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
CS2020-71,IE2020-30(2020-11)

NDN Function Chainingにおける Function選択および Cache更新手法

尾崎　航太†　　中里　秀則†　　金井　謙治†

† 早稲田大学大学院基幹理工学研究科　〒 169–0072 東京都新宿区大久保 3–14–9
E-mail: †kota.wase-tennis@fuji.waseda.jp, ††nakazato@waseda.jp, †††k.kanai@aoni.waseda.jp

あらまし　近年、IoT デバイスの増加に伴い、IoT アプリケーションなどにおいてより高いリアルタイム性が求められている。この問題は、ネットワークインフラストラクチャの場所や性能に依存しないエッジコンピューティングによって対応することができ、私たちはその一つの手法として Service Function Chaining (SFC) と Named Data Networking (NDN) を用いたネットワークサービスである NDN Function Chaining+ (NDN-FC+) の開発に取り組んでいる。本研究では、リアルタイム性をより追求するため、NDN-FC+におけるルータのキャッシュデータを定期的に更新する手法を提案し、評価した。

キーワード　Service Function Chaining(SFC)、Named Data Networking(NDN)、NDN Function Chaining+ (NDN-FC+)、キャッシュデータ

Function Selection and Cache Refresh Mechanisms for NDN Function Chaining

Kota OZAKI†, Hidenori NAKZATO†, and KENJI KANAI†

† graduate School of Information engineering, Waseda University　Okubo 3–14–9, Sinjuku-ku, Tokyo, 169-0072 Japan
E-mail: †kota.wase-tennis@fuji.waseda.jp, ††nakazato@waseda.jp, †††k.kanai@aoni.waseda.jp

Abstract　In recent years, with the increase in IoT devices, real-time performance is required in some IoT applications. This problem can be addressed by edge computing which removes the obstacle imposed by the location and performance of the network infrastructure, and we use Service Function Chaining (SFC) and Named Data Networking (NDN) as one of the methods. We are working on the development of the NDN Service Function Chaining+ (NDN-FC+). In this study, in order to pursue real-time performance, we proposed and evaluated a method to periodically update the cache data of the router in NDN-FC+.

Key words　Service Function Chaining(SFC), Named Data Networking(NDN), Chache data, NDN Function Chaining+ (NDN-FC+)

1. まえがき

近年、IoT デバイスが急速に増加している。これに伴い、IoT アプリケーションやサービスも増加し、自動運転やファクトリーオートメーションのように短い応答時間を必要とする、高いリアルタイム性を求めるアプリケーションも多い。これに対応する一つの解決策として、エッジコンピューティングが提案されているが、ネットワークインフラストラクチャの処理能力及び場所に依存してしまうという問題点がある [1]。私たちはこの問題を解決するため、ネットワークサービスの仮想チェーンを作成する Service Function Chaining (SFC) の考え方を用い、ネットワークサービスの戦略的かつ動的な配置を行うことを提案している。SFC を用いた場合、ネットワーク全体に配置されたコンピューティングリソースに対し、チェーン化された Function を実行することによって IoT データを処理し、目的のアウトプットを取得することができる。この機能の戦略的かつ動的配置を行うことにより、リアルタイム性のみならずネットワーク上の輻輳や負荷の防止をすることもできる。我々はさらに、サービスファンクションの選択に自由度をもたせるために、従来の IP 通信とは異なりコンテンツ名でデータのルーティングを行う Named Data Networking (NDN) を使用する。NDN は従来の IP のような場所に応じたルーティングではなく、コンテンツ名あるいはファンクション名でパケットのルーティングを行うため、複数デプロイされたファンクションイン

