一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
CS2019-68(2019-11)

Dockerによる仮想環境を用いたNDN-FCの設計

熊本　洋平†　　中里　秀則†

† 早稲田大学基幹理工学研究科情報理工・情報通信専攻　〒169–0072 東京都新宿区大久保 3–14–9
E-mail: †yoheimmtw@akane.waseda.jp, nakazato@waseda.jp

あらまし　近年 IoT デバイスが急増し，その結果 IoT アプリケーションやサービスも発展している．ファクトリーオートメーションや自動運転などがその例であるが，これらは低遅延が必要とされる．そこで，より効率的な IoT ネットワークを作成する手段として，Named Data Networking (NDN) と Service Function Chaining (SFC) を組み合わせた NDN-FC と呼ばれる手法が提案されている．本研究では，コンテナ型の仮想化技術を実現する Docker を用いて，より軽量で容易に管理できる NDN-FC を実装する手法を検討した．

Design of NDN-FC Using Virtual Environment by Docker

Yohei KUMAMOTO† and Hidenori NAKAZATO†

† Department of Computer Science and Communications Engineering, Graduate School of Fundamental Science and Engineering, Waseda University　Okubo 3–14–9, Shinjuku-ku, Tokyo 169–0072 Japan
E-mail: †yoheimmtw@akane.waseda.jp, nakazato@waseda.jp

Abstract　In recent years, the number of IoT devices has increased rapidly, resulting in the development of IoT applications and services. Examples include factory automation and automated driving, and these require low latency. Therefore, a method called NDN-FC that combines Named Data Networking (NDN) and Service Function Chaining (SFC) has been proposed as a means of creating a more efficient IoT network. In this study, we examined a method for implementing NDN-FC that is lighter and easier to manage using Docker, which realizes container-type virtualization technology.

1. 研究背景

Docker は，コンテナ型の仮想化を実現するオープンソフトウェアとして，近年開発者やシステム管理者等に広く利用されている．Docker コンテナは軽量で，起動時間が短く，また Docker イメージから同じコンテナを短時間で何個も複製できるという特徴がある．また Docker は，コードの容易な移動を実現する点や，コンテナの管理の容易さ，物理環境に依存しない性質など，開発者にとって嬉しい機能が揃っている [1].

現在、注目され研究開発が行われ、また実利用も始まっている Interneto of Things (IoT) にコンテンツ指向ネットワーク (ICN) 技術 を活用することが提案されている [2]．IoT システムでは，大量の IoT デバイスや低遅延が必要なアプリケーションに対応するため，IoT デバイス近傍で情報処理を行うエッジコンピューティングが活用される．エッジコンピューティングを利用してさまざまな処理を実現する上で， 複数の処理 (Function) を連携させて処理を行う，Service Function Chaining (SFC) を利用することは有効である。

我々は、ICN 環境で SFC を実現する仕組みとして、ICN の実装である Named Data Networking (NDN) において、SFC を実現する NDN-FC を提案し、実装を行っている [3] [4]． これまで NDN-FC は仮想マシンとして実装された Function を連携させる手法で実装が行われてきたが，これらを仮想マシンではなく，Docker コンテナにより実装することで，システムの軽量化，また複製の容易さから多様なトポロジの形成が実現できると考えられる．

本研究では、現在仮想マシンによる実装が行われている NDN-FC の Function を，以上のような特徴を持つ Docker コンテナで実装し，動作検証/性能評価を行った．さらには， NDN の Function を Docker コンテナにより実装することで，動的な Function 配備を目指す展望についても検討する．

