2022 年度　修士論文

ICN サービスメッシュのデータフローの可視化手法の検討

早稲田大学 基幹理工学研究科 情報理工・情報通信専攻

5121F006-5

飯塚　真梧

指導　甲藤二郎 教授
研究指導名　画像情報研究

2023 年 1 月 23 日

指導教授印	受付印

目次

第 1 章 序論

1.1 研究背景

近年，クラウドコンピューティングの発展は著しく，企業や個人を問わず多くの利用者がいる[1]．クラウド環境において，単にクラウドを利用するだけではなく，クラウドの長所を生かし，回復性，管理力，および可観測性のある疎結合システムを実現させるクラウドネイティブという概念が CNCF[2]によって定義されている．クラウドネイティブ技術の 1 つにマイクロサービスが挙げられる．これは，システムを小さいサービス単位に分割して構成することで開発を容易にし，リソースを効率的に利用することができる．しかし，システムをマイクロサービス化すると，従来のモノリシック・アーキテクチャを比較して，サービス間の通信が煩雑となり，管理が難しくなる．そのような環境では，サービス間の通信を可視化し，システムの管理者や利用者が，通信環境を正確かつ容易に把握することが重要であるといえる．一方，近年のネットワーク環境の質的及び量的変化によって，トラヒックの増加も重要課題の 1 つである．そこで，膨大なコンテンツ送受に対応できる従来とは異なる新たなネットワークアーキテクチャの構築することが期待されている．

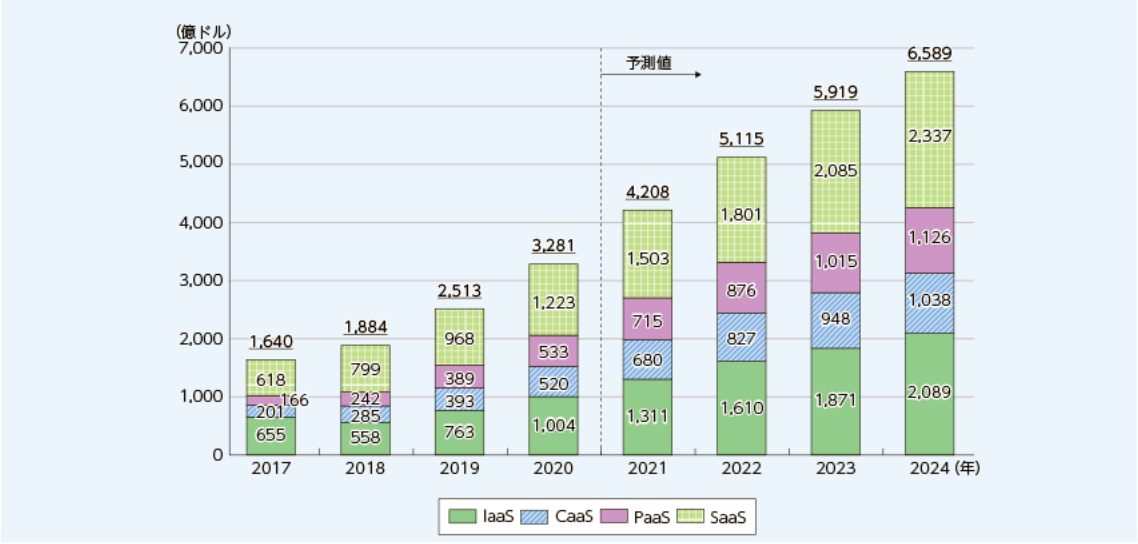


図 1.1.1 世界のパブリッククラウドサービス市場規模（売上高）の推移及び予測[1]

1.2 研究目的

本研究は，マイクロサービス化されたシステムの通信状況を可視化することを目的としている．本研究では，各システムの実装自体を変更することなく，容易に通信ログを取得できるアーキテクチャを構築し，そのログデータを可能な限りリアルタイムで可視化する．可視化の手段として，オープンソースのデータ可視化ツールである Grafana を使用する．また，サービス間の通信手段として，ICN/CCN という新たなネットワークを使用し，その特徴的なデータフローを可視化する．

1.3 本文の構成

第 1 章では本研究の目的を述べた．
第 2 章では関連技術について述べる．
第 3 章では関連研究について述べる．
第 4 章では情報指向サービスメッシュについて述べる．
第 5 章では評価実験について述べる．
第 6 章では総括を述べる．

第 2 章 関連技術

2.1 クラウドネイティブ

CNCF によって定義されるクラウドコンピューティングの利点の活用を前提として設計されたシステム及び技術である[2]．回復性、管理力、および可観測性のある疎結合システムの実現，及びそれらと堅牢な自動化の組み合わせによって，最小限の労力で，頻繁かつ予測どおりにシステムを構成できるようにすることを目的とする．