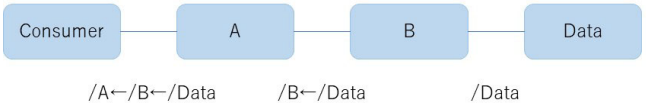

図 1　ICN-FC

スタンスから適切なインスタンスを選択するなど、柔軟なチェインの構成を可能とする。また、NDN は Cache 機能を備えており、ファンクション適用後のデータの一時保存ができるため、ファンクションの処理に費やす時間を省略した SFC の実行が行える。これらを組み合わせて作成された NDN Function Chaining+(NDN-FC+) を用いることにより効率的で管理のしやすい SFC を利用した IoT ネットワークを作成することができる。

本研究では、まずリアルタイム性の実現のため、Consumer のデータ要求時によらず、ルータが定期的に Producer に対しデータの要求を行うことで Cache データを常に更新する手法を提案し評価を行った。さらに、柔軟なファンクションチェインの構成のため、ファンクション処理におけるネットワーク部と加工部を切り離し、ファンクションを複数から選択する手法を提案し評価を行った。

以下、2. で関連研究について、また 3. でこれも本成果に関連の深い NDN-FC+について述べる。では本論文での提案として、4.1 で Function における自律的な Cache の更新について、また 4.2 で Function の呼び出し手法について述べ、5. で提案手法を評価し，6. で結論および今後の課題について述べる。

2. Related Work

情報指向ネットワーク (Information-Centric Networking: ICN) をベースとし、ICN Function Chaining が L. Liu. によって提案された [2]。← 記号を用いて Interest パケットにチェインする Function を記述することによりファンクションチェインを実現する。図 1 に ICN Function Chaining の例を示す。

図 1 における A、B は Function、Data は Producer を表す。ファンクションチェインのために、Consumer が送信する Interest パケットの名前は/A ←/B ←/Data というファンクションチェインになり、Interest パケットは Function 名が/A である Function から順に通過する。Interest パケットが Function を通過するたびにその Function 名が Interst パケットの名前から削除される。Interest パケットの名前から Function 名を削除することにより、Interest パケット名の先頭に記載される Function 名が更新され、Interest パケットが次に送信されるルートが決定される。ICN-FC はビデオファイルの処理を想定して評価されているが、Function 間でコンテンツをセグメント化しないため大きなビデオファイルを転送する方法がない。

3. NDN-FC+

H.Yoshii と Y.Kumamoto はファンクションチェイニングの考え方を用いられるように NDN の拡張を行った。この拡張された NDN は NDN-FC+と呼ばれる [3]。

3.1 Consumer、Producer および Function 間のコンテンツ転送方法

通常の NDN において、コンテンツは Data パケットに収まらないため、コンテンツを取得するためには Consumer と Producer の間で多くの Interest パケットと Data パケットが交換される。そのため、複数の Data パケットを含むコンテンツを転送するために、コンテンツの転送に使用される Data パケットの数を指定する値である Final Block ID を通知するメカニズムがある。サイズの大きなコンテンツを要求する最初の Interest パケットが Producer に送信されると、Producer はコンテンツの Final Block ID および最初のセグメントを含んだ Data パケットを返す。図 2 に、Cache を使用しない場合の Consumer、Function および Producer の簡単な通信手順を示す。図のセグメント化されたコンテンツは次のように処理される [4]。

（1） Consumer は、コンテンツを要求する最初の Interest パケット/test/producer/00/A を送信する。この時、コンテンツ名が/test/producer、Function 名が/A である。

（2） Function は (1) の Interest パケットを受け取り、Producer に向け最初の Interest パケット/test/producer/00 を送信する。

（3） 要求されたコンテンツを保持している Producer は、セグメント化されたコンテンツの Final Block ID および最初のセグメントを含んだ Data パケットを返す。

（4） Function は (3) の Data パケットから得た Final Block ID を使用し、コンテンツを取得するために必要なすべての Interest パケットを Producer に向け送信する。

（5） Producer は (4) の Interest パケットに対応した Data パケットを返す。

（6） Function に全ての Data パケットが到着すると、全てのセグメントからコンテンツを再構成して処理を実行する。その後、処理したコンテンツをセグメント化し、(1) の Interest パケットに対応した Data パケットを返す。このとき、/test/producer/00/A に該当する Data パケットには Final Block ID を含める。

