members, for their research and administrative supports.

I also would like to thank my parents; without their love and selfless support, it would have been difficult for me to finish my master study.

Special thanks to Dr. Ping Du and Mr. Hidenori Inouchi from NAKAO Lab at the University of Tokyo, and Mr. Daisuke Okabe from Hitachi Ltd. for the respectful collaboration and teamwork in the project "5G! Pagoda".

This research was partially funded by 5G! Pagoda project, which is funded by European Commission's H2020 program under grant agreement No.723172 and by the SCOPE project of MIC (Ministry of Internal Affairs and Communications) of Japan.

Abstract

In this research, we leverage the emerging concept of network slicing to enable the end-to-end integrated Information-Centric Networking (ICN) and Content Delivery Network (CDN) for 5th generation mobile networks (5G) networking infrastructure. While CDN is deployed to cache content at the optimal server corresponding to the content and geographical location, this paper aims to focus on verifying the efficiency of the ICN slice for the regional content distribution via the in-network caching and name-based forwarding scheme. Specifically, the ICN slice can be established by the Orchestrator by following the current Software-Defined Network (SDN) and Network Function Virtualization (NFV) standard. Then the slice stitching process will be performed to interconnect two slices after their establishments via the Orchestrator. We also implement an OpenStack-based virtual node which supports both IP and ICN protocols and acts as the ICN-Gateway. The joint testbed evaluations conducted between Japanese side (ICN slice) and European side (CDN slice) show that the deployment of ICN Gateway and the proposed Node ID-based ICN naming structure can improve network performance and avoid network congestion. Typically, by providing content as close as possible to the end-users in the ICN slice, content users in Japanese region can retrieve video contents from Finland with shorter response time, shorter download time, less E2E (End to End) hops and higher throughput.

Keywords: Information-Centric Networking (ICN), CDN (Content Delivery Network), network slicing, ICN/CDN, Video streaming, NFV/SDN.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

In this section, we introduce the literature review and the research purpose.

1.1 Literature Review

The technology of video streaming has been developing for a long time. According to Fig. 1, in the video streaming development's Phase 1 (1995 - 2004), for the first-time people can enjoy video contents via internet. Then in Phase 2, web browsers have become the tool for VoD (Video on Demand) service. In the meantime, RTMP (Real-Time Messaging Protocol) allow service providers using streaming protocol to expend the service range of video streaming. Also, due to the surge in users requesting for video contents, CDN and P2P technology came out as the distributed method to optimize the download time (on users' side) and traffic congestion problem. In recent years (Phase 3), more technologies came to increase video streaming's function and QoE such as RTSP (Real-Time Streaming Protocol), SDN/NFV (Software-Defined Networking/Network Functions Virtualization) and so on. So far, video streaming has taken an important role during people's life, and the related technologies are in constant revolution.

In this study, we have used current video streaming technology such as CDN and SDN/NFV, to meet needs from users' side.

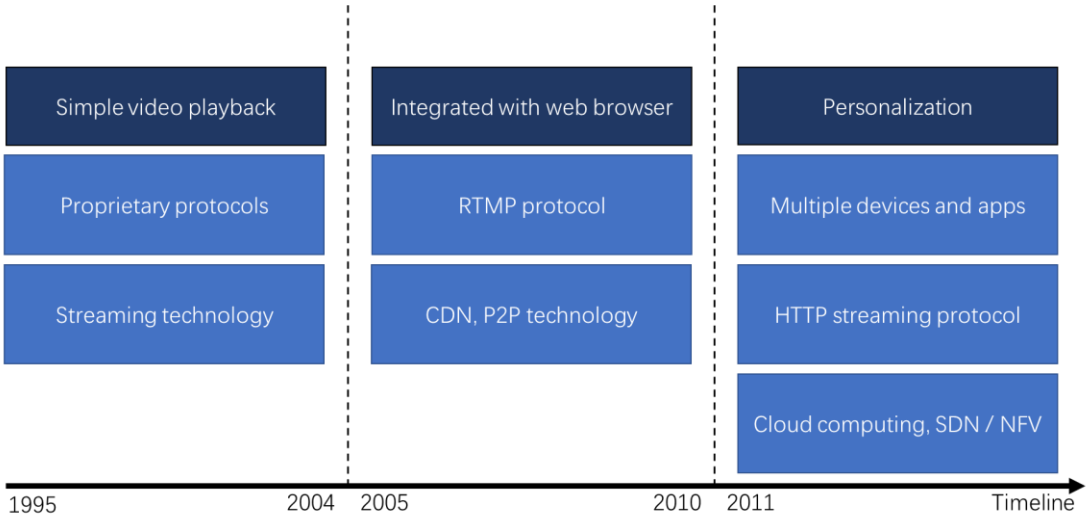

Fig. 1. The Development of Video Streaming

1.2 Research purpose

Nowadays, to match the huge demand for content distribution, we need a new network paradigm shift from IP-based services into information-based services. In this way, we can transfer information in a wide range of services efficiently, especially in the case of High-definition video transmission with high bitrate/speed requirement. In fact, video content has become a major part of the total Internet traffic, and mobile/wireless data traffic is a notable trend for content accesses in the future. According to Cisco's report (Fig. 2), IP video traffic occupied 75 percent of the whole Internet in 2017, and this number will increase to 82 percent by 2022 in which the mobile devices will carry 44 percent of the total IP data traffic [1]. Thus, ensuring mobile user's VoD (Video on Demand) experience is a key to realize efficient content the distribution model for the Internet.

In this context, ICN (Information-Centric Networking) [2] was proposed as a promising future network approach since 2005 [3]. The key features of ICN include in-network caching and using named data instead of IP addresses for forwarding and routing content. However, ICN still has implementation issues [4] because all the content nodes in ICN need to have the memory storage for content caching then make them consume more power compared to the IP routers [5]. Also, the default caching mechanism in ICN [6] produces high cache redundancy by wasting cache space for storing on-path content duplicates as analyzed in our prior work [7].

Currently, many video service providers have selected CDN (Content Delivery Network) as their solution for serving the vast video traffic. The idea of CDN is placing suitable dedicated caches as distributed servers at the edge of various network domains or geographical regions to reduce network load and response time. However, considering the high cost of the cache servers and the video resources, CDN still has the feasibility issue for deployment.

Thus, in this study, we have proposed to integrate CDN and ICN as a 5G network slice for efficient video streaming service which will be detailed in the next sections.

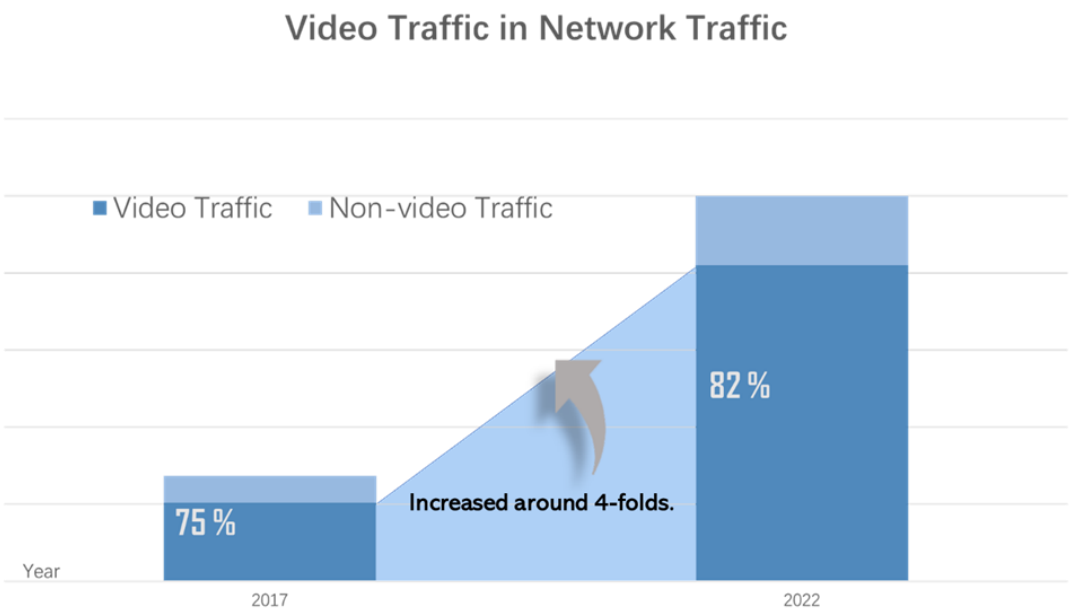

Fig. 2. Video Traffic in Network Traffic

Chapter 2 Related Work

In this section, we introduce the major concepts that are related to our proposal.

2.1 Content-Centric Networking (CCN)

Content-Centric Networking (CCN) [2] is a notable ICN platform that enables users to obtain desired content through its name instead of its location as defined in the existing IP-based Internet architecture. CCN is also implemented using ICN routers with caching function rather than IP-routers to realize efficient content dissemination.

In CCN, there are two types of the packet which are Interest and Data packets. Interests consist of a content name, which is requested by the consumer. Data packets carry content data and act as response for content requests, i.e., Interests. The data transmission unit in CCN is chunk, i.e., a content is split into a number of equally sized chunks.

For the content distribution process, firstly, Interests will be sent by the consumer. Then, the Interests will be broadcasted throughout the whole network. This step can be regarded as a searching strategy: Once a user sends an Interest, it will be sent to the nearest node to minimize the transmission time. Besides, users who are in the same area may express the Interests for the same content so that they can get the desired content from the suitable intermediate ICN nodes without the need of downloading it from the content provider (server).