2. 関連研究

2.1 Named Data Networking (NDN)

NDN は，ICN の実装の一つである [5]．NDN では，IP アドレス等の，コンテンツを収容するホストの位置情報を元に通

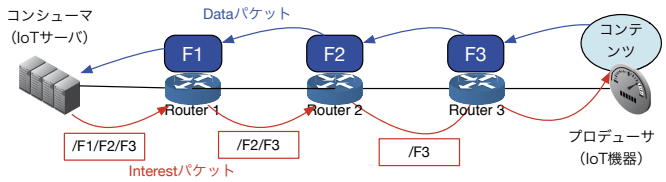

図 1　NDN-FC における SFC

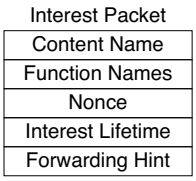

図 2　NDN-FC における Interest パケットのフォーマット

信するのではなく、各コンテンツに付与されたコンテンツ名をルータで識別し，パケットに含まれるコンテンツ名によってパケットのルーティングが行われる．また，サーバ以外のネットワークノード（ルータ）にもキャッシュ機能を持たせ，ネットワークトラヒックを軽減するという特徴を持つ．NDN における通信は Interest パケットと Data パケットの 2 種類によって行われ、前者はコンテンツ名によりコンテンツを要求するために、また後者は要求されたコンテンツを配送するためにそれぞれ利用される．

NDN のルーターでは、Forwarding Information Base (FIB)，Pending Interest Table (PIT)，Content Store (CS) と呼ばれる 3 つのデータ構造を用いてパケットの配送を行う．FIB は、コンテンツ名とネットワークインターフェース (フェイスと呼ぶ) を示す情報が組で格納されたルーティングテーブルで、Interest パケットを転送する際に，その送出フェイスを決めるため利用される．PIT は，到着した Interest パケットのコンテンツ名と，その Interest パケットを受信したフェイスの組を記録する．Data パケットはこの情報を利用して，Interest パケットが通った経路を戻るようにルーティングされ，要求されたコンテンツが要求元に配送される．CS は，Data パケットによって返送されてきたコンテンツを一時的に記憶しておくデータ構造で，コンテンツ名とコンテンツそのものが格納されている．

NDN ルータは，Interest パケットを受け取ると，まずその Interest パケットが要求するコンテンツが CS に保持されているかどうか確認し，保持されていれば，そのコンテンツを使って Data パケットを生成し，Interest パケットは破棄する．保持されたいなければ，Interest パケットが到着したフェイスを PIT に記録し，FIB の情報に基づいて決定されるフェイスに Interest パケットを送信する．

2.2　SFC

SFC は Software-Defined Networking (SDN) と Network Function Virtualization (NFV) の機能を利用して，個々に実装されたネットワークサービスを連鎖させる技術である．NFV の技術を用いて Function をネットワーク内に配備し、SDN の技術を利用してパケットをこれらの Function にルーティングすることができる．これにより Function をチェーン化し、データを順次処理するようにルーティングして，目的の出力を取得させることができる [6]．また，これらの Function を戦略的かつ動的に配備することで，ネットワークの輻輳を防止及び緩和できると期待されている．

3.　NDN-FC

NDN-FC の実装については，既に [4] において報告済であるので，ここではその概要およびその後の拡張について紹介する．

3.1　NDN-FC の概要

NDN のプロトコルは，もともとコンテンツの取得をするためのものであるため，Function を実行する仕組みが含まれていない．NDN-FC では，図 1 に示すような SFC を実行できるように NDN の拡張を行った．図では，コンテンツ名で指定したコンテンツを取得後，そのコンテンツに対して順に Function F3, F2, F1 を実行した後，Interest パケットを発生したコンシューマに結果が戻って来る例を示している．

図 1 のような SFC を可能とするために，Interest パケットに実行する Function を記述する Function Names フィールドを追加した．NDN-FC で拡張された Interest パケットの構造を図 2 に示す．Function Names フィールドには，SFC で実行する一連の Function の Function 名を記述する．

NDN-FC は，NDN のパケット転送機能をソフトウェア実装する Named Data Networking Forwarding Daemon (NFD) [7] を拡張することによって，ルータの情報処理能力を利用した SFC を実現している．

