CCNx-based Cloud-Native Function: Networking and Applications

Yusaku Hayamizu, Atsushi Ooka, Kazuhisa Matsuzono, Hitoshi Asaeda

National Institute of Information and Communications Technology (NICT), Japan

19-21 Sept. 2022

This work was supported in part by the JSPS KAKENHI under Grant Numbers 22K17888.
CCNx-based Cloud-Native Function: Networking and Applications

Yusaku Hayamizu, Atsushi Ooka, Kazuhisa Matsuzono, Hitoshi Asaeda

National Institute of Information and Communications Technology (NICT), Japan

19-21 Sept. 2022
Contents of this tutorial

Check the detailed information of the tutorial in the web*

**Half-day Tutorial: CCNx-based Cloud-Native Function: Networking and Applications**

*https://conferences2.sigcomm.org/acm-icn/2022/tutorial-cefore.html*
Cefore and its Integration with Docker Platforms

Yusaku Hayamizu, Atsushi Ooka, Kazuhisa Matsuzono, Hitoshi Asaeda

National Institute of Information and Communications Technology (NICT), Japan

19-21 Sept. 2022
• Background/Motivation
• Cefore: CCNx-based Extensible Packet Forwarding Engine
• Cefore x Docker Integration
• Sample Scenarios
• Conclusion
Information-Centric Networking (ICN)

- Information-Centric Networking [1]
  - A user retrieves information (contents) by *name* instead of host ID e.g. IP address
  - *in-network caching* (CS: Content Store) enables efficient information delivery
  - supports *multicast communications* by Interest aggregation
• ICN [1]
  – changing NW from “host-centric” to “content-centric”
• CCNx [2,3] / NDN [4]
  – Content-Centric Networking or Named-Data Networking
• Cefore [8]
  – open-source software enabling ICN communications
  – CCNx1.0-compliant packet forwarding engine developed/maintained by NICT
• One missing piece might be...
  – a deployment solution of developed ICN modules into the Internet infrastructures
Aim of Tutorial

1. Introduction of Cefore
   - the Cefore software platform for enabling CCNx-based communications

2. Cefore/Docker integration
   - Cefore’s integration with the emerging Docker technologies for rapid and scalable deployment of ICN
   - NETWORKING

3. Application development with Cefpyco*
   - a Python wrapper program that helps developing CCNx applications
   - APPLICATION

*NOTE: 2nd speaker will present this part
Cefore: CCNx-based Extensible Packet Forwarding Engine
Cefore: Software platform for CCNx-based communications

- CCNx1.0
  - defined in the RFCs 8569 and 8609
  - standardized by IRTF ICNRG

- Cefore
  - originally designed in 2016
  - CCNx1.0 packet (Interest/ContentObject) forwarding/caching engine
  - developed / maintained by NICT
  - open-source, and published in the web* and github+

* https://cefore.net/
+ https://github.com/cefore
Design policies

• Lightweight
  – the software implementation should be compact
  – the platform should be usable for resource-constrained devices, such as sensor nodes

• Usability
  – 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

• Extensibility
  – the platform should be easily extensible to accommodate novel functions to satisfy future network needs
Pluggable architecture of Cefore

- Researchers can install necessary ICN functions depending on their requirements while considering their machine resource constraints

- High-performance
  - router
  - PC, smart phone
  - Raspberry Pi, sensor

- Lightweight

- Packet forwarding: enabled, disabled
- Caching

- 1. Add various computing functions such as caching
- 2. Use additional transport plugin for efficient video streaming transfer
- 3. Just support forwarding function and basic operation
Core components

• cefnetd
  – handles Interest & ContentObject packets as the core packet forwarding daemon
  – a minimum set of ICN functions (FIB&PIT) to achieve the lightweight implementation
  – other compute-intensive functions (e.g. caching and computing) are implemented using plugins or external daemons for providing extensibility and usability
  – [optional] lightweight local-caching function

• csmgrd
  – an external cache daemon interacting with cefnetd, behaving as Content Store (CS)
  – connects to cefnetd via a local socket or TCP

• configuration
  – cefnetd.conf/csmgrd.conf
    • we can tune-up parameters dominant for network performance such as FIB/PIT size, CS capacity, etc.
Plugin extension 1: cefnetd

- forwarding strategy plugin
  - default
  - shortest_path
  - flooding
  - etc

- transport plugin
  - sample tp
  - etc

- xxx plugin
  - new plugins developed in the future

Researchers can develop new mechanisms without modifying codes of the core daemons, i.e., cefnetd/csmgrd by using plugin extension.

