特別研究報告

題目

情報指向ネットワークにおけるネーミングスキーマを用いた移動可能ルータの実現

指導教員
村田 正幸 教授

報告者
高 宇豪

2017 年 2 月 14 日

大阪大学 基礎工学部 情報科学科

平成 28 年度 特別研究報告

情報指向ネットワークにおけるネーミングスキーマを用いた移動可能ルータの実現

高 宇豪

内容梗概

近年、ホストではなくコンテンツ主体のネットワークである情報指向ネットワーク (ICN: Information-Centric Networking) が、IoT 環境における柔軟な通信が可能なネットワークとして注目されている。ICN は TCP/IP とは異なり、通信において機器ごとの IP アドレスの指定が不要であり、多数の機器を扱う IoT 環境やアドレス変更が頻発するモビリティ環境に適している。また、近年では ICN のネーミングスキーマを活用したより柔軟な通信が注目されている。具体的には、コンテンツの名前に優先度の情報を付加してパケット転送方法を柔軟に変更したり、情報の取得以外にもアクチュエーターを作動させたりするといった活用事例が挙げられる。

我々の研究グループでは、ICN における柔軟な制御に関する研究として、センサネットワーク環境や災害時における分断ネットワーク間での通信を実現するためのドローンを用いた移動ルータの研究を行ってきた。これまでの研究では、移動ルータの制御はコンテンツリクエストに応えるための受動的な移動制御のみを対象としていた。しかし、コンテンツの優先度に基づくコンテンツ取得などの戦略を実現するためには、能動的な移動制御が必要不可欠である。移動ルータの移動制御において優先度などの属性を考慮することができれば、従来の手法と比べより柔軟で高速なコンテンツ取得が可能となる。

そこで、本報告では、移動ルータに対し能動的な制御が可能なシステムを実現を目的として、ICN のネーミングスキーマと ICN のストラテジ層を活用した移動制御システムを組み込んだ移動ルータ FR (Flying Router) を提案する。ICN の実装としては NDN (Named Data Networking) を対象とし、NDN のパケットフォワーダである NFD (NDN Forwarding Daemon) の拡張によって移動制御システムを実現するために必要な API、制御用コンポーネント、パケット構造を設計した。具体的には、API に関しては、移動経路を設定するパスプランの生成、一覧表示、実行、一時停止、キャンセル、移動速度の設定のための API を設計した。制御用コンポーネントに関しては、FR をユーザインターフェイス、NFD、制御用ストラテジおよびドローン制御プログラムの 4 つの制御用コンポーネントに分割して設計

を行った。パケット構造に関しては、Interest パケットを API として用いるために、名前に制御関数の名前を含めた制御用 Interest の名前空間を設計した。以上の ICN におけるネーミングスキーマを活用した FR を用いた通信の実証のために実機検証を行い、ICN ネーミングスキーマを FR の移動制御 API として使用可能であり、FR の能動的な制御が可能であることを示した。

主な用語

ICN、ネーミングスキーマ、API、移動ルータ、分断ネットワーク、パスプランニング

目 次

図 目 次

表 目 次

1　はじめに

現在では様々なものがインターネットに接続し、ユーザはインターネットを通じてそれらのIoT機器を制御できるようになった。その環境の中でIoT機器の増加によりネットワークトポロジが複雑になり、IPアドレスの割り当てと解決のコストが増加している[1]。また、現在ではインターネットにおけるコンテンツ取得の利用形態に関しては、ノードのIPアドレスを意識せずにコンテンツ取得を行っている。近年ではそのようなネットワークの利用形態とアーキテクチャの乖離を解決とするアーキテクチャとして、情報指向ネットワーク (ICN: Information-Centric Networking) が注目されている[9]。ホスト指向のネットワークプロトコルであるTCP/IPと異なり、ICNはコンテンツを主体としたネットワークである。コンテンツ取得においてTCP/IPネットワークと比べたICNの利点は、ネットワークが予めコンテンツの名前とFaceのペア情報を共有することで、機器のIPアドレスを指定せずに機器との通信が可能であり、特にネットワークトポロジの変化が頻発するモバイル環境ではネットワークアドレスに依存しないICNが有効である。また、ICNではコンテンツのキャッシング機能により、コンテンツをユーザ近くの通信機器にキャッシュすることでより高速なコンテンツ取得及びサーバの負荷分散が可能である。

また、近年ではICNのネーミングスキーマを活用した柔軟な通信が注目されている。その一つの例として、コンテンツに属性を付加することで柔軟なコンテンツ検索と取得が実現可能であることが挙げられる。また、コンテンツのみならずICNパケットが持つ名前に緊急度や転送先の名前を付加することで、優先度や転送先の変更などの柔軟な転送制御が可能である。さらに、パケットを受信した際にパケットに記載されたコンテンツの名前に基づく処理をエンドノードに定義することで、例えば電灯の点灯消灯やドアの開閉などの多様なアプリケーション層の処理をネットワーク層で実現する事例もICNのネーミングスキーマの活用例として考えられる[2]。

一方、これまではICNネーミングスキーマ及び我々の研究チームではドローンを用いた分断ネットワーク間通信の研究が行われていた。ネーミングスキーマに関する研究に関しては、文献[3]では音声コンテンツの取得において、Interestの名前に音声ファイルの再生時間位置や音声のビットレートなどのパラメータを含めることで実現する音声コンテンツの柔軟な取得方法を提案している。文献[4]ではInterestの名前に暗号化方式の名前などを含めることコンテンツの安全な取得方法を提案している。また、ドローンを用いて分断ネットワーク間通信を実現する研究については、文献[5, 6]ではセンサネットワーク通信においてより高速で省電力のコンテンツ取得のためのデータミュールのパスプランニングを考案している。文献[7]では分断ネットワークにおいてICN移動ルータを用いたコンテンツ取得システムの設計と開発を行っている。文献[8]では分断ネットワーク通信においてICN移動ルー

タのパスプランニングおよびそれを実現するシステムの API の設計を行っている。しかし、文献 [5, 6] ではデータミュールはパスプランニングをロケーションベースで行っており、運搬するコンテンツの特性を考慮した移動制御を行っておらず、また文献 [7] ではコンテンツベースで移動制御を行う移動ルータを提案しているが、コンテンツのリクエストをトリガとする受動的な移動制御しか検討しておらず、ノード数の増加により移動経路に重複が生じてコンテンツ取得が非効率である欠点が存在しており、また文献 [**?**] では移動ルータの能動的な移動制御を実現するシステムを設計したが、その実装と実機検証がまだ行われていない。そこでコンテンツの特性を考慮したパスプランニングに従い、能動的な移動ルータ制御の実現が課題になっている。

本研究では ICN ネーミングスキーマを活用した移動ルータの能動的制御の実現を目的とする。移動ルータを用いたネットワーク通信においてそのような制御のメリットに関しては、コンテンツのより効率的な取得が可能なパスプランニングを適用できるようになり、さらに高速なコンテンツ取得、リクエスト応答の時間短縮及び輻輳やノード故障などの環境の変化への素早い対応が可能である点が挙げられる。また、本研究開発の FR 移動制御システムにより実現する FR の移動制御や分断ネットワーク通信はネーミングスキーマを機器制御の API として活用する一事例として考えられる。

そして、本研究の研究内容に関しては、FR 移動制御システムを考案して API 及びその動作、制御用コンポーネント、パケット構造について設計した。API の設計ではパスプランの生成、一覧表示、実行、一時停止、キャンセル、移動速度の設定の動作に関して設計した。制御用コンポーネントの設計ではユーザインターフェイス、NFD、制御用 Strategy 及びドローン制御プログラムについて設計した。パケット構造に関しては、Interest パケットの名前に制御関数の名前を含めることで、API として用いる制御用 Interest の名前空間を定義した。以上の ICN におけるネーミングスキーマを活用した FR を用いた通信の実証のために実機検証を行い、ICN ネーミングスキーマを FR の移動制御 API として使用可能であり、FR の能動的な制御が可能であることを示した。

2　関連研究

2.1　情報指向ネットワーク

情報指向ネットワーク（ICN: Information-Centric Networking）はホストではなくコンテンツを主体としたネットワークである。本節では最初に ICN の設計思想について述べる。次に ICN におけるパケット、テーブルなどのデータ構造を紹介する。そして、それらを用いた ICN の通信方式を説明する。また、ICN では基本的な通信方式以上の機能を定義でき

るストラテジ層と呼ばれるレイヤーが存在し、それを利用して本研究の移動制御を実現している。最後に、ストラテジ層をサポートしたICN実装の一つであるNDNを紹介する。

ICNはコンテンツに名前を付与し、その名前とネクストホップの組み合わせをノードが保有することでコンテンツの取得経路を識別するネットワークである[9]。ICNの利点としては、TCP/IPネットワークと異なりユーザはIPアドレスを指定する必要がなく、ネットワークが自動的にコンテンツを保有するノードを識別してコンテンツ取得が可能となることが挙げられる。

データ構造については、ICNはInterestとDataの二種類のパケットを所有している。

- Interest

  取得するコンテンツを指定するパケットである。コンテンツ取得の際にコンテンツの名前をInterestの名前に含めてネットワークに転送することでコンテンツを取得する。
  Interestの名前の例：/youtube/forest.mp4

- Data

  コンテンツを格納するパケットである。コンテンツの名前とその名前に対応するコンテンツにより構成される。
  Dataの名前の例：/youtube/forest.mp4

そして、パケットを転送するために、各ノードでは各自のパケット転送用のテーブルであるFIB (Forwarding Information Base)、CS (Content Store)、PIT (Pending Interest Table)を用いる。

- FIB

  Interestパケット転送のためのテーブルである。FIBの各エントリでは一つのInterestと対応するFaceのペアが登録されている。また、Faceは通信路であるリンクオブジェクトとノードのインターフェイスであり、FaceにはFaceの名前が割り当てられている。

- CS

  ノードにキャッシングされたコンテンツを格納するテーブルである。CSを用いることで、コンテンツはエンドノードのみならず中継ノードにも保存でき、Interestに対してより早くコンテンツを返送することが可能となる。

- PIT

  Dataパケット転送のためのテーブルである。PITの各エントリでは一つのDataの名前と対応するFaceのペアが登録されている。Dataの名前とペアになっているFaceにDataパケットを転送することで、最終的にリクエスト元にコンテンツを転送できる。

以上のデータ構造を用いて、ICN では以下の流れでコンテンツの取得を実現している。

1. コンテンツリクエスト元のノードが Interest を生成する。
2. Interest を受信したリクエスト元のノード或いは中継ノードは Interest の PIT エントリを生成する。
3. Interest を受信したリクエスト元のノード或いは中継ノードは FIB を参照して次の転送先に Interest を転送する。
4. Interest がコンテンツの保存してあるノードに到着したら、コンテンツを格納した Data が PIT を参照して転送される。また、参照された PIT エントリは満たされて削除される。
5. コンテンツを格納した Data がリクエスト元のノードに到着する。

また、ICN では名前毎にパケットの受送信の際の制御内容を定義可能なストラテジ層を所有している。ストラテジ層は主にパケットの送受信の際に呼ばれるトリガとその際の制御内容を記述したコールバック関数から構成されており、このコールバック関数を編集することでパケット受送信の際の動作を任意に定義することが可能である。ストラテジ層を用いて実現可能な機能例に関しては、ICN では特定のコンテンツの転送先の変更、マルチキャストやパケットフィルター、ルータでのパケットのキャッシング、ノード内のデータ処理やアプリケーションの呼び出しなどの例が挙げられる。

本研究では ICN の研究プロジェクトの中で NDN Project[10] が開発した ndn-cxx (NDN C++ library with eXperimental eXtensions)[11] 及び NFD (Named Data Networking Forwarding Daemon)[12] を用いて ICN の通信を実現する。ndn-cxx とは ICN のパケットやテーブルなどのデータ構造及び転送するためのアルゴリズムを記述したライブラリである。NFD とは ndn-cxx ライブラリ及びストラテジ層などによりパケットを転送するプログラムである。

2.2 移動ルータを用いた情報指向ネットワーク通信

情報指向ネットワークはモビリティ環境における通信をサポート可能であり [13, 14]、近年の研究では移動可能ノードを用いた通信を ICN で展開する研究が行われている [15, 16]。本節では最初に情報指向ネットワーク通信で移動可能ノード，特に移動可能ルータを用いることの意義とメリットを述べる。次に、移動ルータを用いた情報指向ネットワーク通信モデルとしては通信の流れを説明する。そして、ICN データ構造の視点で通信の流れについて説明する。最後に、本節で説明した通信モデルにおける課題を紹介する。

ICN において移動ルータを取り入れることの意義としては、IP アドレスのようなネットワーク・アドレスが頻繁に変化する移動ルータの通信はロケーションベースの環境より、コンテンツベースの環境の方が親和性が高いことが考えられる。ICN ではネットワーク・アドレスを要しないため、移動ルータのネットワーク・アドレスを考慮せずに通信を実現できる。また、近年では災害時ネットワーク通信や分断されたセンサネットワーク間通信において、ICN を用いた移動ルータの実用が期待されている [7]。それらの場面のネットワークはネットワーク・アドレスの解決が困難であるが、ICN 移動ルータを用いることで IP アドレスを考慮する必要がなく、容易に通信を実現できる。

ICN において移動ルータを使用することのメリットについては、中継ルータが故障した際に移動ルータが代わりに通信を中継でき、耐故障性の高いネットワークを構築できる点が挙げられる。特に災害時のネットワーク通信において、故障した基地局の代わりにパケット運搬が可能であることにより移動ルータが注目されている。また、移動ルータを用いることでネットワークトポロジを変更してサーバや中継ノードでの輻輳解消が可能である点が挙げられる。さらに、ICN のキャッシングにより移動ルータの移動距離が短縮され、より長い時間の稼動が可能である点も考えられる。

移動ルータを用いた情報指向ネットワーク通信モデルにとしては、移動ルータが相互接続のないノードの間で往復してパケットを運搬することで、相互接続のないノード間通信を実現するモデルである。その通信の流れに関しては、最初に移動ルータは移動可能範囲内のノードの場所を把握し、同時に各ノードの持つコンテンツの名前を記録して移動ルータの FIB を構築する。ノード間の通信が発生しない時に全てのノードを巡回してリクエストを収集するが、ノード間の通信が発生した時にコンテンツ取得のために、コンテンツの持つノードへすぐに移動してコンテンツを取得し、或いは巡回を続けてコンテンツを取得するなど移動戦略によって移動経路が変化する。そして、コンテンツの持つノードへ移動してコンテンツを取得してリクエスト元に転送することでコンテンツの取得を実現する。

コンテンツ取得の具体的な動作については、まずユーザのリクエストしたコンテンツの名前を Interest の名前に含め、ユーザの近くに到着した移動ルータにその Interest を転送する。Interest を受信した移動ルータは Interest に対応する PIT エントリを生成し、同時に自身の FIB を参照してリクエストされたコンテンツを持つノードの有無を判断する。コンテンツを持つノードがあればそのノードへ移動するが、コンテンツを持つノードがなければ Interest がタイムアウト後に PIT エントリと共に消滅して通信が終了する。コンテンツの持つノードの近くへ到着した移動ルータは Interest をそのノードに渡し、ノードはコンテンツを ICN の Data パケットに格納して Data を移動ルータに転送する。Data を受信した移動ルータは PIT を参照し、リクエスト元のノードのエントリを見つけてそのノードの位置を取得して移動する。リクエスト元のノードの近くに到着した移動ルータは Data をそのノー