NDN-FC では，NDN ルータが Interest パケットを受信すると，Function Names フィールドを参照し，そこに記述された Function 名のリストから最前部の Function 名を取り出し，その Function 名に該当する FIB のエントリを参照して，Interest パケットを転送する．Function Names フィールドが空の場合には，Content Name フィールドの内容に従って，Interest パケットを転送する．

NDN-FC では，Function はルータを実行するコンピュータ上の一つのサーバプロセスとして実装される．FIB を参照して，Function 名によって，該当するサーバプロセスに転送された Interest パケットは，サーバプロセスで Nonce の更新だけを行い，NDN ルータに向けて再度転送される（図 3．Interest パケットを，この冗長な経路を経由させることにより，Interest パケットに対応して返送されてきた Data パケットを，NDN ルータの機能を使ってこの Function を経由させることができる）．実際の Function での処理は，Data パケットに含まれるコンテンツに対して適用される．

ここで，Nonce の更新を行うのは，同一の Interest パケットを複数回，同一の NDN ルータを経由させるために，転送ループとして誤検出されるのを回避するためである．また同一コンテンツ名の Interest パケットが NDN ルータを経由するたびに

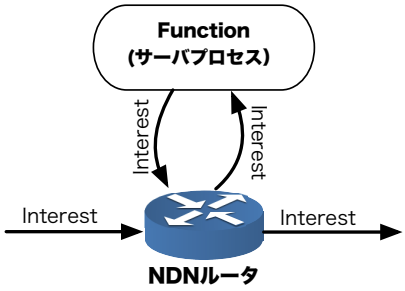

図 3　Interest パケットの Function への転送

PIT エントリを追加することになるが，ループする経路を正しく追跡できるように，PIT エントリの変更も行っている．

3.2　Final Block ID の扱いに関する拡張

ICN で扱うコンテンツは，そのサイズもさまざまであり，必ずしも一つのパケットに収まらず，複数のブロックに分けて転送する必要が生じる．そのため NDN では，あるコンテンツを要求する最初の Interest パケットに対して，そのコンテンツの最後のブロックを示す Final Block ID という情報を Data パケットに含めて返送する．この Data パケットを受け取ったコンシューマは，Final Block ID を参照し，すべてのブロックを受け取るのに必要な Interest パケットを送信し，要求したコンテンツ全体を受け取る．

NDN-FC は，要求したコンテンツに対して，ネットワーク内で，Function の処理を適用する．Function の処理を適用するためには，Function はコンテンツ全体を受け取った上で，処理を適用することになる．この問題に対処するために，NDN-FC では以下のようなプロトコルの拡張が行われている [3].

- 通信の前にコンシューマは Final Block ID のみ を要求する Interest パケットを送信する (例えば/test/file/info という名前の Interest パケットを送信してコンテンツ/test/file の Final Block ID を取得する)
- 取得した Final Block ID を利用してすべての Interest パケットをコンシューマは送信する
- プロデューサは全ての Interest パケットを受信後全ての Data パケットを送り返す

また、NDN-FC では Function の実行によりコンテンツサイズが変わる場合がある．これを解決するために、以下の拡張が加えられた．

- Function 実行後新しい Final Block ID が新しい Data パケットに記録される
- 元の Final Block ID 分の Data パケットを Function が送信する
- Data パケットが到着するとコンシューマは Data パケットに記録された Final Block ID を元のものと比較する
- 差分の Interest パケットを コンシューマ が新たに送信する
- Function は残りの Data パケットを送信する

この機能は，Function の処理によりコンテンツのファイルサイズが大きくなる場合にのみ必要となる．ファイルサイズが小さくなる場合，PIT のエントリが余ることになるが，これはタイムアウトにより消去されるため問題ない．

```
1 FROM ubuntu:16.04
2 RUN apt-get -y update && apt-get -y upgrade
3 RUN mkdir /ndn
4 COPY Consumer-Producer-API /ndn/Consumer-Producer-API
5 COPY ndn-cxx-FCv2 /ndn/ndn-cxx-FCv2
6 COPY ndn-skeleton-apps /ndn/ndn-skeleton-apps
7 COPY NFD-FC /ndn/NFD-FC
```

