令和2年度 修士論文

NDNにおけるSFCの分散制御ファンクション選択手法

Distributed Control Function Selection Method for Service Function Chaining in NDN

指導教授 中里秀則 教授

2021年1月25日

早稲田大学 基幹理工学研究科 情報理工・情報通信専攻
分散コンピューティングシステム研究

5119F102

山口直樹

Naoki Yamaguchi

目 次

第1章　序論

1.1　はじめに

様々なモノをネットワークを通してサーバーやクラウドに繋ぎ、情報のやり取りを行う IoT (Internet of Things) という技術が様々な分野で取り入れられるようになってきている。一方で、IoT 機器の増加に伴い IoT サービスを提供するクラウドサーバやそのコンテンツを配送するネットワークの負荷の増大が予想されている。しかし、IoT が活用されるサービスのうち、医療機器や自動運転、ファクトリーオートメーションといった分野では低遅延であることが必要とされる。

この問題を解決するため、情報を発信する IoT 機器やそのデータを使用する機器の近くに計算リソースを配備することでネットワーク負荷を軽減し、低遅延でデータ処理を行う「エッジコンピューティング」[11] が提案されている。しかし、エッジコンピューティングには、配備された計算リソースに処理が集中するため相応の処理能力が要求されるという問題がある。また、1 つの IoT ネットワークを高いリソース利用効率で運用するために、それらの計算リソースをいかに柔軟に連携させるかが課題である。

この課題を解決するために、Service Function Chaining (SFC) [3] という技術を適用することを考える。SFC は NAT やファイアウォール、ロードバランサなどに代表されるネットワーク機能をファンクションとして定義し、ネットワーク内のコンピューティングリソースに割り当て、要求ファンクションをすべて通過するようにパケットを制御することでネットワーク内処理を実現する技術である。ファンクションチェインを作成することでファンクションが順次実行され、目的の出力を得ることができる。これらのファンクションは Network Function Virtualization (NFV) [1] におけるファンクションのように、汎用 OS 上で柔軟に起動・停止ができるため、ネットワーク内での処理をより効率的にすることができる。これを拡張し、IoT サービスに必要なデータ処理をファンクションとして定義し、ネットワークに配置することで、前述の課題の解決する試みがなされている。この SFC をシームレスで動的に制御するために、一般的なロケーション指向の IP プロトコルとは対照的な、コンテンツ指向プロトコルの 1 つである Named Data Networking (NDN) [13] と呼ばれるプロトコルを採用する。NDN を使用することで、名前によるルーティングを可能にし、ファンクションと IoT データの管理をより直感的で簡単にすることができる。加えて、IoT サービスのファンクションを名前によって管理することで、ファンクションの位置を動的に変更することができる。この SFC と NDN の 2 つの技術を使用することでより効率的な IoT サービスが実現できる。この組み合わせは NDN-FC [4] と呼ばれる。1 つのファンクションのインスタンスにアクセスが集中することを避けるため、同じ種類のファンクションのインスタンスを複数用意し、ネットワークに配備することを考える。そのためには、適切なインスタンスを選択するメカニズムが必要になる。

そこで、本研究では NDN を用いた IoT サービスの SFC において、ネットワーク負荷とファンクションの負荷分散に注目したファンクション選択手法の実現を目指す。先行研究として、DL-MD-C [12] があるが、DL-MD-C はすべてのインスタンスの情報を管理するノードが必要なため、ボトルネックを発生させる原因となる。本研究ではこの問題を回避するために、分散制御によるファンクション選択を提案する。

1.2　本論の構成

本論は全 6 章で構成される。第 2 章では本研究のテーマである SFC 技術について解説し、その関連研究である VNF スケジューリングと IoT データにおける SFC を紹介する。第 3 章では本研究で IoT データの SFC のために用いる通信プロトコル NDN の概要と NDN における SFC、及びそのファンクション選択手法である DL-MD-C について述べる。第 4 章では NDN を用いた IoT データの新しい SFC を提案する。第 5 章では、第 4 章で提案した手法をシミュレーションによって評価し、その結果について考察する。第 6 章では結論と今後の課題について述べる。

第2章　Service Function Chaining (SFC)

この章では、本研究の中心となる SFC とその関連研究について紹介する。

2.1　SFC

一般的なネットワークサービスはファイアウォール、ロードバランサ、NAT、その他サービス特有のネットワーク処理といったネットワーク機能 (NF) が必要となる。これらの処理はそれぞれを適切な順番で行う必要がある。SFC はユーザの要求するサービスに応じて必要なネットワーク処理を担う機器に適切な順番でパケットを経由させる技術である。例えば、あるネットワークサービスをユーザー A が HTTP キャッシュ機能をオプションとして要求しているとき、SFC によりパケットは HTTP キャッシュ→ファイアウォールの順で要求コンテンツが NF を経由し、処理されることでこのサービスは実現される。一方で、ユーザー B はセキュリティ強化のため、この動画視聴サービスをウィルスチェック機能と URL フィルタリング機能をオプションとして要求すると、ウィルスチェック→ URL フィルタリング→ファイアウォールの順で NF を経由するようにパケットを制御することでこのサービスが実現される。

これらの NF は従来、専用のハードウェア上でのみ動作していた。しかし近年の仮想化技術の発展により開発された NFV という技術により、NF を汎用ハードウェア上にソフトウェアとして動作させることが可能になった。この NFV で汎用サーバー上に実装された NF を Virtual Network Function (VNF) と呼ぶ。NFV をサービスに組み込むことで、プロバイダはネットワークの状況やユーザの要求に応じて、VNF の起動や終了をすることが容易になった。このように動的に起動・終了される VNF に対し、SFC を用いることで、コンテンツを柔軟かつ効率的にネットワーク内で処理することが可能なため、ネットワークサービスを提供する際に大きな効果が期待できる。SFC については研究が盛んにおこなわれており、IETF (Internet Engineering Task Force) によって標準化が進められている。

2.2　IoT サービスへの応用

一般的な IoT サービスは、IoT データに対する処理、データの収集、イベント検知、エッジ処理といった複数の要素の組み合わせで成り立っている。これらの処理は一般にクラウドで行われるため、クラウドやそれらを配送するネットワークへの負荷の増大が懸念されている。そこで、現在ファイアウォールやロードバランサ、NAT といったミドルボックスの NF に対して主に利用されている SFC という技術を、本研究では IoT サービスを構成する処理を “ファンクション” として定義し、適用することを考える。これらのファンクションを、ネットワーク内の計算リソース（ルータなど）に配置し、VNF と同様にネットワークの状況やユーザの要求に応じて起動や終了する。そして、IoT データを、SFC の技術を用いてネットワーク内に配置されたファンクションに経由させることで、IoT データが配送される過程での処理を実現することができる。このように、ネットワーク内処理を行うことで、IoT サービスをホストするクラウドにかかる負荷を軽減することができる。本研究では、この IoT サービスにおける SFC をコンテンツ指向プロトコルの 1 つである Named Data Networking(NDN) 上で実装することを考える。NDN における SFC については、第 3 章で述べる。

2.3 SFC の関連研究

2.3.1 VNF スケジューリング

SFC は現在、特に NFV 環境における研究が盛んである。中でも効率的な SFC の実現のために VNF 配置問題と VNF スケジューリング問題について研究が進められている。本稿ではファンクション選択に関わりの深い VNF スケジューリング問題をテーマにした研究について紹介する。

[9] は VNF スケジューリング問題が flexible job-shop 問題によって定式化できることを示した。ジョブショップ問題は実行順序の決まっている複数のタスクで構成されるジョブがあり、いくつかのマシンにそれらのタスクを割り当てる場合、ジョブの処理時間が最短となるようにする問題である。この問題における、タスクを VNF、ジョブを実行順序の決まっている複数の VNF からなるサービス、マシンを VNF が割り当てられた VM として考えることで、flexible job-shop 問題を VNF スケジューリング問題に変換することができる。flexible job-shop 問題の特徴はある 1 つのタスクを処理できるマシンが複数存在しうるという柔軟性にあります。この問題は NP 困難な問題であり、[9] では VNF スケジューリング問題の定式化の方法を示したのみで、具体的な解決方法は提示されていない。また、[9] ではジョブ数にあたるサービスリクエスト数が明確であることが前提となっているため、サービスリクエスト数が不明なオンラインのスケジューリングは想定していない。

これに対し、[7] はオンラインを前提とした VNF マッピング/スケジューリング問題を定式化し、この問題の解決法として 3 つの欲張り法とタブー探索を基にしたヒューリスティックを提案した。

しかし、[9]、[7] はどちらもファンクションノード間に発生するリンク遅延を考慮していない。SFC によって管理される VNF が同じデータセンターに配置されている場合などにはこの影響は小さいかもしれないが、VNF が割り当てられているノード間の距離が大きい場合にはこの遅延を無視することができない。そこで [8] はネットワークサービスの規模が大きくなると伝送遅延が無視できないことを示し、伝送遅延を考慮した新しいスケジューリング問題を定式化した。そして、この問題を最適な混合整数線形計画法（MILP）フレームワークと遺伝的アルゴリズムを基にしたヒューリスティックを使用して評価した。