（7） Consumer は (6) の Data パケットから得た Final Block ID を使用し、コンテンツを取得するために必要なすべての Interest パケットを Function に向け送信する。

（8） Function は (7) から送信された Interest パケットを受信し、それらに対応した Data パケットを返す。

NDN-FC+では接続された隣接ノードにおいて Consumer-Producer 通信を常に行う。これにより、Interest パケットは、Interest パケットによって要求された Function を介して送信されるため、Producer から送信された最初の Data パケットは、チェーン内の Producer からの最初の Function によって受信される。この Function は、最初の Data パケット内のコンテンツの Final Block ID を取得する。Function はこの Final Block ID を使用することで、残りのコンテンツを要求する Interest パケットを自律的に作成する。そうすることで、関数はコンテンツの再構築に必要なすべての Data パケットを受信できる。次

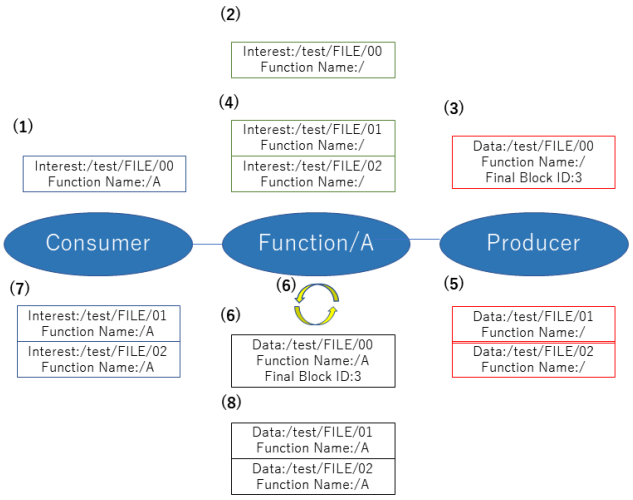

図 2　NDN-FC+におけるコンテンツ転送例

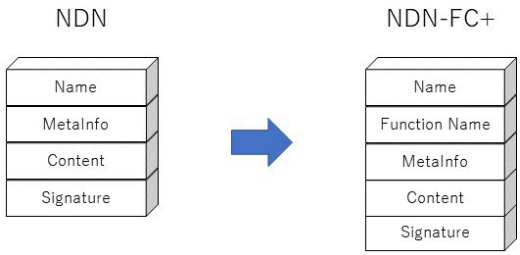

図 3　NDN-FC+における Data パケット形式

に、Function はコンテンツに加工を施した結果をセグメント化し、それらの Data パケットを準備する。最初の Interest はすでに Function によって受信されているため、最初のデータパケットが返され、追加の Interest パケットを待つ。それらが受信されると、残りの Data パケットが返される。

3.2　Interest Function Field

NDN-FC+では、NDN の Interest パケット形式に Function Name フィールドが追加されている [3]。このフィールドは、パケットが Producer に届くまでに通過するべき Function を示す。例えば、Function Name フィールドに/A/B/C と記述されていた場合、Interest パケットを送信するノードは Function /C、/B、/A の順序でコンテンツに Function 加工を適用する。Interest パケットが Function /A に到着すると、Function /A の名前が削除され、Function Name フィールドが/B/C となった新しい Interest パケットが送信される。

3.3　Data Function Field

NDN-FC+では、Interest パケット同様、Data パケット形式にも Function Name フィールドが追加されている。このフィールドは、Data パケットをキャッシュするために、Data パケットが運ぶコンテンツにどのような Function が適用されているかを示す。コンテンツに Function が適用されるたびに、その Data パケットの Function Name フィールドに加工された Function の名前が追加される。図 3 に変更された Data パケットの Function Name フィールドを示す。