For the data retrieval process (Fig. 3), the content name prefix will be searched in the cache memory of the CCN routers, called Content Store (CS). If Interest matches the prefix, data will be returned to the consumer via the corresponding face (network interface). Otherwise, Interest will continue finding the content in Pending Interest Table (PIT) [2]. If the required content exists, content data is sent back by the reverse path of Interest and a new entry of Interest will be added to PIT. If the content still cannot be found, that information will also be recorded in PIT. Then, Interest will be forwarded according to Forwarding Information Base (FIB) [2] for the forwarding procedure [8].

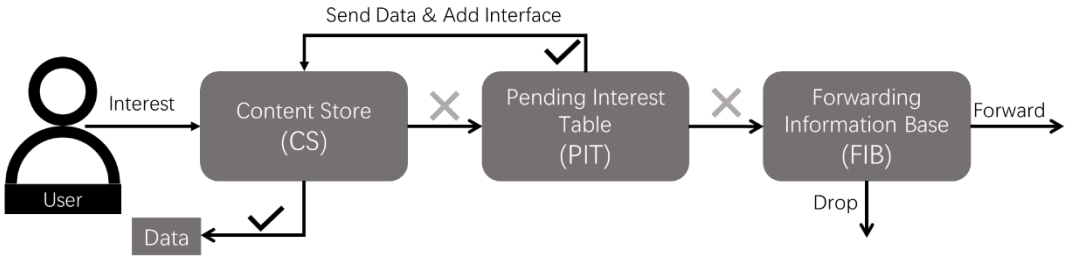

Fig. 3. Forwarding Stratagy in CCN

Besides the name-based forwarding strategy, another key feature of CCN is in-network caching. Different from TCP/IP design, CCN node has its cache memory so that it can store downloaded contents dynamically. In other words, once the content has been downloaded, it will be cached by the CCN node. Hence, the total content downloading time and E2E (End to End) hop count can be reduced.

It should be noted that in this project/study, in order to implement CCN protocol, we choose using CCNx software as the codebase (Fig. 4) to emulate CCN's caching and naming feature above TCP/IP network. By using this emulator, we have the basic condition/environment to demonstrate and evaluate our proposed system in ICN part.

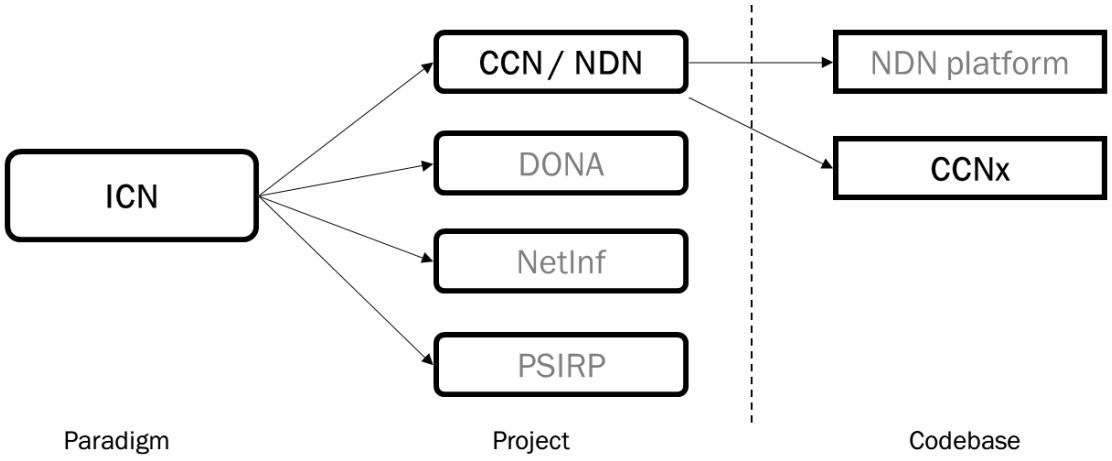

Fig. 4. Projects in ICN

2.2 Content Delivery Network (CDN)

Recently, CDN has been widely implemented to serve the content-based services, represented by the well-known CDN operators, e.g., Netflix, Akamai. CDN deploys edge servers which contain contents from the original content producers/servers. In this way, the data traffic of the original server can be split to multiple mirror cache servers, which leads to relieving the burden of the source server. Additionally, based on users'

geometric information (such as IP addresses), the DNS (Domain Name System) server can identify which edge server is the nearest one to users. Thus, it can allocate the most appropriate server to the user and the downloading time on the user side can be reduced considerably. Hence, the users' QoE (Quality of Experience) can be improved as well. Also, CDN users from different ISPs (Internet Service Provider) and regions can gain similar bitrate experience [9].

However, CDN still has its disadvantages. Firstly, due to the high cost of deploying CDN mirror edge servers, CDN users are requested to pay an additional fee for the premier services. Next, the content updating process (i.e., pushing new video contents to the edge servers) takes time and may not be suitable for every scenario. In general, CDN is primarily used for VoD (Video on Demand) or downloading services. However, in some specific scenarios, users do not need all the cached contents then some of the valuable cache memory is being wasted. Thirdly, though deploying CDN servers can reduce and separate data traffic from the core server, bottleneck sometimes can still happen.

Specifically, when users' requests become huge in one specific area, the data traffic between users and this area's CDN edge cache servers can cause the network bottleneck. In order to prevent this situation, the service providers and CDN operator should increase the number of available CDN caches, which might lead to a higher cost.

2.3 SDN/NFV (Software-Defined Networking/Network Functions Virtualization)

Due to the increasingly diverse need from the user side, ISPs are currently in need of adaptive services to meet users' various demands. However, different from allocating a single service, customizing multiple types of service via physical network resource configuration is challenging. For example, nowadays, FHD (Full High-Definition) or even UHD (Ultra High-Definition) video streaming service and VoIP (Voice over IP) service are among the most popular applications of network service, but they have different optimized network resource configurations. Particularly, HD/UHD video streaming service requires high bandwidth and throughput while VoIP service requires

a stable network environment and low-latency. Moreover, it takes time and a high cost to set up and tune each network service. The introduction of NFV (Network Functions Virtualization) and SDN (Software-Defined Networking) in recent years then aims to satisfy the strong demand of "dynamic network configuration".

NFV uses virtualization technology to split network applications in a simple and adaptive manner. A general NFV architecture usually includes VNFs (Virtualized Network Functions), Hardware resources, Virtualization Layer, NFVI (NFV Infrastructure), NFV Management and Orchestration (NFVM and NFVO). Specifically, NFVI is the key to manage the hardware resources and change the physical hardware into virtualization resources pool so that the computing components can be managed flexibly and conveniently. The VNFs can install and provide service applications. Also, the VNFM and NFVO are responsible for managing and orchestrating the NFV's whole resources and processes [10].

As NFV offers the solution for making physical network resource virtual, SDN is the other technology needed for link virtualization. The most crucial feature of SDN is that it separates the network connection into control-plane and data-plane. By separating the two planes, network control becomes more convenient and flexible since control-plane requires flexibility whereas data-plane requires low-latency. For instance, the processes such as switching routing protocols and generating routing table can be implemented on one control-plane. To complete the separation, a protocol named OpenFlow would be used [11].

2.4 Network Slicing

Network Slicing is a virtualization technology based on NFV/SDN. It enables running multiple logical networks on one common shared-physical network infrastructure to maximize the flexibility and maintainability (Fig. 5). Besides, the network resources are split dynamically so that each logical network's parameter such as capacity, bandwidth, and delay can be customized corresponding to the user requests and network status.

Network slicing plays an essential role as a key concept in the upcoming 5G

network to realize various high-speed network applications, e.g., IoT (Internet of Things) and 8K streaming services [12]. Particularly, the ISPs could manage and customize each network slice simply and comfortably by defining each network service as one dynamic network slice and separate them logically [13].

Therefore, ISP's services in general 5G network should be separated into 3 major parts, which are eMBB (Enhanced Mobile Broadband), uRLLC (Ultra-reliable and Low-latency Communications) and mMTC (Massive Machine Type Communications) as shown in Fig. 5. Commonly, eMBB refers to services that should occupy high bandwidth and large compacity, such as 8K video streaming or even VR (Virtual Reality) in high FPS (frame per second) and high resolution. Thereafter, currently uRLLC is seen as the most popular services type in future mobile network. By shorten the transmission's failure rate and latency, more services which require those features are possible to be realized. For example, remote control and autonomous driving may become common services in the future due to the 5G network's low latency. Finally, mMTC can broaden devices in mobile network. This feature can make enormous IoT and cell phone devices operating normally in the same network. This study concentrates on the design and the implement of ICN/CDN video streaming service as a type of eMBB slicing.

In this research, we apply the emerging concepts of ICN, CDN, and SDN/NFV technologies to realize a network slicing design for efficient content delivery over different multiple geographical locations.

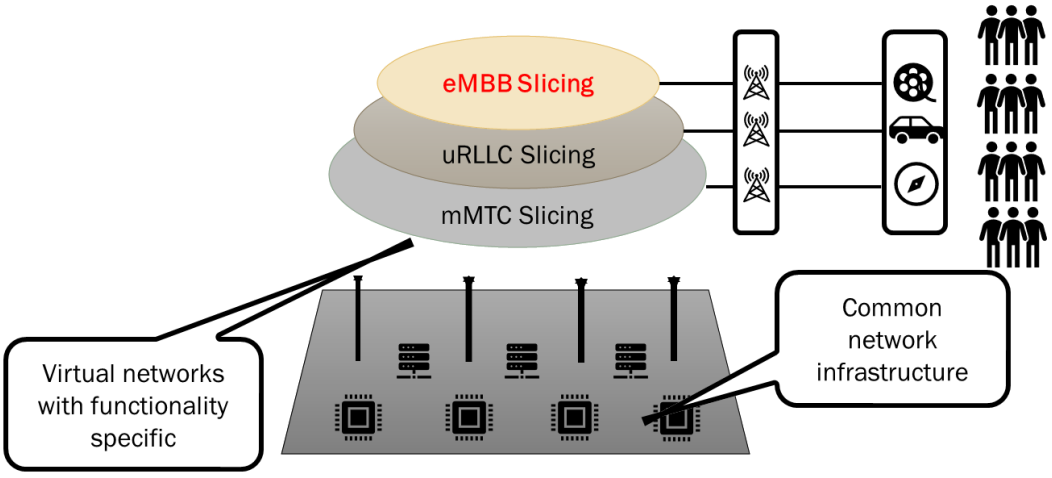

Fig. 5. Network Slicing

Chapter 3 System Design

In this section, we introduce the concept and mechanism of the proposed CDN/ICN content delivery system.