図 4　Dockerfile

表 1　評 価 環 境

	バージョン
Viutualbox	6.0.12
Ubuntu	16.04
Docker	19.03.1

4.　Docker による仮想環境を用いた NDN-FC

本研究では，NDN-FC の軽量化及び管理の簡略化を目指し，NDN-FC の Function を Docker コンテナで作成し、通信させることとし，コンテナの元となるイメージの作成に取り組んだ．イメージをビルドするための Dockerfile は図 4 のようになる．

仮想マシンと同様，元となるイメージには Ubuntu16.04 を模したものを使用し，そこへ NDN-FC のコアとなるツールをホスト OS からコピーしている．ndn-skeleton-apps というファイルには，コンシューマ/プロデューサ/Function のプロセスの実行ファイルが格納されている．

この Dockerfile により作成された Docker コンテナ内で，NFD の公式ドキュメントに従った NFD のインストール，NDN-FC が必要とするツールをそれぞれビルドする作業を行い，イメージとしてコミットした．

5.　性 能 評 価

本研究で作成した Docker コンテナによる NDN-FC ノードの動作検証，及び仮想マシンとの起動時間の比較を行った．仮想マシンは VirtualBox で作成し，ゲスト OS は Ubuntu16.04 とした．評価における環境を表 1 にに示す．

5.1　動 作 検 証

動作検証では，図 5 のように作成したイメージからコンシューマ，Function，プロデューサに対応する Docker コンテナを起動し，また Function にパケットが転送されるように，FIB の設定を各 NFD ルータに施した．その後，コンシューマがプロデューサからコンテンツを取得する動作を検証した．動作検証は，実際の通信を想定し，以下に示す 3 つの場合について行った．

- Function 実行後ファイルサイズが変化しない場合 (15.4kB→15.4kB)
- Function 実行後ファイルサイズが小さくなる場合 (56.9kB→15.4kB)
- Function 実行後ファイルサイズが大きくなる場合 (15.4kB→56.9kB)

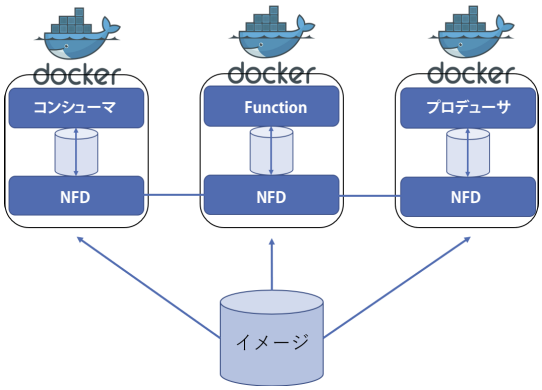

図 5　性能評価のトポロジーの概要

```
root@dd1c00289bc9:/ndn/ndn-skeleton-apps/ndn-cxx-waf/build# ./consumer
Leaving Info: /test/producer/info/test.png
data: /test/producer/info/test.png/%00%00
finalBlockId: 8
-----------------------------------------------------
8
Leaving Content: 0 /A
Leaving Content: 1 /A
Leaving Content: 2 /A
Leaving Content: 3 /A
Leaving Content: 4 /A
Leaving Content: 5 /A
Leaving Content: 6 /A
Leaving Content: 7 /A
Leaving Content: 8 /A
-----------------Received Data-------------------------
data: /test/producer/content/test.png/%00%00
-----------------------------------------------------
-----------------Received Data-------------------------
data: /test/producer/content/test.png/%00%01
-----------------------------------------------------
-----------------Received Data-------------------------
data: /test/producer/content/test.png/%00%02
-----------------------------------------------------
-----------------Received Data-------------------------
data: /test/producer/content/test.png/%00%03
-----------------------------------------------------
-----------------Received Data-------------------------
data: /test/producer/content/test.png/%00%04
-----------------------------------------------------
-----------------Received Data-------------------------
data: /test/producer/content/test.png/%00%05
-----------------------------------------------------
-----------------Received Data-------------------------
data: /test/producer/content/test.png/%00%06
-----------------------------------------------------
-----------------Received Data-------------------------
data: /test/producer/content/test.png/%00%07
-----------------------------------------------------
-----------------Received Data-------------------------
data: /test/producer/content/test.png/%00%08
-----------------------------------------------------
Received All Segments
bufferSize: 15391
Creating File
DONE
```