ドへ転送し、リクエスト元のノードは Data からコンテンツを抽出することでコンテンツを入手する。以上の流れにより一回の通信が完了した。

しかし、ICN 移動ルータの実用のために、本節で説明した通信モデルではまだ様々な課題が存在している。例えば、リアルタイム性のある通信などの様々な状況に対応する移動戦略や長時間稼動のための省エネルギの計算処理及び移動戦略の適用のための制御システムなどがまだ検討されていない。また、コンテンツキャッシングの活用などの移動ルータ間自律協調機構の実装も課題の一つとして挙げられる。

2.3 移動ルータを用いた分断ネットワーク通信

近年では移動ルータを用いる分断ネットワーク通信の研究が行われており、ICN では様々な柔軟な制御が可能であり、その一つの実用例として分断ネットワーク通信における移動ルータの制御の例が挙げられる。本節ではその例に関する従来の研究の内容及び課題について説明する。そして、それらの関連研究の課題に基づいて本研究の目的を示す。

文献 [7] では分断ネットワーク間のコンテンツ取得のために FR を用いることを提案している。ここで FR とはドローンに NFD 等の ICN フォワーダをインストールした小型コンピュータ RPi (Raspberry Pi) を携帯し、RPi が他の ICN ノードと ICN 通信を行い、また RPi 内のドローン制御プログラムを用いてドローンの移動を制御する ICN 移動ルータを指す。また、その文献の研究では ICN 移動ルータを用いた分断ネットワークにおけるコンテンツ取得システムを開発した。

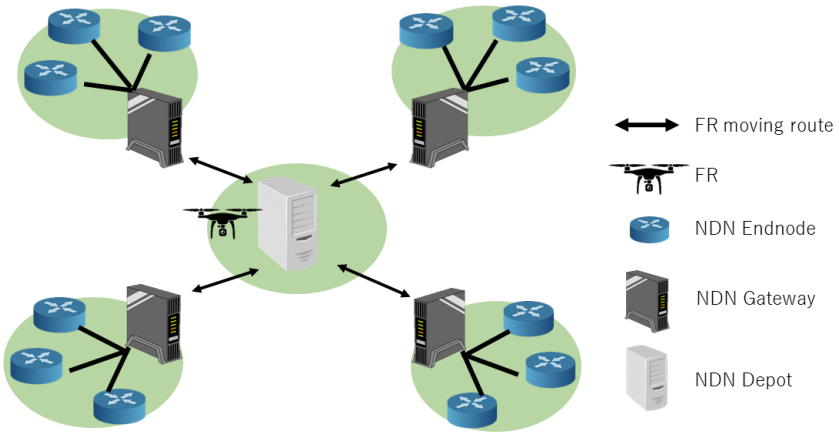

図 1: 移動ルータを含めた分断ネットワーク

上図の分断ネットワークでは一つの指令中心ノードである Depot と複数の分断ネットワークから構成され、FR の初期位置は Depot の通信範囲内にあり、分断ネットワークは複数のエンドノードと一つのゲートウェイから構成される。この研究ではノードの座標とコンテンツの名前のペアを記録するテーブルを考案した。そのテーブルによりコンテンツの名前の指定で FR をコンテンツの保存してあるノードへの移動を指令できる。本研究では分断ネットワーク通信における移動制御 API の実装においてはそのテーブルを FR 移動制御システムのコンポーネント NFD の内部変数として実装する。

しかし、分断ネットワークのノード数の増加により、FR の移動経路にオーバーヘッドが大きくなり、コンテンツ取得時間が非効率である欠点が存在している。それを解決するためにパスプランが必要であるが、この研究ではパスプランを生成する機構とパスプランを適用する機構が検討されておれず、それらの実装が今後の課題になっている。

一方、FR を用いた分断ネットワーク通信のための移動戦略を考案する研究が行われている [8]。また、その研究ではその移動戦略における制御のために ICN ネーミングスキーマを移動制御 API としての使用を提案した。下表は FR を用いた分断ネットワーク通信の移動戦略における移動制御 API をまとめた。

表 1: FR を用いた分断ネットワーク通信の移動戦略における移動制御 API

API 名	/FRControl/{FRName}/DiscoverNetwork/{Length}/{n_FR}
動作概要	複数台の FR がホームを中心とする六角形の地域で分断ネットワークを発見し、その分断ネットワークに名前をつけ、座標を取得する。分断ネットワークの座標を Data パケットに格納してユーザノードに転送する
入力	unsigned long 型の六角形地域の辺の長さ、unsigned int 型のドローンの数
出力	分断ネットワークの座標
API 名	/FRControl/{FRName}/Crawl/{Point}* Point = /{North}/{East}/{Altitude} ｜ {PointName} PointName = {FNName} ｜HOME
動作概要	指定した分断ネットワークへの移動
入力	ndn::Name 型分断ネットワークの名前、或いは long 型のホームから分断ネットワークまでの東西方向距離、南北方向距離
出力	なし

この研究では API の Crawl でパスプランを生成できるが、API に基づいてドローンの制御命令を生成して転送する機構、すなわちパスプランの適用機構の実装が行われていない。上表の API が生成したパスプラン及びパスプランの適用機構を組み合わせれば、能動的にパスプランを考えて移動する移動ルータを実現できる。以上の課題を解決して移動ルータの実用化を進めるために、本研究ではパスプランの適用可能な移動システムを開発し、移動ルータの能動的制御の実現を目的とする。

3　FR 移動制御システムの設計

3.1　設計目標及びその達成のための設計項目

3 章では FR 移動システムの設計について説明するが、本節ではシステムの利用事例などの使用環境と提供する API などのシステム仕様について説明する。最初にシステムの利用事例を述べ、それぞれの事例における必要な動作を列挙する。そして、それらの動作を実現するシステムの API、およびその API を実現するためのドローンとパケットフォワーダの API を列挙する。また、システムのコンポーネントなどの設計は 3.4 節で説明する。

FR 移動制御システムの利用事例及びその利用事例における必要な動作、必要な動作の実行方法は以下の箇条書きで示す。

- 利用事例 1：ICN ネーミングスキーマを用いた FR 移動制御
  - 任意地点への移動
    パスプランの生成、編集、一覧表示、実行、一時停止、キャンセル:FR 移動制御システムの API の/PathCreate、/PathList、/PathRun、/PathPause、/PathCancel
  - 移動方式の設定
    スピードの設定：FR 移動制御システムの API の/SetVehicleSpeed
  - 移動制御用情報の定義、生成、転送、保存
    名前空間、データの保存場所などの移動制御用情報の定義、移動制御用情報の生成、転送、保存：FR 移動制御システムの内部処理
- 利用事例 2：FR を用いた分断センサネットワーク通信
  - 分断ネットワークの発見
    パスプランの地点の算出、パスプランの生成、実行、発見時の移動の停止、移動の再開、発見した分断ネットワークの場所の記録：FR 移動制御システムの API の/DiscoverNetwork

– 分断ネットワークへの移動

分断ネットワークの場所の取得、パスプランの生成、実行、通信時の移動の停止、移動の再開：制御用 Strategy、ドローン制御プログラム：FR 移動制御システムの API の/Crawl

– 移動制御用情報の定義、生成、転送、保存

名前空間、データの保存場所などの移動制御用情報の定義、移動制御用情報の生成、転送、保存：FR 移動制御システムの内部処理

利用事例 1 の API の目的は様々な利用事例に利用事例 1 の API の動作改造により他の利用事例の API の動作を実現可能な基本 API を提供することである。利用事例 1 の動作の実現方法に関しては、FR のドローンの API と NDN の API を組み合わせて FR 制御システムの API の動作を実現する。その中で使用するドローン制御プログラムの API は指定地点への移動と移動速度の設定に関する API が挙げられる。また、その中で使用する NDN の API は主に NDN の API では制御用 Interest と実行結果 Data 受信後の動作設定及び変数の取得と設定に関する API を使用している。

下表は使用するドローン制御プログラムの API の動作概要、入出力をまとめた。

表 2: 使用するドローン制御プログラムの API

API 名	Takeoff(Altitude)
動作概要	陸地にある FR を離陸させ、指定高度まで上昇させる
入力	long 型の高さ
出力	なし

API 名	Land()
動作概要	飛行中の FR を着陸させる
入力	なし
出力	なし

API 名	Simple_goto(Position)
動作概要	指定座標へ移動させる
入力	LocationGlobal の地点座標
出力	なし

API 名	Loiter()
動作概要	FR の移動を停止させる
入力	なし

出力	なし

API 名	Setgroundspeed(Speed)
動作概要	ドローンの移動速度を設定する
入力	unsigned long 型のドローンの移動速度
出力	なし

下表は利用事例 1 に提供する FR 移動制御システムの API の名前、動作概要、入出力をまとめた。

表 3: 利用事例 1 に提供する FR 移動制御システムの API

API 名	/FRControl/{FRName}/PathCreate/{PathName}/{Point}* Point = /{North}/{East}/{Altitude} ｜ {PointName} PointName = {FNName} ｜ HOME
動作概要	パスプランの生成、NFD への保存、編集
入力	ndn::Name 型パスプランの名前、long 型のパスプランの地点の FR からの東西方向距離及び南北方向距離及び高さ、或いは ndn::Name 型の地点の名前
出力	なし

API 名	/FRControl/PathList
動作概要	パスプランの一覧表示
入力	なし
出力	保存したパスプランの各地点の FR からの東西方向距離及び南北方向距離、或いは地点の名前

API 名	/FRControl/{FRName}/PathRun/{PathName}
動作概要	パスプランに沿う移動
入力	ndn::Name 型のパスプランの名前
出力	なし

API 名	/FRControl/{FRName}/PathPause
動作概要	現在実行中のパスプランを一時停止させ、移動を停止させる。一時停止の状態で API を入力するとパスプランを再開させ、移動を再開させる
入力	なし

出力	なし

API 名	/FRControl/{FRName}/PathCancel
動作概要	現在実行中のパスプランをキャンセルし、移動を停止させる
入力	なし
出力	なし

API 名	/FRControl/{FRName}/SetVehicleSpeed/{Speed}
動作概要	機体移動速度を設定する
入力	unsigned long 型の機体移動速度
出力	なし

利用事例 2 の動作の実現方法に関しては、利用事例 1 のシステムの API を組み合わせることで FR のセンサ分断ネットワーク通信での移動を実現する。

下表は利用事例 2 に提供する FR 移動制御システムの API についてまとめた。

表 4: 利用事例 2 に提供する FR 移動制御システムの API

API 名	/FRControl/{FRName}/DiscoverNetwork/{Length}/{n_FR}
動作概要	複数台の FR がホームを中心とする六角形の地域で分断ネットワークを発見し、その分断ネットワークに名前をつけ、座標を取得する。分断ネットワークの座標を Data パケットに格納してユーザノードに転送する
入力	unsigned long 型の六角形地域の辺の長さ、unsigned int 型のドローンの数
出力	分断ネットワークの座標

API 名	/FRControl/{FRName}/Crawl/{Point}* Point = /{North}/{East}/{Altitude} ｜ {PointName} PointName = {FNName} ｜HOME
動作概要	指定した分断ネットワークへの移動
入力	ndn::Name 型分断ネットワークの名前、或いは long 型のホームから分断ネットワークまでの東西方向距離、南北方向距離
出力	なし

本研究では利用事例 1 の API の動作及びそのためのシステムコンポーネントの設計と実

装を行うが、利用事例 2 の API に関する設計と実装は今後の課題とする。

3.2 設計概要

本節では FR 移動制御システムの環境や入出力などのユーザ視点のシステム仕様とシステム構造について説明する。最初に、システムの環境についてネットワークトポロジとノードの通信方法を紹介する。次に、そのシステムの環境におけるシステムの起動方法、ユーザの入力方法、システムの出力および処理の流れについて説明する。そして、システムの処理を実現するコンポーネントおよびその役割を述べる。

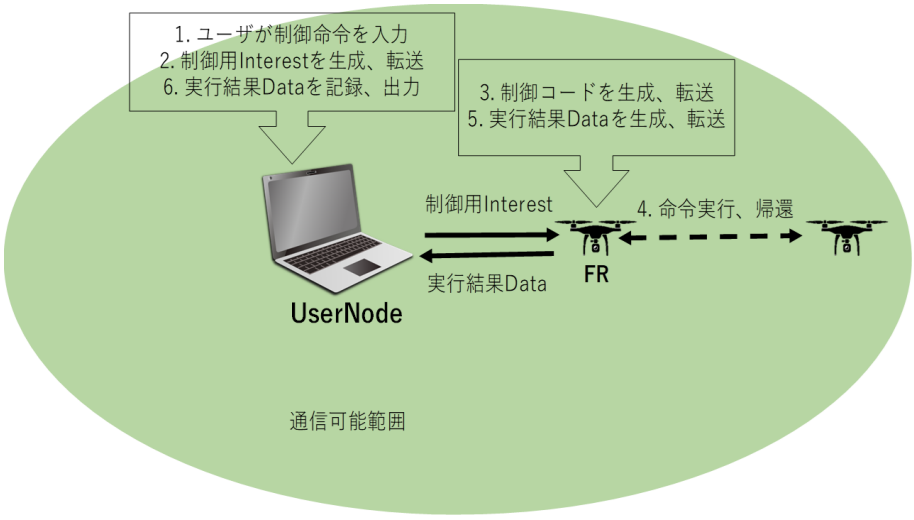

図 2: ICN ネーミングスキーマを活用した FR 移動制御システムのネットワークトポロジ

本システムのネットワークトポロジに関しては、ユーザノード一台と FR 一台がドローンのコントローラの無線 LAN に接続されている。その理由に関しては、FR が異常時にコントローラから動作を停止させることが可能からである。FR では小型コンピュータ RPi(Raspberry Pi) を携帯し、RPi からの命令に従って移動する。コントローラのメーカ仕様ではコントローラは DHCP サーバを有しており、ネットワークのゲートウェイ IP アドレスは 10.1.1.1 であり、ドローンは 10.1.1.10 の固定アドレスを使用しており、10.1.1.100〜250 までは外部機器接続時に使用可能なアドレス範囲である。ここではユーザノードは 10.1.1.101 の固定アドレスを使用し、FR の RPi は 10.1.1.102 の固定アドレスを使用する。これより動的に振り分けられるアドレスの検索の手間を省き、ネットワーク設定を簡単化できる。

また、本システムの入力はユーザからの制御用情報であり、出力は制御の実行結果である。入力に対する処理は FR の機械制御或いはユーザノードと FR 内のデータの操作である。制

御の流れについては、最初にユーザは制御したい対象や制御内容などの制御用情報をユーザインターフェイスに入力する。そして、入力を受け取ったユーザインターフェイスは制御用 Interest を生成し、それらの情報を制御用 Interest の名前に含めて指定する FR に転送する。制御用 Interest が指定した FR に到着すると、FR はその制御用 Interest に基いてドローン制御プログラムの対応関数を呼び出す。ドローン制御プログラムの対応関数でドローンに制御命令を転送して FR の移動制御を行う。そして、移動制御終了後、ドローン制御プログラムの対応関数の実行も完了となり、実行結果を格納した Data パケットを生成してユーザノードへ転送する。Data を受け取ったユーザインターフェイスは標準出力に出力する。

ここで、制御用情報は制御者、制御対象、制御関数、制御関数のパラメータを含めている。分断ネットワークではノード数及び FR の数が多い場合、制御者及び制御対象を明示する必要がある。また、より高度で柔軟な機械制御ではパラメータを必要とすると考えられる。

また、本システムの起動のために一部の初期設定はユーザの手動により行うものであり、それらを以下の箇条書きでまとめた。

1. ユーザノード、ドローン、コントローラの電源を入れる。
2. ユーザノードがコントローラのネットワークに接続する。

   FR の RPi (Raspberry Pi) はコントローラのネットワークに自動的に接続している。
3. ユーザインターフェースを起動。
4. ユーザインターフェースに制御用 Interest の名前を入力する。