ここまで紹介してきた VNF スケジューリングの問題に関する研究はいずれも非常に複雑であるが、本研究で扱う IoT データ SFC はファンクションのインスタンスがネットワーク内のいたるところに配置されているため、より複雑な問題となる。そのため、局所的に最適な選択を継続的に行うことで、計算量を減らし、適切なファンクションインスタンスを選択を目指す。

2.3.2 ICN における SFC

[5] ではシームレスかつ動的で柔軟なファンクションチェイニングを実現する ICN ベースのフレームワークである ICN-FC が提案されている。ICN-FC では、主にビデオ処理や医療シミュレーションなどのアプリケーションレイアでの処理をファンクションとして扱っている。ICN-FC のユースケースとして、複数のカメラによって、あるスタジアムで行われる試合を撮影し、ユーザーにその複数の映像を配送するという例が紹介されている。このとき撮影された複数の映像は、最初に映像を組み合わせるファンクションにより一つの映像として処理され、次に映像データを圧縮するファンクションによりユーザーに適したデータとしてユーザに配送された。ICN-FC は、データの名前とそのデータを処理するファンクションの名前を区別するために、矢印マーク “←” を導入した。例えばファンクション B と C によって処理を施されたデータ A が要求されたとき、ICN-FC では要求コンテンツを得るために、ICN における Interest パケットであるリクエストに/C ←/B ←/A のように名前をつける。この例のようにリクエストに名前をつけたとき、データ A は初めにファンクション B で処理され、次にファンクション C で処理される。ICN-FC は従来のミドルボックスのネットワーク機能ではなく、アプリケーションレイヤの処理に対し、ICN のフレームワークを用いてファンクションチェーンを作成するという点で、共通する部分がある。一方で、ICN-FC では、元のデータとそれに

対するファンクション名をひとまとめに管理しているが、本研究では、データの名前とファンクション名は別々に管理する。このことについては第 3 章で述べることとする。

第3章　Named Data Networking (NDN)

この章では、本研究で IoT データの分散制御 SFC を実現するために扱う通信プロトコルである NDN について紹介するとともに、先行研究である DL-MD-C [12] について述べる。

3.1　Information Centric Network

ネットワーク技術はそもそもは研究目的で開発された。そのため、その主な用途は高性能なコンピュータ資源を共有するための遠隔ログインや、ファイル転送などごく限られたものだった。そのためインターネットは、どこと通信するかに重きを置き、端末の位置を示す IP アドレスに基づいたロケーション指向の通信モデルとして設計された。一方で、ネットワーク技術は、日々の研究により大きな発展を遂げ、専門の研究機関だけでなく、世間一般に普及することになった。現在ではインターネットトラヒックの大部分を Youtube や Hulu、Netflix といった動画コンテンツが占めており、もともとのインターネットの使用目的と大きな齟齬が生まれている。この齟齬を解決するために、現在の使用目的に沿ったインターネットアーキテクチャを設計しようという試みがなされている。これらの現在のコンテンツ指向性に沿った新たな通信モデルは総称して Information Centric Network (ICN) と呼ばれている。本研究では、ICN の中でも研究が最も盛んな NDN を扱う。

3.2　NDN

現在のインターネットの通信モデルは、エンドポイント間のコミュニケーション用に設計された必要最低限の機能を実装した共通のネットワーク層（IP レイヤ）を中心とした砂時計型のモデルをとっている。この thin-waist なモデルは上位及び下位のレイヤへ様々な新しい技術を導入することを可能にしており、現在のインターネットが強力であるひとつの要因といえる。NDN は図 3.1 のように IP ネットワークによく似たアーキテクチャを持っている。一方で、IP ネットワークは通信端末の位置情報を示す IP アドレスに基づいて通信を行うが、NDN はオブジェクトにつけられた名前に従って通信を行う。すなわち、IP ネットワークが特定の宛先アドレスにパケットを届けるネットワークであるのに対し、NDN は特定の名前を持つデータを取り寄せるネットワークであるといえる。この名前による通信が NDN の最大の特徴であり、現在のコンテンツが多く流通するネットワークの需要に即している。

この NDN は、2.2 で述べた IoT サービスにおける SFC に対しても非常に大きな効果を発揮する。一般に IoT サービスは多くの IoT デバイスで構成されるため、これらのデバイスを IP アドレスで管理しようとすると非常に複雑になってしまう [10]。これに対し、NDN はパケットに対して、デバイスの名前や、動画・画像といったコンテンツの名前に加え、各種データの分析・処理やデバイスの制御などの指示の名前も付けることができる。そのため、IoT デバイスに名前を付けることで直感的で簡単な管理を実現できる。さらに、IoT データにファンクションを適用する際にもリクエストをファンクションの IP ではなく名前で行うことができるため、ファンクションの位置の動的な変更にもシームレスに対応することができる。このように、IoT シナリオにおいて NDN というアーキテクチャは大いに有効である。

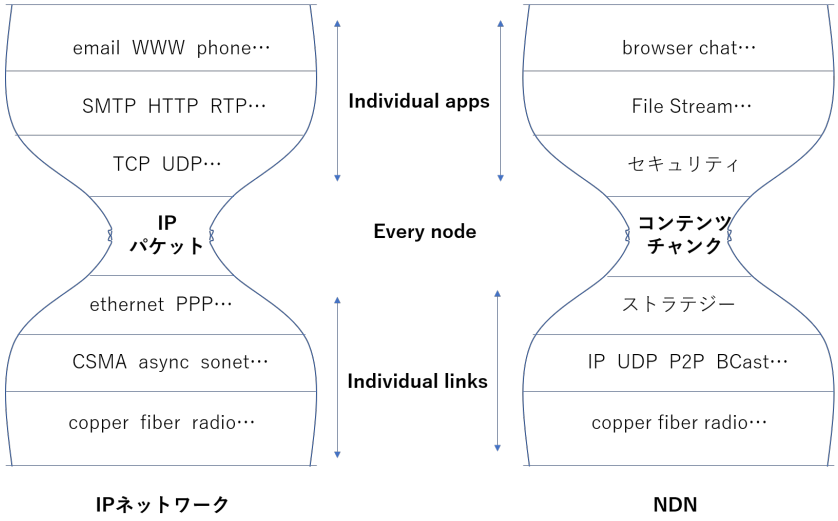

図 3.1: IP ネットワークと NDN のアーキテクチャ

3.3 NDN のアーキテクチャ

3.3.1 NDN のパケット

NDN は Interest パケットと Data パケットという 2 つのパケットの交換によって通信を行う (図 3.2)。NDN の通信モデルは pull 型であり、あるデータを要求する際にコンシューマー (NDN におけるユーザー) は Interest パケットをネットワークに送り出す。この時、Interest パケットは、要求データを識別する名前を Content Name フィールドに URL 形式の階層構造で保持している (例：/waseda.jp/video/sample A.mp4)。そして送信された Interest パケットは、要求データを保持しているプロデューサー (NDN におけるサーバー) に配送される。Interest パケットが到着すると、プロデューサーは Intesest パケットに記述されたコンテンツ名に一致するデータを Data パケットとしてコンシューマに返送する。

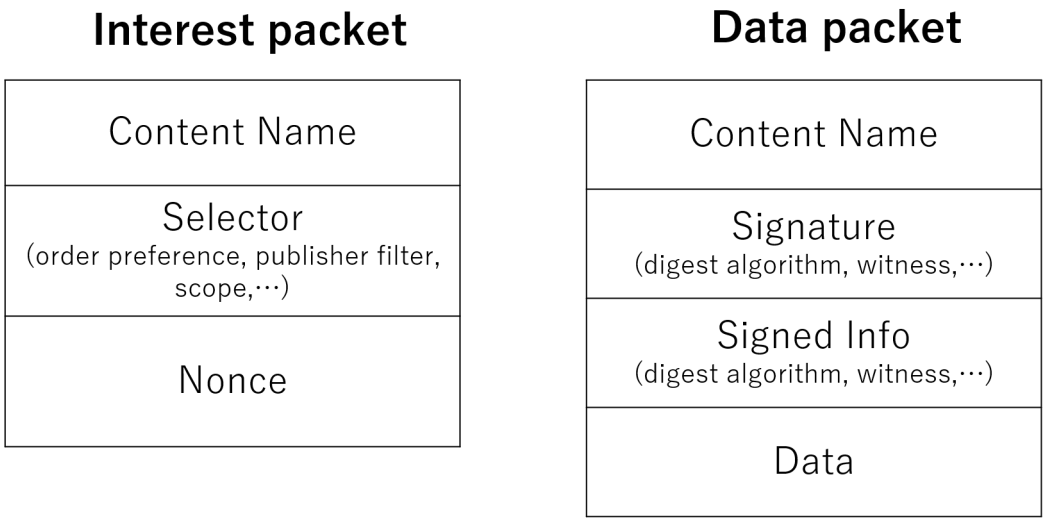

図 3.2: NDN におけるパケットの構造

3.3.2 NDN ノードのデータ構造

NDN は Interest パケットと Data パケットのフォワーディングのために、各ルータは Forwarding Information Base (FIB)、Pending Interest Table (PIT)、Content Store (CS) という 3 つの役割の異なるデータ構造を所持している (図 3.3)。