2.1.1 マイクロサービス

マイクロサービスは，サービスの役割ごとに小さな単位に分割し，それらを組み合わせることで大きなサービスを構成するアーキテクチャである．利点としては，各サービスは独立し，疎結合のため，サービスの変更や追加が容易であり拡張性に優れることや，サービスごとにスケーリングや負荷分散を容易に行えることが挙げられる[3]．一方，サービス間のつながりと構成が複雑になることで，サービス間の接続状態の管理が難しくなる．また障害が発生した場合，サービス全体の構成が複雑であるため，その障害の影響が拡がり易く，原因を特定するのが困難である．

2.1.2 サービスメッシュ

サービスメッシュは，マイクロサービスにおける上述の課題を解決するために考案された機能群である．サービスメッシュの代表的構成は，各サービスをサービス機能と通信機能（サイドカープロキシ）に分離し，サービス間の通信をすべてプロキシ経由で行うことで管理する．それによって，サイドカーがサービスの状態や，サービス間のネットワーク状況を監視し，把握することができる．また，サービスと通信部分の組み合わせを変更することも容易になる．サービスメッシュソフトウェアとして最も有名なものとして Istio[4]が挙げられる．Istio のアーキテクチャは，データプレーンとコントロールプレーンに分離される．データプレーンではサイドカープロキシによってサービス間通信を管理し，コントロールプレーンではプロキシによるトラヒックのルーティング設定を管理する．

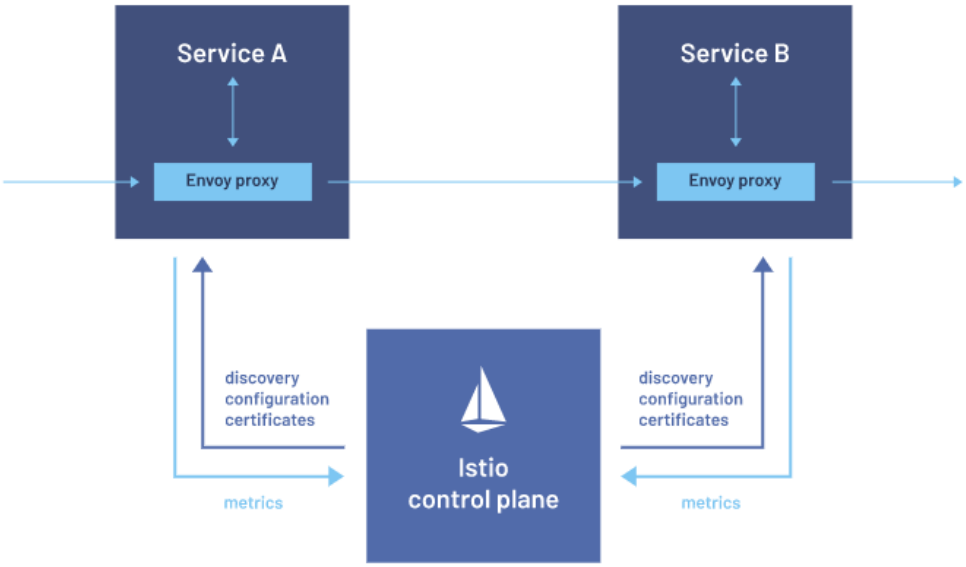


図 2.1.1 Istio アーキテクチャ[4]

2.1.3 サービスチェイニング

ネットワーク仮想化技術の 1 つとしてサービスチェイニングが挙げられる．サービスチェイニングは，パケット制御などのネットワークサービスをサーバ内のソフトウェアで行う技術である[5][6]．同様の目的を持つ概念として NFV(Network Function Virtualization)がある．NFV はネットワーク機能をサーバ上のソフトウェアで実装することを指すが，サービスチェイニングは NFV をサーバ内あるいはサーバ間で連結し，目的を達成することを目指す．

2.2 オブザーバビリティ

可観測性(Observability)は，一連のサービスが想定通りの挙動を示しているかどうかを監視し，外部出力によって得た情報からシステム内部を推定する技術・能力を指す[7]．また，好ましくない挙動を観測した場合にその要因を特定する能力・技術とも言い換えられる．これは，サービスの管理・運用を目的とする．システムの監視（モニタリング）は，従来のモノシリックアーキテクチャにおいても重要であり，新たに提案された概念ではない．しかし，オブザーバビリティはモニタリングとは異なる概念である．マイクロサービスアプリケーションにおいては，サービス間の配置や通信が複雑化している．そのため，あるサービスの障害の根本的要因が他のサービスの状態変化に起因することが頻繁に生じ得る．つまり，単純なモニタリングでは障害の検出やトラブルシューティングが困難である，このことから，マイクロサービスアプリケーションにおいて障害あるいはその予兆を迅速に特定する技術は重要である．

