INVITED PAPER *Special Section on Enhancing Information Centric Networking Technologies Towards Real-world Infrastructure*

Cefore: Software Platform Enabling Content-Centric Networking and Beyond

Hitoshi ASAEDA[†a)], ***Senior Member***, **Atsushi OOKA**[†], **Kazuhisa MATSUZONO**[†], ***and*** **Ruidong LI**[†], ***Members***

SUMMARY Information-Centric or Content-Centric Networking (ICN/CCN) is a promising novel network architecture that naturally integrates in-network caching, multicast, and multipath capabilities, without relying on centralized application-specific servers. Software platforms are vital for researching ICN/CCN; however, existing platforms lack a focus on extensibility and lightweight implementation. In this paper, we introduce a newly developed software platform enabling CCN, named Cefore. In brief, Cefore is lightweight, with the ability to run even on top of a resource-constrained device, but is also easily extensible with arbitrary plugin libraries or external software implementations. For large-scale experiments, a network emulator (Cefore-Emu) and network simulator (Cefore-Sim) have also been developed for this platform. Both Cefore-Emu and Cefore-Sim support hybrid experimental environments that incorporate physical networks into the emulated/simulated networks. In this paper, we describe the design, specification, and usage of Cefore as well as Cefore-Emu and Cefore-Sim. We show performance evaluations of in-network caching and streaming on Cefore-Emu and content fetching on Cefore-Sim, verifying the salient features of the Cefore software platform.

key words: *ICN, CCN, in-network cache, computing in network, Cefore, open source*

1. Introduction

Billions of people with mobile devices and small devices, such as sensors, actuators, and robots, are generating tremendous amounts of data. The Internet adopts the host-centric model, assigning Internet Protocol (IP) addresses to hosts, which enables end-to-end communications. However, this model results in redundant communication overheads as well as large latencies for data retrieval because of duplicate data transmissions from distant servers. To cope with these problems, Information-Centric/Content-Centric Networking (ICN/CCN) has emerged as a novel networking paradigm advocating data retrieval through content identifiers or names, regardless of locations and further addresses. ICN/CCN shifts this communication model to the content-centric model by assigning names to content. By using these content names, an ICN/CCN router intrinsically supports caching and multicast content without relying on centralized or application-specific servers.

Owing to these salient features, ICN/CCN has attracted substantial research attention in the past few years. Many researchers have focused on the ability to efficiently and securely disseminate/retrieve content and have proposed various protocols and mechanisms to enhance ICN/CCN architectures and technical features. We have also studied the essential functions, such as name-based routing [1], [2], transport [3], [4], mobility [5], caching [6], [7], security [8], [9], testbed [10], and measurement [11], [12]. For protocol evaluations, ICN/CCN software implementations play a fundamental role in research in this area, and Community ICN (CICN) [15] and NDN Forwarding Daemon (NFD) [16] are the major implementations that have successfully pushed forward research and satisfied, in part, demand from the ICN/CCN research community in this initial phase.

In addition, there has recently been a strong trend toward the integration of in-network computation with the novel networking architecture to enable efficient data sharing. New computation technologies, such as modern machine learning and blockchain technologies, have the potential to be embedded into networks to provide intelligence and additional functions. Activities such as the IEEE SIG on big data intelligent networking [19] and Computing in the Network (COIN) [20] in the Internet Research Task Force (IRTF) have been initiated for these novel studies. This new trend imposes requirements for the extensibility and ease of use of ICN/CCN software platforms with the capability of embedding various potential functions.

To satisfy the requirements for implementing future networks, we identify the following design requirements for an ICN/CCN software platform: (1) Lightweight: the software implementation should be compact and the platform should be usable for resource-constrained devices, such as sensor nodes; (2) Easy usage: the platform should be easily configured, set up, reloaded, and connected to the experimental environments. Ideally, its emulation/simulation should be easily conducted and tested using real network equipment; (3) Extensibility: the platform should be easily extensible to accommodate novel functions to satisfy future network needs.

To satisfy these requirements, we have developed a software platform, named "Cefore" [13]. The development of Cefore aims to not only provide actual running code for CCN-based communications, but also to initiate new research activities that emerge from or are based on ICN approaches. In Cefore, the forwarding daemon is implemented with a minimum set of functions, and hence the implementation can be lightweight and scalable. Other functions, such as caching and computing functions, are or can be implemented using plugins or external daemons, providing extensibility and

Manuscript received November 29, 2018.
Manuscript revised February 1, 2019.
Manuscript publicized March 22, 2019.

[†]The authors are with the Network System Research Institute at National Institute of Information and Communications Technology (NICT), Koganei-shi, 184-8795 Japan.
a) E-mail: asaeda@nict.go.jp

DOI: 10.1587/transcom.2018EII0001

easy usage. In summary, Cefore consists of the following components: (1) "cefnetd" daemon, which is a lightweight packet forwarding daemon that supports the basic functions of the CCN, such as Interest/Data handling and Forward Information Table (FIB) and Pending Interest Table (PIT) management; (2) "csmgrd" daemon, which implements Content Store (CS) using the UNIX filesystem or memory and interacts with one or more cefnetd daemon(s), is an optional function; (3) arbitrary plugin library implementations that extend the functionality of cefnetd or csmgrd, which achieves extensibility; (4) network tools/commands and sample applications for easy performance measurements. For ease of usage, Cefore can run on diverse OSs, such as Ubuntu, Raspbian Jessie, macOS, and Android.

In addition, Cefore is advantageous for experiments, since we further developed Cefore-Emu and Cefore-Sim for emulation and simulation using Cefore, respectively. Cefore-Emu and Cefore-Sim can be easily configured, reloaded, and used for CCN emulations and simulations. They also support hybrid experimental environments that incorporate physical networks into the emulated/simulated networks. The hybrid experimental environments enable precise performance or quality of experience measurement for wireless communications, mobility, and streaming.

The remainder of this paper is organized as follows. We first introduce two major ICN/CCN implementations as related work in Sect. 2. In Sect. 3, we describe the Cefore software platform enabling CCN. In Sect. 4, we provide the design, operations, and features of the network emulator Cefore-Emu and summarize our experiments related to in-network caching mechanisms and streaming using Cefore-Emu. In Sect. 5, we describe the implementation structure and operations of the network simulator Cefore-Sim and show our experiments with hybrid configurations. Finally, we conclude our work in Sect. 6.

2. Related Work

Reference implementations of the ICN/CCN architecture have emerged. The first released implementation for CCN is known as CCNx version 0 (CCNx-0) [14]. After CCNx-0 became obsolete, CCNx version 1 (CCNx-1.0) was proposed by PARC, Inc., and its packet formats are specified as an IRTF document [17], [18]. Cisco Systems, Inc. has recently acquired the CCN platform (including the CCNx code license) from PARC. CCNx-1.0 is practically merged into Community ICN (CICN) [15].

CICN provides two CCN forwarders, a plugin implementation of the Vector Packet Processing (VPP) framework (named VPP cicn-plugin) for an FD.IO community and a socket-based forwarder (named Metis) that does not require VPP. In VPP, the Data Plane Development Kit (DPDK) is used to bypass kernel processing. The packet processing pipeline in VPP is defined by a graph in which nodes represent specific processes and edges represents their dependency relations. By inserting the cicn-plugin into the graph as a new node, the DPDK-based CCN forwarder can be easily implemented. This forwarder can support up to 14 Mpps of throughput on a 3.5 GHz CPU [28]. However, the specialization in high-throughput forwarding results in installation difficulty, low extensibility, and limited supported OSs (Ubuntu and CentOS). It is also suspected that VPP can be run on top of resource-constrained nodes or devices.