3.1 The Proposed ICN/CDN System

In this research, we combine CDN with ICN to enable an efficient contents delivery network system with low congestion rate. Besides, to meet the future mobile network's needs, we have also proposed using our system as one 5G service slice in the context of the project "5G! Pagoda", a collaborative Europe-Japan research project for softwarized 5G network evolutions [14].

The benefits of integrating ICN and CDN are three-fold as follows:

Firstly, using CDN and ICN can drastically reduce the congestion ratio of the whole network. Particularly, as aforementioned in Section 2, CDN can reduce the data traffic of the core/original server by deploying multiple edge mirror cache servers in various regions. However, when the number of users becomes large enough in one region, the congestion would occur at the edge servers. Therefore, by adding ICN nodes linked to the CDN edge servers, we aim to substantially diminish data traffics of CDN edge cache servers, thanks to the ICN's in-network caching feature [15].

The second benefit is increasing the users' QoE (delay time), especially for contents with high popularity level. By using CDN's cache server for dynamic and optimized content allocations, the download time for the requested content can be reduced considerably [16]. Specifically, this improvement is realized by the efficient retrieval process from the CDN slice to the ICN slice via the appropriate CDN cache and the ICN Gateway.

Additionally, as ICN is a potential future network design at the initial deployment stage and CDN has been a successful content-based business model, by combining CDN and ICN, we can take advantage the merits of both networking models: CDN is used for optimal content allocations at suitable CDN caches nearby the clients while ICN is used for quickly distribute content to users via the ICN nodes with the built-in

dynamic in-network caching feature.

3.2 System Overview

To show that our ICN/CDN system can provide an efficient and realistic video delivery model in real-world, we configure the whole system across different continents. Specifically, the system has been implemented to transfer content objects (videos) published in Finland (Europe EU) to Japan (JP, Asia). In this way, we are expecting to model and realize a promising and practical video streaming system deployment, e.g., Netflix or YouTube videos.

Fig. 6 demonstrates the integrated ICN/CDN system configuration. In our design, the system includes two major parts which are CDN slice for contents provider side ("EU Side") and ICN slice for content distribution to users ("Japanese Side"). Typically, on the EU region, the original CDN content server deployed at Aalto University, Finland (Aalto server) has been set up to publish video contents. Also, we assume that users will request their desired video contents from the Japan region. Thus, CDN cache mirror server and ICN nodes are deployed on the Japan side.

Since our whole work is to realize the 5G network slicing concept, each network slice is implemented based on SDN/NFV technology. It is also necessary to define a regional Orchestrator to manage every VNF instance dynamically. Typically, the EU Orchestrator is responsible for managing each instance and resource pool on the EU side while Aalto servers represent the CDN publishers/providers.

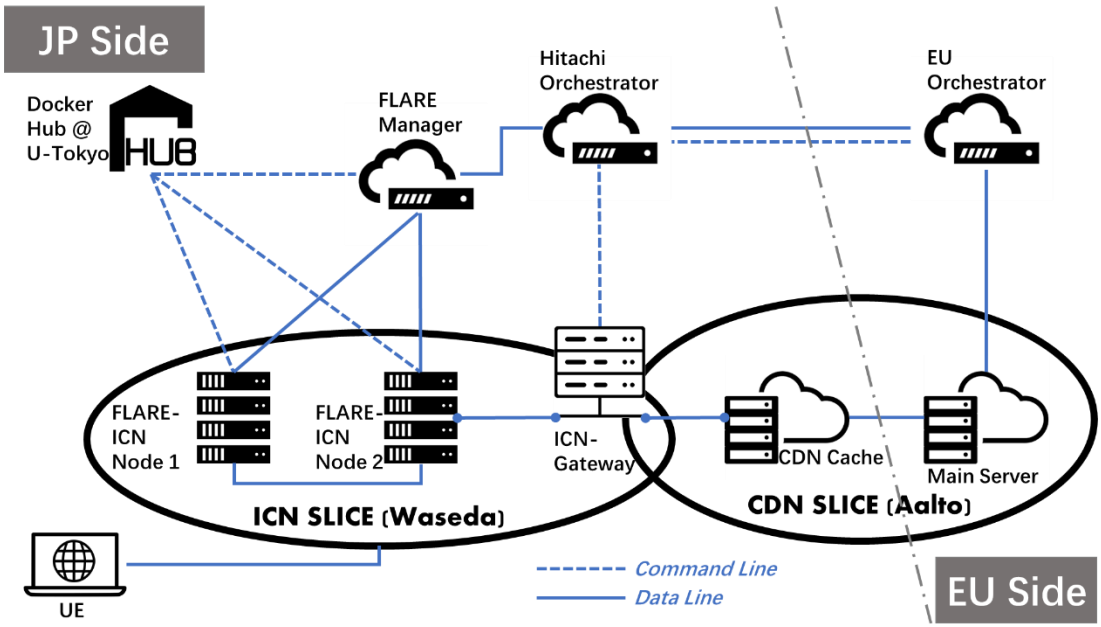

Fig. 6. ICN/CDN System Configuration

The ICN slice configuration is shown in Fig. 7 in which ICN nodes are implemented by JP Orchestrator (Hitachi Orchestrator) using CCN platform on the Japanese side. Firstly, the video contents are stored in the ICN Gateway (an OpenStack based CDN/ICN edge video cache server on the Japan side). Then, we add ICN Gateway FIB entries to CCN nodes for the efficient forwarding process in ICN. Hence, the video content information from the ICN cache can be shared through the Japan domain via the CCN nodes in ICN slice. To reduce the network load and congestion, we use multiple ICN-enabled edge nodes (with transient caches and substantial lower cache storage compared to CDN caches) to separate the content traffic of the ICN Gateway from the other ICN intermediate nodes.

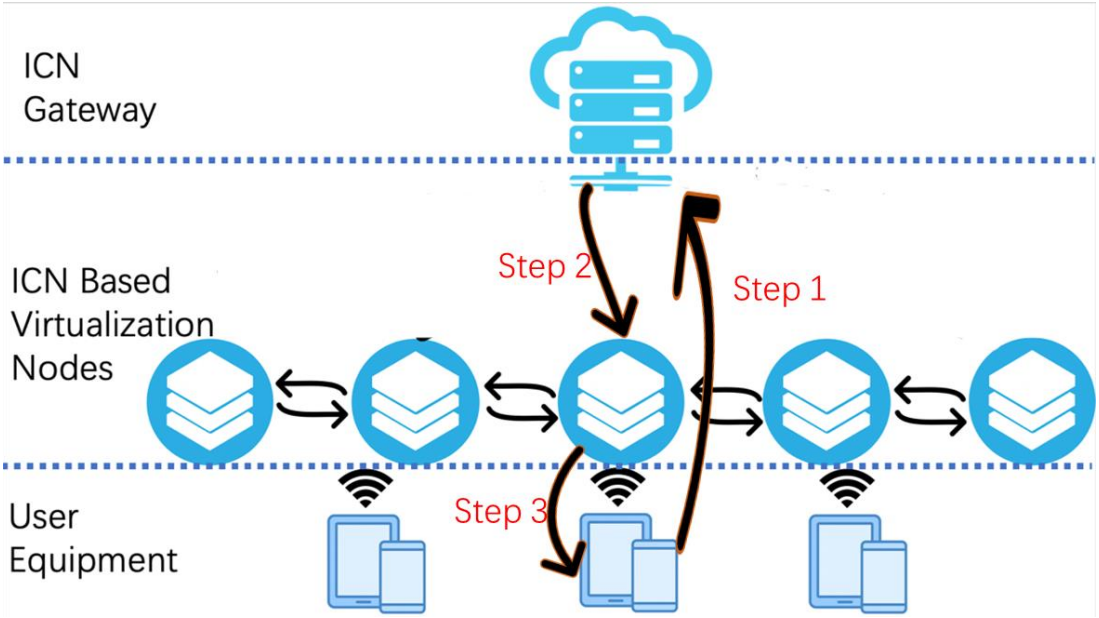

Fig. 7. ICN Slice Configuration

3.3 Virtual Fundamentation of the System

In this part, we will introduce tools that have been used in this study to setup the research environment.

3.3.1 OpenStack

In order to satisfy the SDN/NFV standard, we chose using OpenStack as the IaaS (Infrastructure as a Service) platform of the ICN-Gateway and CDNaaS (CDN as a service, developed by MOSAIC lab at Aalto Univ.) on the Japan side. The merits of using OpenStack are its flexibility and cost. Flexibility means that users are able to establish their own server environment based on their needs, such as how many vCPU they are planning to use or how many GB storage they will need. Thus, with IaaS technology such as OpenStack, a server environment can be customized based on users' requirements, and the computing source can be saved as well. Speaking of cost, usually service providers should set up the hardware machine. However, if service providers need a large number of servers, it will be quite costly, especially considering the price for server machines and space. By using IaaS technology, this high cost can be avoided.