2.2.1 活用データ

オブザーバビリティを実現するために収集すべきデータのとして，メトリクス，ログ，トレースの 3 つが挙げられる[7]．

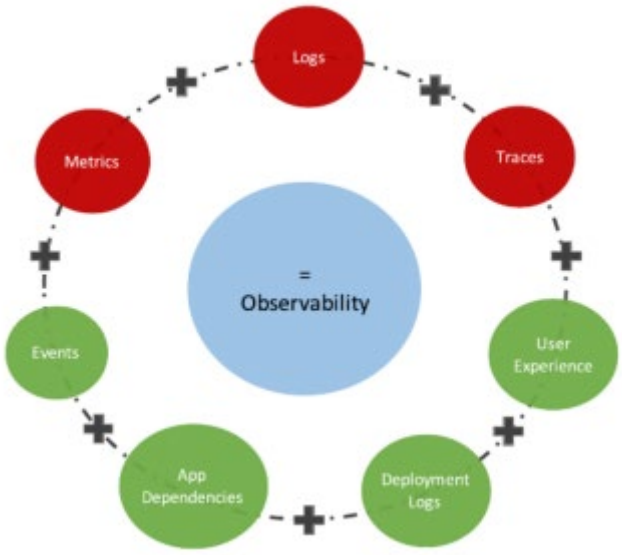


図 2.1.1.1 オブザーバビリティの三本柱（赤）とその他の重要な観測データ[7]

2.2.1.1 メトリクス

メトリクスとは，ある一定期間システムの測定値の集合である．名前，タイムスタンプに加えて，アプリケーション固有で定義された値（インスタンス数，平均応答時間，処理されるリクエスト数）やシステムレベルで定義された値（CPU 使用率，メモリ使用率）等が挙げられる．メトリクスは閾値を設定して，アラートを発生させる等で使用される．

2.2.1.2 ログ

ログとは，アプリケーション内のイベントの集合である，言語やフレームワークに関わらずログ機能は実装されているため，取得することは容易である．何か障害やエラーが発生した際，そのログを分析することで状況把握に役立つ．一方，ログはイベントが羅列されているだけのため，サービス間の関連性を明らかにするには労力を要する．

2.2.1.3 トレース

トレースとは，コンポーネント間のリクエストやデータの行き来等，各コンポーネントの通過する情報を横断的に把握することを目的とする概念である．メトリクスやログは各システムのパフォーマンスを把握するための情報を収集するが，トレースは複数のシステム間の関連性を把握するために利用される．マイクロサービス環境において，システムに障害が起こった際にはトレースデータを分析することで，そのシステム全体の中でボトルネックとなっている箇所を推定することに使用できる．

2.2.2 分散トレーシング

マイクロサービスにおけるトレーシングの手法として，分散トレーシングという概念がある．分散トレーシングでは，各サービスを通過するリクエストの動きを追跡することでシステム全体の状態を監視することを目的とする．分散トレーシングは，主にスパンとトレースという単位により構成される．スパンはシステム内での最小の処理を指し，トレースはスパンの集合である．カスタムヘッダーに ID を挿入し，コレクターによって収集することでシステム全体の可視化を可能にしている．分散トレーシングツールの例としては Jaeger[8]や Zipkin[9]が挙げられる．

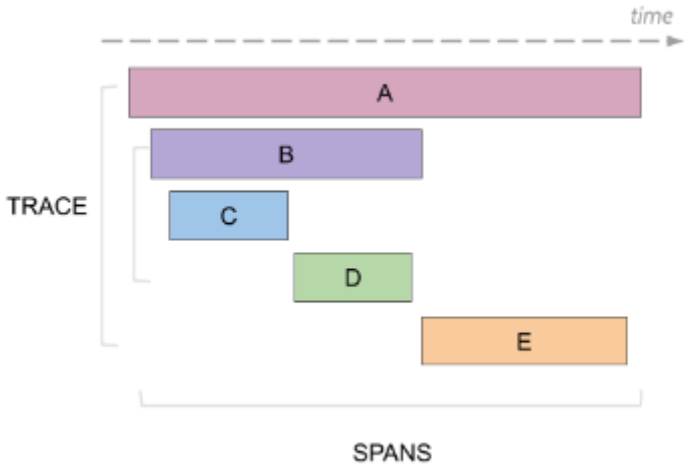


図 2.2.2.1 トレースとスパンの関係性[8]

2.2.3 標準化

上述のオブザーバビリティの三本柱の収集を標準化する取り組みとして OpenTelemetry[10]がある．従来はネットワーク監視のプロトコルとして SNMP(Simple Network Management Protocol)が用いられてきた．しかし，SNMP は request/response 型のデータ送信である．そのため，ログコレクターが収集する項目（CPU 利用率等）のデータを監視対象のデバイスに要求し，それに対しデバイスが応答を返すという仕組みである．そのため，要求と応答情報収集のための処理時間や負荷が増大する点や，ログ収集間隔が長くなってしまい，リアルタイム性が損なわれる点が指摘されてきた[11].また，多数のデバイス（あるいはサービス）に対して，ログ収集を行うには適していない．そこで，マイクロサービス環境に適用できる次世代の監視技術として，Telemetry という概念が提案された．Telemetry は Pub/Sub モデルが採用されており，非同期かつ疎結合でデータを収集できる．そのため，ログ収集の負荷を軽減し，多数のマイクロサービスに対して，短い間隔でログを収集できるようになる[12]．続いて，Telemetry を収集するための方法を標準化する取り組みとして OpenTelemetry が提案された．図 2.2,3,1 に OpenTelemetry の構成を示す．これは，ベンダーに依存せずにオブザーバビリティに必要なデータをバックエンドに送信する方法を標準化することを目的とするプロジェクトである．図 2.2.3.2 にマイクロサービスで構成されたサンプルアプリケーション[13]の例を示す.マイクロサービスでは，多くの場合，言語や実装が異なるサービスの集合によって構成される．そのため，それらに依存せずに，メトリクス，トレースやログ等の情報を収集することは重要である．