図 6　コンシューマの実行結果 (変化しない場合)

5.1.1　ファイルサイズが変化しない場合

コンシューマの実行結果を図 6 に示す．図 6 から，Final Block ID を取得後，同じ数の Interest パケットを送信し，また同じ数の Data パケットを受信したことが確認できる．

次にプロデューサの実行結果を図 7 に示す．図 7 から Final Block ID を送信後，届いた Interest パケットに対して Data パケットを送信したことが確認できる．

最後に Function の実行結果を図 8 に示す．図 8 から届いた Data パケットを再構成し Function の処理を実行していることが確認できる．また図 6 から，コンシューマが最終的に Final Block ID と同じ数の Data パケットを受信したことが確認できる．

5.1.2　ファイルサイズが小さくなる場合

コンシューマの実行結果を図 9 に示す．図 9 から Final Block ID を取得後同じ数の Interest パケットを送信したことが確認

```
root@5a382711f676:/ndn/ndn-skeleton-apps/ndn-cxx-waf/build# ./producer
8
Leaving Data: /test/producer/info/test.png/%00%00
Leaving Data: /test/producer/content/test.png/%00%00
Leaving Data: /test/producer/content/test.png/%00%01
Leaving Data: /test/producer/content/test.png/%00%02
Leaving Data: /test/producer/content/test.png/%00%03
Leaving Data: /test/producer/content/test.png/%00%04
Leaving Data: /test/producer/content/test.png/%00%05
Leaving Data: /test/producer/content/test.png/%00%06
Leaving Data: /test/producer/content/test.png/%00%07
Leaving Data: /test/producer/content/test.png/%00%08
bufferSize: 15391
SENT PNG FILE
```

図 7　プロデューサの実行結果 (変化しない場合)

```
-----------------------------------------------------
Data: /test/producer/content/test.png/%00%05
Prefix: /test/producer/content
Segment No.: 5
Final Block No.: 8
Adding to Buffer
-----------------------------------------------------
Data: /test/producer/content/test.png/%00%06
Prefix: /test/producer/content
Segment No.: 6
Final Block No.: 8
Adding to Buffer
-----------------------------------------------------
Data: /test/producer/content/test.png/%00%07
Prefix: /test/producer/content
Segment No.: 7
Final Block No.: 8
Adding to Buffer
-----------------------------------------------------
Data: /test/producer/content/test.png/%00%08
Prefix: /test/producer/content
Segment No.: 8
Final Block No.: 8
Adding to Buffer
-------------------------------------------------
Creating File
Orig. BufferSize: 15391
Success
new bufferSize: 15391
Final Block ID: 8
SENDING
```

図 8　Function の実行結果 (変化しない場合)

できる．

次にプロデューサの実行結果を図 10 に示す．図 10 から Final Block ID を送信後，届いた Interest パケットに対して Data パケットを送信したことが確認できる．

次に Function の実行結果を図 11 に示す．図 11 から届いた Data パケットを再構成し Function の処理を実行していることが確認でき，処理後 Final Block ID が小さくなったことが確認できる．