Researchers can freely modify and create another “forwarding-strategy/transport” plugin.
Plugin extension 2: csmgrd

- cache plugin
  - FIFO: First-In, First-Out
  - LRU: Least Recently Used
  - LFU: Least Frequently Used

- yyy plugin
  - new plugins developed in the future

- Researchers can develop new mechanisms without modifying codes of the core daemons, i.e., cefnetd/csmgrd
- Researchers can freely modify and create another “cache” plugin
Tools / Utilities 1: cefgetfile/cefputfile

• cefgetfile
  – consumer-like application
  – a sample program for downloading a named content with simple Interest pipelining
  – uses Regular Interest (RGI) for data retrieval

• cefputfile
  – producer-like application
  – a sample program for uploading a named content to CS (csmgrd) running on the localhost
  – ContentObject packets are delivered from the CS to the consumer running cefgetfile
Tools / Utilities 2: cefgetstream/cefputstream

• cefgetstream
  – a consumer-like application
  – a sample program for receiving stream data, e.g. real-time video streaming
  – uses Symbolic Interest* for efficient data transfer of the streaming content

• cefputstream
  – a producer-like application
  – a sample program for sending stream data to downward nodes
  – can control the sending rate of data stream with -r option

Tools / Utilities 3: CCNinfo

• CCNinfo*
  – CCNx network management tool
    • discovery detailed information of CCNx network
      • routing path information
      • RTT between the content forwarder (cache/producer) and the consumer
      • states of in-network cached content (lifetime, etc)
  – the specification is defined in the IRTF ICNRG [10]

*https://datatracker.ietf.org/doc/draft-irtf-icnrg-ccninfo/
## Cefore Components

<table>
<thead>
<tr>
<th>Name/Item</th>
<th>Type</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cefnetd</td>
<td>daemon</td>
<td>Standard</td>
<td>Forwarding daemon</td>
</tr>
<tr>
<td>cefnetdstart</td>
<td>utility</td>
<td>Standard</td>
<td>Utility of starting cefnetd</td>
</tr>
<tr>
<td>cefnetdstop</td>
<td>utility</td>
<td>Standard</td>
<td>Utility of stopping cefnetd</td>
</tr>
<tr>
<td>cefstatus</td>
<td>utility</td>
<td>Standard</td>
<td>Utility of showing cefnetd status on stdout</td>
</tr>
<tr>
<td>cefroute</td>
<td>utility</td>
<td>Standard</td>
<td>Utility of set up cefnetd FIB</td>
</tr>
<tr>
<td>cefctrl</td>
<td>tool</td>
<td>Standard</td>
<td>Function called by cefnetdstop, cefstatus, and cefroute</td>
</tr>
<tr>
<td>cefgetchunk</td>
<td>tool</td>
<td>Standard</td>
<td>Obtain the specified Cob and show the payload on stdout</td>
</tr>
<tr>
<td>cefputfile</td>
<td>tool</td>
<td>Standard</td>
<td>Convert the file to Named Cobs and transmit them to Cefore</td>
</tr>
<tr>
<td>cefgetfile</td>
<td>tool</td>
<td>Standard</td>
<td>Create file from content received by Cefore</td>
</tr>
<tr>
<td>cefputstream</td>
<td>tool</td>
<td>Standard</td>
<td>Convert the stream received from stdin to Named Cobs and transmit them to Cefore</td>
</tr>
<tr>
<td>cefgetstream</td>
<td>tool</td>
<td>Standard</td>
<td>Display the stream received by Cefore on stdout</td>
</tr>
<tr>
<td>cefputfile_sec</td>
<td>tool</td>
<td>develop</td>
<td>Obtain security content from Cefore and output it as a file</td>
</tr>
<tr>
<td>cefgetfile_sec</td>
<td>tool</td>
<td>develop</td>
<td>Convert a file to Named Cob with security features and input it into Cefore</td>
</tr>
<tr>
<td>cefping</td>
<td>tool</td>
<td>cefping</td>
<td>cefping</td>
</tr>
<tr>
<td>cefinfo</td>
<td>tool</td>
<td>cefinfo</td>
<td>cefinfo (aka ccninfo)</td>
</tr>
<tr>
<td>csmgrd</td>
<td>daemon</td>
<td>csmgr</td>
<td>Content Store manager daemon</td>
</tr>
<tr>
<td>csmgrdstart</td>
<td>utility</td>
<td>csmgr</td>
<td>Utility of starting csmgr daemon</td>
</tr>
<tr>
<td>csmgrdstop</td>
<td>utility</td>
<td>csmgr</td>
<td>Utility of stopping csmgr daemon</td>
</tr>
<tr>
<td>csmgrstatus</td>
<td>utility</td>
<td>csmgr</td>
<td>Utility of showing csmgr status on stdout</td>
</tr>
<tr>
<td>Sample Transport</td>
<td>plugin</td>
<td>samtp</td>
<td>Sample transport plugin library</td>
</tr>
<tr>
<td>cefore.lua</td>
<td>application</td>
<td>Standard</td>
<td>Wireshark's LUA script file</td>
</tr>
</tbody>
</table>