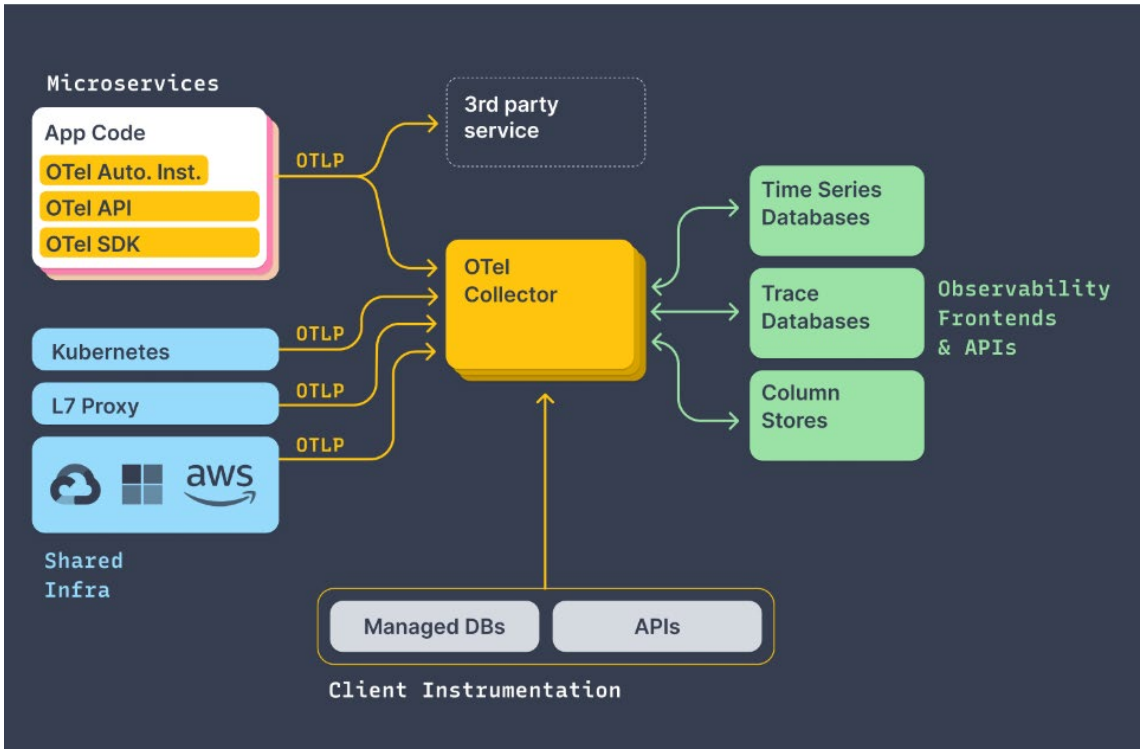


図 2.2.3.1 OpenTelemetry 構成[10]