During this project/study, the detailed hardware configuration of our OpenStack system is shown in Table 1 and Table 2. It should be noticed that due to the OpenStack

requirement of at least two hardware machines (one controller node, one compute node), we have listed two tables to present the configuration.

Table 1. Configuration of the controller node

	Controller Node
CPU	Intel(R) Xeon(R) CPU E3-1240 v3 @ 3.40 GHz
Memory (RAM)	16 GB
Storage (ROM)	HDD 2TB

Table 2. Configuration of the compute node

	Compute Node
CPU	Intel(R) Xeon(R) CPU E3-1245 v3 @ 3.40 GHz
Memory (RAM)	32 GB
Storage (ROM)	HDD 2TB

3.3.2 Docker/Containerization

In recent years, Docker has been standing out as a popular containerization technology in SDN/NFV field. Usually, when users aim to configure a test or staging environment with different OS (operating system), a traditional way is to install a VM (virtual machine) software such as VirtualBox or VMware etc. However, if the test environment or server does not require GUI function (command line only). Containerization is a wise choice because it has lower cost and lighter system requirements. Docker is widely being used as one type of the containerization solution. The biggest reason why Docker is being widely used is its light resource cost and development support. The second point (development support) is that every service can be saved as an image, so that when the service (server) is needed, simply by downloading the image and build it as containers, users can establish their test environment/server in a few seconds. This speed can show more advantages, especially when service in large scale. Thus, Docker is a reasonable choice in slicing technology, since slicing technology requires fast service booting speed and low computing resource cost.

In “5G! Pagoda” project, the Docker image in ICN slice has been built with CCNx software and related tools based on Ubuntu 14.04 environment.

3.3.3 VirtualBox

VirtualBox is a VM software which is free and open source. It is able to create x86 Guest OS on the mainstream Host OS (Windows, Mac OS etc.) using Oracle's xVM virtual platform. Though the system performance is not as smooth as other VM tools, the compatibility of VirtualBox is superior.

In the system design, since our UE's hardware is operated with Windows OS, and our preferred OS should based on Linux OS (with GUI), I decide on using VirtualBox (VM tool) as our "UE" to steaming video contents.

3.4 End-to-End (E2E) Content Delivery

In this part, we briefly present a complete E2E content delivery procedure of the proposed ICN/CDN system. It contains four major steps which are slice establishment, slice stitching, content request, and content delivery.

Firstly, in the "slice establishment" stage, all the CDN and ICN NFV instances are initiated and allocated virtually. During this stage, both JP and EU' Orchestrators will create and configure each instance with a suitable virtual configuration dynamically.

Then, "slice stitching" stage will be executed. After ICN and CDN instances have been established, JP Orchestrator will inform ICN Coordinator and provide the Coordinator the FIB entries of ICN Gateway so that ICN nodes can determine the suitable routing path. After the ICN nodes add the FIB entries of ICN gateway, the slice stitching process is completed.

Next, the user sends the content request, and the content delivery process will be performed. Upon the slice stitching's completion and the video service is triggered at the CDN slice, the user receives the table of contents which lists all of the available video contents' "exact name" (which will be detailed in Section 3.4) with resolution, video name, bitrate and the corresponding CDN cache from CDN coordinator. The user then can choose which video they want to watch. When the user selects the desired video content, the system will generate an ICN Interest to ICN Gateway to check whether this content is already cached in ICN slice or not. If it has been cached, then the target content will be renamed as CCN "exact name" format and sent back to the

User Equipment (UE). When the UE receives the exact content name and acknowledge it, the content transmission can start. Otherwise, if the user selects the desired content and the ICN Gateway does not have the content that the user asked, this Interest will be converted as a content request to the suitable CDN server with respective content. The CDN server then pushes the requested content to ICN Gateway. In this way, users can receive the content in the ICN slice when the same content Interests are received again. Besides, if users require enjoy video contents via the ICN/CDN video streaming system, I also designed a Python based program to show items of video contents. Therefore, users are able to receive contents by click video items via GUI, which can provide a better user experience.

3.5 Naming Strategy in ICN

One key feature of ICN is that its information route is based on the content name instead of an IP address. Specifically, each ICN content has its own unique ICN name without the need of name resolution via the DNS (Domain Name Server) system as of the TCP/IP architecture. However, since in our ICN/CDN video streaming system, both ICN (CCN platform) and IP (CDN) are co-existed, the suitable way to transfer and convert the content name format from IP to CCN is worth to be considered so that UEs in Japan side can receive the desired data efficiently via the Gateway in the ICN slice [17].

Firstly, the initial content name on the CDN side consists of an article name, resolution, bitrate, and video package format. For example, on the Aalto CDN server, a Demo video content is named as "*Demo-1920*1080-3000kbps.avi*".

Normally, a CCN name should contain two base parts which are "Content Source" and "Pattern Parameter". However, since we want to let the user know which ICN node is involved in serving a specific content, we decide to implement another naming format by adding a "Node ID". "Node ID" represents the caching node with the requested content name for content distribution in ICN from CDN slice. Thus, in the simplified case, the "Node ID" will be "ICN Gateway". Besides, we add the content source with location before "Node ID" (left-most position) in the content naming structure. As

"Finland, EU" (Aalto University) is the content publisher's location in our system design, and refers to Fig. 8, the CCN name should be "*/ICN-Gateway/EU/Finland/Aalto-University/Demo-1920*1080-3000kbps.avi*". By using this naming format, the user can be aware of the content transmission so that they can decide whether to receive the content or not. Since the naming format in CCN is Longest-Prefix Match (LPM) for forwarding and routing procedure, we define the proposed CCN name structure as the content "exact name".

Another advantage of putting the Node ID in front of the exact name is to increase the interest-matching possibility. By signing the Node ID to the exact name, as long as Japan users' requests (interests) start with "/<*Node_ID*>", because of LPM forwarding strategy in CCN, it is possible to increase the interests' hitting rate.

Lastly, as for the exact name's design, Node ID (or routable-prefix) can be considered as a "interest direction identifier" in one specific area. Node ID stands for a probable direction of contents retrieval path. Contents' publisher ("ICN-slice owner" in this case) can specify the "interest direction", that it is considered as a reasonably short data-path for users. Moreover, if the requested content has been cached once by the ICN slice, users are able to get faster download time and shorter latency. In this way, the QoE can be increased based on this naming strategy.

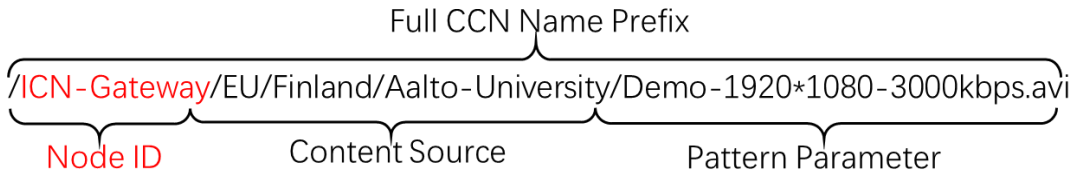

Fig. 8. Naming Structure in CCN Name Format

3.6 ICN Gateway

As shown in Fig. 6 and Fig. 7, between the ICN slice and the CDN slice, we have deployed an additional OpenStack-based node and named it as "ICN-Gateway". The ICN-Gateway enables both CCN and TCP/IP protocol so that the video content from the CDN cache server can be cached inside the CCN's repository. From ICN-Gateway, cached contents would be stored at ICN intermediate nodes then transfer to users for minimizing the latency of subsequent content accesses.

Meanwhile, ICN-Gateway takes responsibility for converting content name from CDN naming format into the proposed CCN's "exact name".

3.7 Demonstration of the Testbed's Environment

As a part in Project "5G! Pagoda", moving Waseda's FLARE environment to U-Tokyo Testbed is necessary for the main demo. Since we have uploaded the ICN slice's docker image on the private Dockerhub, by simply pulling the ICN Docker image, the time of building ICN slice in the testbed can be minimized, especially compared with building a new docker container.

In the demo, as shown in Fig. 9, a similar Waseda ICN slice has been built at the testbed (with FLARE manager and Hitachi Orchestrator's collaboration). With FIB entry's setup and slice stitching (with CDN slice), our testbed's environment preparation has been completed of ICN slice's creation and stitching. The video contents retrieve path is explained in the next paragraph.

Firstly, as it is shown as dotted green arrow in the Fig, from Japanese side, we expect video contents has been stored inside ICN-Gateway. Then, contents are transferred to FLARE1. Thirdly, contents should go through and cached in FLARE2. Finally, UE could enjoy video contents with Internet connection. Thus, according to the test from the UE side, we can enable VoD service on the testbed.