*Details can be found at: https://cefore.net/doc/Readme.html*
``All-in-one package” for CCNx-based communications

- Cefore provides “all-in-one package”

Cefore provides a comprehensive package for CCNx-based communications, including:

- CCNx forwarder (cefnetd)
- External cache (csmgrd)
- External interfaces for consumer and producer:
  - cefgetfile
  - cefgetstream
  - ccninfo
  - etc.
  - cefputfile
  - cefputstream
  - ccninfo
  - etc.

The system is modular with pluggable functions for:

- Forwarding strategy
- Transport
- Etc.

Additionally, there are plugins for:

- Cache plugin/algorithm
- Etc.
Related software program: Cefpyco

- Cefpyco (CEFore Python Compact package)*
  - a Python-based wrapper program that help developing CCNx applications running with Cefore
  - enables easy coding for python programmers (compared to the original C language)
  - Example: sending an Interest packet

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <ctype.h>
#include <cefore/cef_define.h>
#include <cefore/cef_client.h>
#include <cefore/cef_frame.h>
#include <cefore/cef_log.h>

int main(int argc, char *argv[])
{
    CefT_Client_Handle fhdl;
    CefT_Interest_TLVs params_i;
    int res;
    cef_log_init("cefpyco");
    cef_frame_init();
    res = cef_client_init(port_num, conf_path);
    if (res < 0) return -1;
    fhdl = cef_client_connect();
    if (fhdl < 1) return -1;
    memset(&params_i, 0, sizeof(CefT_Interest_TLVs));
    res = cef_frame_conversion_uri_to_name("ccnx:/test", params_i.name);
    if (res < 0) return -1; // Failed to convert URI to name.
    params_i.name_len = res;
    params_i.hoplimit = 32;
    params_i.opt.lifetime_f = 1;
    params_i.opt.lifetime = 4000ull; /* 4 seconds */
    params_i.opt.symbolic_f = CefC_T_OPT_REGULAR;
    params_i.chunk_num_f = 1;
    params_i.chunk_num = 0;
    cef_client_interest_input(fhdl, &params_i);
    if (fhdl > 0) cef_client_close(fhdl);
    return 0;
}
```