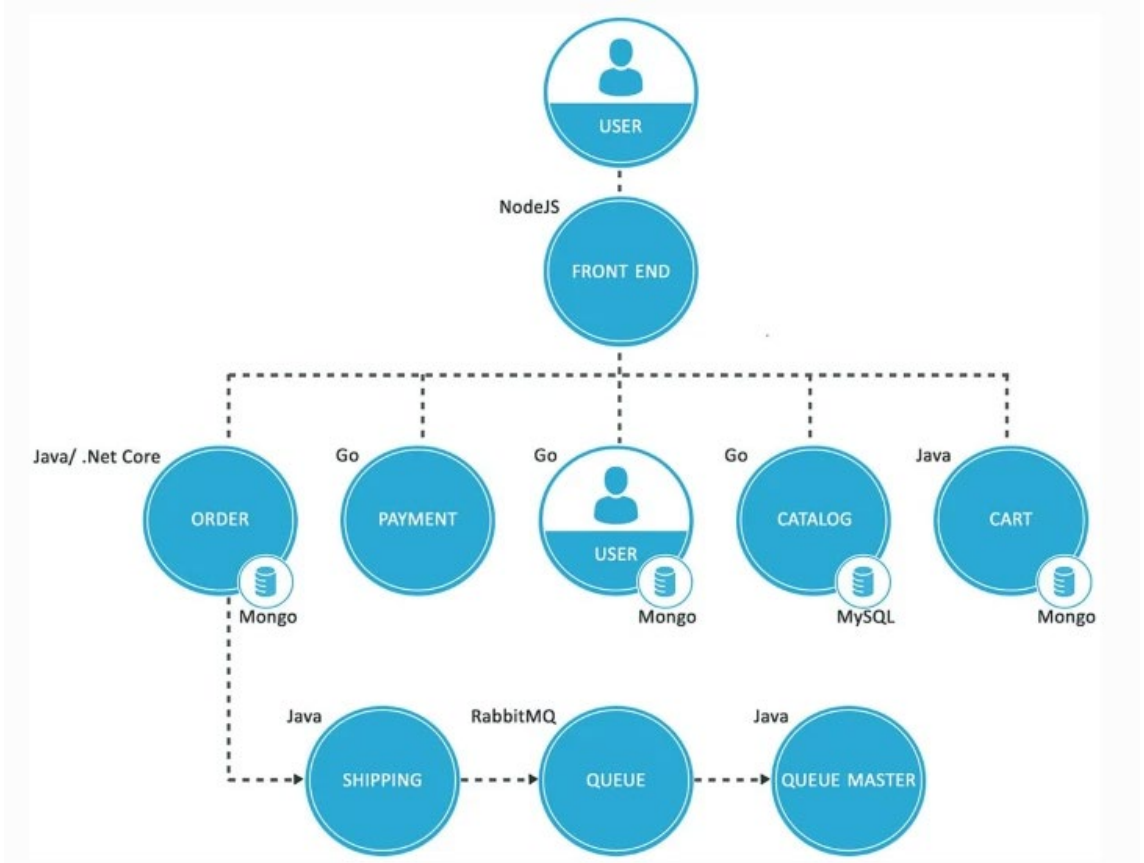


図 2.2.3.2 マイクロサービス（sockshop demo）の構成の例[14]

2.3　ICN/CCN/NDN

従来のネットワークはサーバ間でデータを送受することで情報にアクセスする構造であった．しかし，近年は動画や音楽等の配信コンテンツの普及や，センサノードからの情報収集等を行うことができる IoT の発展によって，ネットワークの利用方法は大きく変化している．そこで新たな利用環境に即したネットワークアーキテクチャとして，情報（コンテンツ）中心のアーキテクチャが研究されている．このアーキテクチャは情報指向ネットワーク（Information Centric Networking）と呼ばれる[15].

2.3.1 概要

ICN の基本概念は，「通信サービスが求めるものはサーバへのアクセスではなく，情報の取得にほかならない」という考えのもと，情報に名前を付与し，その情報を欲するユーザは，その名前のみで求める情報を手に入れられるようにすることである[16]. 従来のネットワークでは，コンテンツがどこにあるかという場所（IP アドレス）の情報を得たうえで，その場所のアドレスを基準にして，情報にアクセスしていた．一方，ICN では，ユーザが情報提供される場所ではなく，名前付きコンテンツを指定し，要求をキャストして情報にアクセスすることで，上述の従来のネットワークにおける制約を排除できる[16].

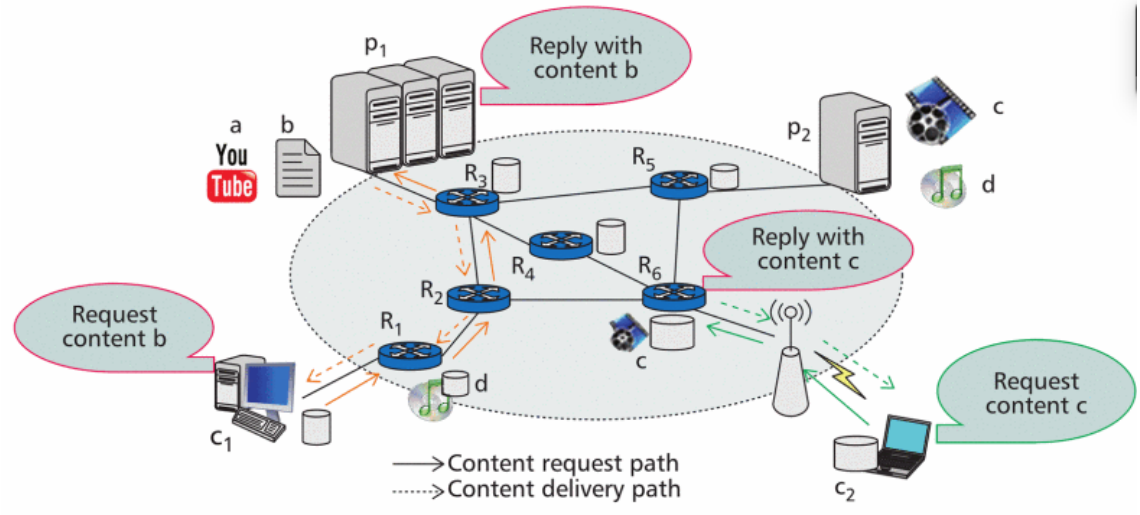


図 2.3.1.1 ICN における基本的なコンテンツ送受[17]

ICN と同様のテーマをもつ CCN(Content-Centric Networking), NDN(NamedData Networking)も存在する[18]が，いずれも ICN の代表的な実装の 1 つである.これ以外にも ICN の概念を利用した実装はあるが，CCN/NDN が代表的な実装であるため，本論文では ICN と表記した場合，CCN/NDN の実装を指すこととする.