In contrast, Metis can be easily installed and used in various OSs, such as Ubuntu, Debian, CentOS, macOS, and Android, although socket-based forwarding is slower than VPP. The software forwarder, metis_daemon, is implemented as a forwarder that accurately follows CCNx specifications [17], [18]. In addition, it is expected to support multi-threading and interface generalization for higher throughput. Currently, the interfaces of listeners (TCP/UDP, Ethernet, etc.), PIT, and CS are modularized, and it is relatively easy to modify the features. However, it is still necessary for developers to fully understand the internal structure of the heavily weighted Metis program.

Named-data networking (NDN) is a different architecture similar to CCN, and an NDN software platform is provided by the NDN project [16]. The software includes an NDN Forwarding Daemon (NFD) and many relevant tools and application programs, enriching the NDN communication architecture. NFD is a socket-based forwarding daemon, as is Metis. Its target is high modularity to promote the development of new algorithms and features as well as experiments that use them. Thus, almost all features in NFD are modularized and their interfaces are detailed in the developer's guide [16]. Currently, more than 30 organizations, including universities and companies, are developing many tools and applications, such as NLSR (Named Data Link State Routing), Mini-NDN (emulator), ndnSIM (simulator), and NDN tools. However, precisely because NDN focuses on avoiding premature standardization rather than interoperability and global rollout, the NDN packet/message format is not officially defined by the IRTF or other Standards Development Organizations. As a result, the NDN platform is frequently modified, yet developers need to always carefully and continuously follow the activities in the NDN project to extend functionality with their own applications. Another consequence is a lack of interoperability between NDN and CICN or other CCNx-1.0-based implementations, as their packet formats are different.

CICN and NFD have successfully satisfied some demands of the ICN/CCN research community in the initial phase, but limitations in terms of lightweight implementation, ease of usage, and extensibility remain. In addition, the recent trend to embed various potential functions in the novel networking architecture (e.g., service function chaining) or computation technologies (e.g., machine learning) necessitates the extensibility and easiness of use for ICN/CCN software platforms. According to the research directions recently discussed in [19], [20], it is important to consider the potential extensibility of the ICN/CCN software platform; yet, how future extensions can be used and linked to CICN and NFD software has not been carefully studied.

3. Cefore: Extensible Packet Forwarding Engine Enabling CCN Communications

To overcome the aforementioned limitations, we developed and released Cefore [13], which is an open-source software platform enabling CCN communications. Basically, Cefore consists of a no-frills forwarding daemon named cefnetd and a CS manager daemon named csmgrd that can manage a UNIX file system-based in-network cache.

Cefnetd is a base component of Cefore and contains the essential functions for forwarding CCNx messages [17] equipped with Forwarding Information Base (FIB) and Pending Interest Table (PIT). Csmgrd is a Content Store (CS) daemon that can be operated on a node that behaves as CS. Cefnetd can connect to csmgrd via a local socket or TCP. Besides these basic functions, Cefore can be easily customized and enhanced by adding plugin libraries. Potentially, researchers can develop new mechanisms and build them into cefnetd and/or csmgrd without modifying these codes. Cefore also includes useful network tools, such as cefputfile and cefgetfile, for uploading and downloading data, respectively, and utilities, such as CCNinfo [12].

Through cefnetd, publishers provide data through the network, and consumers retrieve data by name. A router running cefnetd forwards data requests by means of its FIB, which is populated by name-based routing protocols. Cefnetd enables consumers to retrieve data from in-network cache, which is maintained by own memory or csmgrd. If a router receiving a data request has the requested data in the cache, it sends the data toward the consumer without forwarding the data request ahead.

Figure 1 shows an image of the software components of Cefore and Fig. 2 shows how it can be used with different equipment and for different purposes.

3.1 Cefnetd: Packet Forwarding Daemon

Cefnetd is a packet forwarding daemon that can be used in a resource-constrained environment as well as a high-spec machine. The resident memory size required to run cefnetd is about one-third of the memory usage for starting the Metis forwarder. Even when CS is enabled by csmgrd, cefnetd can maintain memory consumption at the same level, without an increase, in contrast to the increasing memory usage for Metis to cache data. This is because cefnetd cooperates with csmgrd, which provides an external CS process. In fact, the total memory consumption of cefnetd and csmgrd is higher than that of Metis. However, csmgrd has the salient ability to be executed on a computer different from one where cefnetd is executed. For example, if a user wants to run the CCN router function on Raspberry Pi, s/he can launch cefnetd on Raspberry Pi and configure it to cooperate with csmgrd running on another machine, such as a high-end server.

Cefnetd has high extensibility via plugin libraries or cooperation with external processes. Developing a plugin only requires the use of plugin interfaces; therefore, it is generally

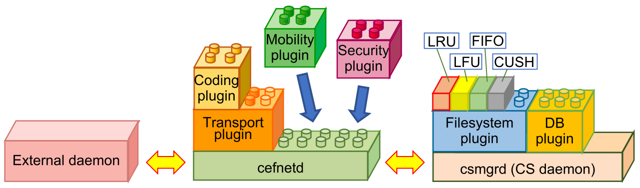


Fig. 1 Cefore and its software components.

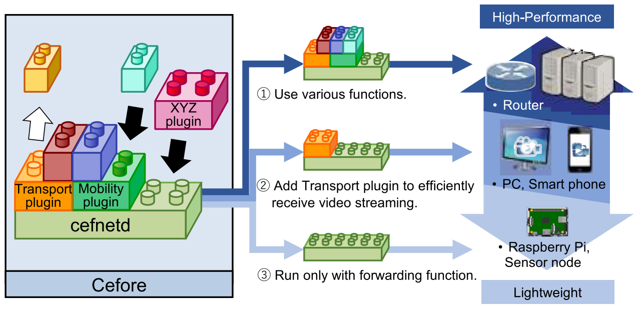


Fig. 2 Cefore applied for heterogeneous environments using arbitrary plugin libraries.

Table 1 Throughput of cefnetd in a physical environment.

OS	CPU	Memory	Throughput
Ubuntu 16.04	2.9 GHz × 4	8 GB	480 Mbps

unnecessary to understand and modify the internal programs of cefnetd, in contrast to Metis or NFD. For example, users can install transport functions for a specific application or mobility functions into cefnetd by developing them as plugins. In addition, arbitrary processes can be implemented as external daemon processes like a costly CS process that is assigned to csmgrd. Cefnetd provides APIs and can connect to such external processes via a local socket or TCP. This cooperation with external processes will activate the potential of other enhanced mechanisms. For example, a network controller daemon enabling intelligent in-network cache management based on artificial intelligence or machine learning will be interconnected with multiple cefnetds in the network.

Users can also easily configure and setup cefnetd. The cefnetd configuration file allows changes in the parameters and forwarding information (e.g., FIB) but also the ability to enable or disable user-defined plugins and cooperation with external processes without re-compilation. Thus, users can concentrate on the development of desired functions without implementing interprocess communication mechanisms.

Table 1 shows the maximum throughput of cefnetd running on a physical environment. In the evaluation, two cefnetd PCs (Intel Core i5) connected via 1 Gbps link acted as a consumer and a producer, and the consumer requested streaming content to the producer.