```python
import cefpyco

with cefpyco.create_handle() as h:
    h.send_interest("ccnx:/test", 0)
```

**NOTE:** the 2nd speaker will deliver the presentation about Cefpyco in detail

*https://github.com/cefore/cefpyco*
Specification

- **OS**
  - Linux (Ubuntu 18.04 or later)
  - macOS (10.15 or later)

- **Packet format**
  - CCNx1.0
  - Type-Length-Value (TLV) format
  - NICT original functions -> Optional Hop-by-hop header

- **Networking**
  - TCP/UDP over IP (overlay)
Cefore x Docker Integration
What is Docker?

- Docker
  - a platform of container-based virtualization technology for quick and scalable deployment of network services

- Benefits
  - Lightweight
    - a Docker container is very lightweight compared with VM
    - we can build many containers in one physical machine
    - this enriches evaluation scenario of ICN networks and improves scalability of experiments
  - Performance
    - Docker containers do not contain OS
    - they can be easily and quickly initiated and terminated
    - this facilitates comfortable test and evaluation of ICN services
  - Scalability
    - there is a requirement that multiple ICN nodes providing different functions co-exist in a network
    - the concept of microservices that each service image is built for each purpose fits this requirement
    - useful option tools such as `docker-compose` can be used for flexibly and quickly setting up Docker containers
Preparation – install Docker

- Ubuntu 20.04
  - follow the official introduction
    - https://docs.docker.com/engine/install/ubuntu/

- macOS
  - web
    - https://www.docker.com/products/docker-desktop/
  - CUI

    ```
    cefore ~ % brew install docker [--cask]
    cefore ~ % open /Applications/Docker.app
    ```

- Windows
  - web
    - https://www.docker.com/products/docker-desktop/
### Specification of host machine

- **CPU**
  - min: 4 cores
  - recommended: 8 cores
  
  * for macOS, the Intel chips are recommended not the Apple silicon chips (M1/M2)

- **Memory**
  - min: 4 GB
  - recommended: 8 GB

NOTE: The host machine spec should be considered according to the scenario of experiments which you want to run with docker containers.
Example scenario of Cefore/Docker-based networking

- Scenario
  - The consumer requests a file
  - The producer responses to the request and send back data
  - The CCNx router stores received data into CS (csmgrd)
Example 1 – writing a Dockerfile

- define a microservice as a ``base” service
  - base function as an ICN node
  - necessary functions for providing ICN services as a container node

base/Dockerfile

```docker
FROM ubuntu:20.04
LABEL maintainer="hayamizu <hayamizu@nict.go.jp>"
RUN mkdir -p /cefore
WORKDIR /cefore
RUN apt update
RUN apt install -y git build-essential libssl-dev automake
RUN apt -y clean
RUN git clone https://github.com/cefore/cefore.git
WORKDIR /cefore/runner_test
```

Afterward, other enhanced ICN services, e.g. ``min” and ``cache,” inherit this ``base” image
Example 2 – writing a Dockerfile

• define a microservice as a `min` service
  – minimum functions serving as a ICN node, i.e., installation & app. preparation

min/Dockerfile

```dlang
FROM cefore/base
WORKDIR /cefore/cefore
RUN ./configure
RUN make; make install; make clean
RUN ldconfig
ENV USER root
COPY ./entrypoint.bash /cefore
ENTRYPOINT /cefore/entrypoint.bash
```

configure & make & install Cefore
set the entrypoint, i.e., just starting Cefore daemon (cefnetd)

define a service “min” that provide minimum ICN functions (application tools)
Example 3 – writing a Dockerfile

- define a microservice as a ```cache```` service

```Dockerfile
FROM cefore/base
WORKDIR /cefore/
core
RUN ./configure --enable-cache --enable-csmgr
RUN make; make install; make clean
RUN ldconfig
RUN echo "CS_MODE=2" > /usr/local/cefore/cefnetd.conf
RUN echo "CACHE_TYPE=memory" > /usr/local/cefore/csmgrd.conf
ENV USER root
COPY ./entrypoint.bash /cefore
ENTRYPOINT /cefore/entrypoint.bash
```

- configure Cefore by enabling ```csmgr/cache```` option
- make & install Cefore
- modify the configuration files.
  - CS_MODE=2 (csmgrd)
  - CACHE_TYPE=memory
- set the entrypoint, i.e., starting Cefore daemons
  (cefnetd & csmgrd)

define “cache” service by adding caching function (cache/csmgrd) to the base ICN functions
• **docker-compose**
  - a tool for defining and running multi-container Docker applications
  - easy service configuration using a YAML file
  - can create and start all the services from the configuration with a single command

-> easy to conduct scenario-based experiments (emulations) like network simulations such as ns-3

*https://docs.docker.com/compose/*

---

Example: `docker-compose.yml`