3.4　パケット転送における各 Interest パケットの区別

NDN ルータは、Data パケットおよび Interest パケットの転送方向を決定する二つのデータ構造を持ち、それぞれ Pending Interest table(PIT)、Forwarding Information Base(FIB) と呼ばれる。PIT は各 Interest パケットがルータに入る際の Face を記録し、Data パケットを Consumer に送り返すために満足されていない Interest パケットの到着インタフェースを記録する。FIB は Interest パケットの転送方向を決定するルーティングテーブルである。NDN-FC+では、PIT のエントリに、元の NDN にある Content Name フィールドに加えて Function Name フィールドを加えている。これにより、Function を通過する前後の Interest パケットを、PIT エントリの Content Name と Function Name で区別している。

4.　提 案 手 法

4.1　Function における自律的な Cache の更新

NDN では、Producer から Conumser までの途中のルータで、Data パケットを Cache することができる。この機能により Consumer が再度同じコンテンツを要求した場合、ルータで Cache された Data パケットを即座に返送することができる。これにより、Cache ルータから Producer までの転送時間を省略することができ、よりリアルタイム性の高い通信を実現できる。しかし、IoT アプリケーションデータのように短いスパンで更新されるコンテンツの場合、Cache されたデータが古くなってしまうためリアルタイム性の追求と乖離してしまう。この問題点の解決策として、Function が配備されたルータ（Function ルータ）による自律的な Cache の更新を行う NDN-FC+の拡張である**自律更新** *NDN-FC+: Autonomic Refresh NDN-FC+ (NDN-FC+AR)* を提案する。Consumer の要求によらず、Function ルータ上のプロセスで Producer に対し自律的に Interest パケットを送信しコンテンツを取得する。

4.1.1　Function における自律的 Interest 送信の手法

Function ルータにおける自律的 Interest 送信の手法として、ルータ上で、当該 Function およびそれによって処理されるコンテンツを要求する擬似 Consumer を用いることを提案する。擬似 Consumer を定期的に起動し、Producer に対してコンテンツ要求を行うことにより Cache されたデータを更新する。このように実装した例を図 4 に示す。

4.1.2　Interest EraseCache Field

図 4 で行われるデータ転送は次のようになる。

（1）　Consumer がコンテンツ要求を行い、Interest パケットおよび Data パケットの送受信が完了する。この時、Function ルータには Function を通過した前および通過した後の Data パケットが Cache される。

（2）　定期的に Function ルータ上で擬似 Consumer が起動され、Producer に向けて Interest 送信を行う。

（3）　Interest パケットが Function ルータに送信されると、Function ルータ上に Cache された Function 適用済みコンテンツの Data パケットにヒットする。そのため、Interest パケットは Producer で更新されたコンテンツに届くことなく、Cache