2.3.2 基本技術

ICN においてユーザが求める情報を取得するまでの通信の過程と，実現のための基本技術について紹介する.

2.3.2.1 送受信方法

ICN ではコンテンツ名を情報識別子として利用することでデータを送受信する.
まずユーザは Interest と呼ばれるデータ要求パケットをブロードキャスト送信する.

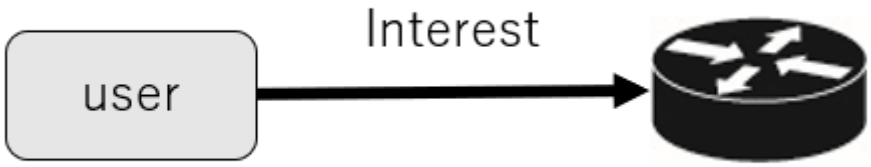
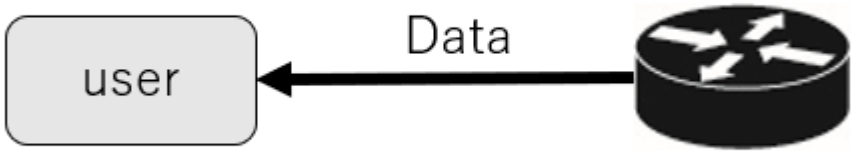


図 2.3.2.1.1 Interest 送信

続いて，Interest を受信したルータは要求されているデータが自身にキャッシュされているかどうかを確認する．もし，キャッシュされている場合は，そのデータをユーザに送り返す.

図



2.3.2.1.2 Data 送信

もし自身のキャッシュに要求されているデータが存在しない場合は，上流のルータもしくは近隣のルータに受信した Interest を再転送する．

図 2.3.2.1.3 Interest 転送

同様の動作を繰り返し，要求されているデータをキャッシュしているルータまで Interest を送信し，到着した際は Interest が送信された経路を逆方向に Data を転送することで目的を達成する．

図 2.3.2.1.4 Data 転送

2.3.2.2 ルータ構成要素

上記のデータ送受の工程において，各ルータは Interest/Data の転送先を決めるテーブルとして FIB(Forwarding Information Base)と PIT(Pending Interest Table)を持つ．FIB は経路制御表の役割を果たす．受信した Interest がもつコンテンツ名に応じて次の転送先が記述されている．PIT は Data 転送を行うために Interest を受信した経路を一時的に保存する．ほかにルータには CS(Content Store)があり，これはキャッシュ領域である．各ルータは以上の 3 つの構成要素からなる．

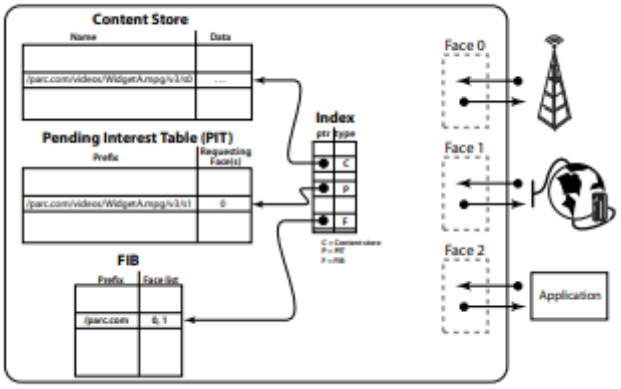


図 2.3.2.2.1 ルータ転送機構[18]

中間ノードにおける FIB と PIT の例を図 2.3.2.2.2 に示す．

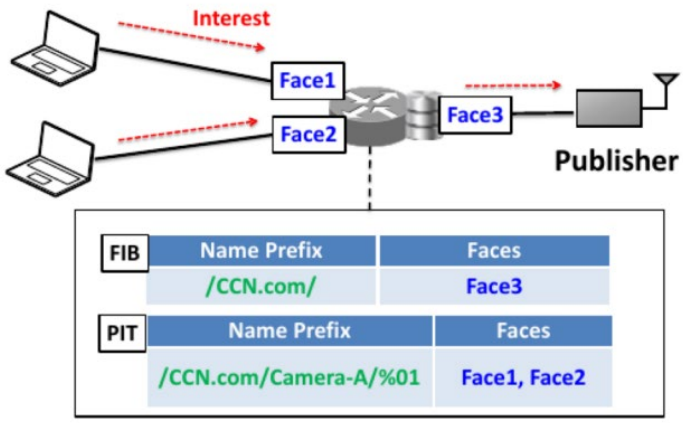


図 2.3.2.2.2　FIB と PIT の内容[19]