また、下図では FR 移動制御システムのコンポーネント及びその動作を示している。

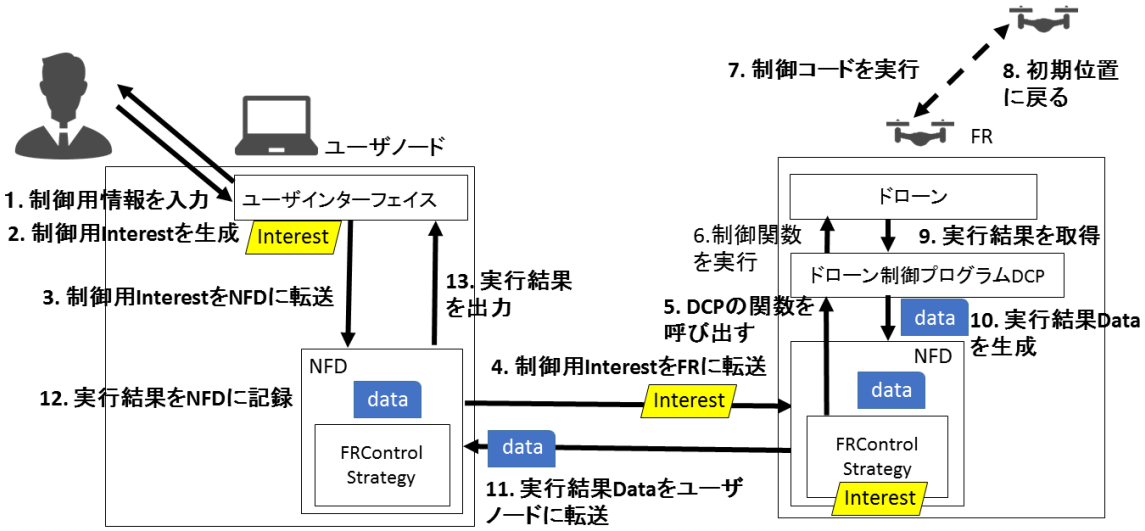

図 3: FR 移動制御システムのコンポーネントダイアグラム

FR 移動制御システムのコンポーネントはユーザインターフェイス、NFD、制御用 Strategy である FRControlStrategy、ドローン制御プログラムであり、本研究ではそれらのコンポーネントの設計と実装を行う。

ユーザインターフェイスはユーザからの制御用情報の入力を受け付け、ユーザへの実行結果の出力を行うプログラムである。ユーザインターフェイスはユーザノードにインストールしている。ユーザインターフェイスは図 9 の「1」、「2」、「3」、「13」を実現する。以下の箇条書きではその役割を示している

- ネットワーク接続などの初期設定の実行
- FR 移動制御のための入力の受付
- 入力に基づく制御用 Interest の生成及び NFD への転送

- 実行結果 Data の内容の出力

また、NFD は Interest と Data を転送するプログラムである。NFD はユーザノードと FR にインストールしている。NFD は図 9 の「4」、「11」、「12」、「13」を実現する。以下の箇条書きではその役割を示して。

- 制御用 Interest の転送及び実行結果 Data の転送
- ノードの名前などの FR 移動制御のための変数の保持

また、制御用 Strategy である FRControlStrategy は制御用 Interest とコンテンツを要求する Interest を判別し、制御用 Interest に基づいてドローン制御プログラムの関数を呼び出す NFD 機能である。FRControlStrategy はユーザノードと FR にインストールされている。FRControlStrategy は図 9 の「5」、「12」を実現する。以下の箇条書きではその役割を示している。

- 制御用 Interest かコンテンツの要求 Interest の判別
- 制御用 Interest の制御関数が定義されているか、パラメータの書式が正しいか、指定したノードに到着したかの確認
- 制御用 Interest に基づくドローン制御プログラムの関数の呼び出し
- 実行結果 Data の記録

また、ドローン制御プログラムはドローン制御動作を記述したプログラムである。ドローン制御プログラムは FR にインストールしている。NFD は図 9 の「6」～「10」を実現する。以下の箇条書きではその役割を示している。

- ドローンとの通信、ドローン制御
- 実行結果 Data の生成

3.3 FR 移動制御システムの API の設計

本節では 3.1 節で述べた利用事例の動作を実現するシステム API の設計について説明する。それぞれの API についての説明は最初に API の書式などの仕様を述べる。次に、API の制御の流れにおいて必要なクラス、メソッド、変数およびそれらの実現方法を説明する。最後に、必要なクラス、メソッド、変数を実装するコンポーネントなどの仕様を説明する。

PathCreate

下表は FR 移動制御システム API の PathCreate の動作概要、パラメータ、出力などについての説明である。

表 5: PathCreate の API 情報

書式	/FRControl/{FRName}/PathCreate/{PathName}/{Point}* Point = {North}/{East}/{Altitude} ｜ {PointName} PointName = HOME ｜ {FNName}
例	/FRControl/FR0/PathCreate/Path0/10/-10/5/-10/10/5
動作概要	パスプランをユーザノードの NFD と FR の NFD に保存する。パスプランの訪問順番はパラメータの先頭から先に訪問し、パラメータの末尾を最後に訪問する順番とする
呼び出しタイミング	FR の NFD で制御用 Strategy の適用後の任意時刻
パラメータ	FRName：制御する FR の名前 PathName : パスプランの名前であり、NDN の受け付ける文字を使用する North：パスプランの地点からホームまでの南北方向の距離である。単位はメートルである。範囲はこの地点からホームまでの直線距離が 800m 以内の実数の数値とする East：パスプランの地点からホームまでの東西方向の距離である。単位はメートルである。範囲はこの地点からホームまでの直線距離が 800m 以内の実数の数値とする Altitude：パスプランの地点の高さである。単位はメートルである。 FNName：ホーム以外の分断ネットワークの名前である
出力	当該パスプランの各地点の座標を標準出力に出力する
例外処理	• FR の NFD で制御用 Strategy の適用前に呼び出しを行った場合、制御用 Strategy の PathCreate() 関数が呼び出されない • パスプランの名前がない、或いは NDN では受け付けない文字を使用した、或いは制御用 Interest の文字数が NDN 規定の文字数範囲を超えた場合、制御用 Strategy の PathCreate() 関数が終了する

	• パスプランの地点がない、或いは最後の地点の東西方向の距離がない、或いは数字以外の文字を使用した、或いは制御用 Interest の文字数が NDN 規定の文字数範囲を超えた場合、制御用 Strategy の PathCreate() 関数が終了する • 地点からホームまでの直線距離が 800m 超えた地点については、地点からホームまでの直線方向で直線距離が 800m の場所へ FR が移動する

制御の流れについては、ユーザインターフェイスが標準入力からパスプランの名前とパスプランの各地点の座標或いは名前を取得し、名前を持つ制御用 Interest を生成してユーザノードの NFD に発送する。制御用 Interest を受け取ったユーザノード NFD はそれをユーザノードの制御用 Strategy と FR の NFD に渡す。制御用 Interest を受け取ったユーザノードの制御用 Strategy はユーザノードの NFD からパスプランのマップを取得して新たなパスプランを設定する。一方、制御用 Interest を受け取った FR の NFD は制御用 Interest を FR の制御用 Strategy に渡し、FR の制御用 Strategy の制御用 Interest に対する動作はユーザノードと同じである。

上記の制御の流れは下図で表している。図中 UI はユーザインターフェイス、U_NFD はユーザノードの NFD、U_S はユーザノードの制御用 Strategy、FR_NFD は FR の NFD、FR_S は FR の制御用 Strategy、DCP はドローン制御プログラム、Drone はドローンを表す。

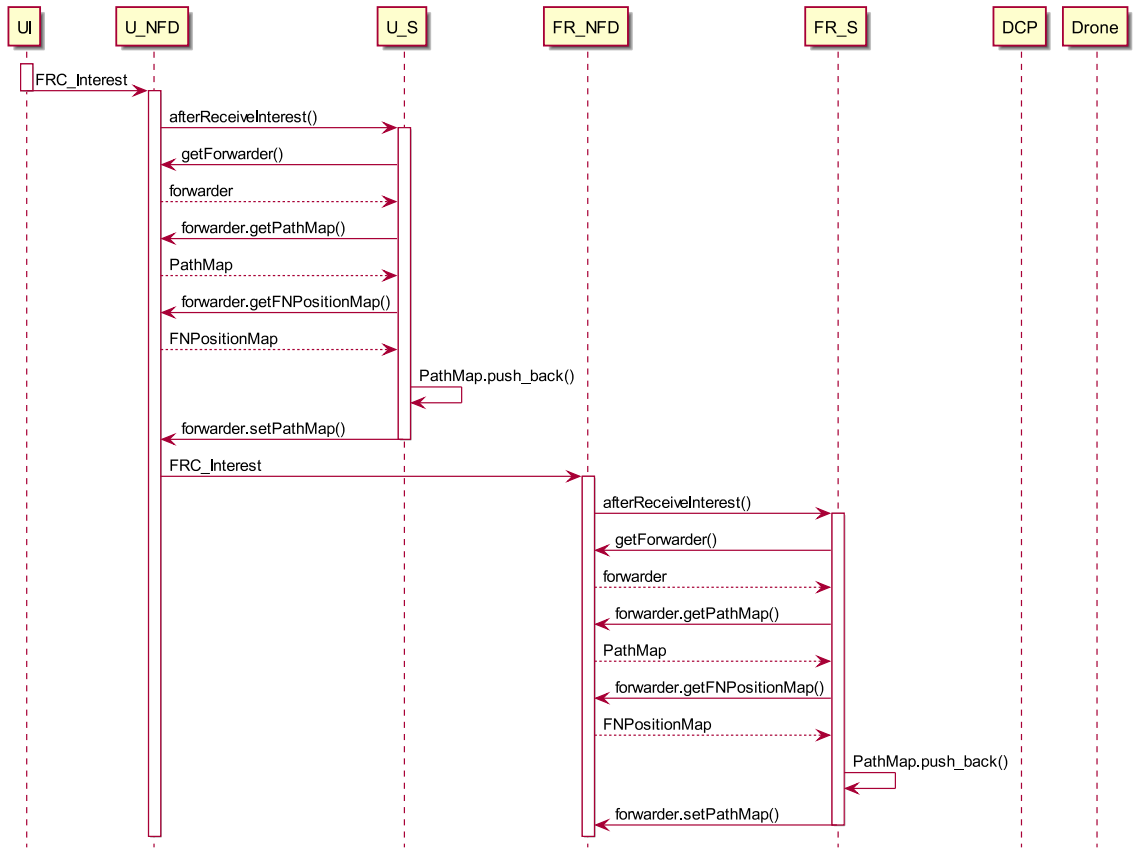

図 4: API の PathCreate のシーケンス図

以下の箇条書きでは上記の流れの各処理の実現方法について説明する。

- 標準入力から制御用情報の取得

  標準入力の文字列を ndn::Name 型のインスタンスに格納すると文字列が扱いやすくなる。標準入力の文字列を ndn::Name のコンストラクタの引数として渡すことで、制御用 Interest の名前の ndn::Name 型インスタンスを生成する。

- 制御用 Interest の生成

  ユーザインターフェイスの FRC_UI クラス void SendFRControlName(ndn::Name InterestName) 関数で実装する。ndn::Interest のコンストラクタに制御用 Interest の名前を引数として渡すことで、制御用 Interest の ndn::Interest 型インスタンスを生成する。

- 制御用 Interest のユーザノード NFD への転送

  ユーザインターフェイスの FRC_UI クラス void SendFRControlName(ndn::Name InterestName) 関数で実装する。FRC_UI クラスのプライベートメンバ m_face を定義し

ておき、m_face.expressInterest() 関数を用いて制御用 Interest を転送する。

- パスプランのマップの取得、新しいパスプラン追加、パスプランのマップの設定

  制御用 Strategy の PathCreate() 関数で実装する。

  PathCreate() のために追加すべきクラス、メソッド及び変数については、FRControlStrategy の親クラスである Strategy クラスで Forwarder 型インスタンスを取得する Forwarder& getForwarder() 関数を定義しておき、Forwarder クラスでは map<ndn::Name, vector<nfd::Point>>型のパスプランのマップと分断ネットワークのマップとそれらの取得と設定を行う関数を定義しておく。forwarder.hpp で地点を表す Point クラスを定義しておく。Point クラスは FR からの南北方向距離、東西方向距離、高さ及び地点の名前の属性を持つ。

  新しいパスプランの追加の方法については、FRControlStrategy::getForwarder() 関数で Forwarder インスタンスを取得し、そのインスタンスのパスプランのマップを取得する。パスプランをマップの「PathName」番目のインデックスのエントリに nfd::Point 型の vector 配列を代入することで新しいパスプランを追加する。最後に、Forwarder::SetPathMap() 関数で新しいパスプランを追加したマップを設定することでパスプランのマップを更新する。

- 制御用 Interest の FR の NFD への転送

  ユーザノードのコマンドラインで「nfdc register ndn;/FRControl udp:10.1.1.102」で転送設定を行うことで制御用 Interest の転送を実現する。

下表では PathCreate で追加するクラス、メソッド、変数について列挙する。

表 6: PathCreate で追加するクラス、メソッド、変数

クラス			
コンポーネント	ファイル	クラス名	説明
NFD	forwarder.hpp	Position	ドローン移動用の座標、ホームからの東西方向の距離、南北方向の距離、高さの属性を有する
UI (User Interface)	FRControl-UI.cpp	FRC_UI	ユーザインターフェイスのクラス

メソッド				
コンポーネント	ファイル	所属クラス	メソッド名	説明
制御用 Strategy	FRControl Strategy.cpp	FRControl Strategy	public void PathCreate(const ndn::Interest interest)	パスプランの生成関数
制御用 Strategy	FRControl Strategy.cpp	FRControl Strategy	public void afterReiceiveInterest()	制御用 Interest 受信時のリスナ関数
NFD	strategy.hpp	Strategy	protected Forwarder& getForwarder()	パスプランのマップと分断ネットワークの座標マップの取得のために Forwarder クラスのインスタンスを取得する関数
NFD	forwarder.hpp	Forwarder	protected ndn::Name& getNodeName()	ノードの名前の取得
NFD	forwarder.hpp	Forwarder	protected void setNodeName(ndn::Name NodeName)	ノードの名前の設定
NFD	forwarder.hpp	Forwarder	protected std::map <ndn::Name, vector<nfd::Point>>& getPathMap()	パスプランのマップの取得
NFD	forwarder.hpp	Forwarder	protected std::map<ndn::Name, nfd::Point>& getFNPositionMap()	分断ネットワークの座標リストの取得

NFD	forwarder.hpp	Forwarder	protected void setPathMap(std::map<ndn::Name, vector<nfd::Point>> PathMap)	パスプランのマップの設定
NFD	forwarder.hpp	Forwarder	protected void setFNPosition(std::vector<nfd::Point> FNPositionList)	分断ネットワークの座標リストの取得

変数				
コンポーネント	ファイル	所属クラス	クラス名	説明
UI	FRControl-UI.cpp	FRC_UI	private ndn::Face m_face	制御用 Interest 転送用
NFD	forwarder.hpp	Forwarder	private ndn::Name m_NodeName	ノードの名前
NFD	forwarder.hpp	Forwarder	private map<ndn::Name, vector<nfd::Point>> m_PathMap	パスプランの名前とパスプランの地点の座標のマップ
NFD	forwarder.hpp	Forwarder	private Forwarder map<ndn::Name, nfd::Point> m_FNPositionList	分断ネットワークの地点のリスト