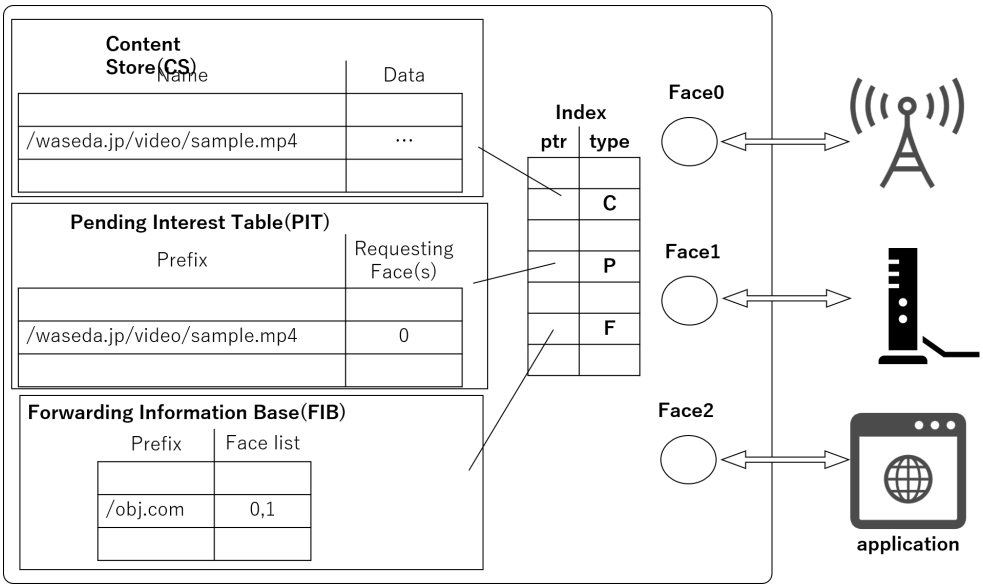

図 3.3: NDN ノードのデータ構造

- Forwarding Information Base (FIB)
  FIB は Interest パケットを、コンテンツ名が一致するデータを保持するを保持しているプロデューサーまで配送するためのデータ構造である。FIB は各コンテンツ名のプレフィックスとそのコンテンツのプロデューサーに向かうための転送先インターフェイスを記録している。NDN ではこのインターフェイスを単にフェイスと呼ぶ。NDN ルーターは Interest パケットが到着すると、FIB から要求コンテンツ名と最長一致するプレフィックスを持つ FIB エントリを検索する。FIB エントリがヒットした場合、そのエントリのフェイスリストに示されているフェイスに Interest パケットを転送する。フェイスリストにフェイスが複数記載されている場合には、フォワーディングストラテジーに従って転送を行う。

- Pending Interest Table (PIT)
  PIT は Data パケットをユーザーのもとに送り返すためのデータ構造である。各 NDN ルーターは Interest パケットが到着すると、その Interest パケットがどのフェイスから送られてきたかを PIT エントリに記述する。そして、Data パケットが到着すると、Data パケットのコンテンツ名が対応する PIT エントリを検索し、その PIT エントリに記載されたフェイスに対して Data パケットを返送する。この働きにより Data パケットは Interest パケットを送出したコンシューマーに届けられる。

- Content Store (CS)
  CS はキャッシュ機能を実現するためのデータ構造である。NDN ルータに Data パケットが到着すると、CS に同じコンテンツ名のデータがないか検索する。同じ名前のデータが存在しない場合、キャッシュポリシーに従い、そのデータをキャッシュするか否か決定する。キャッシュする際に CS のストレージが一杯だった場合、リプレースメントポリシーに従い破棄するデータを決定する。NDN ルータに Interest パケットが到着すると、まず Interest パケットに記載

されたコンテンツ名のデータが CS にあるか検索する。一致するデータが存在する場合にはそのデータを含む Data パケットを作成し、PIT に従ってそれを送り返す。これにより、人気の高いコンテンツを要求する Interest パケットは、中間ノードで要求データを獲得することができる。

3.3.3 NDN のフォワーディング

3.3.2 で述べた 3 つのデータ構造を用いて Interest パケットと Data パケットがどのようにフォワーディングされるか説明する。

- Interest パケットのフォワーディング
  Interest パケットのフォワーディングプロセスの概観は図 3.4 のとおりである。NDN ルーターは Interest パケットを受信すると、まず CS に Interest パケットの持つコンテンツ名と一致するデータがあるか検索する。一致するデータがある場合、ルーターはそのデータを Data パケットにつめてコンシューマーに送り返す。CS に一致するデータがなかった場合、PIT を参照し Interest パケットの持つコンテンツ名と一致するエントリがあるか検索する。エントリがヒットした場合、既に同じコンテンツ名を持つ Interest パケットがプロデューサーに向けて転送されているため、その PIT エントリに Interest パケットの受信フェイスを新たに追加し、Interest パケットは廃棄する。エントリがヒットしなかった場合には、新しくその Interest パケットのコンテンツ名と受信フェイスを記載したエントリを作成し、FIB から Interest パケットのコンテンツ名と一致するエントリを検索し、そのエントリが示すフェイスに Interest パケットを送出する。このとき、FIB に一致するエントリが存在しない場合には、その Interest パケットを廃棄するか NACK を返す。

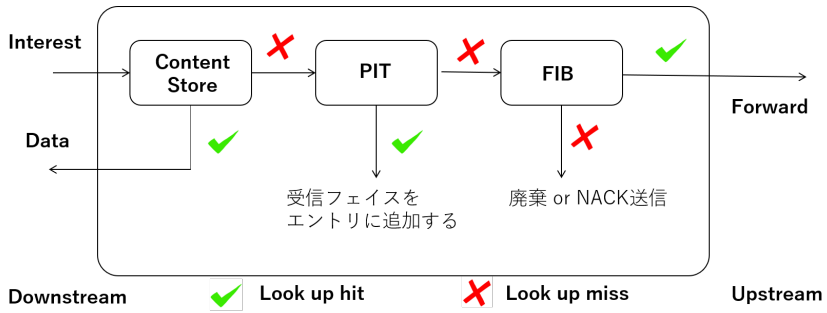

図 3.4: Interest パケットのフォワーディングプロセス

- Data パケットのフォワーディング
  Data パケットのフォワーディングの概観を図 3.5 に示す。NDN ルータは Data パケットが到着すると、まず PIT を参照して Data パケットの持つコンテンツ名と一致するエントリを検索する。エントリがヒットしなかった場合、既に要求データの Data パケットが配送されていると判断し、その Data パケットを廃棄する。エントリがヒットした場合、エントリに記載されたフェイスすべてに Data パケットを返送する。その後、NDN ルーターはキャッシュポリシーに従って Data パケットのコンテンツを CS にキャッシュするか判断する。

3.4 NDN における SFC

SFC を NDN 上で実現する NDN-FC について紹介する。NDN における SFC は、コンテンツを所持した Data パケットがファンクションをホストするノードをリクエスト通りの順番に経由する

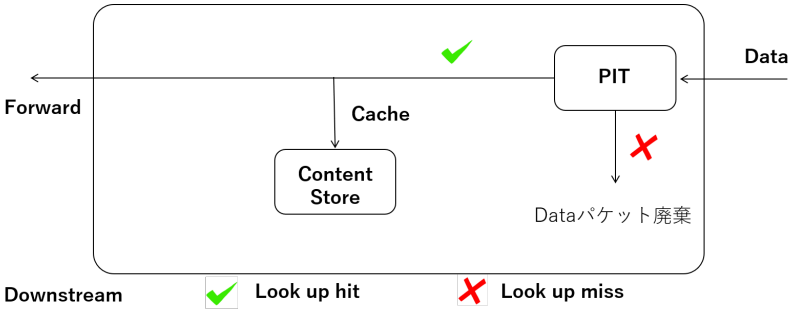

図 3.5: Data パケットのフォワーディングプロセス

ことで、コンテンツへファンクションによる処理が施されることにより実現される。3.3.3 で述べた通り、Data パケットは PIT を参照することで、Interest パケットの通過した経路を逆順に辿っていく。Interest パケットがファンクションをリクエストの逆順で通るように制御する必要がある。そのため、コンシューマーに要求された順番で Data パケットをファンクションに経由させるためには、Interest パケットがファンクションをリクエストの逆順に通るように制御する必要がある。

Interest パケットが、ファンクションを経由するように制御するために、3.3.1 で紹介した Interest パケットの構造に新たに Function Name フィールドを追加する (図 3.6)。Function Name フィールドにコンシューマーの要求するファンクション名をリクエストの逆順で URL 形式によって記述し、Interest パケットの制御のためにこのフィールドを先頭プレフィックスから参照することにする。このとき先に述べたように、Interest パケットは要求されたファンクションを逆順で経由する必要があるため、Function Name フィールドに記述するファンクション名も逆順にする必要がある。例えば、あるコンテンツに対して、F1 → F2 → F3 という 3 つのファンクションによるファンクションチェインを適用したい場合、F3 → F2 → F1 の順に Interest パケットを経由させる必要がある。そのため、Function Name フィールドには/F3/F2/F1 と記述する。