```yaml
version: "3.3"
services:
  producer:
    image: cefore/cache
    container_name: "producer"
    hostname: "producer"
    working_dir: "/cefore"
    networks:
      downward:
        ipv4_address: 10.0.1.10
  router:
    image: cefore/cache
    container_name: "router"
    hostname: "router"
    working_dir: "/cefore"
    networks:
      downward:
        ipv4_address: 10.0.1.20
  consumer:
    image: cefore/min
    container_name: "consumer"
    hostname: "consumer"
    working_dir: "/cefore"
    networks:
      downward:
        ipv4_address: 10.0.1.100
networks:
  downward:
    name: downward
driver: bridge
ipam:
  driver: default
config:
  - subnet: 10.0.1.0/24
```
Cefore/Docker Basic Operation
Manual installation of Cefore

- Downloading source codes
  - [https://cefore.net/](https://cefore.net/)
  - [https://github.com/cefore/cefore](https://github.com/cefore/cefore)

- Installing dependencies

  ```
  $ sudo apt-get install libssl-dev automake
  ```

- Installing Cefore

  ```
  $ unzip cefore-0.9.0b.zip
  $ cd cefore-0.9.0b
  $ autoconf
  $ automake
  $ ./configure --enable-csmgr --enable-cache
  $ make
  $ sudo make install
  $ sudo ldconfig
  # binaries are to be installed in the /usr/local/bin, sbin
  ```

Please see more details Section 2 “Installation” of README.
*https://cefore.net/doc/Readme.html*
Starting Docker containers

• Build a Docker image

```bash
% docker build -f Dockerfile -t cefore/base
```

• Check the status of built container images

```bash
% docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
cefore/cache latest 59ef9d12b859 12 minutes ago 881MB
cefore/min latest 935747e7cc84 12 minutes ago 881MB
cefore/base latest 3694f2af7dd7 13 minutes ago 873MB
```

• Start a container

```bash
% docker run --name consumer -it IMAGE_ID /bin/bash
root@7db7391ba03c:/cefore/runner_test#
```

• Login to the container (NOTE: run ``exec'' command in another terminal)

```bash
% docker exec -it consumer /bin/bash
root@7db7391ba03c:/cefore/runner_test#
```
Stopping Docker containers

- Stop a Docker container
  
  ```bash
  root@d731a71974f:/cefore/cefore# exit
  
  or
  
  % docker stop CONTAINER_ID
  ```

- Remove a container
  ```bash
  % docker rm -f CONTAINER_ID
  ```

- Remove a container image
  ```bash
  % docker rmi IMAGE_ID
  ```

- Purge all the build cache and images*
  ```bash
  % docker builder prune
  WARNING! This will remove all dangling build cache. Are you sure you want to continue? [y/N] y
  ```

* This command should be run carefully
Starting / Stopping daemons (cefnetd/csmgrd)

- Start cefnetd
  ```
  % cefnetdstart
  ```

- Stop cefnetd
  ```
  % cefnetdstop
  ```

- Start csmgrd
  ```
  % csmgrdstart
  ```

- Stop csmgrd
  ```
  % csmgrdstop
  ```

NOTE: When you configure to use both cefnetd and csmgrd, first you need to start csmgrd, and then start cefnetd.
• Checking the status of cefnetd
  – cefstatus
    • CCNx ver.
    • Rx/Tx Interest #
    • Rx/Tx ContentObject #
    • Cache Mode
    • Face Table
    • FIB
    • PIT

  – cefstatus -v
    • confirm the version of Cefore running on the container