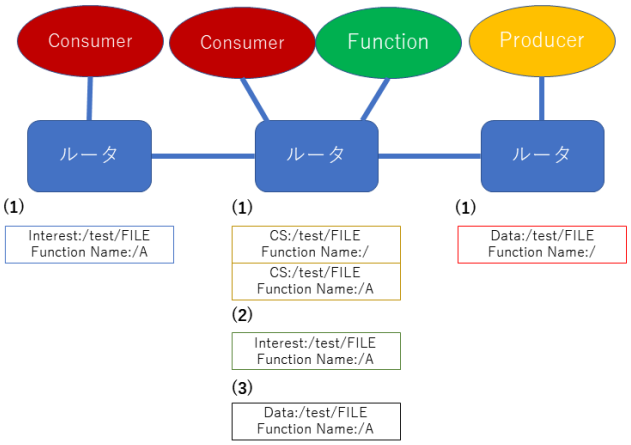

図 4　擬似 Consumer の自律的 Interest 送信

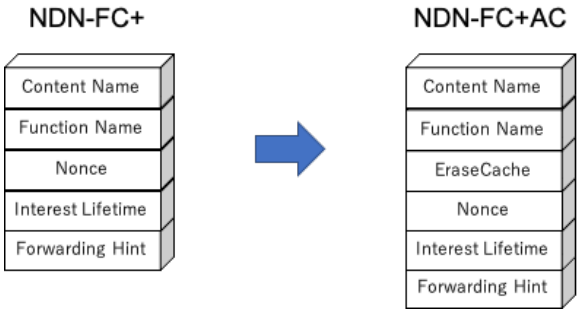

図 5　Interest パケット形式における EraseCache 追加

された Data パケットは更新されずに擬似 Consumer に返る。

このように、擬似 Consumer で Interest を生成して送信した場合、Cache されたデータにヒットをしてしまうことから、Cache データの更新が行えない。そのため、Cache データの更新を行うために Cache データを一度削除する手法を提案する。この提案では、NDN-FC+における Interest パケット形式に EraseCache フィールドを追加する。このフィールドには 0 または 1 が記載され、1 が記載された Interest パケットが Cache を使用したルータに送信されると、Cache データの中からコンテンツ名が一致するものを選択し削除する。こうすることにより、Interest パケットは Cache データにヒットすることなく Producer 方向に転送されるため、Cache データを更新することができる。Consumer が送る Interest パケットはこのフィールドを 0 とし、疑似 Consumer が送る Interest パケットはこのフィールドを 1 とすることで、Function ルータでは Cache ヒットせずに Cache データを更新することができ、Consumer は更新された Cache データからコンテンツを入手することができる。EraseCache フィールドは、Function 通過後、ルータから他のルータに送信されるときに 0 にリセットされる。このようにすることで、Interest パケットが次のルータに送信された際、Cache データを削除せずにヒットすることができる。図 5 に変更された Interest パケットの EraseCache フィールドを示す。

4.2　Function 呼び出し手法

NDN-FC+AC において、Function を実行する上では、コンテンツに対して Function を適用するだけでなく、コンテンツの分割/再構成を行い、Interest パケットと Data パケットの通信を行う必要がある。このコンテンツの分割/再構成と通信を行う部分は、各 Function で定義された処理とは独立で、すべての Function を処理する上で共通に必要な処理である。この共通部分を**ネットワーク処理部**として、Function 固有の処理を行う *Function* **処理部**から独立させることとした。この分離により、同一ルータ上に配備された複数の Function 処理部の中から、適切な Function 処理部をネットワーク処理部によって選択することができる。

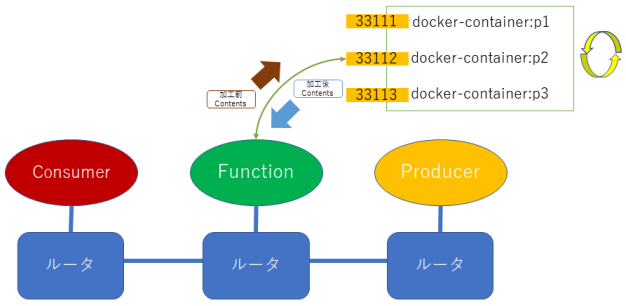

図 6　NDN-FC+AC における Function 呼び出し

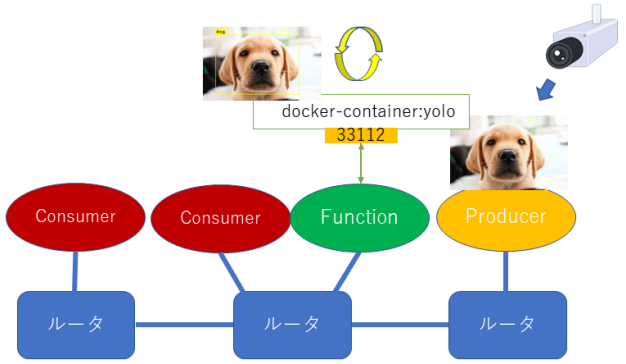

図 7　各要素の動作確認における NDN-FC+AC の構成

本研究での実装では、Function 処理部を Docker コンテナとし、Function 呼び出し時にルータから起動されるネットワーク処理部から、Docker コンテナとソケット通信を行うものとなっている。図 6 に NDN-FC+AC における Function 呼び出しを示す。