最後に図 12 により，コンシューマが小さくなったファイルサイズ分の Data パケットを受信したことが確認できる．

5.1.3　ファイルサイズが大きくなる場合

コンシューマの実行結果を図 13 に示す。図 13 から，まず最初に受け取った Final Block ID の情報に基づき，九つの Interest パケットが送信されている．その後，最初の Data パ

```
root@ab722ccb1128:/ndn/ndn-skeleton-apps/ndn-cxx-waf/build# ./consumer
Leaving Info: /test/producer/info/test.png
data: /test/producer/info/test.png/%00%00
finalBlockId: 33
-----------------------------------------------------
33
Leaving Content: 0 /A
Leaving Content: 1 /A
Leaving Content: 2 /A
Leaving Content: 3 /A
Leaving Content: 4 /A
        ⋮  途中省略
Leaving Content: 28 /A
Leaving Content: 29 /A
Leaving Content: 30 /A
Leaving Content: 31 /A
Leaving Content: 32 /A
Leaving Content: 33 /A
```

図 9　コンシューマの実行結果 (小さくなる場合/送信)

図 10　プロデューサの実行結果 (小さくなる場合)

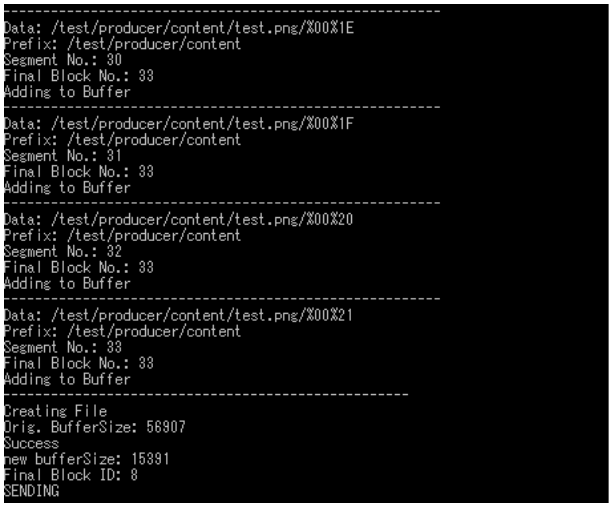

図 11　Function の実行結果 (小さくなる場合)

ケットを受け取った後，追加で 25 個の Interest パケットが送信されている．これは，Function での処理後，コンテンツサイズが大きくなり，Final Block ID が 33 となり，その情報が最初の Data パケットに載って配送されるためである．

次にプロデューサの実行結果を図 14 に示す．図 14 から Final Block ID を送信後，届いた Interest パケットに対して九つの

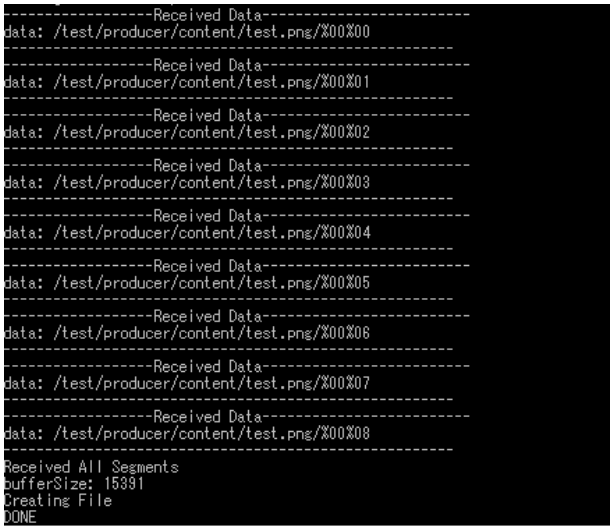

図 12　コンシューマの実行結果 (小さくなる場合/受信)