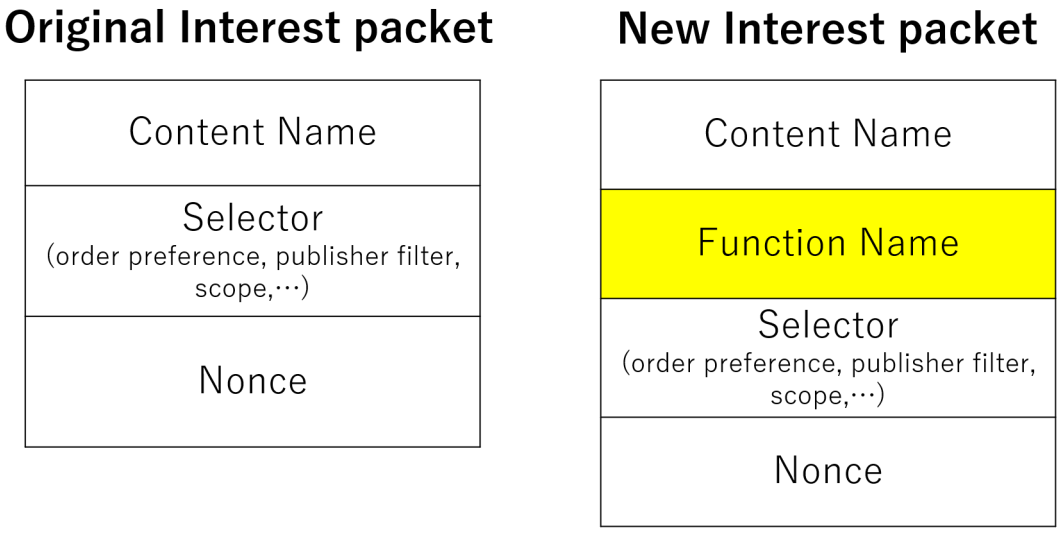

図 3.6: NDN-FC のパケットフォーマット

NDN-FC のフォワーディングプロセスについて説明する。Function Name フィールドにファンクション名が記載されている Interest パケットがルーターに到着すると、コンテンツ名ではなくそのファンクション名を参照してフォワーディングを行う。FIB には従来のエントリに加えファンクション名とその転送先を示すエントリも登録されているため、Interest パケットの持つ Function Name

フィールドを参照し、先頭のファンクション名と一致する FIB エントリを検索し、送信先フェイスを決定する。その後、Interest パケットが要求ファンクションに到達すると、そのファンクションは Interest パケットの Function Name フィールドの先頭にある自分の名前と一致するファンクション名を削除する。この動作の繰り返しによって Function Name フィールドのプレフィックスがすべて削除された場合、各ルーターはもともとのフォワーディングに従い、Interest パケットの要求するコンテンツ名のプロデューサーにパケットを転送する。

図 3.7 を用いて説明する。まず、User が Function Name フィールドに/F3/F2/F1 という記載のある Interest パケットをネットワークに送信する。すると、この Interest パケットを受け取ったルーター 1 は Function Name フィールドに記載があるので、その先頭プレフィックスを参照する。このとき、先頭プレフィックスは”F3”するのは Function3 なので、F3 のエントリを参照し記載してある送り先のフェイスに Interest パケットを転送する。Interest パケットが F3 に到達すると、F3 は Interest パケットの Function Name フィールドの先頭にある自身のファンクション名を削除する。これによって、Function Name フィールドは/F2/F1 になる。その後、この Interest パケットは同様の手順によって、Function Name フィールドに従って F2 → F1 へとルーティングされる。この Interest パケットは F1 に到達すると、Function Name に記載されたプレフィックスがすべて削除されるため、Interest パケットのコンテンツ名に従ってプロデューサへ転送される。

Interest パケットがファンクションを経由してプロデューサーに到達すると、プロデューサーは Interest パケットのコンテンツ名が示すコンテンツを持った Data パケットをコンシューマーに返送する。Data パケットは PIT を参照しながら Interest パケットの経路を逆順に辿ってコンシューマーまで返送される。その過程で、Data パケットは Interest パケットとは逆順にファンクションを経由する。Data パケットがファンクションに到達したとき、ファンクションは Data パケットの持つコンテンツに特定の処理を加える。Interest パケットの経路をたどることで、要求されたファンクションをすべて経由し、Data パケットがコンシューマーのもとに届けられると、SFC が完了する。図 3.7 の例では、Interest パケットが F3 → F2 → F1 の順で経由して配送されるため、Data パケットは逆順に F1 → F2 → F3 の順でファンクションノードを経由し、処理される。

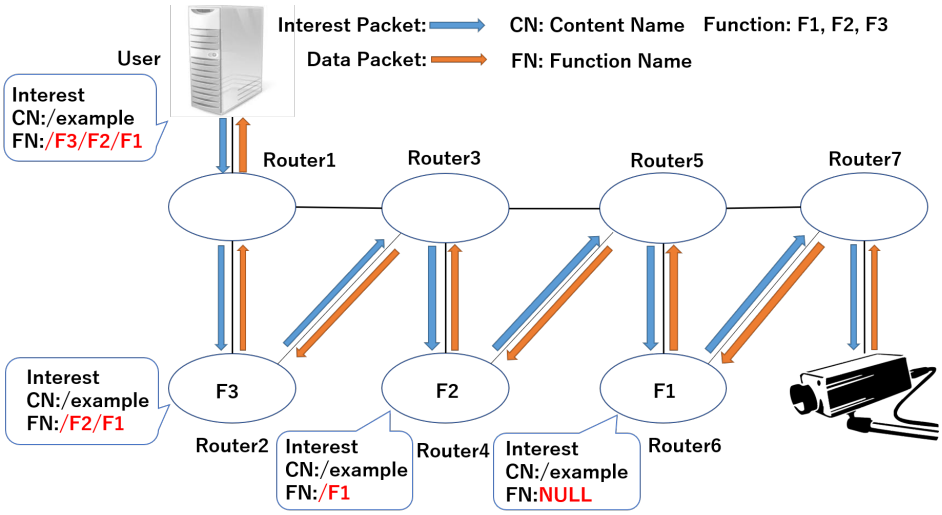

図 3.7: NDN-FC のフォワーディング

3.5 DL-MD-C

SFC において、1 つのファンクションのインスタンスにアクセスが集中することを避けるため、同じ種類のファンクションのインスタンスを複数用意し、ネットワークに配備することを考える。その

ためには、適切なインスタンスを選択するメカニズムが必要になる。[12] は、ファンクションの負荷を分散し、サービス実行の遅延を最小限に抑えるために、ネットワーク負荷とファンクション混雑度に焦点を当てた、NDN-FC を想定したファンクション選択手法である DL-MD-C(Lord Distribution and Minimum Delay in Centralized) を提案した。DL-MD-C では、同じ種類のファンクションのインスタンスが複数配備されるとき、ファンクション名の末尾に識別子を追加することでこれらを区別しルーティングを可能にする。例えば F1 というファンクションのインスタンスを 3 つネットワークに配備するとき、それぞれに F1a、F1b、F1c という名前をつけることで名前の衝突を回避する。コンシューマーは SFC リクエストのために Interest パケットのファンクション名にプレフィックス列を記入する際に、評価関数に従ってファンクション名のプレフィックスそれぞれを識別子付きのものに置き換える。DL-MD-C の概観を図 3.8 を用いて述べる。User が/F3/F2/F1 というファンクションチェインを要求するとき、ネットワーク全体を監視しているコーディネーターとのやり取りによって、各ファンクションインスタンスの情報を得る。その情報と評価関数に従って、F1、F2、F3 のインスタンスの中からそれぞれ適したものを選択し、ファンクションチェインを/F3b/F2a/F1c のように識別子付きのものに置き換え、Interest パケットの Function Name フィールドに記入する。その後、Interest パケットは 3.4 で述べた、NDN-FC のフォワーディングプロセスに従い配送される。

この手法は、ファンクション呼び出し回数をファンクション混雑度の指標として、SFC 実行中に通過したホップ数をネットワーク負荷の指標として、ファンクション選択のための評価関数 3.1 を定義した。この式について詳しく説明する。n 個のノードから成るネットワーク内に m 個のファンクションが配備されているとする。このとき、ある IoT サービス S における一連の i 個のファンクション $F_S = f_1, f_2, \cdots, f_i$ で構成される SFC リクエストを想定する。ここで f_k はファンクションを表す。F_s は $f_1 \to f_2 \to \cdots \to f_i$ の順に処理する必要がある。ファンクション f_j が割り当てられたノードの集合を N の部分集合 $N(f_j)$ とすると、ファンクション f_j が配備されたあるノード $n_k(n_k \in N(f_j))$ において、過去 x 回のファンクション呼び出しの間にファンクション f_j が呼び出された回数を C_{jk} で表す。また、あるノード $n_k(n_k \in N)$ から任意のノード $n_l(n_l \in N)$ までのホップ数を $H_{k,l}$ で表す。ただし、$n_k \in N(f_j), n_l \in N(f_{j-1})$ であり、f_0 はエンドユーザーである。以上より、あるサービス S におけるノード n_k でのファンクション選択基準 U_k は式 (3.1) で定義され、DL-MD-C はこれが最小となるノード $n_k(n_k \in N(f_j))$ をコンシューマーが各ファンクションに対して決定する。ここで、係数 α、β はそれぞれホップ数とファンクション呼び出し回数に対する重みであり、任意の定数である。

$$U_k = \sum_{j=1}^{i} (\alpha H_{kl} + \beta C_{jk}), \tag{3.1}$$

DL-MD-C は、最小の U_k を求めるためにすべてのファンクションの呼び出し回数とネットワークトポロジ情報を監視するコーディネーターが必要となり、[12] ではシミュレーター上のオブザーバーがその役を務めていたため、理想的な経路制御を実現できていたが、実際のネットワークではこのコーディネーターがボトルネックになってしまう。また、最小の U_k を求めるためには、ファンクションチェインを構成するファンクションのインスタンスの組み合わせを全て調べる必要があり、計算コストが高いという問題がある。