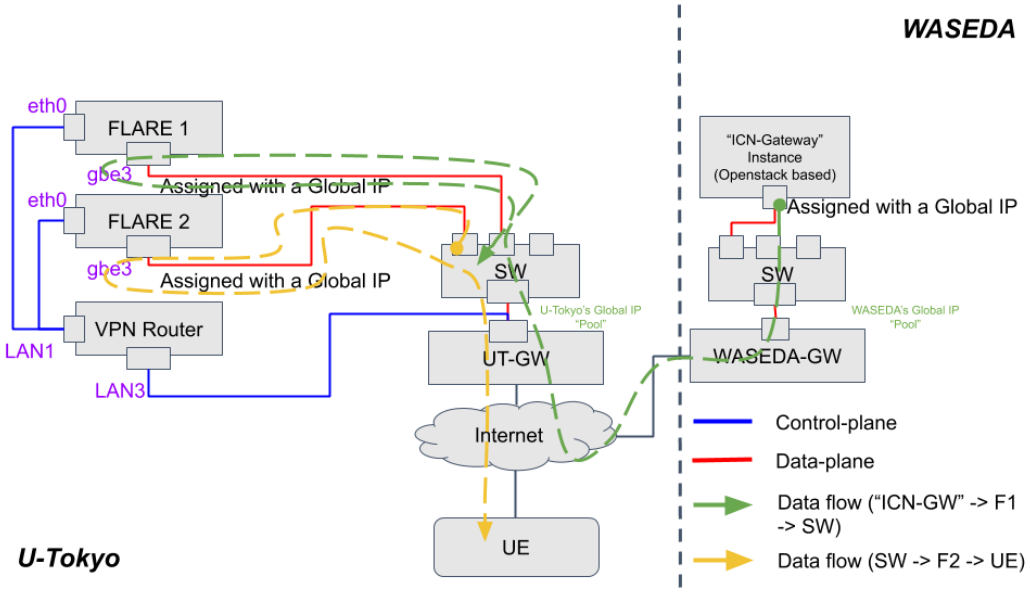

Fig. 9. ICN Slice's Network Topology of the Testbed

Chapter 4 System Evaluations

In this part, we design the test environment and evaluate our system's performance.

4.1 FLARE-based ICN Nodes

Regarding the proposed ICN/CDN system implementation, we build the joint test-bed based E2E content delivery at Waseda University in which the virtual content nodes image and configuration setting are installed in deeply programmable nodes, namely FLARE, developed by the University of Tokyo. We select FLARE as it enables an Open Deeply Programmable Switch/Network Node Architecture to verify the merits of the proposal over multi-domain test-bed with multi-core processors toward 5G slicing [18]. As shown in Fig. 10 (refers to red line as "ssh" control line and blue line for transmitting data between FLAREs), FLARE also realizes resource isolation with lightweight control plane and data plane programmability. We then implemented ICN Based Virtualization nodes on FLARE, and the hardware configuration of FLARE is shown in Table 3. Specifically, we use Docker as the container technology to implement ICN nodes' virtualization over FLAREs.

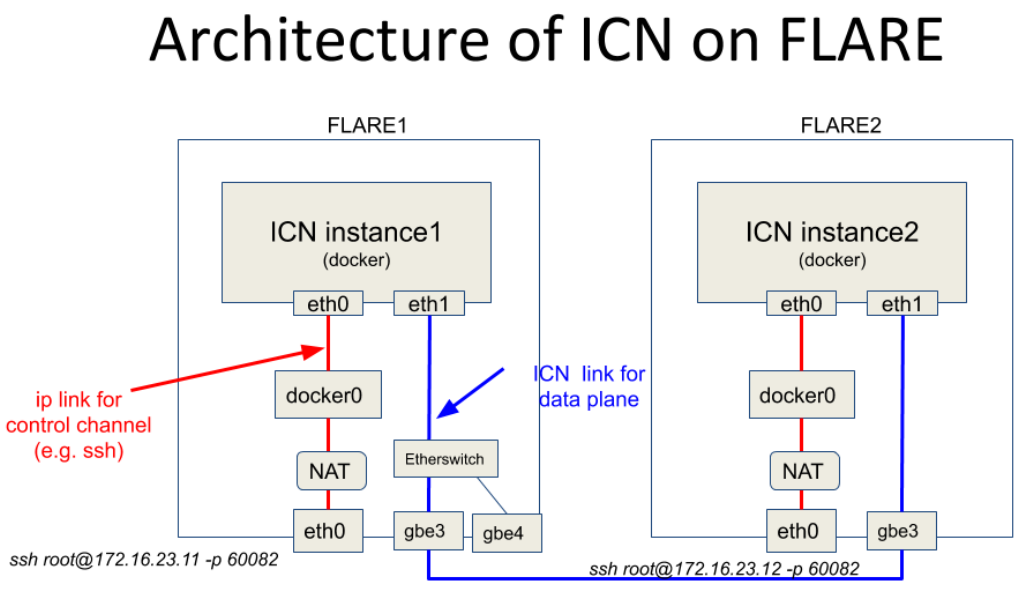

Fig. 10. Architecture of ICN Slices on FLARE

Also, since Hitachi. Ltd. acts as the Orchestrator of the Japan domain (Fig. 6), ICN nodes can be established and managed dynamically so that the ICN slice follows the network slicing standardization [19].

Table 3. FLARE's detailed configuration.

Parameter	Value
Spec	72 core EZ-Chip Network processor
SFP+	2 ports, up to 128GB memory / 1TB SSD
GbE	24 ports and 10 GbE
Power	Redundant Power supply

4.2 The Proposed System Configuration

Note that in this research, we focus on the ICN Slice design for content distribution when the content objects are already stored at the ICN Gateway from the CDN Slice, i.e., suppose that Optimal VNFs Placement in CDN Slicing over Multi-domain is already performed. Then, this paper is different from our prior work in the same research theme which presented the overall integrated ICN/CDN system design [16].

For the experiment evaluation, we set up an ICN slice configuration as shown in Fig. 11. Particularly, at first, an OpenStack based ICN-Gateway caches the test video content. Then, upon receiving the message from Orchestrator (Hitachi Orchestrator), the ICN coordinator can receive the FIB entry of ICN-Gateway. By using this information, FLARE-ICN Node 1 can be set up and make a connection with ICN-Gateway in CCN protocol (slice stitching procedure). Next, two ICN nodes are connected, and finally, on the UE side, UE 1 connect with ICN Node 1 while UE 2 makes a connection with ICN Node 2 via the CCN protocol (followed by orange arrows in Fig. 11). Also, it should be noted that, UE 1 and UE 2 do not stand for two single devices, but two type of UEs in two different areas (UEs in area 1 and UEs in area 2).We use this system configuration as the evaluation scenario model to verify the benefit of using CCN nodes with in-network caching feature so that the video contents with high popularity in a geographical domain can be transmitted to the user side efficiently with minimized response time [20].

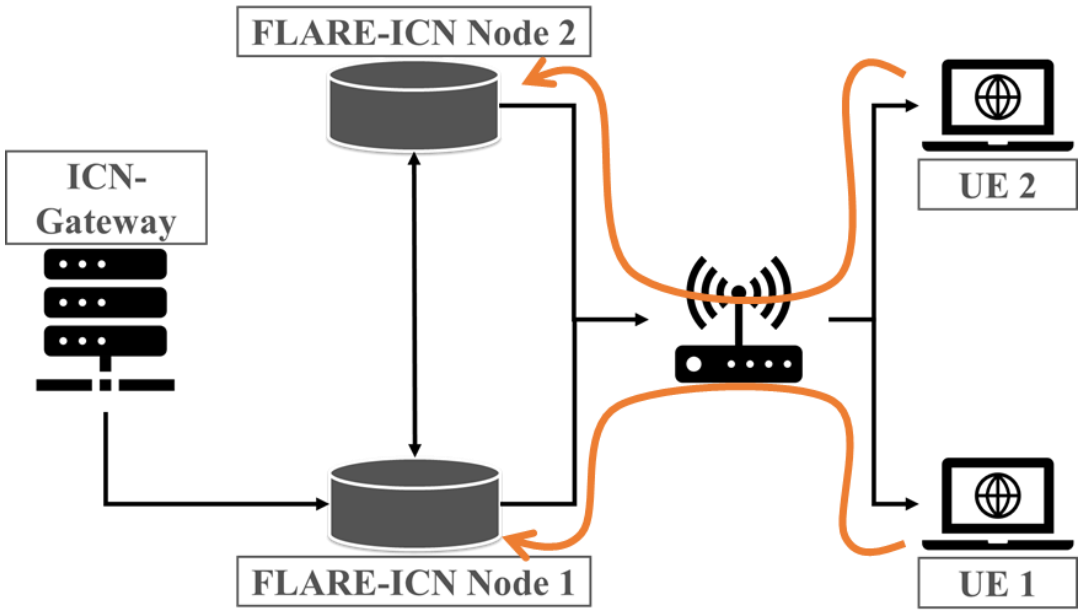

Fig. 11. FLARE-based Testbed Configuration

4.3 Test Scenarios

After the slice stitching procedure is completed by the Orchestrator at the Japan side via information exchanges from Gateway at ICN slice, we perform the test scenarios using the above testbed configuration. In particular, we conduct four different test scenarios to verify the efficiency of the proposed ICN/CDN system for content distribution in which the content delivery is conducted twice for each scenario as follows:

- *Scenario 1*: Firstly, the content request is sent from UE1. Then, for the second time, the content request (for the same content) is also sent by UE1.
- *Scenario 2*: First content request is sent from UE2, and a second-time request is from UE1 for the same content.
- *Scenario 3*: The first-time request is from UE1, and then UE2 will send Interest for the same content.
- *Scenario 4*: Both requests are sent by UE2.

It should be noted that the test content file in each Scenario has the same size (either 1 MB or 10 MB). Then, we have evaluated ICN slice performance by measuring

downloading time, E2E hops count, throughput, and Round-Trip Time (RTT) [21] which will be detailed in the next subsection.

4.4 The Integrated ICN/CDN System Performance Evaluations and Discussion

The four above network metrics are evaluated as follows:

4.4.1 Downloading Time

Downloading time means spending time since a user sends the first content Interest until the requested file's last chunk is transmitted to the user. Shorter download time refers to a higher transmission rate. As shown in Fig. 12 and Fig. 13, we have measured downloading time using the 1 MB and 10 MB sized file and, in both cases, the second time request's download time is much smaller than the first time. Thus, as long as the content has been cached by ICN nodes once (i.e., the requested content is stored in ICN slice), the buffering time for streaming service on the user side can be reduced considerably. As a result, QoE can be ensured, especially in the case of popular content [22].