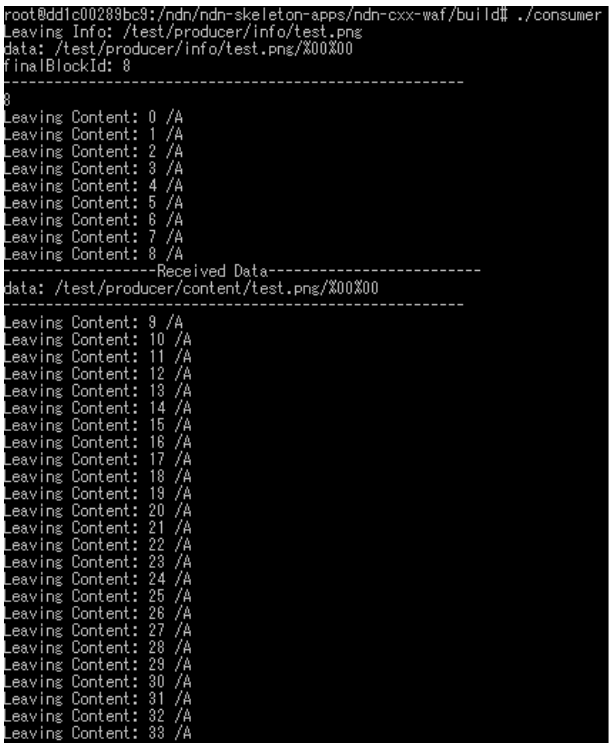

図 13　コンシューマの実行結果 (大きくなる場合/送信)

図 14　プロデューサの実行結果 (大きくなる場合)

Data パケットを送信したことが確認できる．追加でコンシューマは送信した Interest パケットは，途中の Function で消費され，プロデューサには配送されていない．

次に Function の実行結果を図 15 に示す．プロデューサから届いた九つの Data パケットを再構成した上で，Function の処理を実行し，処理後 Final Block ID が大きくなったことが確認できる．

最後に図 16 により，コンシューマが大きくったファイルサイズ分の Data パケットを受信したことが確認できる．

5.2　仮想マシンとの起動時間の比較

起動時間の比較では，本研究で作成した NDN-FC ノードと同等の動作をする仮想マシンを用意して比較した．Docker コンテナは，Windows PoweShell で Measure-Command コマンドを用い，Docker コンテナがイメージから起動されるまでの処理時間を計測した．仮想マシンは、dmesg コマンドを用い，カーネルが起動する際の処理と時間を表示させて計測した．それぞれメモリの上限を 3072MB とし，5 回測定した結果を表 2 に示す．表 2 より，今回作成した Docker コンテナのほうが起動時間が早くなっていることが確認できる．

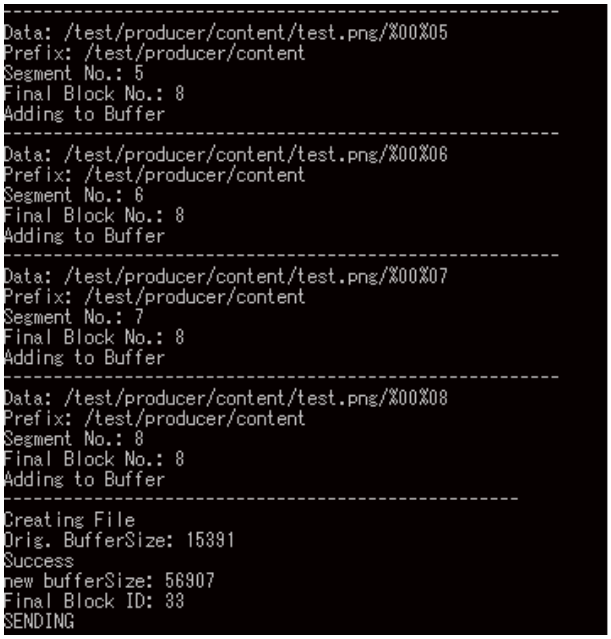

図 15　Function の実行結果 (大きくなる場合)

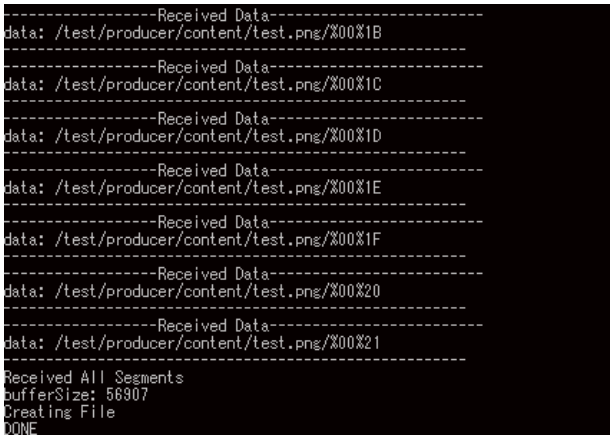

図 16　コンシューマの実行結果 (大きくなる場合/受信)