3.2 Csmgrd: CS Manager Daemon Using UNIX Filesystem

A significant feature in CCN is that a portion of the routers holds cache for close data retrieval. However, the consumption of a large memory for cache affects the efficient usage of the router resources, which will be a substantial burden, especially for resource-constrained devices. To address this issue, Cefore implements a CS manager daemon (csmgrd) that enables data caching on a UNIX filesystem, which can be co-located or separated with the cefnetd routers. More precisely, Cefore provides three types of CS implementations: a local cache memory on cefnetd (called local cache), a local cache memory on csmgrd (called on-memory cache), and a cache space on a UNIX filesystem on csmgrd (called on-filesystem cache).

Users can enable or disable CS for cefnetd. If they enable CS for cefnetd, they select the cache type either from local cache, on-memory cache, or on-filesystem cache. The on-filesystem cache has been introduced in detail for the implementation of the global ICN testbed called CUTEi [10], and we herein provide a simple introduction. On-filesystem cache avoids the memory occupation of routers. Furthermore, the on-filesystem cache system accommodates two kinds of caches: "individual cache" and "shared cache." Individual cache is accessible for only one dedicated router, while shared cache is accessible for a set of routers in the same group to avoid duplicated caching in the neighborhood. Multiple cefnetd daemon processes can share the cached contents.

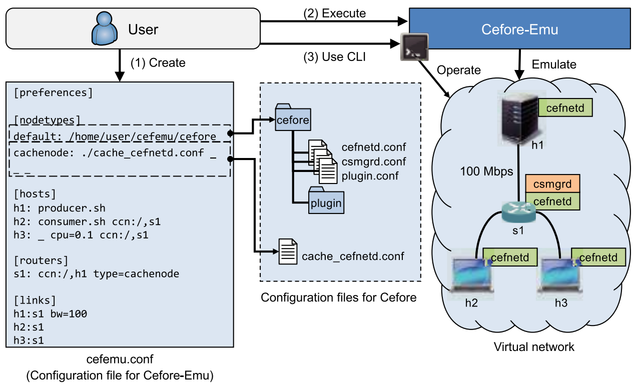
To implement the shared cache, csmgrd provides two sub-functions, "cache expiration control" and "cache write control". Cache expiration control enables csmgrd to expire (and discard) the cached content according to its expiration time. It is necessary for shared cache, because the expiration time of the content stored by one router may be updated by another router and cannot be detected by any router. The cache write control prevents routers from writing duplicate content (chunk) into shared cache when the same content is already stored.

In addition, csmgrd has another sub-function, "cache buffer control", which temporarily keeps content in memory without direct write. That is, the data are written into the on-filesystem cache after the temporary memory space is full (or certain timeout). The cache buffer control reduces frequent disk I/O process calls to avoid the caching performance penalty comparing with on-memory cache.

4. Cefore-Emu

4.1 Overview

Based on the Cefore software platform, we further developed a network emulator named Cefore-Emu that runs on a Linux machine. Cefore-Emu is a fork of the Mininet-HiFi emulator [21] to integrate it with Cefore. Like Mininet-HiFi, Cefore-Emu creates a container by using the following features: a process group, a network namespace, and virtual network interfaces. The process group and the network namespace are private. That is, they belong to a particular container and cannot be seen by the other containers. In addition, cgroups in Linux, which realizes the private process group, has a promising feature in which it isolates and limits the resource usage of each process group. The virtual network interfaces are connected via virtual network links. The properties of virtual links (e.g., bandwidth, delay, and packet loss) can be configured by using the Linux traffic control (tc) command.



```
[preferences]

[nodetypes]
default: /home/user/cefemu/cefore
cachenode: ./cache_cefnetd.conf

[hosts]
h1: producer.sh
h2: consumer.sh ccn:/,s1
h3: _ cpu=0.1 ccn:/,s1

[routers]
s1: ccn:/,h1 type=cachenode

[links]
h1:s1 bw=100
h2:s1
h3:s1
```

Fig. 3 An example of the configuration files and emulated network topology for Cefore-Emu.

For the operations of Cefore-Emu, Fig. 3 illustrates an example in which a simple network is emulated using Cefore-Emu. A Cefore-Emu user needs to take the following actions: (1) prepare the configuration file for Cefore-Emu, cefemu.conf, (2) execute the cefemu command to set up an emulated network, and (3) use a command line interface (CLI) to operate the emulated network. The configuration file, cefemu.conf, is simpler than the original python script used by Mininet-HiFi. It defines a topology as a collection of nodes and links. The detailed properties of nodes and links can also be configured in cefemu.conf. The nodetypes definition allows users to define the behaviors of the daemons, such as cacheable router or non-cacheable router. The host and router definitions can specify an upper limit of CPU usage for nodes, FIB entries, and node type (defined in nodetypes). In this example, the user configures the topology as follows: for each named content (i.e., ccn:/), the neighbor of h2 is s1, the neighbor of h3 is s1, and so on. To run applications on top of Cefore-Emu, the user's programs, such as producer.sh and consumer.sh, are also specified in the definitions of hosts and routers. The links definition indicates that the link bandwidth between h1 and s1 is 100 Mbps. After the user prepares the configuration files, s/he can start network emulation by executing the cefemu command, which automatically launches cefnetd and csmgrd (if configured) on each emulated node.

Compared with existing emulators, Cefore-Emu provides the following three unique features. First, a topology

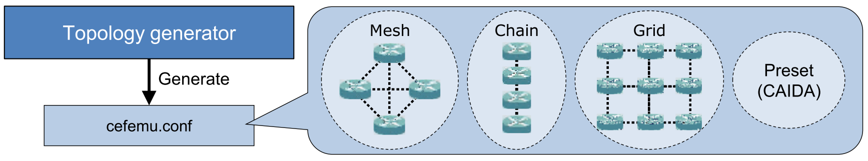
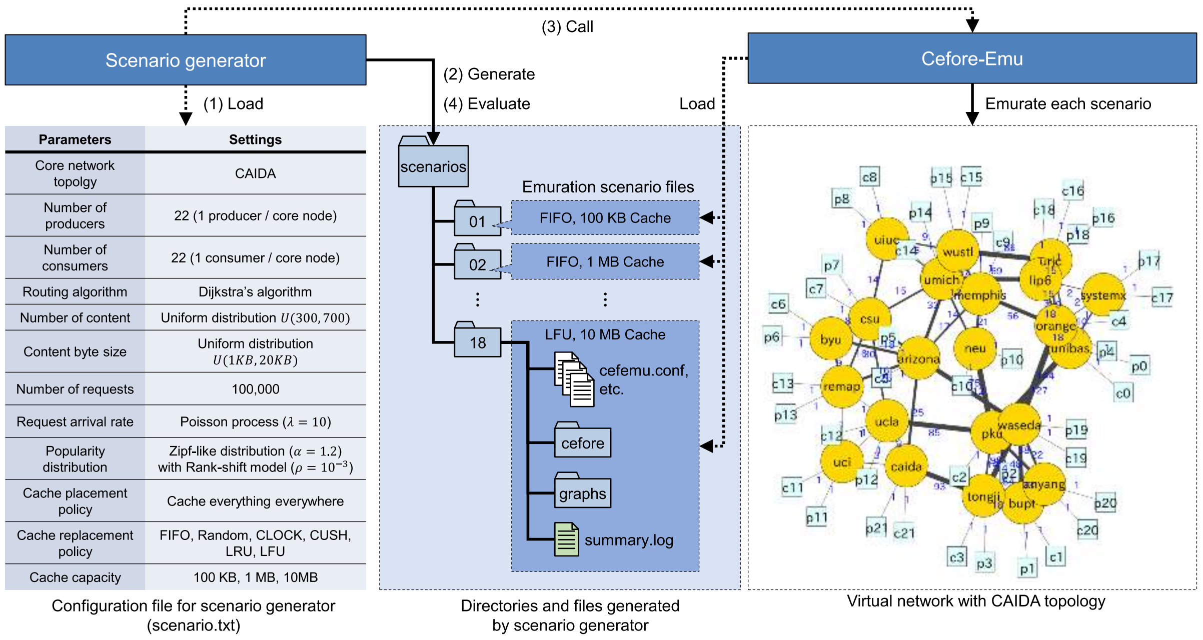


Fig. 4 Topology generator for Cefore-Emu.