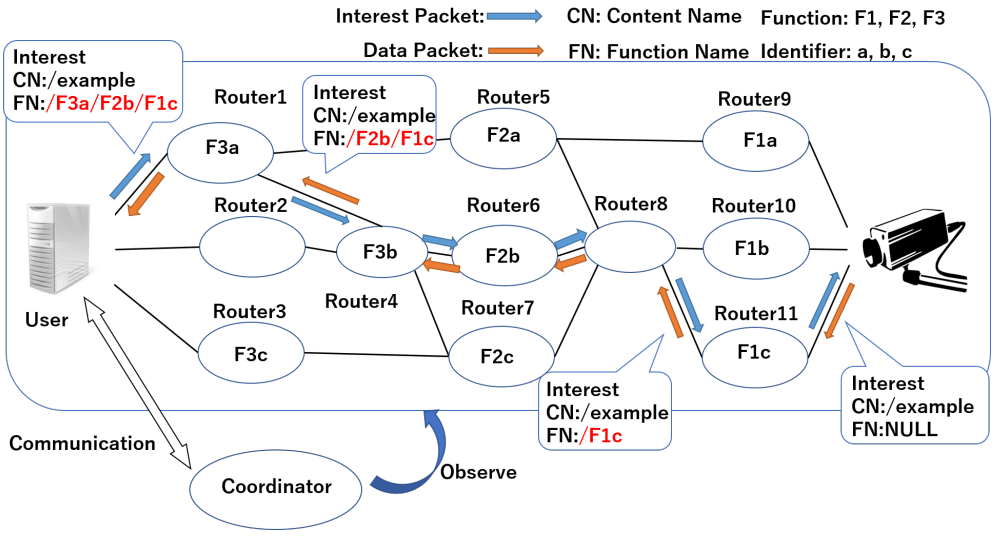
Interest Packet:
CN: Content Name
Function: F1, F2, F3
Data Packet:
FN: Function Name
Identifier: a, b, c
Interest
CN:/example
FN:/F3a/F2b/F1c
Router1
F3a
Interest
CN:/example
FN:/F2b/F1c
Router5
F2a
Router9
F1a
Router2
Router6
Router8
Router10
F3b
F2b
F1b
User
Router3
Router4
Router7
Router11
F3c
F2c
F1c
Interest
CN:/example
FN:/F1c
Interest
CN:/example
FN:NULL
Communication
Coordinator
Observe

図 3.8: DL-MD-C

第4章　提案手法

3.5 で述べた DL-MD-C の問題を回避するため、局所的に最適なファンクション選択を行う手法について提案及び実験・評価を行う。この章では、提案手法の概要及び実装方法を述べていく。

4.1　提案手法の概要

本稿で提案するファンクション選択手法は、コンシューマーがコーディネーターとのやり取りによって一元的にファンクションチェインの各プレフィックスに対しファンクション選択を行う DL-MD-C と対照的に、コンシューマーは Interest パケットのファンクション名の先頭プレフィックスに対してのみ選択を行い、ファンクションノードに Interest パケットが到着することでファンクション名の先頭プレフィックスが削除される度に新たに先頭になったプレフィックスに対して選択を行う。すなわち、集中制御である DL-MD-C 対し、提案手法は分散制御であるといえる。各ノードは各ファンクションインスタンスの呼び出し回数と各ファンクションインスタンスまでのホップ数を記録するテーブルを管理しており、データパケットに便乗した情報によってこのテーブルを更新する。そのため、DL-MD-C においてボトルネックの原因となるようなコーディネーターを提案手法は必要としない。

提案手法は DL-MD-C の問題点を回避することができるが、近似アルゴリズムであり、厳密解を保証するものではない。また、本研究では 1 つのノードがホストするファンクションインスタンスは 1 つまでを想定している。ネットワーク内のファンクションインスタンスの数がノードの数より少ない限り、この想定は有効である。ただし、提案手法は 1 つのノードが複数の関数をホストしていても適用できる。

4.2　Function Information Table とパケットの拡張

提案手法を実現するために、各ファンクションインスタンスまでのホップ数と呼び出し回数を記録するために Function Information Table というデータ構造を新たに各ファンクションノードとコンシューマーに追加する。さらに、FIT の維持のために、Interest パケットと Data パケットをそれぞれ拡張する。これらについて詳しく説明する。

4.2.1　Function Information Table

コンシューマーやファンクションをホストするルーターが、他のファンクションインスタンスの情報を参照しファンクション選択を行うために、各ノードにそれらの情報を記録しておくためのデータ構造を追加する。このデータ構造を Function Information Table (FIT) と呼ぶ。FIT はエントリごとにファンクションインスタンスの名前とそのインスタンスまでのホップ数およびそのインスタンスの呼び出し回数を記録する。さらに、ファンクションをホストしているノードの FIT はほかのファンクションインスタンスの情報に加えて、自身の呼び出し回数も記憶する。この自身の呼び出し回数の情報は、ある一定の期間 T で初期化される。また、コンシューマーやファンクションノードでファンクション名がそのインスタンス名に置き換えられたとき、選択したインスタンスの呼び出し回数をインクリメントする。図 4.1 を例にとって FIT の働きを詳しく説明する。コンシューマーの FIT は他のファンクションインスタンスまでのホップ数 (Partial Hop Count: PHC) および呼び出

し回数 (Function Call Count: FCC) を記憶している。これらの情報は、4.2.2 で述べる Data パケットに追加された働きによって更新される。コンシューマーが/F3/F2/F1 というファンクションチェインを要求するとき、先頭ファンクションである F3 のうちどのインスタンスを選択するかを、この FIT を参照して決定する。このとき、F3a を選択した場合、F3a の FCC を 1 増加させる。

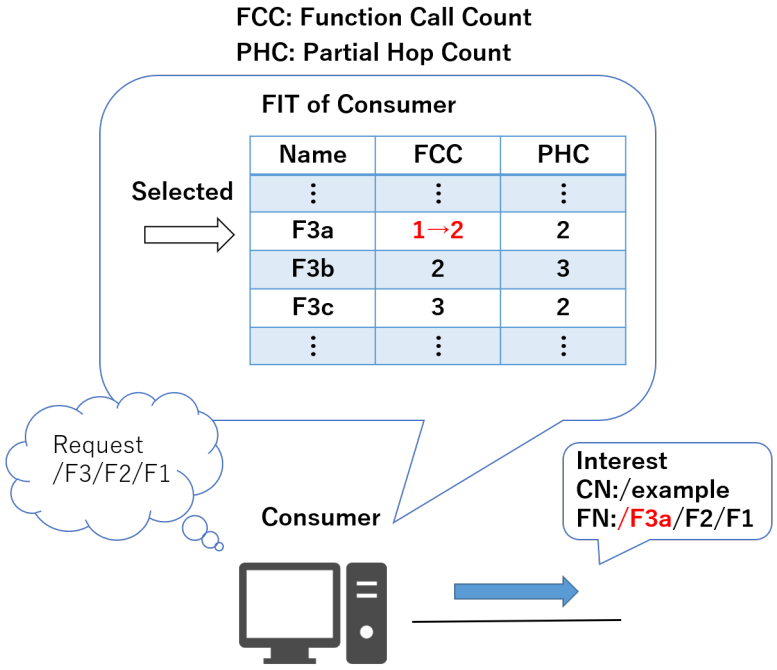

図 4.1: FIT の構造

4.2.2 パケットの拡張

ノード間で情報を交換し、FIT を維持するために、パケットを拡張する。Interest パケットには Function Full Name フィールド、Data パケットには Function Name フィールド、Partial Hop Count フィールド、Function Call Count フィールドを追加した。Data パケットに追加された Function Name フィールドは FIT の更新のためにひとつ前のファンクションインスタンスの名前を認識するために使用され、Interest パケットに追加された Function Full Name フィールドはこれを初期化するために使用される。Partial Hop Count フィールドと Function Call Count フィールドはファンクションインスタンスの情報を Data パケットに付与するため用いられる。これらについて、詳しく説明する。