NFD	forwarder.hpp	Point	public long North, East, Altitude; public ndn::Name PointName	地点の属性、それぞれはホームからの東西方向の距離、南北距離の距離、高さ、地点の名前

PathList

下表はFR移動制御システムAPIのPathListの動作概要、パラメータ、出力などのAPI情報についての説明である。

表 9: PathListのAPI情報

書式	/FRControl/PathList
例	/FRControl/PathList
動作概要	保存したパスプランを標準出力に出力する
呼び出しタイミング	FRのNFDで制御用Strategyの適用後の任意時刻
パラメータ	なし
出力	保存したパスプランの名前及びその各地点を標準出力に出力する
例外処理	• FRのNFDで制御用Strategyの適用前に呼び出しを行った場合、制御用StrategyのPathList()関数は呼び出されない • 保存したパスプランがない場合、何も出力せずに制御用StrategyのPathList()関数が終了する

ユーザインターフェイスが制御用Interestを生成してユーザノードNFDへ転送する。制御用Interestを受信したNFDはそれを制御用Strategyに渡す。制御用Strategyはユーザノード NFDのインスタンスを取得し、そのインスタンスのパスプランのマップを取得して標準出力に出力する。

上記の制御の流れはシーケンス図で表している。図中UIはユーザインターフェイス、U_NFDはユーザノードのNFD、U_Sはユーザノードの制御用Strategy、FR_NFDはFRのNFD、FR_SはFRの制御用Strategy、DCPはドローン制御プログラム、Droneはドロー

ンを表す。

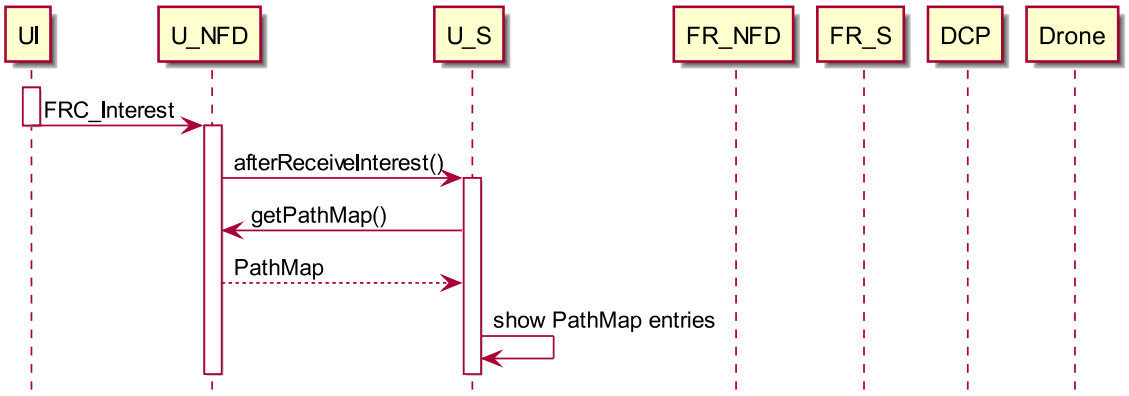

図 5: API の PathList のシーケンス図

以下の箇条書きでは上記の流れの各処理の実現方法について説明する。

- 制御用 Interest の生成、制御用 Interest のユーザノード NFD への転送、パスプランのマップの取得

  制御用 Strategy の PathList() 関数で実装し、実現方法は API の PathCreate と同じである。

下表では API の PathList で追加するメソッドを示す。

表 10: API の PathList の追加するメソッド

コンポーネント	ファイル	所属クラス	メソッド名	説明
制御用 Strategy	FRControlStrategy.cpp	FRControlStrategy	public void PathList()	パスプランの一覧を出力する関数

PathRun

下表は FR 移動制御システム API の PathRun の動作概要、パラメータ、出力などの API 情報についての説明である。

表 11: PathRun の API 情報

書式	/FRControl/{FRName}/PathRun/{PathName}
例	/FRControl/FR0/PathRun/Path0
動作概要	パスプランを実行し、終了後ホームに帰還する
呼び出しタイミング	PathCreate 実行後の任意時刻
パラメータ	PathName : パスプランの名前である。NDN の受け付ける文字を使用する
出力	1 秒ごとに現在地からパスプランの次の地点までの距離を出力する
例外処理	• FR の NFD で制御用 Strategy の適用前に呼び出しを行った場合、制御用 Strategy の PathRun() 関数が呼び出されない • パラメータのパスプランの名前が指定されていない、或いは NDN では受け付けない文字を使用した、或いは制御用 Interest の文字数が NDN 規定の文字数範囲を超えた場合、制御用 Strategy の PathRun() 関数が終了する • パスプランの名前がマップに保存されていない場合、制御用 Strategy の PathRun() 関数が終了する • パスプラン実行中に他のパスプランを実行すると、現在のパスプランの各地点を全て訪問してから FIFO(First In, First Out) の順番で後に他のパスプランを実行する。

制御の流れについては、ユーザインターフェイスが標準入力からの制御用情報に基づいて制御用 Interest を生成し、それをユーザノード NFD に転送する。制御用 Interest を受信したユーザノード NFD はそれを FR の NFD に渡す。制御用 Interest を受信 FR の NFD はそれを制御用 Strategy に渡し、制御用 Strategy でパスプランのマップを取得してパスプランの地点の座標をファイルに書き込む。ファイル書き込み終了後、制御用 Strategy は FR にあるドローン制御プログラムを実行させ、ドローン制御プログラムが地点座標のファイルを読み込み、地点へ移動する関数を実行する。

上記の制御の流れはシーケンス図で表している。図中 UI はユーザインターフェイス、U_NFD はユーザノードの NFD、U_S はユーザノードの制御用 Strategy、FR_NFD は FR の NFD、FR_S は FR の制御用 Strategy、DCP はドローン制御プログラム、Drone はドローンを表す。

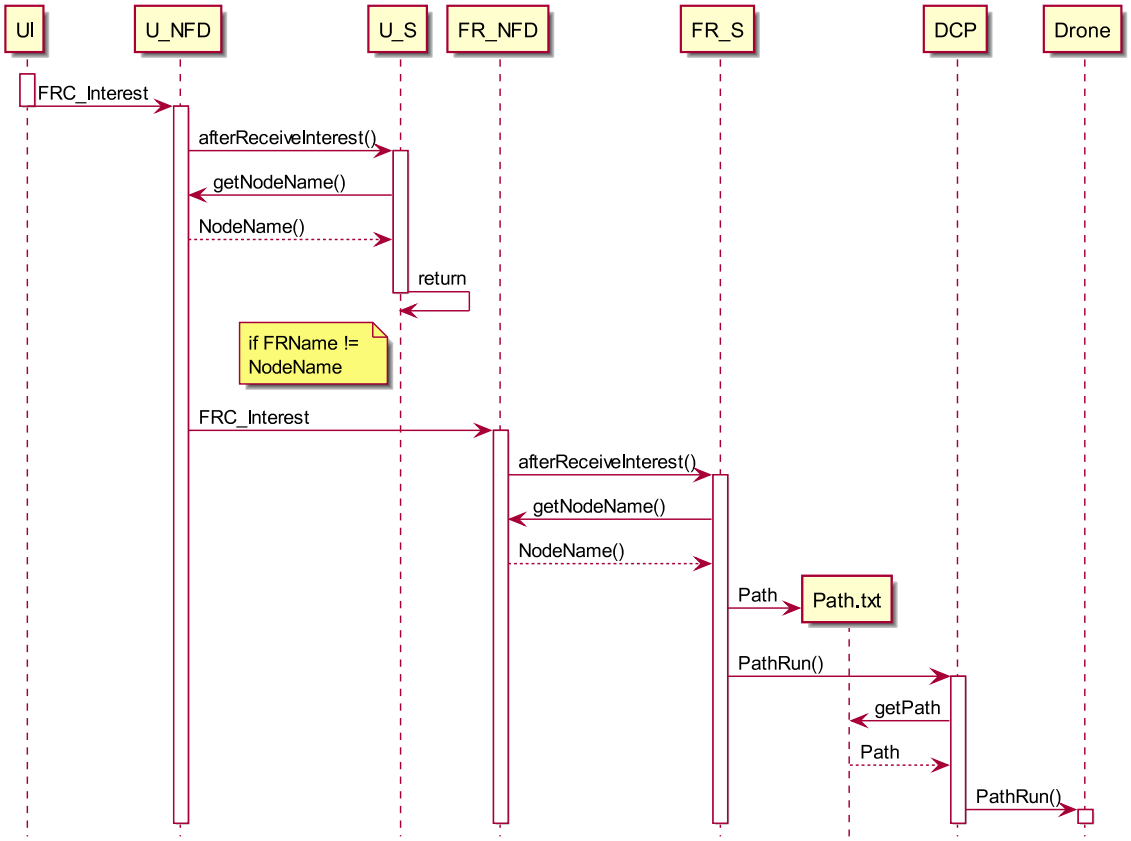

図 6: API の PathRun のシーケンス図

以下の箇条書きでは上記の流れの各処理の実現方法について説明する。

- 制御用 Interest の生成、制御用 Interest のユーザノード NFD への転送、パスプランのマップの取得

  PathCreate と同じであり、ここでは省略する。

- 地点座標の記載してあるファイルの生成

  ファイルでは一つの地点に対して一行を使い、座標を/North/East/Altitude 或いは/FNName の形で記述する。

- ドローン制御プログラムの実行、移動を実現する関数の呼び出し

  制御用 Strategy の public void PathRun() 関数で実装する。外部コマンドの実行関数 std:: system(std::string) でドローン制御プログラムを実行する。ドローン制御プログラムにコマンドライン引数を渡すことでドローン制御プログラム内のドローンの移動を実現する関数を呼び出す。

- ドローン制御プログラムにおけるパスプラン実行の実装方法

  ドローン制御プログラムの PathRun() 関数で実装する。dronekit の vehicle.commands 配列にパスプランの各地点への移動を意味するエントリを追加し、vehicle.mode に VehicleMode ('AUTO') を代入することで、配列にあるエントリを順番に実行して移動を実現できる。

- 現在地からパスプランの次の地点までの距離の出力

  dronekit-python が提供した example の mission_basic.py の中の distance_to_current_waypoint() を利用することで現在地からパスプランの次の地点までの距離の出力を実現できる。

下表では API の PathRun で追加するファイル及びメソッドを示す。

表 12: API の PathRun の追加するファイルとメソッド

コンポーネント	ファイル	説明
DCP (Drone Control Program)、制御用 Strategy	Path.txt	DCP で移動命令の生成用

コンポーネント	ファイル	所属クラス	メソッド名	説明
制御用 Strategy	FRControl Strategy.cpp	FRControl Strategy	public void PathRun()	ドローン制御プログラムの PathRun() 関数の呼び出し
DCP	DCP.py	なし	PathRun()	パスプランの地点へ順番に移動する関数

PathPause

下表は FR 移動制御システム API の PathPause の動作概要、パラメータ、出力などの API 情報についての説明である。

表 13: PathPause の API 情報

書式	/FRControl/{FRName}/PathPause
例	/FRControl/FR0/PathPause
動作概要	現在実行中のパスプランが一時停止/再開
呼び出しタイミング	PathRun 或いは PathGoto の実行で移動中の時、或いはそれらの実行で一時停止中の時
パラメータ	なし
出力	なし
例外処理	• PathRun 及び PathGoto 両方の実行時ではない時に呼び出しを行うと、ドローン制御プログラムの PathPause() 関数が終了する

制御の流れについては、ユーザインターフェイスで生成した制御用 Interest が FR の NFD に到着し、制御用 Strategy でノードの名前を確認する。制御用 Interest の指定 FR 名と FR のノード名が一致したら、ドローン制御プログラムのパスプランの一時停止/再開関数を呼び出す。

上記の制御の流れはシーケンス図で表している。図中 UI はユーザインターフェイス、U_NFD はユーザノードの NFD、U_S はユーザノードの制御用 Strategy、FR_NFD は FR の NFD、FR_S は FR の制御用 Strategy、DCP はドローン制御プログラム、Drone はドローンを表す。

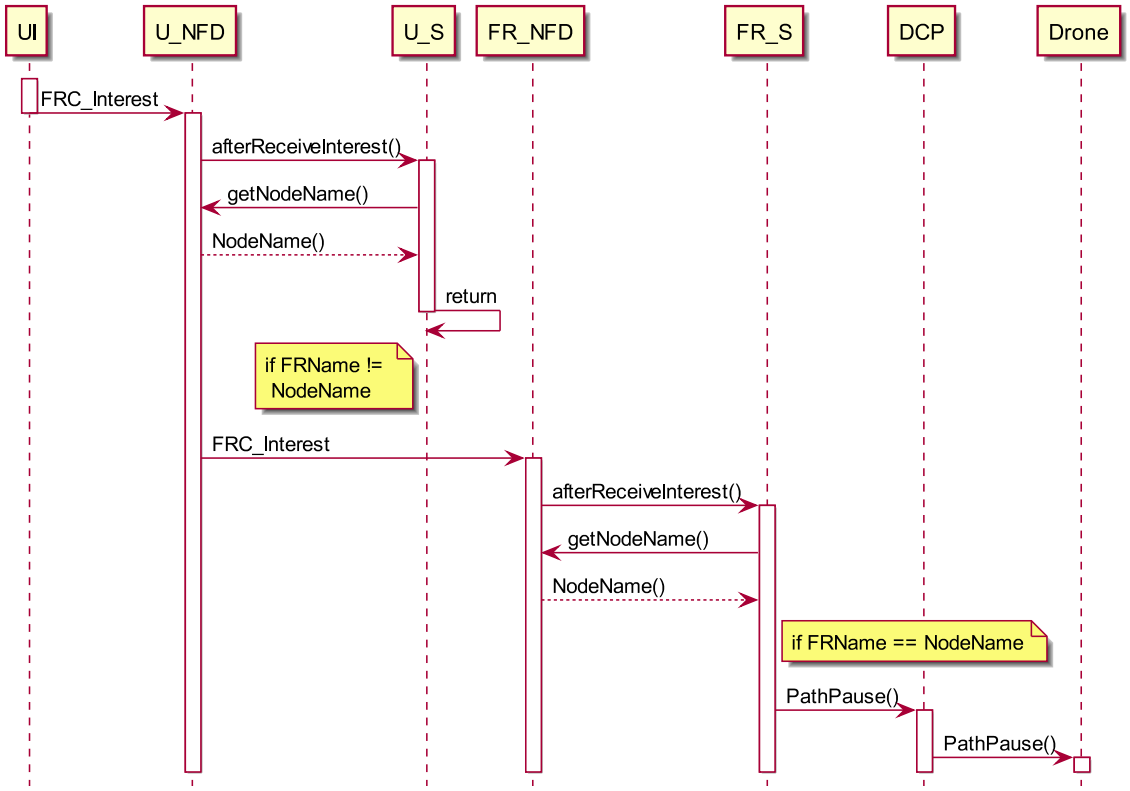

図 7: API の PathPause のシーケンス図

以下の箇条書きでは上記の流れの各処理の実現方法について説明する。

- 制御用 Interest の生成、制御用 Interest のユーザノード NFD への転送

  実現方法は API の PathCreate で記述してあり、ここでは省略する。

- ドローン制御プログラムの実行、パスプランの一時停止/再開関数の呼び出し

  ドローン制御プログラムの PathPause() で実装し、実現方法は API の PathRun で記述してあり、ここでは省略する。

- ドローン制御プログラムのパスプランの一時停止/再開の実装

  ドローン制御プログラムの PathPause() 関数で実装する。vehicle.mode に VehicleMode ('GUIDED') を代入することでパスプランの一時停止及び移動の停止が実現可能であり、VehicleMode ('AUTO') を代入することでパスプランの再開及び移動の再開が実現可能。