Parameters	Settings
Core network topolgy	CAIDA
Number of producers	22 (1 producer / core node)
Number of consumers	22 (1 consumer / core node)
Routing algorithm	Dijkstra's algorithm
Number of content	Uniform distribution $U(300, 700)$
Content byte size	Uniform distribution $U(1KB, 20KB)$
Number of requests	100,000
Request arrival rate	Poisson process ($\lambda = 10$)
Popularity distribution	Zipf-like distribution ($\alpha = 1.2$) with Rank-shift model ($\rho = 10^{-3}$)
Cache placement policy	Cache everything everywhere
Cache replacement policy	FIFO, Random, CLOCK, CUSH, LRU, LFU
Cache capacity	100 KB, 1 MB, 10MB

Fig. 5 Scenario generator for Cefore-Emu.

generator (Fig. 4) and a scenario generator (Fig. 5) are provided to easily initiate emulation environments. In particular, when writing a configuration file based on a real topology that consists of a large number of nodes and links, there is a risk of human error in writing the nodes' and links' properties, such as delay and bandwidth. For Cefore-Emu, the topology generator creates the cefemu.conf file based on the specified topology, such as a mesh, chain, tree, or grid topology, or the Internet topology provided by CAIDA [23]. After a user specifies a set of parameters (as shown in scenario.txt in Fig. 5), the scenario generator loads it and generates (or modifies if exists) cefemu.conf, Cefore configuration files (cefnetd.conf, csmgrd.conf, and plugin.conf), and scripts for producers and consumers. If multiple values are specified in the parameters in scenario.txt, the scenario generator creates the configuration files and scripts for the different sets of parameters. Figure 5 shows that eighteen scenarios are generated by the combinations of six cache replacement policies and three cache sizes. The scenario generator then conducts the experiments using Cefore-Emu, and finally reports about the evaluation results in a summary.log file located in each scenario directory. These two generators significantly reduce the burden of configuration when conducting a large number of evaluation scenarios.

The second feature is a "reloading function." If users develop a new mechanism and install it into cefnetd or csmgrd running on Cefore-Emu nodes, it often changes the parameters of the user-defined mechanisms according to the emulation results. When users modify one or some of these configurations, they can reload the configurations of the daemons on Cefore-Emu nodes by using the reload command without restarting the whole emulation. The reloading function allows users to quickly use multiple parameters or various proposals for CCN, as the time to set up and clean up an emulated network is not always negligible.

Third, Cefore-Emu enables a "hybrid emulation" that incorporates physical networks into the emulated network. Although results given by emulation for wired networks can be slightly stable and modeled feasibly, those for wireless networks or mobile communications are sometimes far from the real results as the characteristics of the wireless medium are too complicated and highly sensitive in the network conditions. The hybrid experimental environment we developed allows users to leverage the fidelity of a physical environment including the wireless medium to generate an arbitrarily customizable network on the fly. For ease of configuration, Cefore-Emu adds a new node parameter in cefemu.conf to specify a physical interface including wireless interfaces. Users can incorporate physical interfaces into emulated nodes by merely setting the optional parameter (e.g., a wireless interface wlan0 can be incorporated to a node by specifying "exintf=wlan0" as the parameter of the

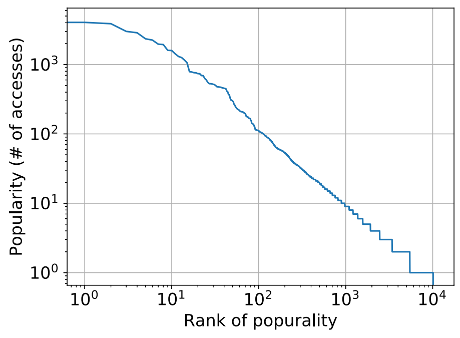


Fig. 6 Popularity distribution.

node).

4.2 Cefore-Emu in Action

4.2.1 Comparing Cache Replacement Policies

We measured CCN performance for different cache replacement policies in a large-scale emulated network, as shown in Fig. 5. The evaluation here was performed using the tools described in Sect. 4.1. The parameters used in this emulation are listed in scenario.txt in Fig. 5. The topology included 66 nodes consisting of 22 core routers based on the CAIDA topology (depicted by large yellow circles) and 44 edge nodes attached to them (depicted by small blue rectangles). The edge nodes named “pN” and “cM” ($1 \leq$ N, M ≤ 22) represent producers and consumers, respectively. The nodes are connected by 94 links. The numbers on the links shown in the figure indicate their link delays in milliseconds, which are used to execute Dijkstra’s algorithm. In this experiment, we do not consider dynamic routing for simplicity.

In this emulation, producers upload 11,307 content objects (cobs) and consumers issue 100,000 requests (i.e., Interests) to download the content, allowing repetition. The content popularity basically follows a Zipf-like distribution with $\alpha = 1.2$ and the rank-shift model [24] to realize the dynamic change in popularity. Figure 6 shows the generated popularity distribution, which roughly follows a power law with a fat tail, as is commonly observed for network content.

Cefore-Emu supports Cache Everything Everywhere (CEE) (also known as Leave-Copy Everywhere (LCE)) as the cache placement policy, where all cacheable nodes try to cache all transferred content. We examined six cache replacement policies: first-in first-out (FIFO), random replacement (Rand), least recently used (LRU), least frequently used (LFU), CLOCK (a low-overhead version of LRU), and CLOCK-Pro using switching hash-tables (CUSH) [7]. In this emulation, all core routers have the cache mechanisms installed and the cache is enabled. Each cache has a capacity ranging from 100 KB to 10 MB. These sizes approximately correspond to 0.1 – 10% of the total size of all content in the network.

The evaluation results are shown in Figs. 7 and 8, which depict cache hit rates and path stretches, respectively. The cache hit rate is defined as the fraction of the number of

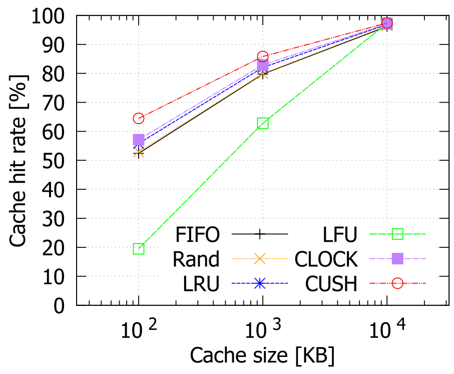


Fig. 7 Cache hit rate.