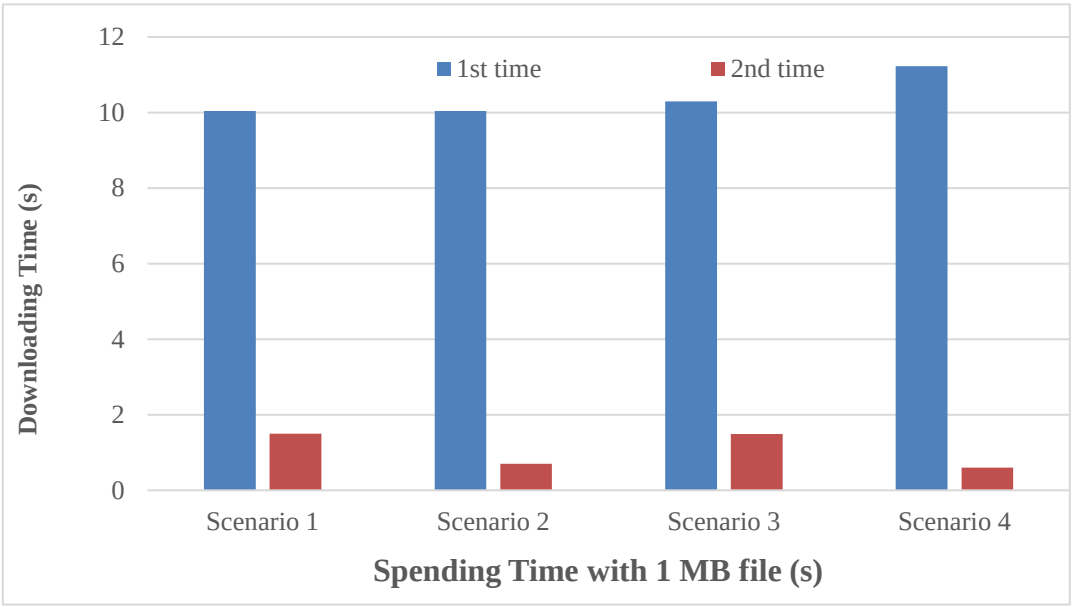

Fig. 12. Download time with 1 MB file

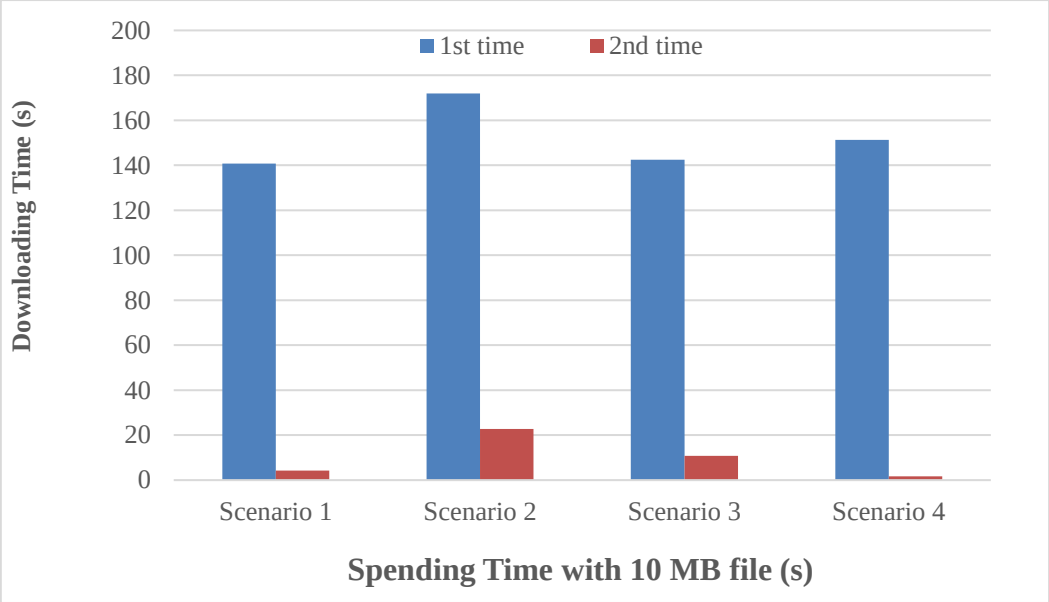

Fig. 13. Download time with 10 MB file

4.4.2 E2E Hop Counts

Similarly, when measuring the number of hops counts between UEs and the content source, we realize that the second time request's hops are always less than that of the first time (Fig. 14). The reason is that appreciates to the merits of in-network caching feature in ICN, for all the four test scenarios, the contents will be cached at the nearest ICN nodes after the first request.

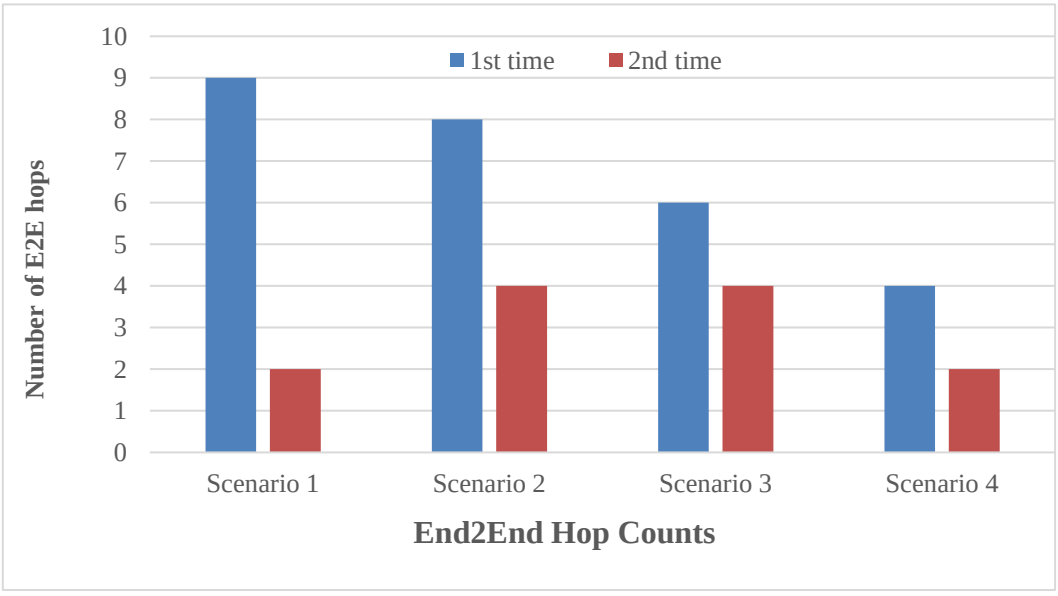

Fig. 14. End-to-End Hop Counts

4.4.3 Round-trip Time (RTT)

RTT measures the period since a packet is sent until it is responded. As shown in Fig. 15, RTT gets smaller after the first Interest when we test a content file in CCN's default chunk size (4 KB). As the requested content is stored at the nearest nodes (ICN node 1 or ICN node 2) after the first request, the subsequent requests for the same content become smoother, i.e., a reduced RTT shows better QoE on the user side.

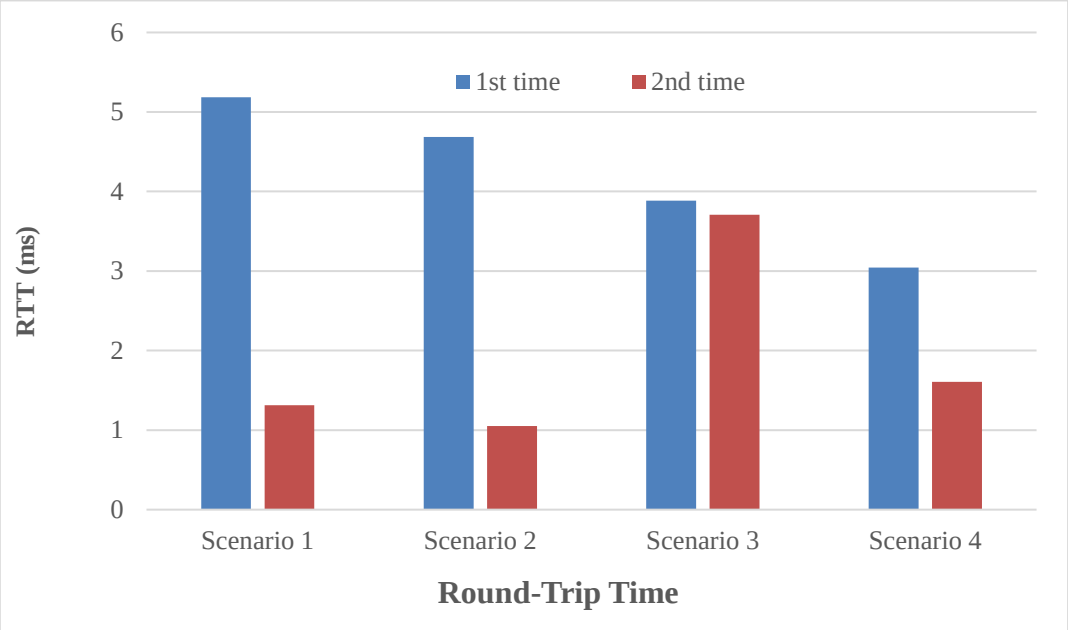

Fig. 15. Round-Trip Time

4.4.4 Throughput

Throughput is a key performance of the network, and the same tendency can be realized when measuring throughput in both cases of 1 MB and 10 MB test contents (Fig. 16 and Fig. 17). Specifically, in scenario 2 and scenario 3, the second time requests always get higher throughput compared to the first time. However, in scenario 1 and scenario 4, the throughput performance is decreased. The reason is that since our UEs are also equipped with CCN protocol, UEs will cache content into their repository with the built-in in-network caching feature as long as they retrieve the content once. This result explains why when users send the same Interest as the first time, they do not have a high throughput via their network interfaces. This deployment then also leads to less heavy data traffics for a stable network with low congestion rate.