下表では API の PathPause で追加するメソッドを示す。

表 14: API の PathPause の追加するメソッド

コンポーネント	ファイル	所属クラス	メソッド名	説明
制御用 Strategy	FRControl Strategy.cpp	FRControl Strategy	public void PathPause()	ドローン制御プログラムの PathPause() 関数の呼び出し
DCP	DCP.py	なし	PathPause()	パスプランの一時停止/再開

PathCancel

下表は FR 移動制御システム API の PathCancel の動作概要、パラメータ、出力などの API 情報についての説明である。

表 15: PathCancel の API 情報

書式	/FRControl/{FRName}/PathCancel
例	/FRControl/FR0/PathCancel
動作概要	現在実行中のパスプランをキャンセルし、移動を中止させる
呼び出しタイミング	PathRun 或いは PathGoto の実行で移動中の時、或いはそれらの実行で一時停止中の時
パラメータ	なし
出力	なし
例外処理	• PathRun 及び PathGoto 両方の実行時ではない時に呼び出しを行うと、ドローン制御プログラムの PathCancel() 関数が終了する

制御の流れについては、ユーザインターフェイスで生成した制御用 Interest が FR の NFD に到着し、制御用 Strategy でノードの名前を確認する。制御用 Interest の指定 FR 名と FR のノード名が一致したら、ドローン制御プログラムのパスプランのキャンセル関数を呼び出す。

上記の制御の流れはシーケンス図で表している。図中 UI はユーザインターフェイス、U_NFD はユーザノードの NFD、U_S はユーザノードの制御用 Strategy、FR_NFD は FR の

NFD、FR_S は FR の制御用 Strategy、DCP はドローン制御プログラム、Drone はドローンを表す。

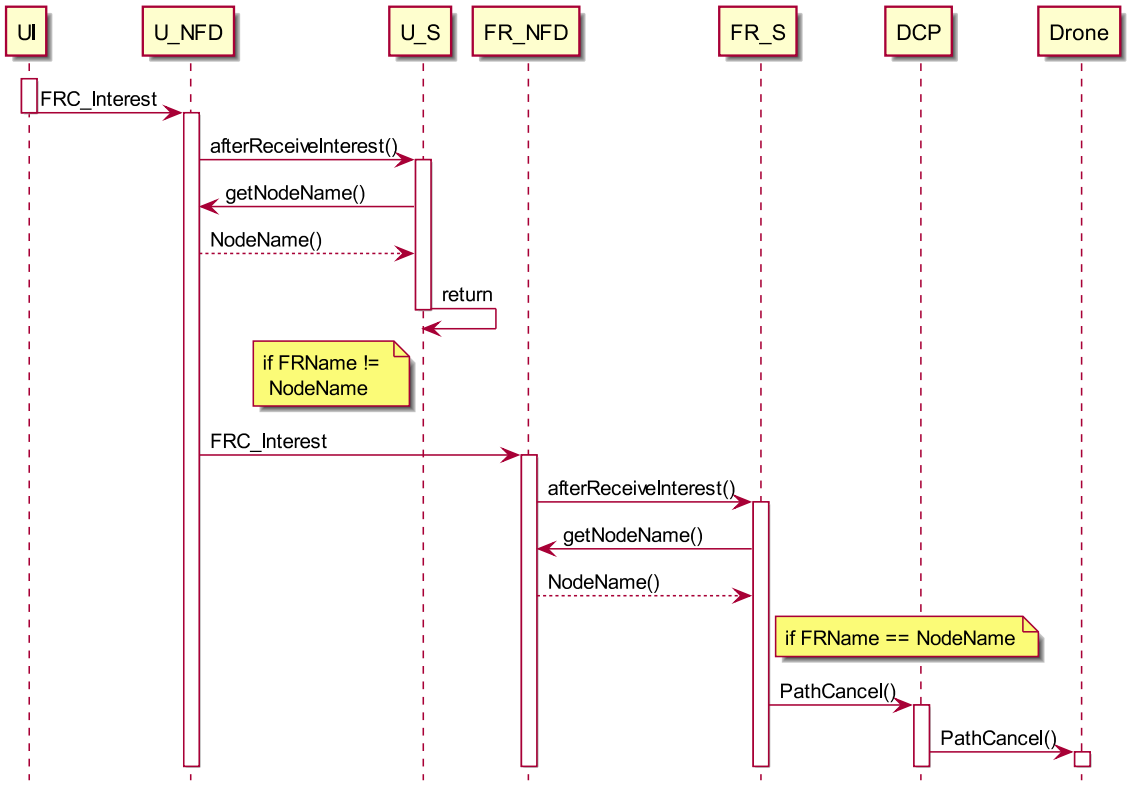

図 8: API の PathCancel のシーケンス図

以下の箇条書きでは上記の流れの各処理の実現方法について説明する。

- 制御用 Interest の生成、制御用 Interest のユーザノード NFD への転送

  実現方法は API の PathCreate で記述してあり、ここでは省略する。

- ドローン制御プログラムの実行、パスプランの一時停止/再開関数の呼び出し

  制御用 Strategy の PathCancel() 関数で実装し、実現方法は API の PathRun で記述してあり、ここでは省略する。

- ドローン制御プログラムのパスプランのキャンセルの実装

  ドローン制御プログラムで PathCancel() 関数で実装する。vehicle.commands 配列のエントリを削除し、vehicle.commands.upload() でパスプランの削除を行う。また、VehicleMode ('Brake') を代入することで移動の停止及び現在地での待機を実現できる。

下表では API の PathCancel で追加するメソッドを示す。

表 16: API の PathCancel の追加するメソッド

コンポーネント	ファイル	所属クラス	メソッド名	説明
制御用 Strategy	FRControlStrategy.cpp	FRControlStrategy	public void PathCancel()	ドローン制御プログラムの PathCancel() 関数の呼び出し
DCP	DCP.py	なし	PathCancel()	パスプランのキャンセル

SetVehicleSpeed

下表は FR 移動制御システム API の SetVehicleSpeed の動作概要、パラメータ、出力などの API 情報についての説明である。

表 17: SetVehicleSpeed の API 情報

書式	/FRControl/{FRName}/SetVehicleSpeed/{Speed}
例	/FRControl/FR0/SetVehicleSpeed/5
動作概要	ドローン速度の指定
呼び出しタイミング	FR の NFD で制御用 Strategy の適用後の任意時刻
パラメータ	Speed : ドローンの移動速度を表す正実数であり、単位はメートル毎秒である
出力	なし
例外処理	• Speed で零或いはマイナスの値或いはドローン機体の制限速度を超えた値を入力した場合、ドローンの速度が変化せずにドローン制御プログラムの SetVehicleSpeed() 関数が終了する

制御の流れについては、ユーザインターフェイスで生成した制御用 Interest が FR の NFD に到着し、制御用 Strategy でノードの名前を確認する。制御用 Interest の指定 FR 名と FR のノード名が一致したら、ドローン制御プログラムの移動速度の設定関数を呼び出す。

上記の制御の流れはシーケンス図で表している。図中 UI はユーザインターフェイス、U_NFD はユーザノードの NFD、U_S はユーザノードの制御用 Strategy、FR_NFD は FR の NFD、FR_S は FR の制御用 Strategy、DCP はドローン制御プログラム、Drone はドローンを表す。

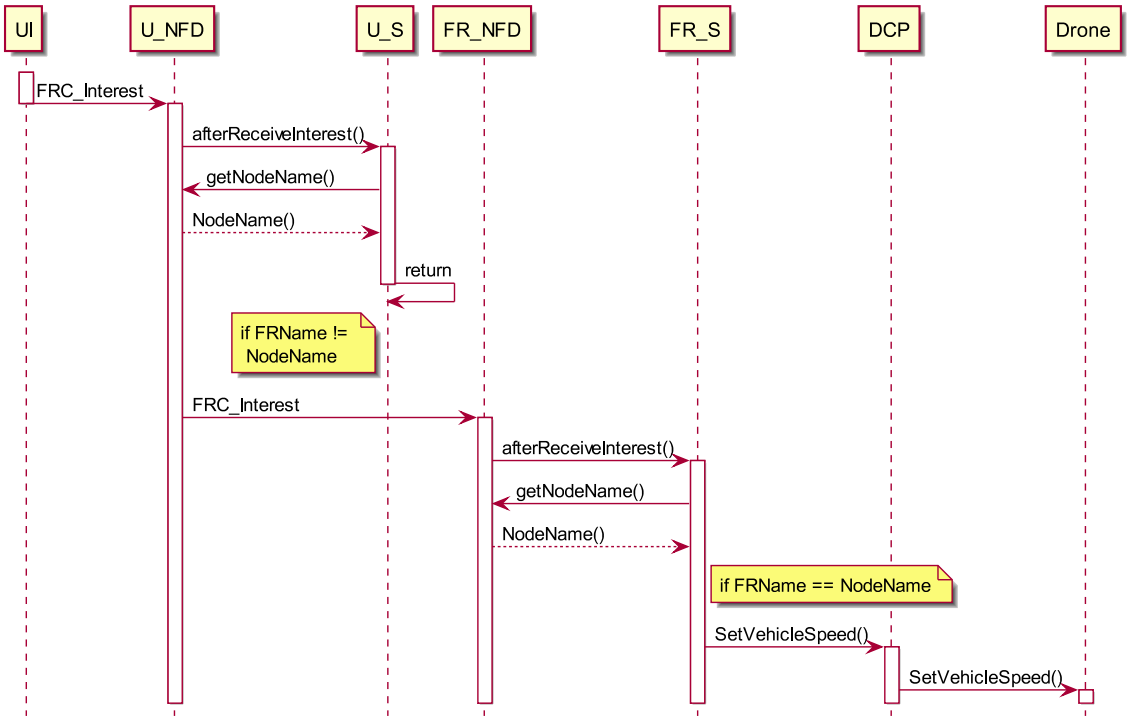

図 9: API の SetVehicleSpeed のシーケンス図

以下の箇条書きでは上記の流れの各処理の実現方法について説明する。

- 制御用 Interest の生成、制御用 Interest のユーザノード NFD への転送

  実現方法は API の PathCreate で記述してあり、ここでは省略する。

- ドローン制御プログラムの実行、パスプランの一時停止/再開関数の呼び出し

  制御用 Strategy の SetVehicleSpeed() で実装し、実現方法は API の PathRun で記述してあり、ここでは省略する。

- ドローン制御プログラムの移動速度の設定の実装

  ドローン制御プログラムで SetVehicleSpeed() 関数で実装する。vehicle.groundspeed に速度を表す正実数を与えることで移動速度を設定できる。

下表では API の SetVehicleSpeed で追加するメソッドを示す。

表 18: API の SetVehicleSpeed の追加するメソッド

コンポーネント	ファイル	所属クラス	メソッド名	説明
制御用 Strategy	FRControlStrategy.cpp	FRControlStrategy	public void SetVehicleSpeed(unsigned long Speed)	ドローン制御プログラムの SetVehicleSpeed() 関数の呼び出し
DCP	DCP.py	なし	SetVehicleSpeed()	移動速度の変更

3.4 FR 移動制御システムのパケット構造とコンポーネントの設計

3.1〜3.3 節ではシステムの各コンポーネントの仕様と実装する関数などについて説明した。3.4 節ではコンポーネント間の通信用のパケットおよび各コンポーネントごとのクラス、メソッド、変数の実装方法について説明する。最初に FR 移動制御システムのパケットについては制御用 Interest と実行結果 Data の名前空間を説明する。次に、コンポーネントについてはユーザインターフェイス、制御用 Strategy、ドローン制御プログラムの設計及びコンポーネント内の関数の実装方法について説明する。

3.4.1 パケット構造

本節では制御用 Interest の概要、制御用 Interest の名前空間、実行結果 Data の概要、実行結果 Data の名前空間について説明する。

制御用 Interest は FR の移動制御の実行を指示するための ICN パケットである。制御用 Interest は/FRControl の prefix を持ち、名前以外のパケット構造は ICN の Interest パケットと同じである。制御用 Interest の用途は制御用 Interest の送受信の際に制御用 Strategy 内の移動制御を実現する関数を呼び出すことにある。また、役目を果たした制御用 Interest の処理方法に関して、Interest の保存してある PIT エントリは InterestLifeTime 後に自動的に削除される。

以下の枠では制御用 Interest の名前の書式及びその各要素について説明する。

書式：/FRControl/{FRName}/{Function}/{Parameter}

FRControl： 制御用 Interest であることを判断する prefix であり、「FRControl」と固定する
FRName： 制御する FR の名前であり、本システムではドローンは一つなので名前は「FR0」とする
Function： 制御関数の名前
Parameter：関数のパラメータ

/FRControl の prefix を持つ Interest の中で以上の書式ではないものは破棄される。また、/FRControl の prefix を持たない Interest はコンテンツを要求する Interest としてみなし、制御用 Strategy の処理を呼び出さずに NFD のフォワーディングのアルゴリズムに従って処理される。下図では制御用 Interest の名前空間を示している。括弧で囲まれている名前は実際の利用例によって変化するものであり、それ以外の名前は利用例によって変化しないものである。

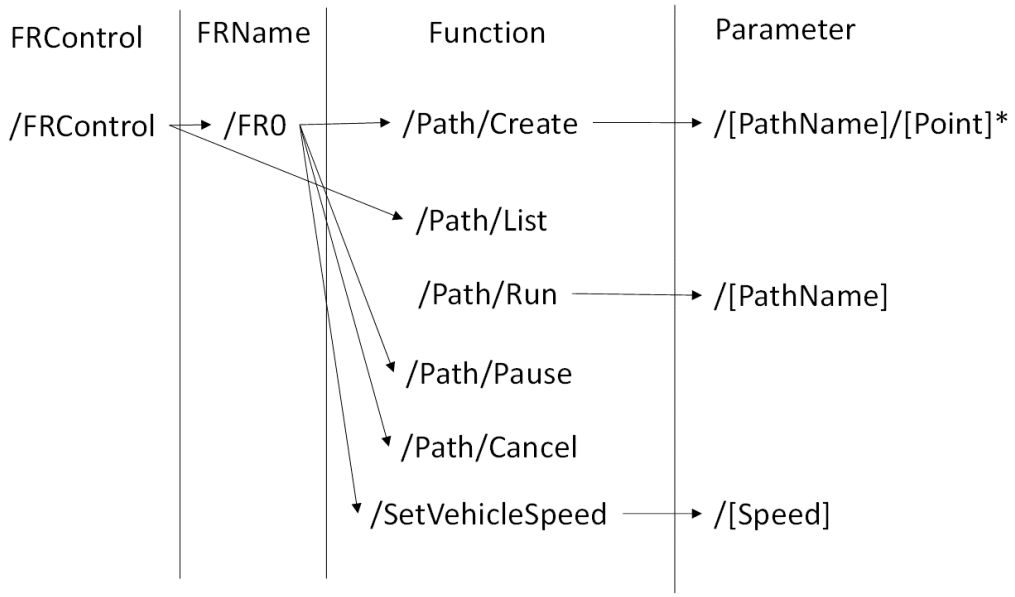

図 10: 制御用 Interest の名前空間

また、実行結果 Data は FR 移動制御の API の実行後の戻り値を格納した ICN パケットである。実行結果 Data は ICN の Data パケットと同じ構造を持ち、格納したコンテンツは API の戻り値である。本システムでは DiscoverNetwork 以外の API は戻り値を返す必要が