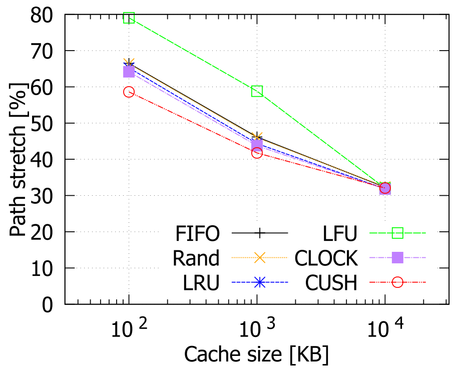


Fig. 8 Path stretch.

cache hits to the total number of requests, excluding the first request for each content. The path stretch is defined as the fraction of the sum of actually transferred hops of all packets to the sum of hops without in-network caches. CUSH outperforms the other algorithms when the cache size is not large. LFU is significantly degraded in contrast to common experiments adopting static popularity because its focuses on only the number of accesses to content. This frequency-based strategy fills the cache with outdated content and pushes out content that newly (dynamically) becomes popular.

Although the evaluation requires experimental effort to conduct 18 emulation scenarios (6 cache replacement policies $\times$ 3 cache sizes), implementing many scenarios is easy owing to the support of the topology generator and the scenario generator. The scenario generator can also visualize the topology and popularity distribution; in fact, the topology graph in Fig. 5 and the popularity distribution graph in Fig. 6 were generated by the scenario generator.

4.2.2 Measuring the Streaming Performance of Hybrid Emulation

To examine the feature of hybrid emulation, we conducted an experiment in the hybrid experimental environment enabled by Cefore-Emu incorporating physical networks into the emulated virtual network. The advantage of the incorporation is that it allows experiments in realistic environments that are too complicated to be simulated/emulated. In this exper-

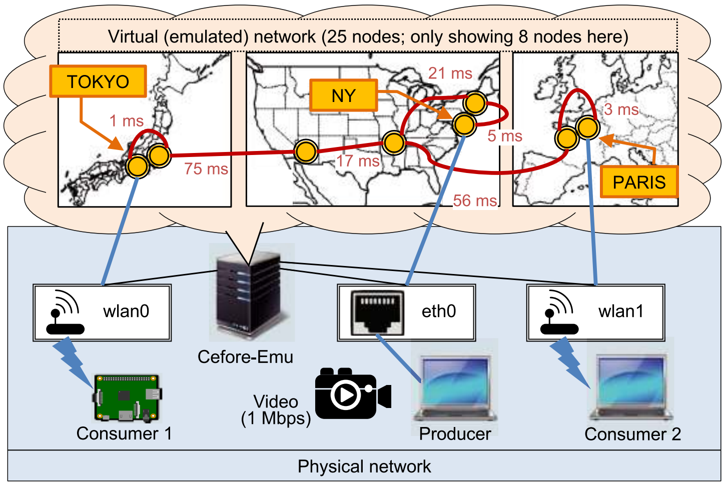


Fig. 9 Cefore-Emu hybrid experimental environment that combines a virtual intercontinental network and physical equipments including wireless medium.

Table 2 Experimental computers.

Node	OS	CPU	Memory	HDD
Cefore-Emu	Ubuntu 16.04	Intel Core i5-3470T (2.90 GHz × 4)	8 GB	512 GB
Producer	Ubuntu 16.04	Intel Core i5-3470T (2.90 GHz × 4)	8 GB	512 GB
Consumer 1	Raspbian 9.3 (Raspberry Pi 3 B+)	ARMv8 (900 MHz × 4)	1 GB	16 GB (microSDHC)
Consumer 2	macOS High Sierra	Intel Core i7-3667U (2.00 GHz × 4)	8 GB	512 GB

iment, three physical devices (two laptops and a Raspberry Pi) are connected to a virtual network that emulates intercontinental communication, as shown in Fig. 9. The virtual network consists of the core topology used in the previous evaluation and three additional nodes, TOKYO (in Tokyo, Japan), NY (in New York City, USA), and PARIS (in Paris, France). Figure 9 shows only eight nodes for simplicity, although there are a total of 25 nodes in the virtual network.

The virtual network is emulated by the Cefore-Emu node, which is connected by three physical computers. Producer connects to NY via the wired interface eth0 and publishes 720 Kbps of streaming video content to Consumer 1 and Consumer 2, which are connected to TOKYO and PARIS via wireless interfaces wlan0 and wlan1, respectively. The specifications of computers used in this experiment are shown in Table 2. As described in Sect. 4.1 and depicted in Fig. 9, the incorporation of physical interfaces into Cefore-Emu is easily realized by adding the parameter "exintf" to the nodes. The experiment is conducted in two environments: (a) an environment without movement where the consumers stay 1 m away from the access points (APs) and (b) an environment with movement where the both consumers are carried from 1 to 14 m away from the APs at a constant speed for 70 s. To demonstrate the instability of a real wireless network, the round-trip time (RTT) is measured in the experiments. The RTT is defined as the elapsed time between when a consumer sends out a request and when the consumer receives a corresponding content object (cob). The theoretical RTTs of Consumer 1 and Consumer 2 are 238 ms and 170 ms, respectively. If the junction node caches a requested cob, the RTT is shortened by 52 ms.

Figures 10 and 11 plot the RTTs and error rates per 100 cobs of requests made from 10 s after requesting the first cob. The x-coordinate of each point corresponds to the time when the request is sent by the consumer. The color of points is darkened every 1,000 cobs to visualize the difference in request timing between two consumers. Because the RTTs of unsatisfied requests are undefined, the y-coordinate of the point for a packet loss is assumed to be a certain constant value. The averages and standard deviations of RTTs are shown in Table 3.

These results reflect the fluctuations of the wireless medium. The jitter of Consumer 1 is larger than that of Consumer 2 because of the poor performance of Raspberry Pi. In Fig. 10, Consumer 1 can partially receive a cob within 200 ms, which is faster than the theoretical RTT between NY and TOKYO. This is because the requests of Consumer 1 hit in-network cache at the junction node. This is indicated by the color change in Fig. 10: the requests of Consumer 1 are issued after those of Consumer 2 (e.g., when 40 s has elapsed). In Fig. 11, the further Consumer 1 moves from the AP, the longer the RTT of Consumer 1 becomes. The packet loss rate also increases and Consumer 1 loses the connection at 48 s after receiving the 2,577th cob.

Table 4 shows the maximum throughput of cefnetd on the emulated environment. We configured two emulated nodes and measured the maximum throughput for transferring streaming content between them. Although the throughput was decreased because of the emulation, Cefore-Emu still achieved 303 Mbps data forwarding.