Function Full Name フィールドは選択されたファンクションインスタンスの識別子を持つ完全なシーケンスをプロデューサーに伝達するために利用される。コンシューマーやファンクションノードでファンクションインスタンスが選択される度に、選択されたファンクションインスタンスの名前がこのフィールドに追加される。これによって、実行される SFC のファンクションインスタンスのシーケンスをプロデューサーに伝えることができる。プロデューサーは Data パケットを作成する際に、この Function Full Name フィールドの値を Data パケットの Function Name フィールドにコピーする。ある User が/F3/F2/F1 というファンクションチェインを要求する場面を例にとって、4.2 を用いて拡張された Interest パケットの働きを説明する。まず、User がファンクション F3 のインスタンスとして F3a を選択したとき、Function Name フィールドの先頭プレフィックスである F3 を F3a に置き換えるとともに、Function Full Name フィールドに/F3a と記載する。つぎにこの Interest パケットが F3a をホストする Router1 まで到達すると、Function Name フィールドの先頭にある F3a が削除され、次に先頭に来るファンクション F2 のインスタンスが選択される。このとき F2b が選択され

ると、Function Name フィールドの先頭が F2b に置き換えられ、Function Full Name フィールドの末尾に F2b が追加される。同様に、ファンクション F1 においても選択が行われ、F1c が Function Full Name フィールドに追加された Interest パケットが NDN-FC のフォワーディングプロセスによってプロデューサーまで配送されたとき、Function Full Name フィールドには/F3a/F2b/F1c という選択されたファンクションインスタンスのシーケンスが記入されている。こうして得られたシーケンスをプロデューサーは Data パケットの作成時に Function Name フィールドに記入する。

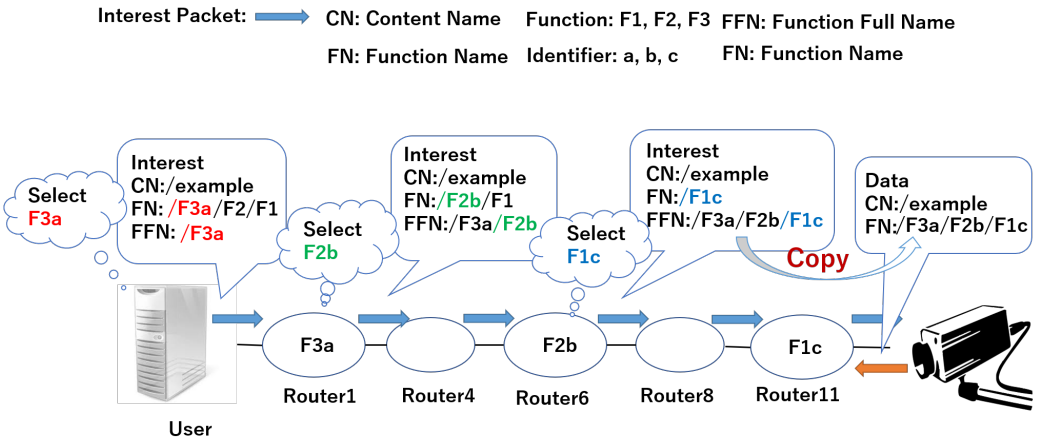

図 4.2: Interest パケットの拡張

Data パケットに追加された Function Name フィールドは、Data パケットに便乗した情報を用いて FIT を更新するために、ひとつ前のファンクションインスタンスの名前を認識する際に使用される。先述の Function Full Name フィールドの働きによってこのフィールドはファンクションインスタンスの名前のシーケンスを持っている。Data パケットが 1 つ目のファンクションに到達したとき、Data パケットはファンクションの情報を持っておらず、FIT を更新する必要がないため、Function Name フィールドは参照されない。 2 番目以降のファンクションもしくはコンシューマーに到達したとき、すなわち Data パケットは FIT を更新するための情報を持っている場合、Function Name フィールドの末尾にあるファンクションインスタンスの名前が抽出され、削除される。抽出されたインスタンス名はひとつ前のインスタンスを示すので、FIT からそのインスタンス名と一致するエントリを検索し、Data パケットの情報を用いてエントリを更新する。

Data パケットに追加された Partial Hop Count フィールドはファンクションノード間のホップカウントを測定するためのフィールドである。Data パケットが生成されたとき、このフィールドの値は NULL である。ファンクションのインスタンスが実行され、Data パケットを送出するときこのフィールドには 1 が代入される。各ノードで Data パケットを送出するとき、このフィールドの値が NULL でないなら、このフィールドを 1 加算する。

一方で、Function Call Count フィールドはファンクションインスタンスの呼び出し回数を伝達するためのフィールドである。Data パケットが生成されたとき、このフィールドの値は NULL である。ファンクションのインスタンスが実行され、Data パケットが送出されるとき、そのインスタンスの呼び出し回数をこのフィールドに代入する。

Data パケットがプロデューサーから送出され、1 つ目のファンクションに到達したとき、その Data パケットの Partial Hop Count フィールドと Function Call Count フィールドの値は NULL であり、FIT を更新する必要はない。一方で、Data パケットが 2 つ目以降のファンクションやコンシューマーに到達したときには FIT を更新する必要があるため、Function Name フィールドの末尾からファンクションのインスタンス名を抽出し、FIT からそのインスタンス名と一致するエントリを検索してホップ数と呼び出し回数の情報を更新する。

拡張された Data パケットの働きを図 4.3 を例にとって説明する。Data パケットの Function Name

フィールドには/F3a/F2b/F1c という記載があり、F1c → F2b → F3a の順でファンクションを経由する。Data パケットがプロデューサーから送出され、1 つ目のファンクションインスタンス F1c が割り当てられているノードである Router11 に到達したとき、この Data パケットは FIT を更新するためのデータを持っていないため、Router11 の FIT の更新は行われない。そしてコンテンツに F1c の処理が加えられた後、Router11 は自身に割り当てられた F1c の呼び出し回数を Function Call Count フィールドに記入し、Partial Hop Count フィールドの値を 0 に初期化する。一方で、ファンクション F2 のインスタンス F2b を割り当てられた Router6 に先の Data パケットが届いた際には、Data パケットがインスタンス F1c の情報を持っているため、FIT の更新が行われる。このとき、Data パケットの Function Name フィールドの末尾が更新すべきエントリのファンクションインスタンス名 (この場合では F1c)、Partial Hop Count フィールドがそのインスタンスまでのホップ数、Function Call Count フィールドの値がそのインスタンスの呼び出し回数を示している。このとき、FIT の更新のために抽出された Function Name フィールドの末尾のインスタンス名は削除される。

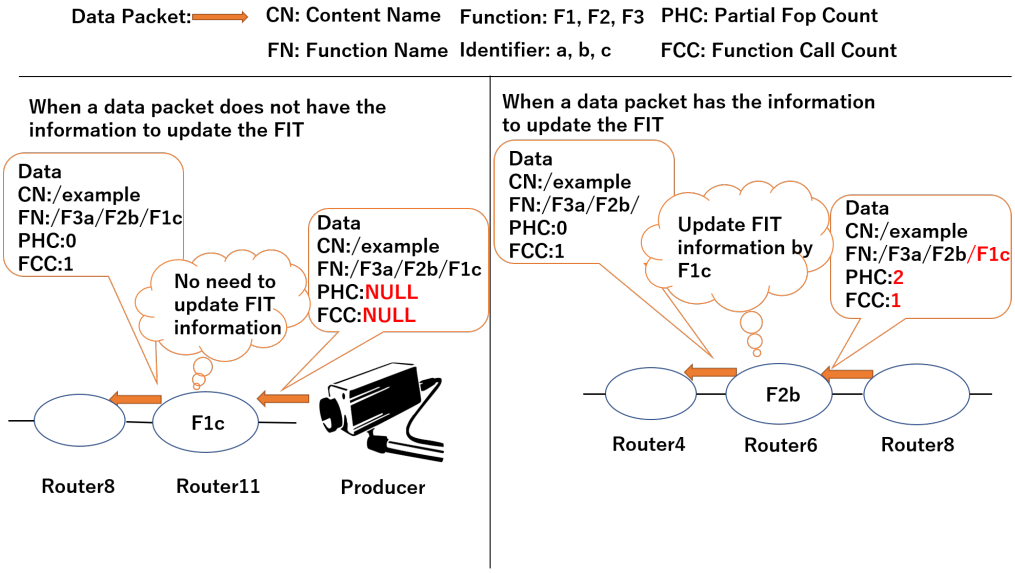

図 4.3: Data パケットの拡張

4.3　アルゴリズム

3.4 で述べた通り、Interest パケットの Function Name フィールドに記載がある場合、そのパケットは Function Name フィールドに記載されたシーケンスの最初のファンクションに向けて転送される。すなわち、ファンクション名に記載されたプレフィックスは先頭のもの以外は転送の目的では無視できる。特定のファンクションインスタンスが割り当てられていないファンクション名が Interest パケットの Function Name フィールドの先頭に現れるたびに、すなわちコンシューマーが Interest パケットを作成したとき、または Interest パケットがファンクションノードに到達し先頭プレフィックスが削除されたときに、そのノード n_k は 4.2.1 で述べた FIT を参照し、先頭に現れたファンクション名と一致するファンクションインスタンスのエントリを検索する。そして各エントリからファンクションノード間のホップ数 H_{kl} とファンクション呼び出し回数 C_{jl} を取得する。このとき、ノード n_l はそのエントリが示すファンクションインスタンスが割り当てられているノードである。そして、式 (4.1) の V_l の値を最小化するノード n_l が選択される。ここで、α と β はそれぞれ任意の定数である。

$$V_l = (\alpha H_{kl} + \beta C_{jl}) \tag{4.1}$$

コンシューマー n_0 が/F3/F2/F1 という SFC リクエストを送信するとする (図 4.4)。まず、コンシューマーは FIT から得た F3 のインスタンスに関する情報と評価関数式 (4.1) に従い、関数 F3 のインスタンスの中から一つを選択する。ここでは F3a が選択されたとすると、この Interest パケットのファンクション名は/F3a/F2/F1 という風に書き換えられ、F3a に転送される。その後、F3a をホストしているノードに、この Interest パケットが到達すると、先頭の F3a というプレフィックスが削除される。その後、新たに先頭になった F2 が再度式 4.1 によって評価され、選択されたファンクションインスタンス名に置換される。