5.　評　　価

5.1　本提案手法の評価シナリオ

本提案手法の評価として、各要素の動作確認およびコンテンツ取得時間の比較を行った。これらの評価において、それぞれ図 7、図 8 に示す NDN-FC+AC の構成を使用する。

図 7 の構成に関して、Producer はカメラで取得した画像データを「/test/producer/test.jpg」というコンテンツ名で Consumer 側に供給する。また、Producer に保存されている画像データ「test.jpg」は 5 秒に一度更新される。Function ルータは自律的に Interest パケットを送信する擬似 Consumer を定期的に起動する。さらに、ルータは Function をコンテンツに適

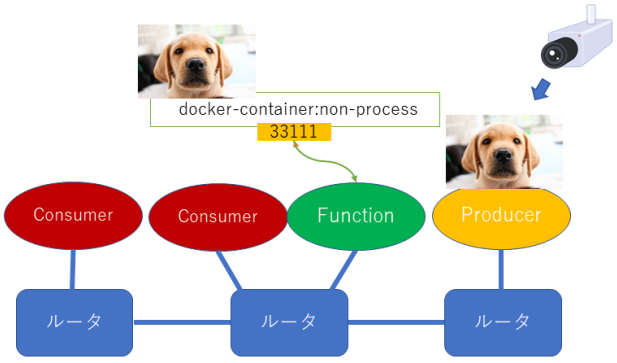

図 8　コンテンツ取得時間の比較における NDN-FC+AC の構成

```
----------------------------------------------------
Creating File
Orig. BufferSize: 17645
Success
Send to Docker
complete
complete2
Return from Docker
test.jpgnew bufferSize: 14878
Final Block ID: 8
SENDING
----------------------------------------------------
```

図 9　ネットワーク処理部と Function 処理部間のデータ通信

用するために、Function 処理部を実装する Docker コンテナと通信を行う。コンテンツは Function として実行される、リアルタイムオブジェクト検出アルゴリズムである YOLO により、物体検知および画像加工が施される。

図 8 の構成では、Producer は図 7 と異なり、定期的に更新されない画一の画像データを Consumer 側に供給する。Function ルータにおいても図 7 と同様に自律的に Interest パケットを送信する擬似 Consumer を定期的に起動する。Function はコンテンツ加工時に Docker コンテナと接続し、データ通信を行う。この際、Docker コンテナにおけるコンテンツの加工は行わず、元の状態のコンテンツを Function に返す。

5.2　各要素の動作確認

Function ルータのみ Cache を有効にし、擬似 Consumer を 5 秒に一度起動するように設定した。この設定の上、Consumer から Interest パケットを送信した際の、Function 実行の実験を行った。結果を図 9、図 10 に示す。

図 9 は、ネットワーク処理部と Funciton 処理部（Docker コンテナ）がデータ通信を行う箇所である。Interest パケットに対応した Data パケットをすべて受信し、コンテンツの再構成をした後、そのコンテンツを Docker コンテナに送信している。図 10 はコンテンツを加工する Function 処理部のプロセスを示したものである。Function 処理部はネットワーク処理部からの接続が来るまで待機し、接続しコンテンツが送られてくるとオブジェクト検知を開始する。すべての動作が終わると、加工した画像をネットワーク処理部に送り返し、再度待機状態に入る。加工されたコンテンツを受け取ったネットワーク処理部はそのコンテンツをセグメント化し、Consumer 側に送信する準備をする。このような結果から、ネットワーク処理部および

```
root@fce4f7ceec3d:/src# python3 main.py
[callServiceFunction] waiting
[callServiceFunction] accepted
[callServiceFunction] received
[callServiceFunction] callServiceFunction
[callServiceFunction] start service function
layer     filters    size              input                output
    0 conv     32  3 x 3 / 1   608 x 608 x   3   ->   608 x 608 x
32  0.639 BFLOPs
    1 conv     64  3 x 3 / 2   608 x 608 x  32   ->   304 x 304 x
64  3.407 BFLOPs
  103 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 1
28  0.379 BFLOPs
  104 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 2
56  3.407 BFLOPs
  105 conv    255  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 2
55  0.754 BFLOPs
  106 yolo
Loading weights from yolov3.weights...Done!
receive.jpg: Predicted in 81.415915 seconds.
chair: 94%
chair: 86%
chair: 67%
[callServiceFunction] complete
[callServiceFunction] send results
[callServiceFunction] send complete
[callServiceFunction] waiting
```