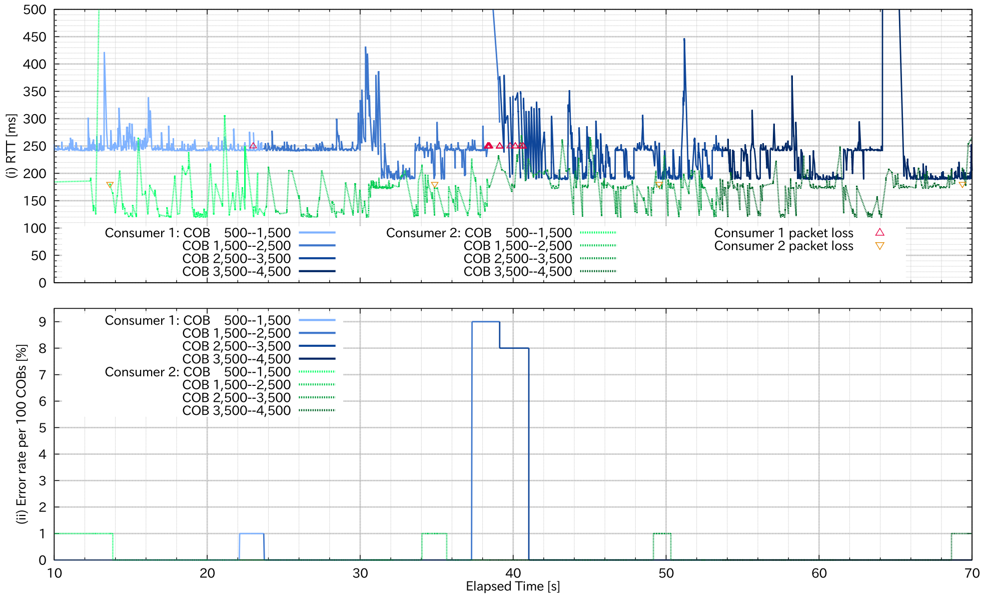


Fig. 10 Evaluation of video streaming on the Cefore-Emu hybrid experimental environment (without movement).

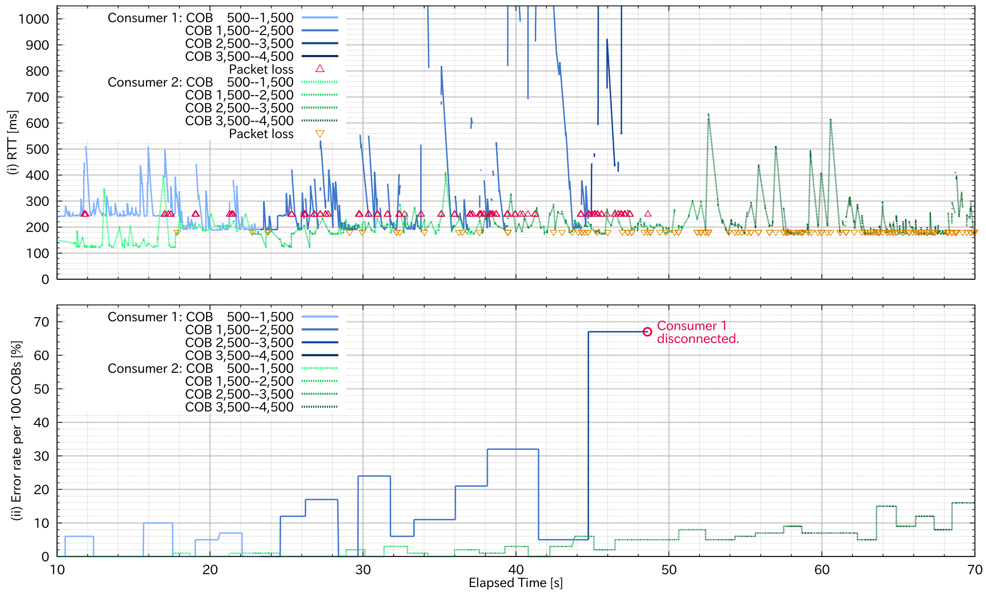


Fig. 11 Evaluation of video streaming on the Cefore-Emu hybrid experimental environment (with movement).

Table 3 RTT statistics.

Computer	Ave. [ms]	Dev. [ms]
(a) Consumer 1	233.46	70.46
(a) Consumer 2	172.14	53.55
(b) Consumer 1	280.26	232,78
(b) Consumer 2	221.13	116.56

Table 4 Throughput of cefnetd in emulation.

OS	CPU	Memory	Throughput
Ubuntu 16.04	2.9 GHz × 4	8 GB	303 Mbps

5. Cefore-Sim

5.1 Overview

To further improve the ease of usage of Cefore, we developed Cefore-Sim, which is an NS-3 [26] based simulation platform. With Cefore-Sim, real implemented Cefore programs can be simulated within NS-3. The main objectives of Cefore-Sim are to enable flexible and faster development and to provide a repeatable and fully controllable evaluation environment for Cefore applications and protocols.

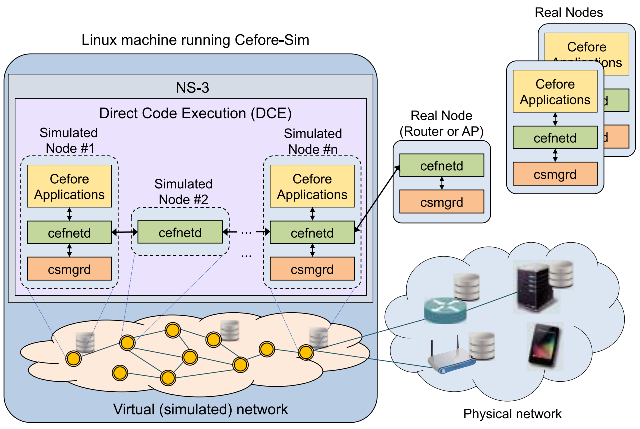


Fig. 12 An example of the Cefore-Sim framework.

Figure 12 illustrates the Cefore-Sim framework. Direct Code Execution (DCE) [25] is utilized to allow Cefore developers and experimenters to directly run the unmodified Cefore implementations, such as cefnetd, csmgrd, and cefgetfile, on each simulated node in an NS-3 network. Cefore-Sim gets rid of the duplicate effort of writing both simulation and real implementation code. It facilitates the development and testing of the original Cefore applications and protocols in a complex scenario and/or large-scale simulated network in NS-3. For instance, we can easily run a mobility scenario under mixed wireless networks (e.g., WiFi, WiMax, and LTE) by writing the simulation scenario script using NS-3 wireless-related modules provided as defaults. This feature eliminates the high cost of real equipment setups. Moreover, the simulation scenario is fully controllable and reproducible, which enables users to debug the implementations running with Cefore and explore performance for further improvements.

In contrast, the NS-3-based NDN simulator (called ndnSIM) provides a simulator platform for evaluating NDN applications and/or protocols [27]. Unlike Cefore-Sim, ndnSIM imposes several requirements to run a real implementation on the simulator. For instance, a tested application must use a subset of specific APIs provided, and it should not use disk operations to avoid undefined behavior, unless application instances access unique parts of the file system. It is not realistic for developers or experimenters to expect NDN real implementations to run on the simulator without modifications, which requires the duplicate effort of developing source codes for both the simulation and real implementations.

To build and use Cefore-Sim, NS-3 and DCE must first be installed on a Linux machine. To use the Cefore executable binary for Cefore-Sim, Cefore source codes need to be recompiled using some specific compile/link options, because DCE needs to relocate the executable binary in memory. Next, it is necessary to copy the target executable produced in a specified directory in the variable environment `DCE_PATH` so that DCE can find it (see [25] for more detail). To use the resulting Cefore executable, we need to write the NS-3 DCE scenario script with the help of a set of DCE Helper classes (e.g., DceManagerHelper and DceApplicationHelper). As in a typical NS-3 script without DCE, the other required descriptions (e.g., setup of the nodes and topology) must be specified in the NS-3 DCE script.

Cefore-Sim also provides a "hybrid simulation" environment in which the simulated Cefore nodes running on a single physical node can communicate and interact with other real physical Cefore nodes via real networks (e.g., Ethernet or WiFi). To enable this feature, Cefore-Sim utilizes "tap-bridge-module" provided by NS-3 as default. The tap-bridge creates a software channel between a virtual tap interface on the physical node and a simulated node in the NS-3 network, and enables the exchange of Interest/Data packets between these. We can use "TapBridgeHelper" in the scenario script to create the interface. This function can be useful for developers and experimenters when (1) real network environments must be combined with the simulated network in order to realize a more realistic scenario (e.g., various real-world traffic flows are required) and (2) the node running Cefore-Sim does not have sufficient computational capability (e.g., the experiment requires some applications requiring high CPU and memory usage, such as high-quality video streaming servers).