```bash
root@router:/cefore# cefstatus
Version  : 1
Port     : 9896
Rx Interest : 0 (RGL[0], SYM[0], SEL[0])
Tx Interest : 0 (RGL[0], SYM[0], SEL[0])
Rx ContentObject : 0
Tx ContentObject : 0
Cache Mode  : Excache
Faces : 6
  faceid = 4 : IPv4 Listen face (udp)
  faceid = 0 : Local face
  faceid = 5 : IPv6 Listen face (udp)
  faceid = 6 : IPv4 Listen face (tcp)
  faceid = 16 : Local face
  faceid = 7 : IPv6 Listen face (tcp)
FIB(App) :
  Entry is empty
FIB :
  Entry is empty
PIT(App) :
  Entry is empty
PIT :
  Entry is empty
```

```bash
root@consumer:/cefore# cefstatus -v
Cefore version 0.9.0b
```
Checking the status of csmgrd

- Checking the status of csmgrd
  - csmgrstatus NAME_PREFIX

+ initial state

```
root@producer:/cefore# csmgrstatus ccnx:
```

Connect to 127.0.0.1:9799  
***** Connection Status Report  *****
All Connection Num : 1  

***** Cache Status Report  *****
Number of Cached Contents : 0

+ after putting the 3 contents

```
root@producer:/cefore# csmgrstatus ccnx:
```

Connect to 127.0.0.1:9799  
***** Connection Status Report  *****
All Connection Num : 1  

***** Cache Status Report  *****
Number of Cached Contents : 3

```
[0] Content Name : ccnx:/ccc  
Version : None  
Content Size : 4 Bytes  
Cache Hit : 0  
Request Count : 0  
Freshness : 290 Sec  
Elapsed Time : 8 Sec
```

```
[1] Content Name : ccnx:/bbb  
Version : None  
Content Size : 4 Bytes  
Cache Hit : 0  
Request Count : 0  
Freshness : 283 Sec  
Elapsed Time : 14 Sec
```

```
[2] Content Name : ccnx:/aaa  
Version : None  
Content Size : 4 Bytes  
Cache Hit : 0  
Request Count : 0  
Freshness : 275 Sec  
Elapsed Time : 22 Sec
```

+ specify name prefix

```
root@producer:/cefore# csmgrstatus ccnx:/aaa
```

Connect to 127.0.0.1:9799  
***** Connection Status Report  *****
All Connection Num : 1  

***** Cache Status Report  *****
Number of Cached Contents : 1

```
[0] Content Name : ccnx:/aaa  
Version : None  
Content Size : 4 Bytes  
Cache Hit : 0  
Request Count : 0  
Freshness : 268 Sec  
Elapsed Time : 29 Sec
```
- **cefroute**
  - Insertion
    - `cefroute add ccnx:/aaa udp 10.0.0.1`
  - Deletion
    - `cefroute del ccnx:/aaa udp 10.0.0.1`
- **Alternative: preparing an FIB configuration file**
  - `/usr/local/cefore/cefnetd.fib`
    - Cefnetd automatically loads this file when starting its process
- **Routing Protocol**
  - TBA