2.3.2.3 ネーミング

ICN においては，要求するデータを識別するためにネーミングを行う．各ルータはキャッシュされているかどうかを正しく判断するために，ネーミングは唯一の識別子となる必要がある．ネーミングの例として図 2.3.2.3.1 のように定義されている．

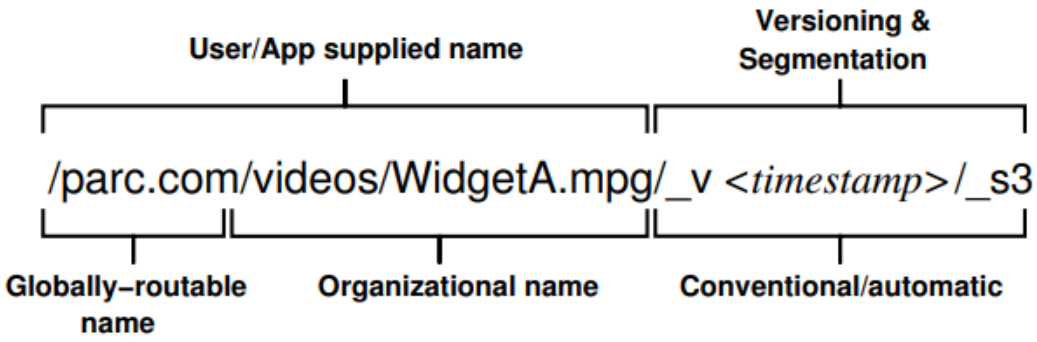


図 2.3.2.3.1 ネーミングの例[18]

第 3 章 関連研究

3.1 オブザーバビリティ実現のためのアーキテクチャ

3.1.1 概要

マイクロサービス環境における可視化を実現するためのフレームワークとして，以下の 4 つの役割を担う論理的なレイヤーが必要である[20].

①データの収集及び検索

②データの保存

③データの処理及び相関関係

④可視化とアラート

上記の各過程について様々な研究がなされている.

例えば[20]では図 3.1.1 のようなオブザーバビリティアーキテクチャが提案されている.

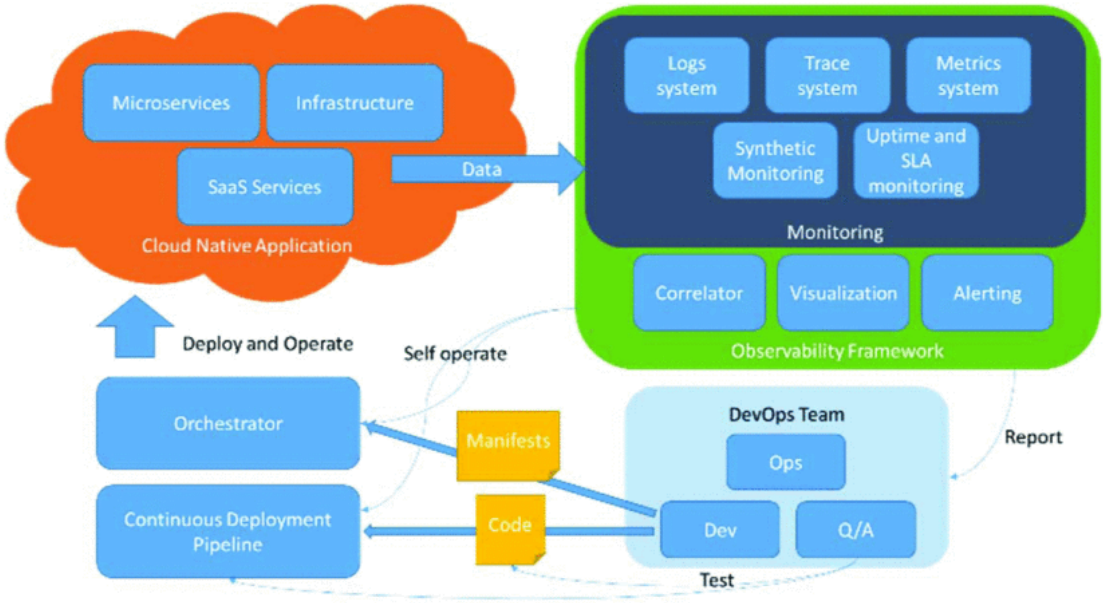


図 3.1.1 マイクロサービスとオブザーバビリティフレームワークの相互関係[20]