ないため、実行結果 Data の生成を行わないようにした。DiscoverNetwork では発見した分断ネットワークの座標と名前を戻り値とする。また、名前空間に関しては実行結果 Data は制御用 Interest と一対一の関係であり、実行結果 Data の名前空間は制御用 Interest と同じである。

3.4.2 ユーザインターフェイス

ユーザインターフェイスはネットワークなどの初期設定及び API の入力受付を行う C++ プログラムである。下表ではユーザインターフェイスの入出力、動作及びその実現方法などのコンポーネント情報について説明する。

表 19: ユーザインターフェイスのコンポーネント情報

動作概要	ネットワーク設定、NFD の転送設定などの初期設定、制御用 Interest の生成、NFD への転送
起動タイミング	ユーザノードと FR の電源が入った後に手動的に起動する
入力	標準入力から API の名前を入力し、その書式は 3.1 で述べた形とする
出力	NFD へ制御用 Interest を転送
例外処理	NDN の名前の規則で定義されていない文字を入力した場合、当入力を無視して次の入力を求める
使用ツール、言語	C++ 11、ndn-cxx 0.5.0、NFD 0.5.0
その他	• ndn-cxx のチュートリアルプログラム consumer.cpp を改造することで実装する • ユーザノードにインストールする

以下の箇条書きではユーザインターフェイスの主要動作及びその実現方法を説明する。

- ユーザノード NFD の起動

  NFD の API である「nfd-start」を用いて起動する

- ユーザノード NFD の制御用 Interest 転送先の設定

  NFD の API である「nfdc register ndn:/FRControl/User/FR0 udp://10.1.1.102」を用いて設定する。

- FR の RPi における NFD の起動と制御用 Strategy の適用

ユーザノードで ssh で遠隔ログインし、起動では NFD の API である「nfd-start」を用いて起動し、制御用 Strategy の適用では NFD の API である「nfdc set-strategy ndn:/FRControl ndn:/localhost/nfd/strategy/FRControl」を用いて適用する。

- 標準入力の受付

  FR 制御関数名とパラメータのみの入力を受け付け、/FRControl/FR0 は入力後自動的に先頭に付加される。

- 制御用 Interest の生成、転送

  生成では NDN の関数である「Interest(string InterestName)」を用いて生成する。転送では NDN の関数である「face.expressInterest()」を用いて転送する。

下表ではユーザインターフェイスで追加するクラス、メソッドについて説明する。

表 20: ユーザインターフェイスで追加するクラス、メソッド

クラス名	FRC_UI
説明	ユーザインターフェイスのクラス
属性及びその説明	• private ndn::Face m_face：制御用 Interest 転送用のフェイス
メソッド及びその説明	• public void SendFRControlInterest(ndn::Name InterestName) 制御用 Interest の生成、転送関数

メソッド名	public void FRC_UI::SendFRControlInterest(ndn::Name InterestName)
動作概要	制御用 Interest の生成、NFD への転送
引数	• ndn::Name InterestName：制御用 Interest の名前
戻り値	なし
その他	InterestLifeTime はデフォルトで 1 秒
実装方法	• 制御用 Interest の生成：NDN の Interest クラスのコンストラクタである ndn::Interest(const char* InterestName) を使用する • 制御用 Interest の転送：ndn::Face::expressInterest (ndn::Interest interest) 関数を使用する

3.4.3 NFD

NFD はパケットの転送、テーブルなどの変数の保存及び取得を行うプログラムである。下表では NFD の入出力、動作及びその実現方法などのコンポーネント情報について説明する。

表 21: NFD のコンポーネント情報

動作概要	パケットの転送、変数の保存及び取得、Strategy の呼び出し
起動タイミング	ユーザインターフェイス起動後に呼び出される
入力	Interest か Data
出力	Strategy に制御用 Interest か実行結果 Data、或いは他のノードの NFD へ Interest か Data を出力
例外処理	到着 Interest に対応する FIB エントリが存在しない場合、InterestLifeTime 後に Interest を破棄する
使用ツール、言語	C++ 11、ndn-cxx 0.5.0、NFD 0.5.0
その他	• パスプランのマップ、分断ネットワークの座標リスト及びそれらの取得、設定関数を追加する • ユーザノード、FR にインストールする

以下の箇条書きでは NFD に追加する動作及びその実現方法を説明する。

- パスプランのマップ、分断ネットワークの座標マップ及びそれらの取得、設定関数

  パスプランのマップは ndn::Name 型のパスプランの名前と nfd::Point 型の地点座標か地点名前のマップであり、nfd::Point クラスを Forwarder.hpp に追加する。nfd::Point クラスの属性はホームからの南北方向の距離、東西方向の距離、高さ、名前である。また、分断ネットワークの座標リストは nfd::Point 型の配列である。両マップは nfd::Forwarder クラスの private メンバとして定義し、それらの取得、設定関数は public メソッドで定義する。

下表では NFD で追加するクラス、メソッド、変数について説明する。

表 22: NFD で追加するクラス、メソッド、変数

クラス名	nfd::Point
説明	ノードの名前、座標を持つクラス
属性及びその説明	• public ndn::Name PointName：ノードの名前 • public long North：FR からの南北方向の距離、正は北で負は南、単位はメートル • public long East：FR からの東西方向の距離、正は東で負は西、単位はメートル • public long Altitude：高さ、単位はメートル
メソッド及びその説明	なし

メソッド名	protected nfd::Forwarder& Strategy::getForwarder()
動作概要	Forwarder 内変数の値の取得と設定のために、nfd::Forwarder インスタンスの取得
引数	なし
戻り値	nfd::Forwarder インスタンス
実装方法	Strategy クラスの private m_forwarder を戻り値にする

メソッド名	protected ndn::Name& getNodeName()
動作概要	ノードの名前の取得
引数	なし
戻り値	ノードの名前
実装方法	FRControlStrategy クラスの private m_NodeName を戻り値にする

メソッド名	protected setNodeName(ndn::Name NodeName)
動作概要	ノードの名前の取得
引数	ノードの名前
戻り値	なし
実装方法	FRControlStrategy クラスの private m_NodeName にノードの名前を代入する

メソッド名	protected std::map<ndn::Name, vector<nfd::Point>>& getPathMap()
動作概要	パスプランのマップの取得

引数	なし
戻り値	パスプランのマップ
実装方法	FRControlStrategy クラスの private m_PathMap を戻り値にする

メソッド名	protected void setPathMap(std::map<ndn::Name, vector<nfd::Point>> PathMap)
動作概要	パスプランのマップの設定
引数	パスプランのマップ
戻り値	なし
実装方法	FRControlStrategy クラスの private m_PathMap にパスプランのマップを代入する

メソッド名	protected std::vector<nfd::Point>& getFNPositionList()
動作概要	分断ネットワークの地点リストの取得
引数	なし
戻り値	分断ネットワークの地点リスト
実装方法	FRControlStrategy クラスの private m_FNPositionList を戻り値にする

メソッド名	protected getFNPositionList(std::vector<nfd::Point> FNPositionList)
動作概要	分断ネットワークの地点リストの設定
引数	分断ネットワークの地点リスト
戻り値	なし
実装方法	FRControlStrategy クラスの private m_FNPositionList に分断ネットワークの地点リストを代入する

変数名	所属クラス	説明
private ndn::Name m_NodeName	nfd::Forwarder	ノードの名前
private std::map<ndn::Name, vector<nfd::Point>> m_PathMap	nfd::Forwarder	パスプランのマップ
private std::vector<nfd::Point> m_FNPositionList	nfd::Forwarder	分断ネットワークの座標リスト

3.4.4 制御用 Strategy

制御用 Strategy とはドローンの移動制御コードの生成と転送、実行結果の生成と保存を行う NDN の拡張ストラテジである。下表では制御用 Strategy の入出力、動作及びその実現方法などのコンポーネント情報について説明する。

表 24: 制御用 Strategy のコンポーネント情報

動作概要	API の関数の実行、ドローン制御プログラムの呼び出し
起動タイミング	ユーザインターフェイス起動後に呼び出される
入力	制御用 Interest
出力	ドローン制御プログラムにコマンドライン引数を渡し、或いは標準出力にパスプランなどを出力する
例外処理	制御用 Interest の FRName と FR のノードネームが一致しない場合、関数が終了する 制御用 Interest の制御関数名が定義されていない、或いは制御関数のパラメータが定義されていない場合、関数が終了する
使用ツール、言語	C++ 11、ndn-cxx 0.5.0、NFD 0.5.0
その他	• /FRControl の prefix を持たない Interest は FRControl-Strategy クラスを呼び出さず、他のストラテジに適用するかストラテジを適用せず、NFD の定義に従って転送される

	• ndn::Strategy の子クラスとして実装する • ユーザノード、FR にインストールする

以下の箇条書きでは制御用 Strategy の主要動作及びその実現方法を述べる。

- 制御用 Interest の受け取り

  NFD で afterReiceiveInterest() を呼び出して制御用 Interest を渡している。制御用 Strategy の afterReiceiveInterest() 関数で受け取った後の動作を記述する。

- API の関数の実行、ドローン制御プログラムの呼び出し

  3.1 で述べた API はそれぞれ PathCreate()、PathList() などの関数で実装する。それらの関数では主にドローン制御プログラムを外部コマンドの実行関数 std::system() で呼び出し、或いは NFD に保存してあるデータに対して操作を行っている。ドローン制御プログラムにはドローンの制御命令を記述している。

下表では制御用 Strategy で追加するクラス、メソッドについて説明する。

表 25: 制御用 Strategy で追加するクラス、メソッド

クラス名	ndn::FRControlStrategy：public Strategy
説明	FR 制御用 Strategy
属性及びその説明	なし
メソッド及びその説明	• void afterReceiveInterest(const ndn::Interest interest)：制御用 Interest 受信時のリスナ • void PathCreate(const ndn::Interest interest)、void PathList()：API の PathCreate と PathList の動作を実現する関数であり、標準出力にパスプランを出力する • void PathRun(const ndn::Interest interest)、void PathPause()、void PathCancel()、void SetVehicleSpeed(const ndn::Interest interest)：ドローン制御用プログラム内の API の PathRun、PathPause、SetVehicleSpeed の動作を実現する関数を呼び出す

メソッド名	void afterReceiveInterest(const ndn::Interest interest)

動作概要	制御用 Interest 受信時のリスナ、中には API に対応する関数を定義している
引数	制御用 Interest
戻り値	なし
実装方法	FRControlStrategy.hpp 及び FRControlStrategy.cpp でその動作定義し、コマンドラインで「nfdc set-strategy」でストラテジを適用する

メソッド名	void PathCreate(const ndn::Interest interest)
動作概要	パスプランの生成、NFD での保存
引数	制御用 Interest
戻り値	なし
実装方法	getForwarder() で forwarder インスタンスを取得し、forwarder.getPathMap() と forwarder.setPathMap() でパスプランを設定する

メソッド名	void PathList()
動作概要	パスプランの表示
引数	なし
戻り値	なし
実装方法	getForwarder() で forwarder インスタンスを取得し、forwarder.getPathMap でパスプランのマップを取得して標準出力に出力する

メソッド名	void PathRun(const ndn::Interest interest)
動作概要	指定パスプランの各地点の座標をファイルに出力し、ドローン制御プログラムのパスプラン実行関数を呼び出す
引数	制御用 Interest
戻り値	なし
実装方法	外部コマンドの実行関数 std::system() でドローン制御プログラムにコマンドライン引数「-m PathRun」を渡すことで、ドローン制御プログラム内の PathRun() を実行させる

メソッド名	void PathPause(const ndn::Interest interest)
動作概要	ドローン制御プログラムのパスプラン一時停止関数を呼び出す

引数	制御用 Interest
戻り値	なし
実装方法	外部コマンドの実行関数 std::system() でドローン制御プログラムにコマンドライン引数「-m PathPause」を渡すことで、ドローン制御プログラム内の PathPause() を実行させる

メソッド名	void PathCancel(const ndn::Interest interest)
動作概要	ドローン制御プログラムのパスプランキャンセル関数を呼び出す
引数	制御用 Interest
戻り値	なし
実装方法	外部コマンドの実行関数 std::system() でドローン制御プログラムにコマンドライン引数「-m PathCancel」を渡すことで、ドローン制御プログラム内の PathCancel() を実行させる

メソッド名	void SetVehicleSpeed(const ndn::Interest interest)
動作概要	ドローン制御プログラムの移動速度変更関数を呼び出す
引数	制御用 Interest
戻り値	なし
実装方法	外部コマンドの実行関数 std::system() でドローン制御プログラムにコマンドライン引数「-m SetVehicleSpeed -s {speed}」を渡すことで、ドローン制御プログラム内の SetVehicleSpeed() を実行させる

3.4.5 ドローン制御プログラム

ドローン制御プログラムとは制御用 Strategy からコマンドライン引数を受け取って呼び出され、ドローン制御関数の実行及びドローンとの通信を行うプログラムである。下表ではドローン制御プログラムの入出力、動作及びその実現方法などのコンポーネント情報について説明する。

表 27: ドローン制御プログラムのコンポーネント情報

動作概要	ドローンとの通信及びドローン制御関数の実行
起動タイミング	ユーザインターフェイス起動後に呼び出される
入力	制御用 Strategy からコマンドライン引数を受け取り、パスプランの地点座標が記載されているファイルを読み込む
出力	ドローンに制御命令、NFD に制御用 Interest か実行結果 Data を転送する
例外処理	ドローンかコントローラが電源を入れていない、或いはドローンと通信できない場合、プログラムが終了する
使用ツール、言語	Python 2.7.0、Dronekit-Python 2.0、ndn-cxx 0.5.0、NFD 0.5.0
その他	• Dronekit-Python が提供する examples の mission-basic.py を改造して実装する • FR にインストールする

以下の箇条書きではドローン制御プログラムの主要動作及びその実現方法を述べる。

- ドローンインスタンスの取得

  Dronekit-Python の vehicle.connect(connect_string, port) 関数で指定 IP アドレスを持つドローンのインスタンスを取得できる。本システムでは 10.1.1.10:14550 で FR0 のインスタンスを取得する。

- ドローン制御関数の実行

  3.1 で述べた API はそれぞれ PathCreate()、PathList() などの関数で実装する

下表ではドローン制御プログラムで追加するメソッドについて説明する。

メソッド名	PathRun()
動作概要	ファイルから地点座標を取得し、地点座標へ移動するコマンドを配列に追加することでパスプランを実行する
引数	なし
戻り値	なし

実装方法	ファイルから地点座標を取得し、ドローン制御プログラムの get_location_metres() 関数で東西南北方向の距離の座標を GPS 座標の地点 point に変換し、vehicle.commands に地点座標へ移動するコマンド Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, mavutil.mavlink.MAV_CMD_NAV_WAYPOINT,0,0,0,0,0,0, point.lat, point.lon, point.alt) を追加し、vehicle.commands.upload() 関数でドローンに転送する

メソッド名	PathPause()
動作概要	ドローンのモードの切り替えることでパスプランの一時停止を実現する
引数	なし
戻り値	なし
実装方法	ドローンのモードが AUTO であれば GUIDED にして移動を停止させ、GUIDED であれば AUTO にして移動を再開させる

メソッド名	PathCancel()
動作概要	コマンド配列を空にして、さらにモードを変えることでパスプランのキャンセルを実現する
引数	なし
戻り値	なし
実装方法	vehicle.commands.clear() で配列を空にし、vehicle.commands.upload() でドローンに転送し、さらに vehicle.mode に VehicleMode('BRAKE') を代入する
メソッド名	SetVehicleSpeedl()
動作概要	インスタンスの速度属性に値を代入することでパスプランのキャンセルを実現する
引数	なし
戻り値	なし
実装方法	vehicle.groundspeed に速度の正実数を代入する