One of the concerns of using NS-3 simulated time clock is the timing accuracy with respect to real environments. Although Cefore-Sim using DCE provides accurate results whatever the scale of the experiment, it runs slower than real time depending on the scale of scenario used. Such conditions are undesirable when Cefore-Sim is connected to real networks via tap-bridge-module. However, this issue can be overcome by using Message Passing Interface (MPI)-based distributed simulation built into NS-3 [29].

5.2 Cefore-Sim in Action

Here, similar to Cefore-Emu, we present a simple experiment to examine the feature of the hybrid simulation provided by Cefore-Sim. The experiment considers a cache-enabled WiFi scenario in which a WiFi access-point enables in-network cache (csmgrd), as shown in Fig. 13. This experiment requires two physical nodes: one is used to run Cefore-Sim, and the other is used as a content publisher. Since the publisher with csmgrd requires a certain amount of CPU and memory, we separately use the non-simulated node as a content publisher to make the csmgrd stable. We investigate the in-network cache efficacy regarding the flow completion time (FCT) of Cefore content retrieval by the cefgetfile command, compared with TCP/IP-based content retrieval using HTTP (wget).

In this experiment, three WiFi consumers retrieve the content (about 5.3 MB) published by the producer through IEEE 802.11g using the NS-3 default settings. In the case of the Cefore experiment, the content is divided into small content objects by cefputfile, the sizes of which are 1024 bytes. Cefore consumers invoke cefgetfile to send the Interests for each data object until the download is completed.

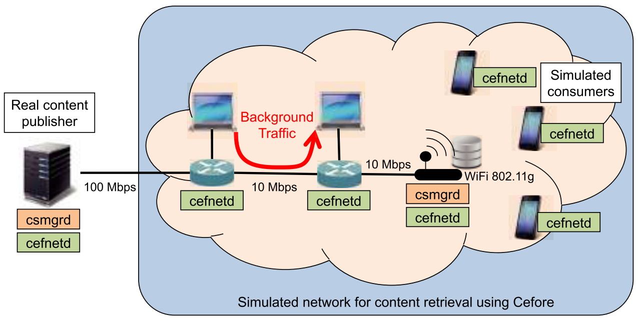


Fig. 13 Network topology used for the Cefore-Sim experimentation.

The pipeline size of cefgetfile, which represents the maximum number of outstanding Interests sent to the network, is set to 16 or 32. The Interest lifetime, defined as the time limit to expire in PIT, is set to 100 ms. Cefore consumers thus retransmit the Interest that could not receive the corresponding content object for 100 ms. In the case of the TCP/IP-based experiment, consumers download the content using wget. Considering a realistic scenario, the start time for each consumer varies using an exponential distribution with a mean value of 5 s and a maximum value of 10 s.

In both cases, the bandwidth/propagation-delay for each link in the NS-3 network is set to 10 Mbps/5 ms. The physical node running Cefore-Sim is connected to the real content publisher via an Ethernet link of 100 Mbps. Considering the realistic size of the in-network cache space affected by the ability to perform memory/storage lookups at line speed and/or the processing load required to store/forward content objects, the size of the in-network cache enabled by the WiFi AP is set to 1 GB. In the link between the two simulated routers, background traffic is generated as a UDP flow by using the iperf command tool to create a congested link. The UDP traffic rate varies to cause different rates of packet loss in the link in order to investigate how the loss conditions impact the performance of content retrieval in association with the WiFi in-network cache.

Figure 14 shows the average FCTs of wget using TCP/IP and Cefore's cefgetfile with and without in-network cache at the AP (hereinafter, referred to as cefgetfile-cache and cefgetfile-noncache, respectively). The FCTs of wget and cefgetfile-noncache become longer with increasing background traffic rate. Wget achieves a lower FCT (i.e., higher throughput) than that of cefgetfile-noncache with a pipeline size of 16 because the pipeline size cannot fully consume the available bandwidth due to the low Interest transmission rate. However, in the case of the pipeline size of 32, cefgetfile-noncache achieves a lower FCT than wget. This result indicates that the Cefore consumer needs to configure the pipeline size appropriately according to the network condition in order to achieve higher throughput. Note that, depending on the network conditions, cefgetfile with a substantially larger pipeline size adversely decreases the throughput due to further network congestion caused by a much larger volume of Interest/Data traffic.

On the other hand, cefgetfile-cache with a pipeline size

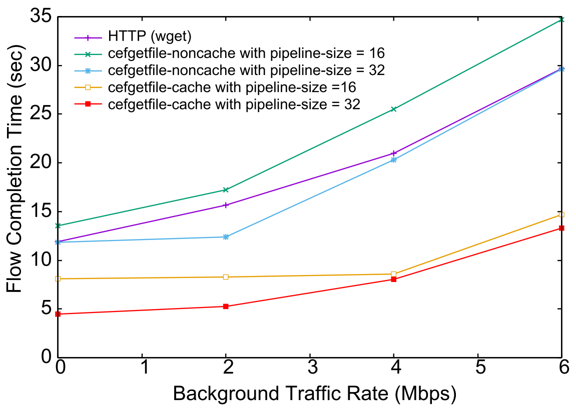


Fig. 14 Results for each flow completion time (FCT) under the background traffic using UDP at different rates.

Table 5 Throughput of cefnetd in simulation.

OS	CPU	Memory	Throughput
Ubuntu 14.04	2.5 GHz × 4	16 GB	303 Mbps

of 16 maintains an FCT of about 8 s, except in the case of background traffic of 6 Mpbs; the FCT is far superior to those of both wget and cefgetfile-noncache. For a pipeline size of 32, the FCT becomes better owing to the higher Interest transmission rate. Since the AP with in-network cache stores the received content objects during the first download attempt by a consumer, the second and third downloads of the content objects can be served directly by the AP, which makes the consumers likely to be unaffected by network congestion due to the background traffic. This result also demonstrates that the in-network cache enabled by the access point reduces the traffic load between the publisher and the in-network cache point, which is an important feature in view of the current rapid growth in mobile traffic volume. Although this experiment simply investigates the in-network efficacy, we can easily and flexibly execute more complex scenarios such as vertical/horizontal handover scenarios by using the wireless and mobility modules provided by NS-3. In this way, we can easily investigate and evaluate Cefore functions by using real Cefore implementations, which promotes further developments and improvements.

We also assessed the maximum throughput achieved by Cefore-Sim running two simulated nodes on the Ubuntu machine as shown in Table 5.

6. Conclusion and Outlook

Software platform development plays a fundamental role in researching and evaluating proposed protocols and mechanisms. To study the ICN/CCN architecture or future networking, existing ICN/CCN software platforms, still lack a focus on extensibility and lightweight implementation. To solve these problems, we developed the software platform Cefore, which satisfies the requirements of lightweight implementation, easy usage, and extensibility. Cefore can run on a resource-constrained node/device and is easily extensible through plugin libraries or external software implementations. For easy usage and scalable experiments, we also developed Cefore-Emu and Cefore-Sim whose design, specification, and usage method were elaborated in this paper.