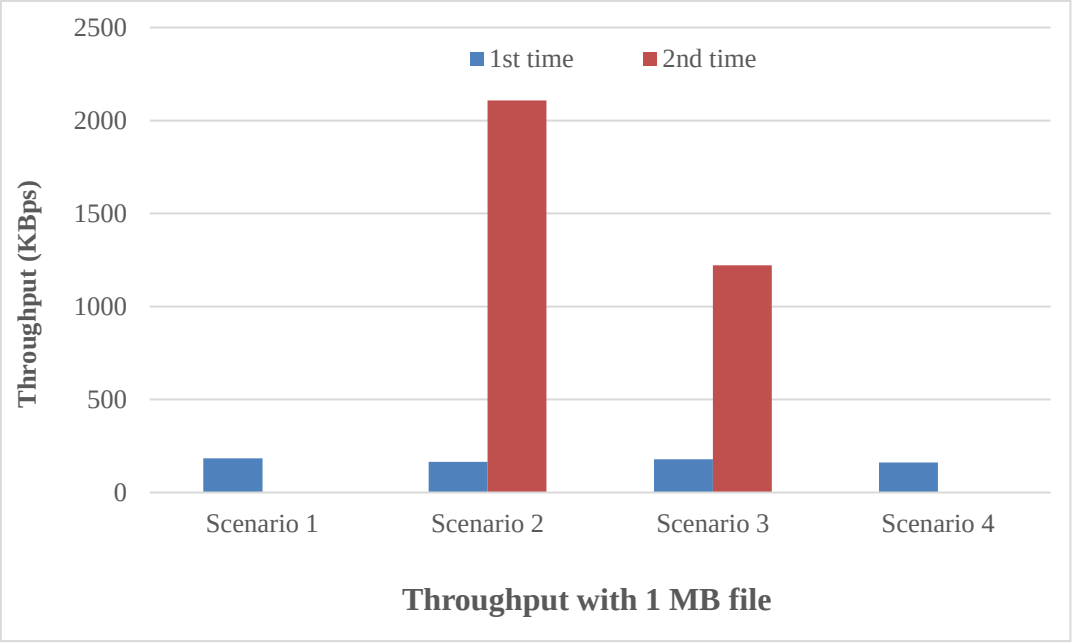

Fig. 16. Throughput with 1 MB file

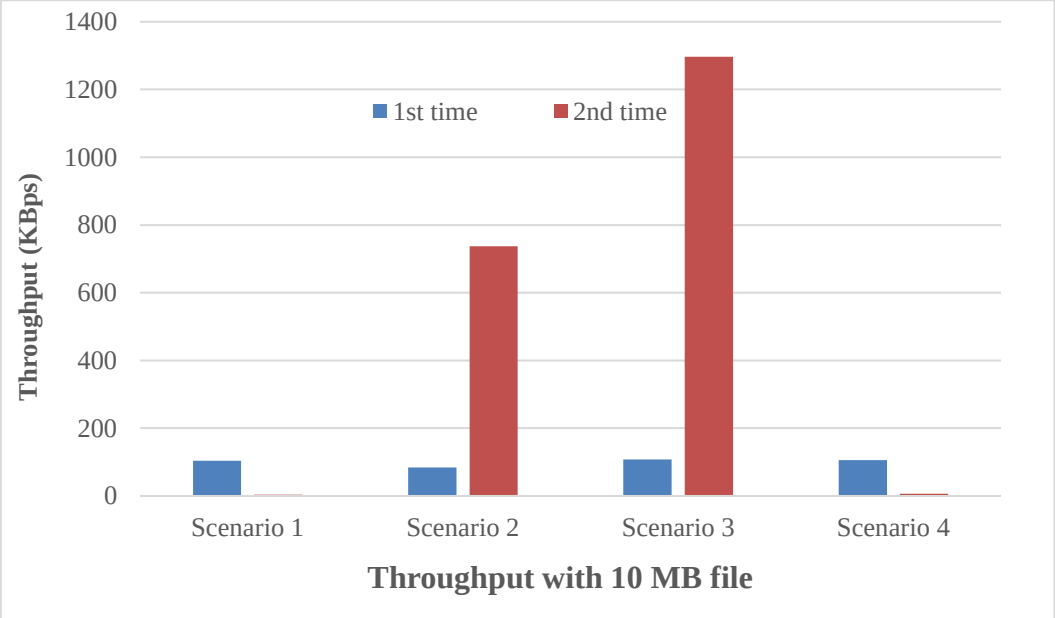

Fig. 17. Throughput with 10 MB file

Overall, the above scenarios show that our proposed system can improve the network performance efficiently right after a requested content is stored in the ICN slice.

Chapter 5 Conclusion

In this study, we have proposed, designed, implemented, and evaluated the combined ICN/CDN architecture as a video streaming service. The joint-testbed evaluations between Japan and Europe show that this approach can reduce the download time effectively, especially when transmitting contents with high popularity. This realizes a potential and feasible network design for efficient video streaming service by leveraging SDN/NFV technologies and combining the benefits of both ICN and CDN for video content distribution.

The concept and design of function chaining design for optimal VNF allocation in network slicing of the integrated ICN/CDN will be the focus of our future work.

Research Achievement

[1]. Chengkai Yan, Tai Rong, Zheng Wen, Xin Qi, Takuro Sato. “Video streaming based on ICN”. IEICE General Conference, March 2018, Tokyo, Japan

[2]. Yan, Chengkai et al. “The Implementation of Integrated ICN and CDN as a Video Streaming Service”. IEICE General Conference, March 2019, Tokyo, Japan

[3]. Yan, Chengkai et al. “Design and Implementation of Integrated ICN and CDN as a Video Streaming Service.” The 17th International Conference on Wired/Wireless Internet Communications (IFIP WWIC 2019), June 2019, Bologna, Italy

References

[1] Cisco Visual Networking Index: Forecast and. Trends, 2017–2022. [Available Online]

[2] V. Jacobson, et al. “Networking named content” In Proc. of ACM CoNEXT’09, Dec.2009.

[3] G. Xylomenos et al., "A Survey of Information-Centric Networking Research," IEEE. Commun. Surveys & Tutorials, 2013, pp. 1–26.

[4] Q. N. Nguyen et al., "A Context-Aware Green Information-Centric Networking. Model for Future Wireless Communications," in IEEE Access, vol. 6, pp. 22804-22816, 2018.

[5] Q. N. Nguyen, M. Arifuzzaman, T. Miyamoto and S. Takuro, "An Optimal. Information Centric Networking Model for the Future Green Network," 2015 IEEE Twelfth International Symposium on Autonomous Decentralized Systems, Taichung, 2015, pp. 272-277.

[6] M. Arifuzzaman, Y. Keping, Q. N. Nguyen and S. Takuro, "Locating the content in. the locality: ICN caching and routing strategy revisited," 2015 European Conference on Networks and Communications (EuCNC), Paris, 2015, pp. 423-428.

[7] Nguyen, Q.N. et al., “PPCS: A Progressive Popularity-Aware Caching Scheme for. Edge-Based Cache Redundancy Avoidance in Information-Centric Networks”. Sensors 2019, 694.

[8] Y. N. Rohmah, D. W. Sudiharto and A. Herutomo, "The performance comparison of. forwarding mechanism between IPv4 and Named Data Networking (NDN). Case study: A node compromised by the prefix hijack," 2017 3rd International Conference on Science in Information Technology (ICSITech), Bandung, 2017, pp. 302-306.

[9] G. Zhang, W. Liu, X. Hei and W. Cheng, "Unreeling Xunlei Kankan: Understanding. Hybrid CDN-P2P Video-on-Demand Streaming," in IEEE Transactions on Multimedia, vol. 17, no. 2, pp. 229-242, Feb. 2015.

[10] Network Functions Virtualisation (NFV); Management and Orchestration; Report. on Architectural Options, ETSI Group Specification NFVIFA 009, Jul. 2017.

[11] D. Kreutz, et al., “Software-Defined Networking: A Comprehensive Survey," in. Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, Jan. 2015”.

[12] T. Taleb, et al., "PERMIT: Network Slicing for Personalized 5G Mobile. Telecommunications," in IEEE Communications Magazine, vol. 55, no. 5, pp. 88-93, May 2017.

[13] J. Zhang, X. Zhang and W. Wang, "Cache-Enabled Software Defined. Heterogeneous Networks for Green and Flexible 5G Networks," in IEEE Access, vol. 4, pp. 3591-3604, 2016.

[14] 5G! Pagoda Project, [Available Online] https://5g-pagoda.aalto.fi/

[15] X. Wang, et al."Cache in the air: exploiting content caching and delivery techniques. for 5G systems," in IEEE Communications Magazine, vol. 52, no. 2, pp. 131-139, February 2014.

[16] I. Benkacem, M. Bagaa, T. Taleb, Q.N. Nguyen, T. Tsuda, and T. Sato, “Integrated. ICN and CDN Slice as a Service,” in IEEE Globecom’18, Abu Dhabi, UAE, Dec. 2018.

[17] K. Kanai et al., "Proactive Content Caching for Mobile Video Utilizing. Transportation Systems and Evaluation Through Field Experiments," in IEEE Journal on Selected Areas in Communications, vol. 34, no. 8, pp. 2102-2114, Aug. 2016.

[18] A. Nakao, et al., "End-to-end network slicing for 5G Mobile networks", Journal of. Information Processing, Vol 25, pp153-163, 2017.