4　実機検証

本研究の FR 移動制御システムは移動ルータを用いた様々な場面における移動を実現するものである。本章では FR 移動制御システムの動作を確認することで、本システムを用いることで移動ルータに対して能動的制御が可能であることを示す。最初に実機検証を行うための実装環境、使用機材、実験環境を述べる。次に、システムの検証方法として使用データと操作手順について説明する。そして、検証方法に従って行った検証の結果を示すことで、ICN のネーミングスキーマは移動制御 API としての使用が可能であること、および本システムを用いることで移動ルータに対して能動的制御が可能であることを示す。最後に、本システムでは移動ルータに対してドローンのコントローラの通信範囲内で移動制御を行っているが、その通信範囲外の移動制御を行う方法を考察で説明する。

4.1　実装環境、使用機材、実験環境

下表では FR 移動制御システムの実装環境に関して使用したソフトウエアやツールを列挙する。

表 29: 実装環境

ツール	ツールバージョン	説明
ndn-cxx	0.5.0	ストラテジ層が正常に動作
NFD	0.5.0	ストラテジ層が正常に動作
DroneKit-Python	2.0	ドローンの制御ツール
Mission Planner	1.3.44	動作確認で使用し、ドローンの状態を GUI で表示するツール
Python	2.7.0	ドローンの制御ツールの言語
c++	C++11	ndn-cxx と NFD とユーザインターフェイスの言語
ユーザノード OS	Ubuntu 14.04 LTS	NFD が正常に動作
FR の RPi の OS	Raspbian Sep 2016	NFD が正常に動作

本研究では下表で示した機材を使用している。

表 30: 使用機材の説明

機材	機材型番	説明
ドローン	3D Robotics Solo	プログラムによる制御可能、WiFi による通信可能、GPS 搭載
小型コンピュータ	Raspberry Pi 3 Model B	5V1A で電源供給可能、WiFi 搭載、ndn-cxx、NFD などのアプリケーションを所持

図 11: FR の全体図

ドローンと RPi の接続に関しては、ドローンと RPi は USB ケーブルによるデータ交換をせず、ドローンから RPi への電力供給のみを行う。その理由はドローンの仕様でドローンから RPi を USB ホスト及び USB デバイスとして識別できなかったからである。ただし、ドローンと RPi の通信はコントローラのネットワークで実現する。

電力供給の方法としては、ドローンの Accessory Bay に TX24-30R-12ST-H1E のコネクタを挿し込み、コネクタのピンから半田でジャンプワイヤーと繋ぎ、ジャンプワイヤーを RPi の GIPO のピンに挿し込むことで電力を供給できる。

FR のソフトウエア構成に関しては、ドローンには pip、DroneKit-Python、RPi には ndn-cxx、NFD をインストールしている。

また、実機検証の場所に関しては、後述の検証方法では高さ 15 メートル、辺の長さ 10

メートルの正方形地域のパスプランを使用している。安全のため、高さ 30 メートル、辺の長さ 20 メートルの正方形地域の場所を実験場所とする。本研究では大阪大学吹田キャンパス工学部グラウンドを使用していた。

4.2 検証方法

本研究の目的は移動戦略に従った FR の能動的制御の実現である。移動戦略はとある移動戦略のポリシーに従って移動制御 API を組み合わせることで実装可能である。そこで、各移動制御 API が設計通りに正しく動作することを保障すべきである。以下では ICN ネーミングスキーマを FR 移動制御 API としての使用が可能であること、および本システムを用いることで FR の能動的制御が可能であることを示すために、各移動制御 API の動作を確認する。本節ではそれぞれの API の使用データ、検証手順及び予想動作について説明する。

PathCreate と PathList

PathCreate と PathList の検証では生成したパスプランの一覧の出力を観測することで動作の正しさを検証する。使用データに関しては、パスプラン一覧の表示で全てのパスプランを出力することを検証するために、パスプランを二つ以上用意する。ここでは便宜のために二つのパスプラン Path0、Path1 を用意する。二つのパスプランの各地点座標を出力させるために、各パスプランには二つ以上の地点を用意する。各パスプランの各地点の出力を観測しやくすること及び API の入力の短縮のために、二つのパスプラン Path0 と Path1 を用意し、Path0 に (100, 100, 10) と (-100, -100, 10) の二つの地点を持たせ、Path1 に (200, 200, 10) と (-200, -200, 10) の二つの地点を持たせる。 ここで (X、Y、Z) は FR から、X が正なら北方向 X メートル、負なら南方向 X メートル、Y が正なら東方向 Y メートル、負なら Y メートル離れており、高さが Z メートルの地点を表す。

検証手順及び予想動作に関しては、最初にユーザインターフェイスに「PathCreate/Path0/100/100/10/-100/-100/10」と「PathCreate/Path1/200/200/10/-200/-200/10」を入力し、標準出力に Path0、Path1 の地点座標が出力されることを確認する。次にユーザインターフェイスに「PathList」を入力し、標準出力に Path0 と Path1 の地点座標が出力されることを確認する。以上の動作を確認できれば、PathCreate と PathList は仕様通りに正しく動作したと判断できる。

PathRun

PathRun の検証では生成したパスプランに沿う FR の移動を観測することで動作の正しさを検証する。使用データに関しては、一つのパスプランの全ての地点を訪問することを確認するために、二つ以上の地点を含むパスプランを用意する。ここでは FR の移動経路を観測しやすくするために、正方形の頂点である (5, 5, 10)、(-5, 5, 10)、(-5, -5, 10)、(5, -5, 10) の四つの地点を含むパスプラン Path0 を用意する。

検証手順及び予想動作に関しては、ユーザインターフェイスに「PathCreate/ Path0/5/5/10/-5/5/10/-5/-5/10/5/-5/10」、「PathRun/Path0」の順番で入力し、FR が正方形の経路に沿う移動を確認する。また、FR の状態を無人飛行機エミュレータ Mission Planner で受け取り、Mission Planner での FR の経路を確認する。以上の動作を確認できれば、PathRun は仕様通りに正しく動作したと判断できる。

PathPause

PathPause の検証では実行中のパスプランの一時停止を観測することで動作の正しさを検証する。使用データに関しては、API 命令の転送に十分な時間があり一定以上の移動時間を要するパスプランが相応しく、便宜のため上述の PathRun のパスプランを使用する。

検証手順及び予想動作に関しては、ユーザインターフェイスに「PathCreate/Path0/5/5/10/-5/5/10/-5/-5/10/5/-5/10」、「PathRun/Path0」の順番で入力し、次に FR 移動中の任意時刻にユーザインターフェイスに「PathPause」を入力し、FR の移動停止を確認する。また、Mission Planner で移動の停止を観測し、標準出力にパスプランの残りの未訪問地点の数を確認する。移動停止状態でユーザインターフェイスに「PathPause」を入力し、FR の移動が再開し、元のパスプランに沿う移動を確認する。以上の動作を確認できれば、PathPause は仕様通りに正しく動作したと判断できる。

PathCancel

PathCancel の検証では実行中のパスプランのキャンセル及び移動の停止を観測することで動作の正しさを検証する。使用データに関しては、API 命令の転送に十分な時間があり一定以上の移動時間を要するパスプランが相応しく、便宜のため上述の PathRun のパスプランを使用する。

検証手順及び予想動作に関しては、ユーザインターフェイスに「PathCreate/Path0/5/5/10/-5/5/10/-5/-5/10/5/-5/10」、「PathRun/Path0」の順番で入力し、次に FR 移動中の任意時刻にユーザインターフェイスに「PathCancel」を入力し、移動の停止を観測する。また、

Mission Planner で移動の停止を観測し、ドローン制御プログラムの終了を確認する。以上の動作を確認できれば、PathCancel は仕様通りに正しく動作したと判断できる。

SetVehicleSpeed

SetVehicleSpeed の検証では移動時の速度変化を観測することで動作の正しさを検証する。使用データに関しては、API 命令の転送に十分な時間があり一定以上の移動時間を要するパスプランが相応しく、便宜のため上述の PathRun のパスプランを使用する。

検証手順及び予想動作に関しては、ユーザインターフェイスに「SetVehicleSpeed/1」、「PathCreate/ Path0/5/5/10/-5/5/10/-5/-5/10/5/-5/10」、「PathRun/Path0」の順番でを入力し、FR の移動速度が約 1 メートル毎秒であることを確認する。また、Mission Planner でスピードが 1 メートル毎秒であることを確認する。次にユーザインターフェイスに「/SetVehicleSpeed/3」を入力し、FR の移動速度の増加を確認する。また、Mission Planner でスピードが 1 メートル毎秒以上であることを確認する。以上の動作を確認できれば、SetVehicleSpeed は仕様通りに正しく動作したと判断できる。

4.3 検証結果

PathCreate と PathList

ユーザインターフェイスに「PathCreate/Path0/100/100/10/-100/-100/10」と「PathCreate/ Path1/200/200/10/-200/-200/10」を入力し、図 12 の赤い枠の各地点の座標の出力を確認した。

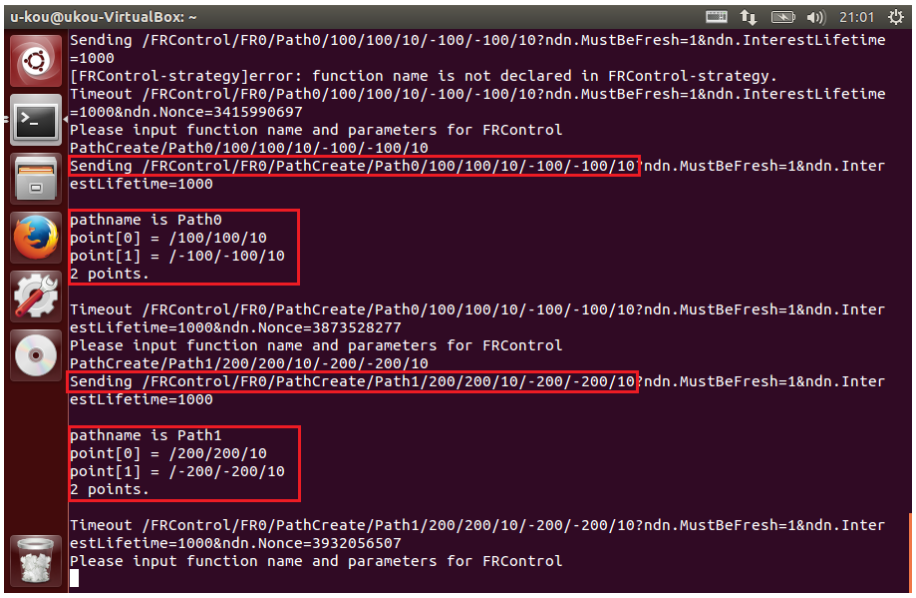

図 12: API の PathCreate のコマンドラインにおける出力結果

次に、ユーザインターフェイスに PathList を入力し、図 13 の赤い枠の保存された各地点の座標の出力を確認した。以上より、API の PathCreate と PathList の動作が正しく行われたと判断する。

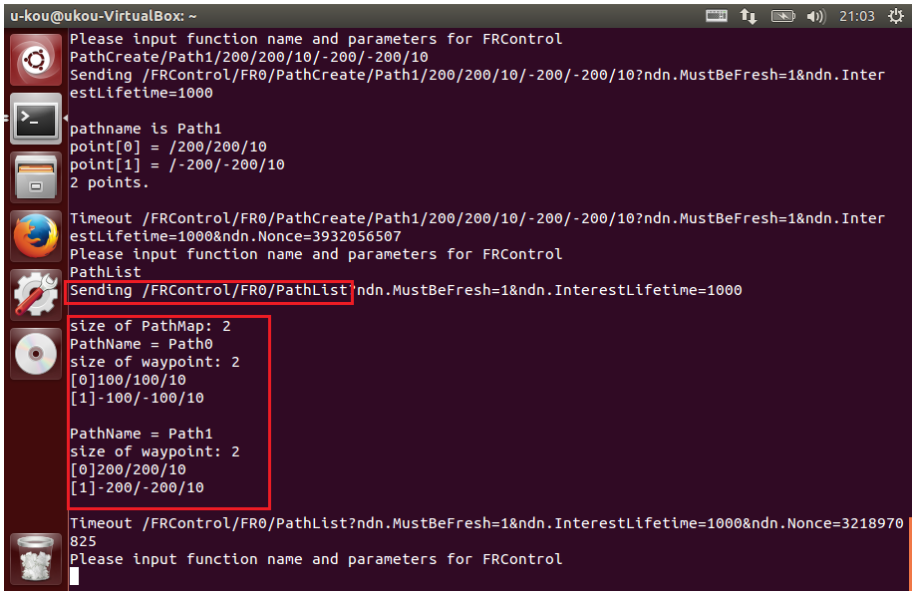

図 13: API の PathCreate のコマンドラインにおける出力結果

PathRun

ユーザインターフェイスに「PathCreate/Path0/5/5/10/-5/5/10/-5/-5/10/5/-5/10」、「PathRun/Path0」を入力し、実機検証では FR の移動経路が正方形であることを確認した。また、Mission Planner で FR の移動経路を確認した。以上より、API の PathRun の動作が正しく行われたと判断する。

図 14: API の PathRun の Mission Planner における出力結果

PathPause

ユーザインターフェイスに「PathCreate/Path0/5/5/10/-5/5/10/-5/-5/10/5/-5/10」、「PathRun/Path0」を入力し、パスプランの全ての地点に訪問する前に PathPause を入力し、実機検証では FR の移動停止を確認した。また、コマンドラインでは図 15 の赤い枠のパスプラン一時停止のメッセージの出力を確認し、Mission Planner では FR の移動停止を確認した。

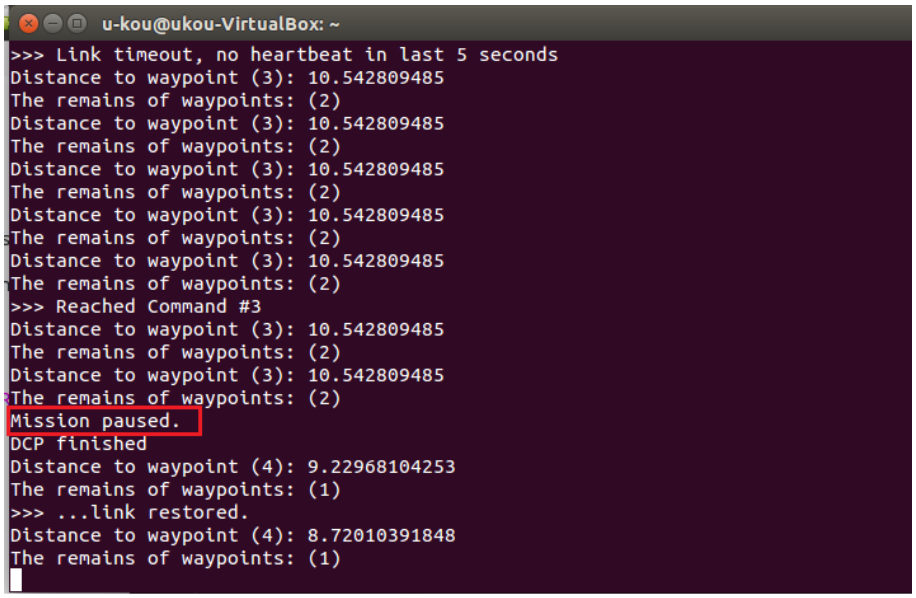

図 15: API の PathPause のコマンドラインにおける出力結果 パスプラン一時停止

図 16: API の PathPause の Mission Planner における出力結果 パスプラン一時停止

パスプランの再開に関しては、一時停止の状態でユーザインターフェイスに PathPause を入力し、実機検証では FR のパスプランに沿う移動を確認した。また、コマンドラインでは図 17 の赤い枠のパスプラン再開のメッセージの出力を確認し、Mission Planner では FR のパスプランに沿う移動を確認した。以上より、API の PathPause の動作が正しく行われたと判断する。