```
root@producer:/cefore# cat /usr/local/cefore/cefnetd.fib
ccnx:/example udp 10.0.1.1
```

+ Setting FIB using cefnetd.fib
```
root@producer:/cefore# cefstatus
Version : 1
Port : 9896
Rx Interest : 0 (RGL[0], SYM[0], SEL[0])
Tx Interest : 0 (RGL[0], SYM[0], SEL[0])
Rx ContentObject : 0
Tx ContentObject : 0
Cache Mode : Excache
Controller : 192.168.0.99
Faces : 7
  faceid = 4 : IPv4 Listen face (udp)
  faceid = 0 : Local face
  faceid = 16 : address = 10.0.1.1:9896 (udp)
  faceid = 17 : Local face
  faceid = 5 : IPv6 Listen face (udp)
  faceid = 6 : IPv4 Listen face (tcp)
  faceid = 7 : IPv6 Listen face (tcp)
FIB(App) :
  Entry is empty
FIB : 1
  ccnx:/example
    Faces : 16 (-s-) RtCost=0
PIT(App) :
  Entry is empty
PIT :
  Entry is empty
```
Tuning up Cefore

- **Primary parameters**
  - cefnetd.conf
    - CS_MODE
      - 0: no content store [default]
      - 1: cefnetd’s local cache
      - 2: external content store (csgmrd)
    - FORWARDING_STRATEGY
      - default: Forward the Interest to a face in the longest-prefix-matched(LPMed) FIB entry [default]
      - flooding: Forward the Interest to all the faces registered in the LPMed FIB entry
      - shortest_path: Forward the Interest to the face that has the minimum routing cost in the LPMed FIB entry
  - csmgrd.conf
    - CACHE_CAPACITY
      - The maximum number of cached ContentObjects in csmgrd
      - 819,200 [default]
    - CACHE_TYPE
      - filesystem: cache located on UNIX filesystem [default]
      - memory: cache located on memory (RAM)
    - CACHE_ALGORITHM
      - libcsmgrd_fifo
      - libcsmgrd_lru
      - libcsmgrd_lfu
### Configuration parameters*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEF_LOG_LEVEL</td>
<td>Specifies the log output type for the cefnetd.</td>
<td>0</td>
</tr>
<tr>
<td>PORT_NUM</td>
<td>Port number cefnetd uses.</td>
<td>9896</td>
</tr>
<tr>
<td>PIT_SIZE</td>
<td>Max number of PIT entries.</td>
<td>2048</td>
</tr>
<tr>
<td>FIB_SIZE</td>
<td>Max number of FIB entries.</td>
<td>1024</td>
</tr>
<tr>
<td>CS_MODE</td>
<td>ContentStore mode Cefore uses.</td>
<td>0: No cache used 1: cefnetd's local cache 2: csmgrd</td>
</tr>
<tr>
<td>LOCAL_CACHE_CAPACITY</td>
<td>Max number of Cobs to use for the local cache in cefnetd.</td>
<td>65535</td>
</tr>
<tr>
<td>CSMGR_NODE</td>
<td>csmgrd's IP address</td>
<td>localhost</td>
</tr>
<tr>
<td>CSMGR_PORT_NUM</td>
<td>TCP port number used by csmgrd to connect cefnetd.</td>
<td>9799</td>
</tr>
<tr>
<td>FORWARDING_STRATEGY</td>
<td>Forwarding strategy when sending Interest messages.</td>
<td>0</td>
</tr>
</tbody>
</table>

*README is available at: https://cefore.net/doc/Readme.html*
• Tuning socket buffer size
  – Ubuntu
    
    ```
    $ sudo sysctl -w net.core.rmem_default=10000000
    $ sudo sysctl -w net.core.wmem_default=10000000
    $ sudo sysctl -w net.core.rmem_max=10000000
    $ sudo sysctl -w net.core.wmem_max=10000000
    ```

  – macOS
    
    ```
    $ sudo sysctl -w net.local.stream.sendspace=10000000
    $ sudo sysctl -w net.local.stream.recvspace=10000000
    ```

NOTE: Experientially, we would recommend to increase the socket buffer size of kernel parameters in advance, when you conduct an experiment with high-speed data rate.
Sample Scenarios
Basic scenario – file transfer

- Scenario
  - producer puts a video file (ccnx:/video.mp4) to its csmgrd with cefputfile
  - consumer requests the cached content with cefgetfile
  - check statistics information of consumer (throughput, latency, etc)
  - check the status of cefnetd/csmgrd in the router node
Basic scenario – Video streaming

- Scenario
  - producer publishes the stream data (video.mp4) toward the consumer with cefputstream
  - consumer sends Symbolic Interest to receive the data with cefgetstream
  - The host OS is waiting for the playback with ffplay in advance
• IEICE ICN summer workshop 2021 [fully-online]
  – Cefore/Docker hands-on
  – Multicast video streaming using Cefore/Docker platforms*
    • The producer is located at NICT (Tokyo)
    • The consumers receive the video streaming from their homes/schools/companies
*You can get sample codes from https://github.com/cefore/2021-hands-on [materials are in Japanese only]
Conclusion

• Cefore
  – CCNx-based extensible packet forwarding engine
  – All-in-one package for CCNx-based communications

• Docker integration
  – Quick and scalable deployment of CCNx functions

• Sample scenarios
  – File transfer
  – Video streaming

• Future work
  – A possibility of collaboration with the emerging Docker orchestration technologies such as Kubernetes


Thank you.