6.　今後の展望

現在，NDN-FC に係わる研究としては，ネットワークの状況に応じて，Function を動的に及び戦略的に配備する研究を行なっている．図 17 のような仮想的なネットワーク内のルータに，SFC 構築機構がネットワークの状況を分析しながら動的に Function を配備するシステムの設計/実装を行っている．これにより，効率的にパケットを処理できるよう柔軟に対応するシステムが実現する．Function のプロセスのみを実行する Docker コンテナを実装することで，再配備の際，必要な Function を

表 2　起動時間の比較

回数	docker の起動時間 (s)	VM の起動時間 (s)
1	4.831765	62.47054
2	2.869693	47.32371
3	2.591142	50.25787
4	2.723558	46.92482
5	2.753842	45.98574

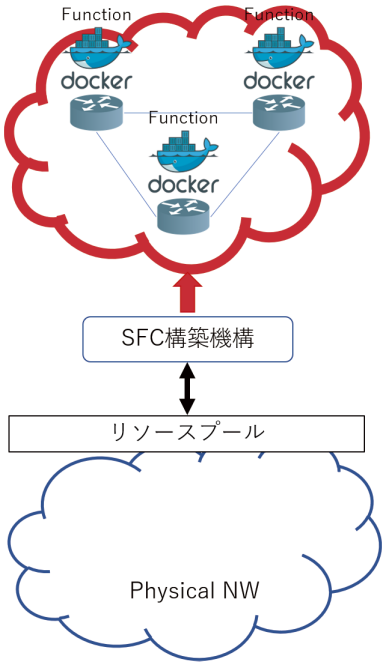

図 17　動的な Function の配備

Docker コンテナとして必要なルータに立てることができると考えられる．それに際し，Function のプロセスのみを実行する Docker コンテナを実装，さらに Docker コンテナに NFD ルータから Interest/Data パケットが届くような FIB の設定が必要となる．さらには，IoT ネットワークで必要となる push 型の通信を NDN において実装する研究にも取り組んでいく．

謝辞　本研究成果は，総務省の（平成 30 年度）戦略的情報通信研究開発推進事業（国際標準獲得型）「スマートシティアプリケーションに拡張性と相互運用性をもたらす仮想 IoT-クラウド連携基盤の研究開発 (Fed4IoT)」によるものである．

文　　献

[1] “Docker 概要”. http://docs.docker.jp/engine/understanding-docker.html

[2] W. Shang, A. Bannis, T. Liang, Z. Wang, Y. Yu, A. Afanasyev, J. Thompson, J. Burke, B. Zhang, and L. Zhang, “Named data networking of things (invited paper),” 2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI), pp.117–128, April 2016.

[3] 吉井宏希，“Real world implementation of function chaining in named data networking,” Master’s thesis, 早稲田大学 基幹理工学研究科 情報理工・情報通信専攻, 2 月 2019.

[4] 吉井宏希，中里秀則，“NDN におけるファンクション・チェイニングの実装,” 技術研究報告 CS2018-39，電子情報通信学会，7 月 2018.

[5] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, “Named data networking,” SIGCOMM Comput. Commun. Rev., vol.44, no.3, pp.66–73, July 2014.

[6] 北田裕之，小島久史，高谷直樹，相原正夫，“将来ネットワークに向けたサービスファンクションチェイニング 技術,” NTT 技術ジャーナル，vol.26，no.5，pp.10–13，2014.

[7] “NFD - named data networking forwarding daemon”. http://named-data.net/doc/NFD/current/