図 10　Function 処理部によるコンテンツ加工

```
outRecord SequenceNumber for /test/producer/content/test.jpg/%00%05= 1FaceId: 268
1603121428.042008 DEBUG: [Forwarder] onIncomingInterest face=275 interest=/test/produc
er/content/test.jpg/%00%06
1603121428.042049 DEBUG: [Forwarder] EraseCacheNumber is 1
1603121428.042071 DEBUG: [Forwarder] RESET CACHE RESET CACHE RESET CACHE
1603121428.042098 DEBUG: [Forwarder] /test/producer/content/test.jpg/%00%06
1603121428.042128 DEBUG: [ContentStore] findforerase /test/producer/content/test.jpg/%
00%06 L
1603121428.042173 DEBUG: [ContentStore]   DoErase /test/producer/content/test.jpg/%00%
06
1603121428.042218 DEBUG: [Forwarder] Function is /
1603121428.042254 DEBUG: [ContentStore] findforuse /test/producer/content/test.jpg/%00
%06 L
1603121428.042332 DEBUG: [ContentStore]   no-match
1603121428.042355 DEBUG: [Forwarder] onContentStoreMiss interest=/test/producer/conten
t/test.jpg/%00%06
inRecord SequenceNumber for /test/producer/content/test.jpg/%00%06= 1FaceId: 275
1603121428.042404 DEBUG: [Forwarder] onOutgoingInterest face=268 interest=/test/produc
er/content/test.jpg/%00%06
outRecord SequenceNumber for /test/producer/content/test.jpg/%00%06= 1FaceId: 268
```

図 11　EraseCache による NDN-FC+AC の動作

Function 処理部は、正常に動作していると考えられる。

次に、Function ルータにデータが Cache されている状態のもとで、Function ルータ上で疑似 Consumer の最初の起動をさせ、Function ルータ上における自律的な Interest パケットの送信による Cache の動作実験を行った。結果を図 11 に示す。

図 11 は、Function ルータに Interest パケットが入った際のルータの動作を示したものである。Interest パケットの Erase-Cache フィールドには 1 と記載されているため、Cache されているデータで Interest パケットとコンテンツ名が同じものが削除される。本実験では、Interst パケットとコンテンツ名が同じであり Function Name「/」および「/A」の二つの Data パケットが Cache されているため、これらは Cache から削除される。その後、ルータ内で Interest パケットとコンテンツ名が同様のものを Cache から探しているが、それらは全て削除されているためヒットしない。そして、Cache ヒットしなかったことにより、Interest パケットは Producer 方向に転送される。Function 上で自律的に生成される Interest パケットが全て図 11 と同様の振る舞いをすることにより、全ての Interest パケットが Producer 方向に転送され、それらに対応する Data パケットが新しく Cache に保存されるため、Function ルータに Cache されている画像データを 5 秒に一度更新することができた。