We performed performance evaluations of in-network caching and streaming on Cefore-Emu and content fetch on Cefore-Sim, demonstrating the salient features of the Cefore software platform. These evaluations are made with the hybrid experimental environments that incorporate physical networks into the emulated/simulated networks. The hybrid experimental environments give a great chance to obtain precise performance or Quality of Experience (QoE) measurement for wireless communications, mobility, and streaming; therefore these implementations contribute to the future study of CCN technologies applied to IoT (Internet of Things) or M2M (Machine to Machine) environments or ultra-high-quality video streaming.

References

[1] R. Li, K. Matsuzono, H. Asaeda, and X. Fu, "Consecutive caching and adaptive retrieval for in-network big data sharing," Proc. IEEE ICC, Kansas City, USA, May 2018.

[2] R. Li and H. Asaeda, "MWBS: An efficient many-to-many wireless big data delivery scheme," IEEE Trans. Big Data, DOI: 10.1109/TBDATA.2018.2878584, 2018. (Accepted)

[3] K. Matsuzono and H. Asaeda, "NRTS: Content name-based real-time streaming," Proc. IEEE CCNC, Las Vegas, USA, Jan. 2016.

[4] K. Matsuzono, H. Asaeda, and T. Turletti, "Low latency low loss streaming using in-network coding and caching," Proc. IEEE Infocom, Atlanta, USA, May 2017.

[5] K. Matsuzono and H. Asaeda, "NMRTS: Content name-based mobile real-time streaming," IEEE Commun. Mag., vol.54, no.8, pp.92–98, Aug. 2016.

[6] H. Shimizu, H. Asaeda, M. Jibiki, and N. Nishinaga, "Local tree hunting: Finding the closest contents in an in-network cache," IEICE Trans. Inf. & Syst., vol.E98-D, no.3, pp.557–564, March 2015.

[7] A. Ooka, E. Suyong, S. Ata, and M. Murata, "Scalable cache component in ICN adaptable to various network traffic access patterns," IEICE Trans. Commun., vol.E101-B, no.1, pp.35–48, Jan. 2018.

[8] R. Li, H. Asaeda, and J. Li, "A distributed publisher-driven secure data sharing scheme for information-centric IoT," IEEE Internet Things J., vol.4, no.3, pp.791–803, June 2017.

[9] R. Li, H. Asaeda, and J. Wu, "DCAuth: Data-centric authentication for secure in-network big-data retrieval," IEEE Trans. Netw. Sci. Eng., DOI: 10.1109/TNSE.2018.2872049, 2018. (Accepted)

[10] H. Asaeda, R. Li, and N. Choi, "Container-based unified testbed for information-centric networking," IEEE Netw. Mag., vol.28, no.6, pp.60–66, Nov. 2014.

[11] H. Asaeda, K. Matsuzono, and T. Turletti, "Contrace: A tool for measuring and tracing content-centric networks," IEEE Commun. Mag., vol.53, no.3, pp.182–188, March 2015.

[12] H. Asaeda and X. Shao, "CCNinfo: Discovering content and network information in content-centric networks," IRTF Internet Draft (work in progress), Oct. 2018, Available at: https://tools.ietf.org/html/draft-irtf-icnrg-ccninfo

[13] "Cefore," Available at: https://cefore.net/, accessed Nov. 1, 2018.

[14] "CCNx," Available at: https://github.com/ProjectCCNx/ccnx, accessed Nov. 1, 2018.

[15] "CICN," Available at: https://wiki.fd.io/view/Cicn, accessed Nov. 1, 2018.

[16] "Named Data Networking," Available at: http://named-data.net/, accessed Nov. 1, 2018.

[17] M. Mosko, I. Solis, and C. Wood, "CCNx messages in TLV format," IRTF Internet-Draft (work in progress), July 2018, Available at: https://tools.ietf.org/html/draft-irtf-icnrg-ccnxmessages

[18] M. Mosko, I. Solis, and C. Wood, "CCNx Semantics," IRTF Internet-Draft (work in progress), June 2018, Available at: https://tools.ietf.org/html/draft-irtf-icnrg-ccnxsemantics

[19] IEEE SIG on Big Data Intelligent Networking (BDIN), https://github.com/IEEETCBDIN/Home/blob/master/index.md/, accessed Nov. 9, 2018.

[20] Computing in the Network (COIN), https://trac.ietf.org/trac/irtf/wiki/coin, accessed Nov. 1, 2018.

[21] "Mininet," Available at: http://mininet.org/, accessed Nov. 1, 2018.

[22] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," Proc. ACM CoNEXT'12, Dec. 2012.

[23] "CAIDA Data," Available at: https://www.caida.org/data/, accessed Nov. 1, 2018.

[24] J. Ratkiewicz, S. Fortunato, A. Flammini, F. Menczer, and A. Vespignani, "Characterizing and modeling the dynamics of online popularity," Phys. Rev. Lett., vol.105, no.15, p.158701, Oct. 2010.

[25] "Direct Code Execution (DCE)," Available at: https://github.com/direct-code-execution/ns-3-dce, accessed Jan. 26, 2019. direct-code-execution/, accessed Nov. 1, 2018.

[26] "NS-3: discrete-event network simulator for Internet systems," Available at: https://www.nsnam.org/, accessed Nov. 1, 2018.

[27] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2: An updated NDN simulator for NS-3," NDN Project, Technical Report NDN-0028, Revision 2, 2016.

[28] L. Muscariello, "CICN community information-centric networking," Tutorial at ACM SIGCOMM ICN, Sept. 2017.

[29] H. Tazaki, F. Urbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, and W. Dabbous, "Direct code execution: Revisiting library OS architecture for reproducible network experiments," Proc. ACM Conext, California, USA, Dec. 2013.

Hitoshi Asaeda is an executive researcher at the Network System Research Institute, National Institute of Information and Communications Technology (NICT). He received a Ph.D. from Keio University. He previously worked at IBM Japan, Ltd. and as a research engineer specialist at INRIA, France. He was a project associate professor at Keio University in 2005–2012. He is a chair of the IEICE technical committee on ICN and a program officer for several international projects. He serves as a TPC member for premier conferences such as IEEE Infocom, WCNC and ACM ICN, and was a guest editor-in-chief of the special series of IEICE Trans. Commun. in 2016. His research interests include routing, streaming, distributed computing, and large-scale testbeds. He is actively working in the IETF standards body. He is a senior member of the IEEE and a member of the ACM.

Atsushi Ooka is a researcher at the Network System Research Institute, NICT. He received M.E. and Ph.D. degrees in the Graduate School of Information Science and Technology, Osaka University in 2014 and 2017, respectively. His research interests include the design and implementation of a router in content-centric networking. He is a member of the IEEE.

Kazuhisa Matsuzono is a researcher at the Network System Research Institute, NICT. He received a Ph.D. from Keio University in 2012. He was a post-doctoral fellow at INRIA in 2013. His research interests include transport protocols for multimedia flows, network coding, and information-centric networks. He is a member of the IEEE.

Ruidong Li received a bachelor's degree in engineering from Zhejiang University, China in 2001 and a doctorate of engineering from the University of Tsukuba in 2008. He is a senior researcher at the Network System Research Institute, NICT. He is a chair of the IEEE SIG on Big Data Intelligent Networking and serves as co-chair for the Young Researcher Group in the Asia Future Internet Forum (AsiaFI). His current research interests include future networks, big data networking, information-centric networks, internet of things, network security, and wireless networks. He is a member of the IEEE.