この操作を繰り返すことで、適切なファンクションインスタンスを選択しながら、Interest パケットの Content Name フィールドで指定されたプロデューサーに Interest パケットが配送される。

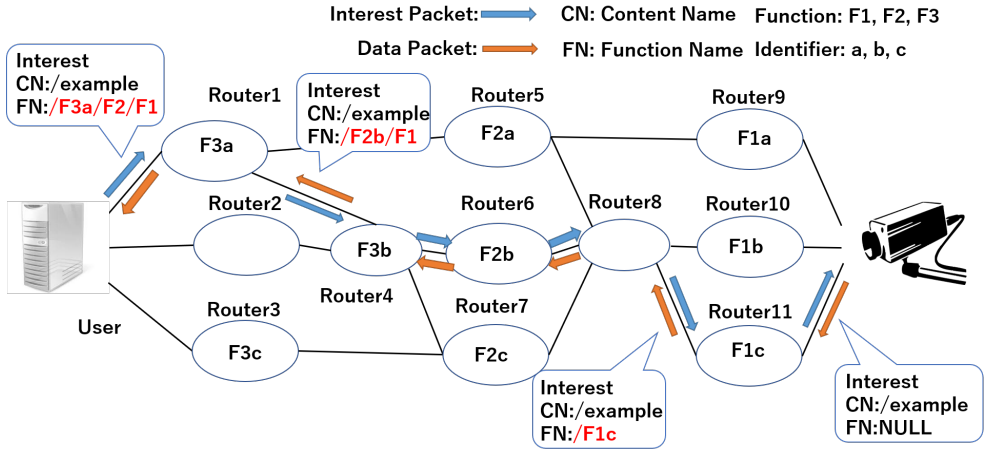

図 4.4: 分散制御によるフォワーディング

第5章　評価

この章では第 4 章で提案した分散制御がたファンクション選択手法の性能をシミュレーションにより評価し、その結果について考察する。

5.1　実験環境

提案手法の有効性を評価するためにシミュレーションによる評価を行った。 シミュレーションでは、NDN のシミュレーターである ndnSIM [6] を使用して、ファンクションインスタンスの負荷とファンクションチェインで構成されるサービスの実行遅延を測定した。 測定には、A 24-node US nation-wide トポロジ [2] を使用した (図 5.1)。

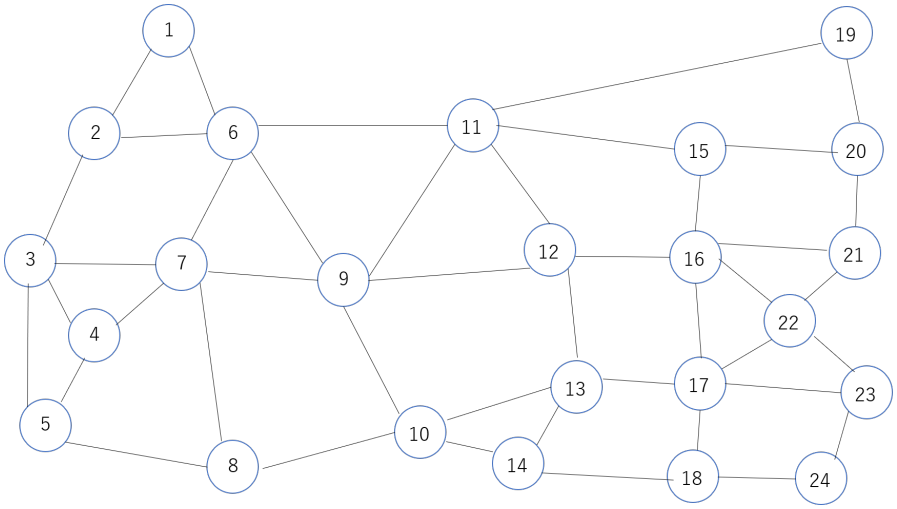

図 5.1: A 24-node US nation-wide トポロジ

このネットワークに、5 種類のファンクション F1、F2、F3、F4、F5 のインスタンスをそれぞれ 3 つずつ配置した。ファンクションのインスタンスはそれぞれその種類ごとに、末尾の “a”、“b”、“c” という識別子によって、(例えば F1a、F1b、F1c のように) ファンクション名と区別される。ネットワークには 4 つのコンシューマーが接続される。コンシューマーはそれぞれ SFC 要求を生成する。加えて、ネットワークにはプロデューサーが 2 つ接続される。コンシューマーは、表 5.1 に示すあらかじめ用意された 12 種類の SFC リクエストから 1 つを定期的に選択し、Interest パケットを送信する。 この環境で、コンシューマーから 300 件の SFC リクエストを行う実験を行った。比較のために、提案手法の他に 3 つのファンクション選択手法 DL-MD-C、Hop-first、Lord-first を用意し、これらについても実験を行った。表 5.2 は、このシミュレーションにおける提案手法と DL-MD-C のパラメーターを示す。また、DL-MD-C には [12] と同様にファンクション選択のために通信を必要としないコーディネーターを用意して計測を行う。これは、提案手法のパフォーマンスを理想的なファンクション選択と比較するためである。

ファンクション選択手法の Hop-first と Load-first は以下の通りである。

表 5.1: 12 種類の SFC リクエスト

Type	Request
1	F1→F2→F4
2	F1→F2→F5
3	F2→F1→F4
4	F2→F1→F5
5	F1→F3→F4
6	F1→F3→F5
7	F3→F1→F4
8	F3→F1→F5
9	F2→F3→F4
10	F2→F3→F5
11	F3→F2→F4
12	F3→F2→F5

表 5.2: シミュレーションにおけるパラメータ

Parameter	DL-MD-C	Distributed Control
α	1	1
β	1	1
x	30[times]	-
T	-	50[ms]

- Hop-first
  Hop-first は、ファンクション選択時にファンクションインスタンス間のホップ数の情報のみを参照し、ホップ数が最も少なくなるファンクションインスタンスを選択するファンクション選択手法である。すなわち、Hop-first は、式 4.1 のパラメーター $(\alpha, \beta) = (1, 0)$ に等しい。

- Lord-First
  Lord-first は、ファンクション選択時にファンクションインスタンスの呼び出し回数の情報のみを参照し、呼び出し回数が最も少ないものを選択する手法である。 Lord-first は、式 4.1 のパラメーター $(\alpha, \beta) = (0, 1)$ に等しい。

5.2　ファンクション分散

ファンクションの負荷がどの程度分散されているかを評価した。図 5.2 は、それぞれの手法について、ファンクションインスタンスごとの呼び出し回数を示している。分散度をより明確に示すために、[12] で定義された**ファンクション分散度**を各手法に対して求める。ファンクション分布度は、実際の呼び出し回数とそのファンクションが均等に呼び出されたときの呼び出し回数の差の絶対値の和で求められる。すなわち、ファンクション分散度 D は次のように定義される。

$$D = \sum |(\text{実際のファンクション呼び出し回数}) - (\text{均一な呼び出し回数})|.$$

ファンクション分散度 D が 0 に近いほど、各ファンクションが均等に呼び出されたことを意味する。図 5.3 に各手法のファンクション分散度を示す。

図 5.3 を見ると、提案手法のファンクション分散度は、DL-MD-C と同様に Lord-First と Hop-First の間の値を示しているが、その値はわずかに異なっている。この原因を探るために図 5.2 を見ると、提案手法の F1 から F3 までのファンクション呼び出し回数は、DL-MD-C と似たような特性を示している。一方で、F4 と F5 は F1 と F3 に比べて差異が大きくなっている。この差異の原因としては、図 5.1 を見るとわかるように、F4 と F5 は F1 から F3 と比べて呼び出し頻度が低いことがあげられる。DL-MD-C には全体を監視するコーディネーターが存在するため、呼び出し頻度に関わらず各ファンクションの呼び出し頻度の情報を得ることができる。一方で、提案手法は FIT の更新が Data パケットに依存している。そのため、ファンクションの呼び出し頻度は FIT を管理し維持するためにも重要な要素であると考えられる。