5.3　コンテンツ取得時間の比較

コンテンツ取得時間を以下の三つの条件で比較する。

表 1　コンテンツ取得時間

case	time(ms)
(1)	229.19
(2)	419.24
(3)	77.59

（1） 全てのルータで Cache を有効にせず、NDN-FC+で実行した場合

（2） 全てのルータで Cache を有効にせず、NDN-FC+においてネットワーク処理部と Function 処理部を分離し実行した場合

（3） Function ルータのみ Cache を有効にし、NDN-FC+AC で実行した場合

これら全ての条件において、通信処理を公平にするため、17,726bytes の同一の画像データを配送コンテンツとする。また、NDN-FC+における Function や NDN-FC+AR における Function 処理部の処理速度が異なる可能性がある。その場合、加工方法によりコンテンツ取得時間に差異が生まれ、遅延時間の測定が不正確になってしまう恐れがある。そこで、Function 処理部でのコンテンツ加工はせずに元の状態のコンテンツを Function ルータに返すこととしてコンテンツ取得時間の比較を行った。コンテンツ取得時間はそれぞれ 100 回の計測を行った平均値である。コンテンツ取得時間を表 1 に示す。

表 1 からわかるように、ネットワーク処理部と Function 処理部を分離して実行する場合、NDN-FC+の Consumer-Function-Producer 通信によるコンテンツ取得時間に比べ大幅に遅延している。これは、Function ルータと Docker コンテナ間の通信時間および Docker コンテナにおけるプロセスに費やす時間によるものだと考えられる。

一方、Function ルータ上で擬似 Consumer を定期的に起動することによって Cache を更新する場合は、コンテンツ取得時間を大きく短縮することができた。これは、Consumer から Function に Interest パケットが入った際に常に Cache にヒットすることにより、Function-Producer 間の通信時間および Function、Producer の実行プロセスに費やす時間を省略することができるためだと考えられる。

6. 結論および今後の課題

本論文では、IoT ネットワーク環境としての NDN-FC+における Cache データの更新、および NDN-FC+の Function におけるネットワーク処理部と Function 処理部を分離することの必要性を説明した。また、NDN-FC+における Cache データの更新と NDN-FC+の Function におけるネットワーク処理部と Function 処理部の分離をサポートするアーキテクチャを提案した。本論文の性能評価においては、それぞれの提案の有効性を示した。

今後の課題として、Cache データの更新については排他制御のアーキテクチャを検討する必要がある。本提案のアーキテクチャでは、異なる Consumer が同じルータに対し同時期に Interest パケットを送信した場合、不具合が生じるという結果となった。これは本提案に限らず、NDN-FC+が広域なネットワークになるにあたり解決しなければならない問題である。NDN-FC+の Function におけるネットワーク処理部と Function 処理部の分離については、Docker コンテナにおける通信時間も考慮したうえでの最適ルート探索をする必要がある。本提案で実装したアーキテクチャは、Consumer からコンテンツまでのホップ数を減少させることはできるが、Docker コンテナとのデータ通信時間やコンテナ内でのプロセス時間により、もともとの NDN-FC+の Function 実行に比べ処理に時間を要する。そのため、これらのメリットとデメリットを踏まえたうえで最適なルーティングを行う必要がある。

謝辞　本研究の成果は，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）【JPJ000595】「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] W. Shi and S. Dustdar, “The promise of edge computing,” Computer, vol.49, no.5, pp.78–81, 2016.

[2] L. Liu, Y. Peng, M. Bahrami, L. Xie, A. Ito, S. Mnatsakanyan, G. Qu, Z. Ye, and H. Guo, “ICN-FC: An information-centric networking based framework for efficient functional chaining,” 2017 IEEE International Conference on Communications (ICC), pp.1–7, 2017.

[3] Y. Kumamoto, H. Yoshii, and H. Nakazato, “Real-world implementation of function chaining in Named Data Networking for IoT environment,” 2020 IEEE ComSoc International Communications Quality and Reliability Workshop (CQR), pp.1–6, May 2020.

[4] Y. Kumamoto and H. Nakazato, “Implementation of NDN function chaining using caching for iot environments,” Proceedings of the 2020 Cloud Continuum Services for Smart IoT Systems (CCIoT’20), pp.20–26, 2020.