[19] M Arumaithurai et al., “Exploiting ICN for flexible management of software-defined networks,” in ACM-ICN '14 Proceedings of the 1st ACM Conference on Information-Centric Networking, pp. 107-116, Paris, France — September 24 - 26, 2014.

[20] Jianfeng Guan, Wei Quan, Changqiao Xu and Hongke Zhang, "The comparison. and performance analysis of CCN under mobile environments," 2014 IEEE 3rd International Conference on Cloud Computing and Intelligence Systems, Shenzhen,

2014, pp. 292-296.

[21] A. Kerrouche, et al., "AC-QoS-FS: Ant colony based QoS-aware forwarding. strategy for routing in Named Data Networking," 2017 IEEE ICC, Paris, 2017, pp. 1-6.

[22] Yan, Chengkai et al. “Design and Implementation of Integrated ICN and CDN as a Video Streaming Service.” The 17th International Conference on Wired/Wireless Internet Communications (IFIP WWIC 2019).

Appendix

Script of Docker File (on Cent OS 6.4)

```
1.  ################
2.  # Pull base image
3.  # You can find other versions of Ubuntu by command below.
4.  # $ curl -
    s https://registry.hub.docker.com/v1/repositories/ubuntu/tags | sed 's/,/\n/
    g' | grep name | cut -d '"' -f 4
5.  ################
6.  FROM centos:6
7.
8.  RUN yum update -y
9.  ################
10. # Install packages for building ndnx
11. ################
12. RUN yum groupinstall -y 'development tools' &&\
13.     yum install -
    y https://repos.fedorapeople.org/repos/openstack/EOL/openstack-
    icehouse/epel-6/iproute-2.6.32-130.el6ost.netns.2.x86_64.rpm &&\
14.     yum install -y expat-devel sqlite-devel boost-devel openssl-devel &&\
15.     yum install -y libpcap-devel &&\
16.     yum install -y java-1.7.0-openjdk java-1.7.0-openjdk-devel &&\
17.     yum install -y sudo libxml2 net-tools asciidoc
18.
19. ENV JAVA_HOME /usr/lib/jvm/java-1.7.0-openjdk.x86_64
20.
21. ################
22. # Build pkg-config
23. ################
24. RUN yum install -y wget &&\
25.     wget https://pkg-config.freedesktop.org/releases/pkg-config-
    0.29.tar.gz &&\
26.     tar xvf pkg-config-0.29.tar.gz
27. WORKDIR /pkg-config-0.29
28. RUN ./configure --with-internal-glib && make && make install
29.
30. ################
31. # Build ndnx
32. ################
33. WORKDIR /
```

```
RUN wget http://ftp.yz.yamagata-u.ac.jp/pub/network/apache/ant/binaries/apache-ant-1.9.11-bin.tar.gz &&\
    tar zxvf apache-ant-1.9.11-bin.tar.gz
ENV PATH $PATH:/apache-ant-1.9.11/bin
RUN yum install -y git-core
RUN git clone https://github.com/named-data/ndnx.git
WORKDIR /ndnx
RUN git checkout master
RUN  ./configure &&\
     yum install -y make &&\
     make &&\
     wget http://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo &&\
     yum install -y apache-maven &&\
     mvn clean package &&\
     make test &&\
     mkdir -p /usr/local/man/man1 &&\
     make install &&\
     (cd javasrc; ./jrun; make install)
################
# Run ndnx
# Docker container changes own status to "stopped" if no foreground process is running.
# "tail -f /dev/null" command keeps running in foreground and container status "running".
################
#CMD ndndstart && ndnr &
#ENTRYPOINT ["ndndstart", " && ", "ndnr", "&"]
```

Script of Docker File (on Ubuntu 16.04)

```
################
# Pull base image
# You can find other versions of Ubuntu by command below.
# $ curl -s https://registry.hub.docker.com/v1/repositories/ubuntu/tags | sed 's/,/\n/g' | grep name | cut -d '"' -f 4
################
From ubuntu:16.04

################
# Install packages for building ndnx
################
```

```
RUN apt-get -y update &&\
    apt-get -y install git &&\
    apt-get -y install build-essential libcrypto++-dev libsqlite3-dev libboost-all-dev libssl-dev &&\
    apt-get -y install pkg-config libpcap-dev &&\
    apt-get -y install default-jre default-jdk &&\
    apt-get -y install sudo libxml2-utils net-tools
#################
# Build ndnx
#################
RUN git clone -b master https://github.com/named-data/ndnx.git
WORKDIR ndnx
RUN  ./configure &&\
     apt-get -y install make &&\
     make &&\
     apt-get -y install maven &&\
     mvn clean package &&\
     make test &&\
     mkdir -p /usr/local/man/man1 &&\
     make install

#################
# Run ndnx
# Docker container changes own status to "stopped" if no foreground process is running.
# "tail -f /dev/null" command keeps running in foreground and container status "running".
#################
CMD ndndstart && ndnr &
```

UE Program – “Contents Table” (in Python)

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-

import tkinter
from tkinter import messagebox, scrolledtext
import subprocess
import select
import os
import signal

WINDOW_SIZE = "600x400"
```

```
12.
13. # Run Button
14. btn_upd_1 = None
15. btn_upd_2 = None
16. # Output textbox
17. txt_out = None
18. # current script job/process
19. cur_process = None
20. # mark if Tk is closing
21. close_flag = False
22. # Tk root
23. root = tkinter.Tk()
24.
25.
26. def disable_allbtn():
27.     global close_flag
28.     global btn_upd_1
29.     global btn_upd_2
30.     if close_flag:
31.         return
32.     btn_upd_1.configure(state=tkinter.DISABLED)
33.     btn_upd_2.configure(state=tkinter.DISABLED)
34.
35. def enable_allbtn():
36.     global close_flag
37.     global btn_upd_1
38.     global btn_upd_2
39.     if close_flag:
40.         return
41.     btn_upd_1.configure(state=tkinter.NORMAL)
42.     btn_upd_2.configure(state=tkinter.NORMAL)
43.
44. def cb_runbash_withoutput(script):
45.     global cur_process
46.     global txt_out
47.     global root
48.     global close_flag
49.     # clear output
50.     txt_out.delete(1.0, tkinter.END)
51.     # create subprocess
52.     process = subprocess.Popen(script, shell=True, preexec_fn=os.setsid,
53.         stderr=subprocess.PIPE, stdout=subprocess.PIPE)
54.     # read subprocess stderr/stdout in non-blocking
55.     outpoll = select.poll()
```

```
    errpoll = select.poll()
    outpoll.register(process.stdout, select.POLLIN)
    errpoll.register(process.stderr, select.POLLIN)
    # store current subprocess
    cur_process = process
    while True:
        output = ""
        errput = ""
        # poll stdout, timeout 2ms
        if outpoll.poll(2):
            output = process.stdout.readline().decode()
            # insert stdout to txt_out
            if output and not close_flag:
                txt_out.insert(tkinter.INSERT, output)
        # poll stderr, timeout 2ms
        if errpoll.poll(2):
            errput = process.stderr.readline().decode()
            # insert stderr to txt_out
            if errput and not close_flag:
                txt_out.insert(tkinter.INSERT, errput)
        # scroll text if have output
        if (output or errput) and not close_flag:
            txt_out.see(tkinter.END)
        # do eventloop when not closing
        if not close_flag:
            root.update()
        # process end
        if output == "" and process.poll() is not None:
            break
    # get process return code
    code = process.poll()
    # reset current subprocess
    cur_process = None
    return code

# callback
def cb_update_1():
    global close_flag
    disable_allbtn()
    code = cb_runbash_withoutput("./table.sh /satolab/demo.mp4")
    # exit when closing
    if close_flag:
        return
    if code != 0:
```

```
100.         messagebox.showerror(title="Error", message="run bash script failed
    ")
101.     else:
102.         messagebox.showinfo(title="Info", message="run bash script success"
    )
103.     enable_allbtn()
104.
105. def cb_update_2():
106.     global close_flag
107.     disable_allbtn()
108.     code = cb_runbash_withoutput("./table.sh /satolab/demo.mp4")
109.     # exit when closing
110.     if close_flag:
111.         return
112.     if code != 0:
113.         messagebox.showerror(title="Error", message="run bash script failed
    ")
114.     else:
115.         messagebox.showinfo(title="Info", message="run bash script success"
    )
116.     enable_allbtn()
117.
118.
119. def on_closing():
120.     global cur_process
121.     global root
122.     global close_flag
123.     # if cur_process still running
124.     if cur_process:
125.         if messagebox.askokcancel("Quit", "There is task running, do you wa
    nt to quit?"):
126.             # kill subprocess
127.             os.killpg(os.getpgid(cur_process.pid), signal.SIGTERM)
128.             cur_process = None
129.         else:
130.             return
131.     # mark closing
132.     close_flag = True
133.     # close window
134.     root.destroy()
135.
136. root.title("Video-Contents-Table")
137. root.geometry(WINDOW_SIZE)
138. # register cloing event callback
```

```
root.protocol("WM_DELETE_WINDOW", on_closing)

btn_upd_1 = tkinter.Button(root, text="Content-1", command = cb_update_1)
btn_upd_1.pack(fill=tkinter.X, pady=10)
btn_upd_2 = tkinter.Button(root, text="Content-2", command = cb_update_2)
btn_upd_2.pack(fill=tkinter.X, pady=10)

txt_out = scrolledtext.ScrolledText(root)
txt_out.pack(fill=tkinter.BOTH, expand=True, padx=20, pady=20)

tkinter.mainloop()
```

Table.sh

```
#!/bin/bash
name=$1
vlc ccnx://$name
```