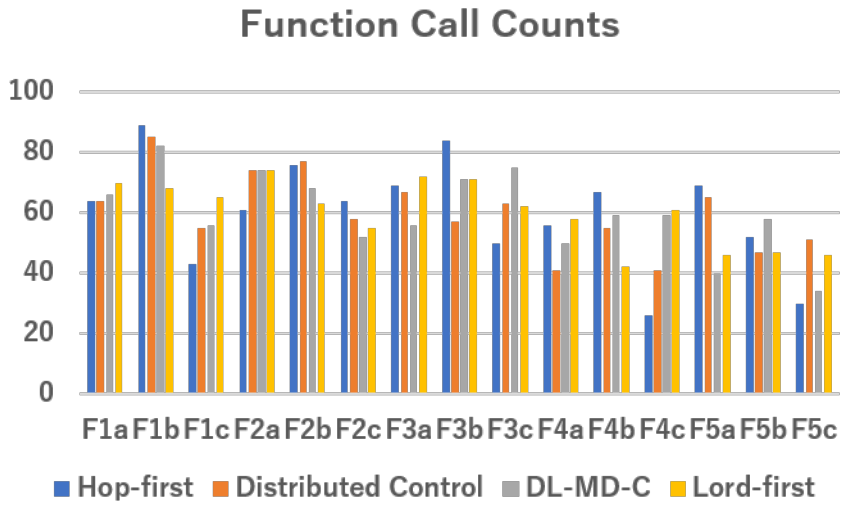

図 5.2: ファンクション呼び出し回数

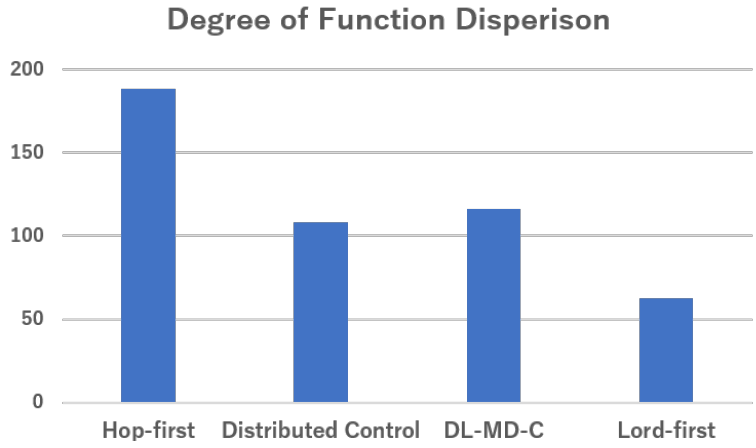

図 5.3: ファンクション分散度

5.3　サービス実行遅延

ユーザーが SFC リクエストを送信してから、各ファンクションによって処理された Data パケットをユーザーが受信するまでの遅延を測定した。ファンクションが Data パケットを処理するために必要な時間は $40ms$ とする。様々なネットワーク負荷に対するパフォーマンスを比較するために、リクエスト頻度を $10/s$ から $5/s$ ずつ増やし、$30/s$ まで測定した。 このとき、平均サービス実行遅延を図 5.4 に示す。 提案手法は DL-MD-C とほぼ同等のパフォーマンスを示した。しかし、リクエスト頻度を $30/s$ に上げると、提案手法の実行遅延は DL-MD-C より若干大きくなる。 これは、それぞれのファンクション呼び出し頻度の定義による差異であると考えられる。DL-MD-C はネットワーク全体を監視し、過去 x 回のファンクション呼び出しのうち何回対象のファンクションインスタンスが呼び出されたかでファンクション呼び出し回数を求めるため、常に適度なサイズのファンクション呼び出し回数を獲得できる。それに対し、提案手法ではファンクション呼び出し数を一定の期間 T でリフレッシュするため、短期間で過剰なリクエストが発信された場合にホップ数に対して大きな呼び出し回数を抱えてしまい、適切なファンクション選択を難しくしてしまう。

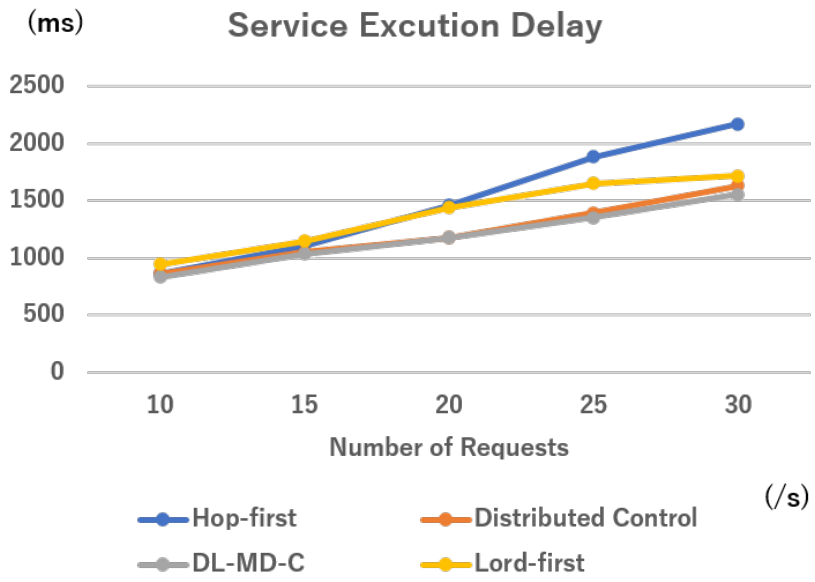

図 5.4: 平均サービス実行遅延

第6章　結論

本稿では、実際のネットワーク上で NDN を用いた SFC による IoT データのネットワーク内処理を行うためのファンクション選択手法を提案し、その実装方法について述べ、その評価と考察を行った。リクエストがファンクションノードに送られるたびにファンクションインスタンスを選択する分散制御により、ボトルネックとなるコーディネーターの存在と計算量の増大の問題を回避することを目指した。提案手法を用いることで、理想的なファンクション選択手法に非常に近いパフォーマンスを発揮することができる。提案手法のファンクション分散度とサービス実行遅延を評価すると、提案手法は、ネットワークに負荷をかけずにネットワーク内の情報を収集するコーディネーターの存在を前提とした DL-MD-C に近い性能を示した。

今後の課題として、NDN の特徴であるネットワークノードでのキャッシュの適用があげられる。ファンクションで処理したデータを事前にキャッシュしておくことで、サービス実行遅延を大幅に短縮できると考えられる。ただし、提案手法をそのまま適用すると、キャッシュ使用時には FIT を更新するための情報が古くなっていることが懸念される。また、本稿ではファンクションの選択のみを検討しているが、ネットワークを効率的に利用するためには、どこにファンクションインスタンスを配置するかも重要な問題である。将来的には、ファンクションの配置と選択をうまく刷り合わせをする必要がある。

参考文献

[1]

[2] Dragos Andrei, Biswanath Mukherjee, and Dipak Ghosal. Online scheduling of large file transfers over lambda grids.

[3] Joel Halpern, Carlos Pignataro, et al. Service function chaining (sfc) architecture. In *RFC 7665*. 2015.

[4] Yohei Kumamoto, Hiroki Yoshii, and Hidenori Nakazato. Real-world implementation of function chaining in Named Data Networking for IoT environment. In *2020 IEEE ComSoc International Communications Quality and Reliability Workshop (CQR)*, pages 1 – 6, May 2020.

[5] 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. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7, 2017.

[6] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnSIM 2: An updated NDN simulator for NS-3. Technical Report NDN-0028, Named Data Networking Project, November 2016.

[7] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and S. Davy. Design and evaluation of algorithms for mapping and scheduling of virtual network functions. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, pages 1–9, 2015.

[8] L. Qu, C. Assi, and K. Shaban. Delay-aware scheduling and resource optimization with network function virtualization. *IEEE Transactions on Communications*, 64(9):3746–3758, 2016.

[9] J. F. Riera, E. Escalona, J. Batallé, E. Grasa, and J. A. García-Espín. Virtual network function scheduling: Concept and challenges. In *2014 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, pages 1–5, 2014.

[10] 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). In *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 117–128, 2016.

[11] Weisong Shi and Schahram Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, 2016.

[12] Y. Shiraiwa and H. Nakazato. Function selection algorithm for service function chaining in NDN. In *2019 IEEE ComSoc International Communications Quality and Reliability Workshop (CQR)*, pages 1–5, April 2019.

[13] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. Named data networking. *SIGCOMM Comput. Commun. Rev.*, 44(3):66–73, July 2014.

研究業績

1. Naoki Yamaguchi and Hidenori Nakazato. 2020. Distributed control function selection method for service function chaining in NDN. Proceedings of the Workshop on Cloud Continuum Services for Smart IoT Systems. Association for Computing Machinery, New York, NY, USA, 26–31. DOI:https://doi.org/10.1145/3417310.3431397

謝辞

本研究を進めるにあたり、中里秀則教授から多くの助言や指導をいただきました。そのことに関しまして、深く感謝申し上げます。また、中里研究室の皆様には議論を通して多くの気づきや支えを受けました。ここに感謝の意を表しまして、謝辞と致します。

2021 年 1 月 25 日 山口直樹