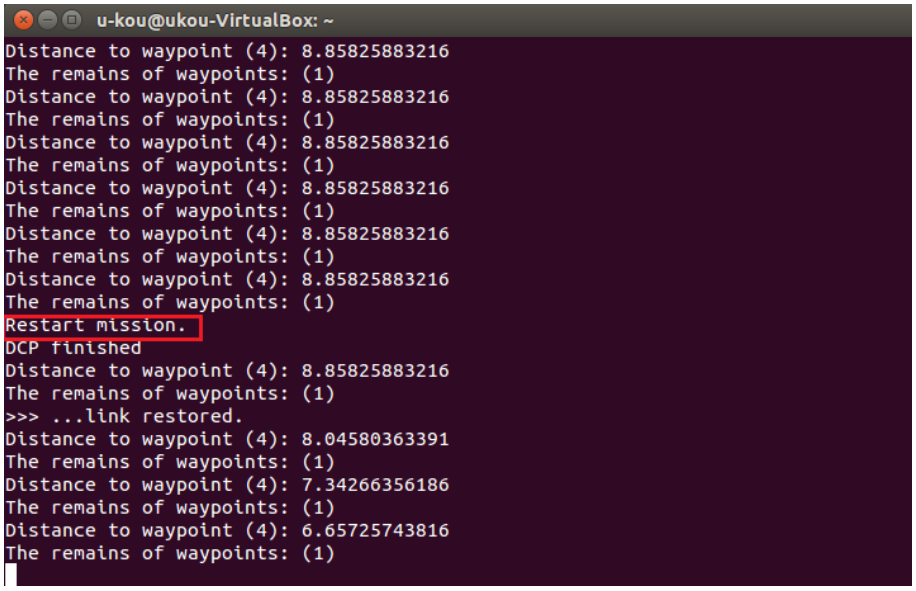

図 17: API の PathPause のコマンドラインにおける出力結果 パスプラン再開

図 18: API の PathPause の Mission Planner における出力結果 パスプラン再開

PathCancel

ユーザインターフェイスに「PathCreate/Path0/5/5/10/-5/5/10/-5/-5/10/5/-5/10」、「PathRun/Path0」を入力し、パスプランの全ての地点に訪問する前に PathCancel を入力し、実機検証では FR の移動停止を確認した。また、コマンドラインでは図 19 の赤い枠のパスプラン

キャンセルのメッセージの出力を確認し、Mission Planner では FR の移動停止を確認した。以上より、API の PathCancel の動作が正しく行われたと判断する。

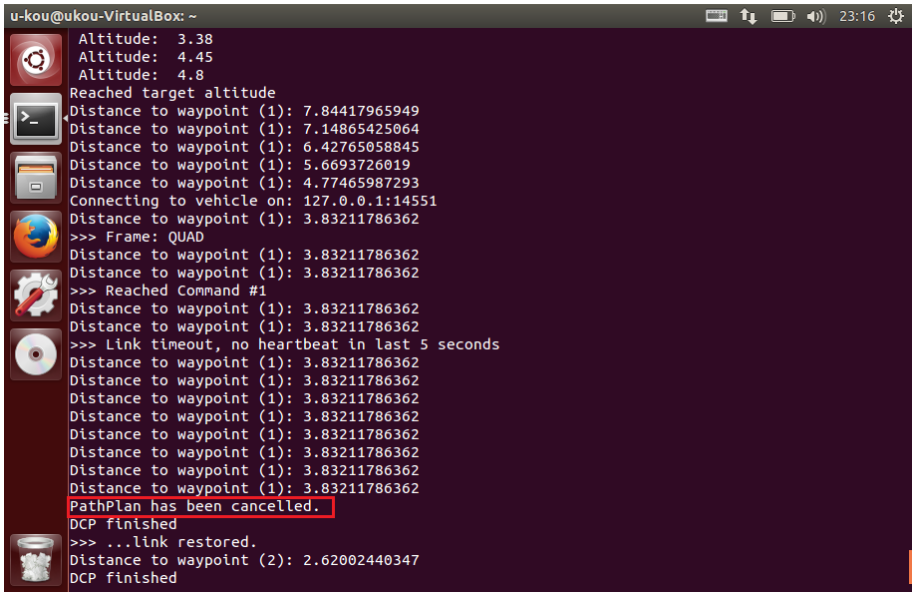

図 19: API の PathCancel のコマンドラインにおける出力結果

図 20: API の PathCancel の Mission Planner における出力結果

SetVehicleSpeed

ユーザインターフェイスに SetVehicleSpeed/1、「PathCreate/Path0/5/5/10/-5/5/10/-5/-5/10/ 5/-5/10」、「PathRun/Path0」を入力し、実機検証では FR が 1 メートル毎秒の速度で移動することを確認した。また、Mission Planner では FR の移動速度が 1 メートル毎秒以下であることを確認した。

図 21: API の SetVehicleSpeed の Mission Planner における出力結果 速度変更前

速度の変更を確認するために、ユーザインターフェイスに SetVehicleSpeed/3 を入力し、実機検証では FR の速度の増加を確認した。また、コマンドラインで図 22 の赤い枠の速度変更のメッセージの出力を確認し、Mission Planner では FR の移動速度が 1 メートル毎秒以上 3 メートル毎秒以下であることを確認した。以上より、API の SetVehicleSpeed の動作が正しく行われたと判断する。

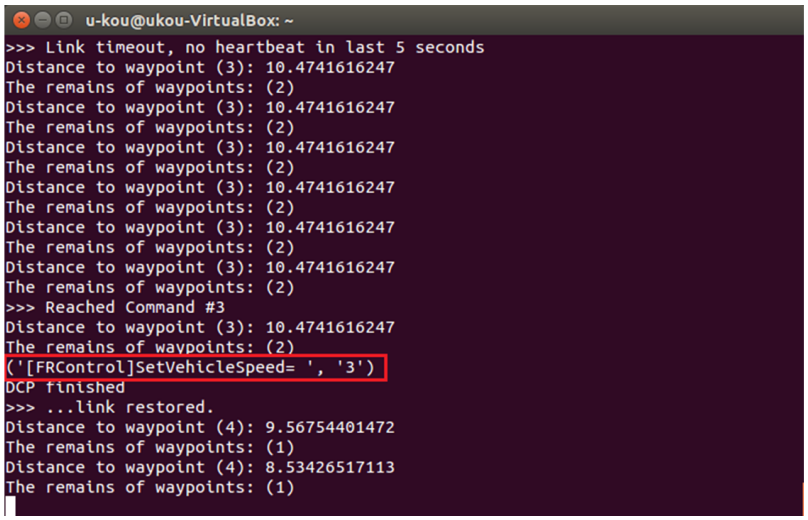

図 22: API の SetVehicleSpeed のコマンドラインにおける出力結果 速度変更後

図 23: API の SetVehicleSpeed の Mission Planner における出力結果 速度変更後

4.4 考察

本研究で使用する移動ルータは安全のためにコントローラの通信範囲内でしか移動できないが、実際の利用形態ではさらに広範囲に移動する必要がある。そこで、コントローラ通信範囲外の任意場所における移動ルータの制御の実現が課題になっている。その制御の意義としては、移動ルータの異常動作による安全問題の回避、およびパスプランの変更などの制

御によるコンテンツ取得時間の短縮が考えられる。そこで、コントローラ通信範囲外にある移動ルータに対する制御は、異常動作を行った移動ルータに対するもの、および正常に動作している移動ルータに対するものの分ける。異常動作の移動ルータに対する制御方法としては、まずはコントローラ通信範囲の外の領域に移動ルータの許可される移動範囲を決定し、その範囲の境界に移動ルータに対して制御命令を転送する制御用移動ルータを設ける。そして、範囲を超えようとする移動ルータに対してホーム帰還などの命令を転送する制御方法が考えられる。一方、コントローラ通信範囲外の正常に動作している移動ルータの場合、想定内のパスプランを行っていることが考えられる。そのような移動ルータに対する制御方法としては、ユーザノードから制御用移動ルータを派遣し、制御対象の移動ルータに追いつき制御命令を転送することで制御する手法が考えられる。

5　おわりに

本稿では最初に ICN の基本動作及びストラテジ層とネーミングスキーマの特徴について紹介した。その特徴によりストラテジ層とネーミングスキーマを使用して移動制御を行うことが可能である。次に、ICN 通信において移動ルータを用いることの意義、その通信モデル及び利点を説明した。ICN ネーミングスキーマを API として用いることの利点は、移動ルータに対して能動的で柔軟な制御が可能であることを示した。そして、移動ルータを用いた分断ネットワーク通信のコンテンツ取得のためのシステムおよび移動戦略などの関連研究の内容および課題を紹介した。それらの研究において移動戦略に基づいた能動的制御の実現が課題になっている。そこで、それらの課題の解決および移動ルータの実用のために、本研究では ICN のネーミングスキーマを用いた移動ルータの能動的制御の実現を目的とする。そして、本稿の 3 章では FR 移動制御システムの提供する API 及びその動作、実装方法を説明した。API の設計ではパスプランの生成、一覧表示、実行、一時停止、キャンセル、移動速度の設定の動作を考案した。また、それらの API の実装のために、移動制御システムのパケットである制御用 Interest と実行結果 Data の仕様の実装方法、および各コンポーネントで使用するクラス、メソッド、変数の実装方法について説明した。次に、本稿の 4 章では実機検証について、最初に使用ツールや OS などの実装環境と実機検証を行う場所などの実験環境について紹介した。次に、各 API 検証方法を述べ、実機検証では各 API の仕様通りの動作を確認した。実機検証により、ICN ネーミングスキーマを FR 移動制御 API の使用が可能であること、および本システムを用いることで FR に対する能動的制御が可能であることを示した。システムの設計及び実験結果に基づいて、考察ではコントローラ通信範囲外の移動ルータに対して制御を行う方法を考察した。本研究の結論としては、ICN ネーミングスキーマの移動制御 API として使用は可能であり、本システムを用いることで FR を能動的に制御できることである。

今後の課題としては、まずは分断ネットワーク通信における FR のパスプランの生成機構を実装することで、一台の FR を用いた分断ネットワーク通信を実現する。次に、FR 移動制御システムを複数台の FR に導入し、複数台の FR による分断ネットワーク通信を実現する。また、分断ネットワーク通信以外の事例に関して FR 移動制御システムの拡張を行う。

謝辞

本報告を終えるにあたり、ご多忙の中論文に対する指摘を頂き、研究に対する態度を改めてくださった村田正幸教授、ならびに論文の添削と助言を頂き、NFD 勉強会を招いてくださった大阪市立大学の阿多信吾教授、中間報告会に論文の章立てのアドバイスをくださった大下裕一助教授、荒川伸一准教授、小南大智助教授、久世尚美特任助教授、出張や普段の事務の手続きを代わりに行って頂いた秘書の寺前香子氏と石井裕子氏に厚く感謝申し上げます。本稿の動機及びアイデアをくたさった参考文献の著者の皆様、それらの文献に出会わせてくださった Google 検索エンジン、本研究の実装及び実験で使用したプログラミング言語、ツール、ドローン、Raspberry Pi などの開発チームメンバに心より感謝いたします。本研究で生かした研究における心構えを教えて頂き、新人輪講の進行及び指導をして頂いた井上昴輝氏、乙倉麻里氏、宮川裕考氏、河島滉太氏、山西広平氏、平野康晴氏、村上雅哉氏、志垣沙衣子氏にお礼申し上げます。研究チームのミーティング及び普段の研究時に論文の誤りの指摘及びアドバイスをくださった先輩としては、ドローンの及び NFD の使い方などの専門知識と研究室の生活を教えて頂き、本稿の添削および実機検証の手伝いをしてくださった北川拓氏、本稿の方針のアドバイスを頂き、いつも研究の相談や助言をしてくださった大岡睦氏、志望理由や論文の添削をしてくださった田中達也氏に深謝いたします。また、普段から情報交換や共同作業を共にしてきた金田純一氏、神田幸大氏、川崎慈英氏、山内雅明氏、経済的及び精神的に支えてくれた家族及び友達がいつもお世話して頂き、誠にありがとうございました。

参考文献

[1] E. Baccelli, C. Mehlis, O. Hahm, T. C. Schmidt, and M. Wählisch, "Information centric networking in the IoT: Experiments with NDN in the wild," in *Proceedings of the 1st ACM Conference on Information-Centric Networking*, Sep. 2014, pp. 77–86.

[2] W. Shang, A. Bannis, and T. Liang, "Named data networking of things," in *Proceedings of 2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, Apr. 2016, pp. 117–128.

[3] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, and R. L. Braynard, "VoCCN: Voice-over Content-Centric Networks," in *Proceedings of the 2009 Workshop on Re-architecting the Internet*, ser. ReArch '09, Dec. 2009, pp. 1–6.

[4] B. Li, D. Huang, Z. Wang, and Y. Zhu, "Attribute-based access control for icn naming scheme," *IEEE Transactions on Dependable and Secure Computing*, vol. PP, no. 99, pp. 1–1, Apr. 2016.

[5] R. Sugihara and R. K. Gupta, "Path planning of data mules in sensor networks," *ACM Trans. Sen. Netw.*, vol. 8, no. 1, pp. 1:1–1:27, Aug. 2011.

[6] J. S. Liu, S. Y. Wu, and K. M. Chiu, "Path planning of a data mule in wireless sensor network using an improved implementation of clustering-based genetic algorithm," in *Proceedings of 2013 IEEE Symposium on Computational Intelligence in Control and Automation (CICA)*, April 2013, pp. 30–37.

[7] T. Kitagawa, S. Ata, and M. Murata, "Retrieving information with autonomously-flying routers in information-centric network," in *Proceedings of 2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–6.

[8] T. Kitagawa, "Mobility-controlled flying routers for information centric networking," Master's thesis, Osaka University, Feb. 2017.

[9] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, Dec. 2009, pp. 1–12.

[10] L. Zhang, D. Estrin, and J. Burke, "Named data networking (NDN) project," PARC, Tech. Rep. NDN-0001, Oct. 2010. [Online]. Available: http://named-data.net/wp-content/uploads/TR001ndn-proj.pdf

[11] NDN Project, "ndn-cxx: NDN C++ library with eXperimental eXtensions," http://named-data.net/doc/ndn-cxx/current/, accessed: 2016-11-01.

[12] ——, "NFD - Named Data Networking Forwarding Daemon," http://named-data.net/doc/NFD/current/, accessed: 2016-11-01.

[13] G. Tyson, N. Sastry, I. Rimac, R. Cuevas, and A. Mauthe, "A survey of mobility in information-centric networks: Challenges and research directions," in *Proceedings of the 1st ACM Workshop on Emerging Name-Oriented Mobile Networking Design - Architecture, Algorithms, and Applications*, Jun. 2012, pp. 1–6.

[14] H. Farahat and H. Hassanein, "On the design and evaluation of producer mobility management schemes in named data networks," in *Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, Nov. 2015, pp. 171–178.

[15] M. A. Yaqub, S. H. Ahmed, S. H. Bouk, and D. Kim, "Interest forwarding in vehicular information centric networks: A survey," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, Apr. 2016, pp. 724–729.

[16] T. Wei, L. Chang, B. Yu, and J. Pan, "MPCS: A mobility/popularity-based caching strategy for information-centric networks," in *Proceedings of Global Communications Conference*, Dec. 2014, pp. 29–34.