また，このフレームワークによって得られるデータによって有効になる機能として，以下の 4 点が挙げられている．

①カスタムヘルスチェック

②自動スケーリング（図 3.1.2）

③異常検出アラート

④新たなメトリック生成

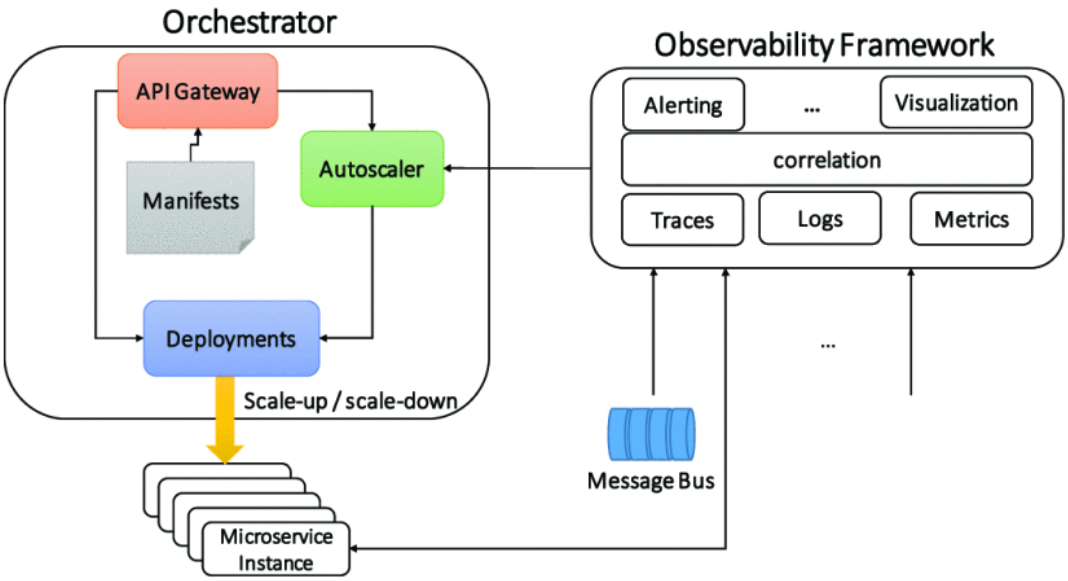


図 3.1.2 自動スケーリングオーケストレーターの構成[20]

3.1.2 データ収集・保存

オブザーバビリティのためのデータを収集するためには，各サービスにデータ収集のためのライブラリを事前に挿入する必要がある．しかし，それぞれのマイクロサービスは環境や言語が異なり，さらにサービスの変更が頻繁に行われる状況を考慮すると，事前にライブラリを挿入すると，非常に複雑な構成を要し，アプリケーションの管理や拡張を難しくする．データ収集のための実装方法は①手動コーディング②フレームワークの導入③動的バイナリ計装の 3 つの方法に分類される[21]．各方法におけるメリットとデメリットを表 3.1.2.1.1 に示す．

表 3.1.2.1.1　実装方法のメリットとデメリット[21]

	メリット	デメリット
手動コーディング	実装が容易	複雑さを増長させる
フレームワーク	一部の実装でよい	オーバーヘッドが生じる
動的バイナリ計装	追加実装が不要	言語に依存

一方，各サービスからログを収集する仕組みとしては，各サービスにエージェントを置き，そのエージェントがログをコレクターにプッシュする方法と，サービス自身がログを生成し直接コレクターにプッシュする方法がある．

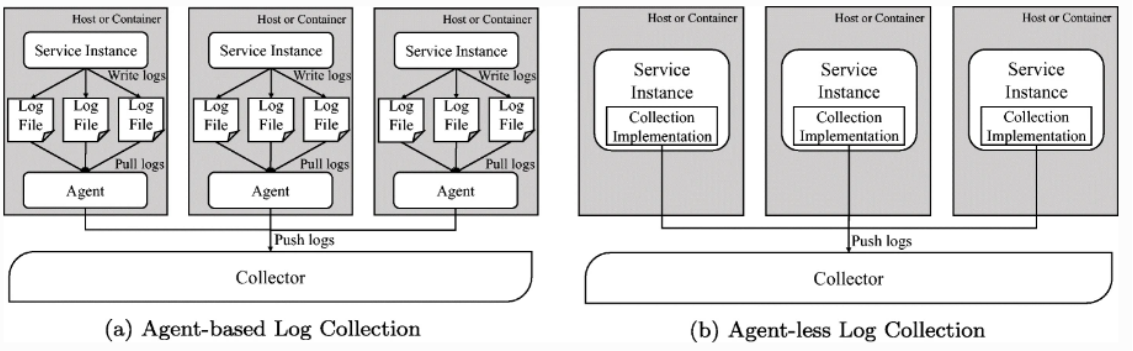


図 3.1.2.1.1 ログ収集アーキテクチャの種類[21]

[22]では，リクエストからメタデータを抽出するコンポーネントをサイドカー内に挿入する手法が提案されている．サイドカーを用いるサービスメッシュでは，サービスと通信部が分離されている．そのため，このような構造にすることによって，サービス内部には変更を加えることがなく，実装できる．

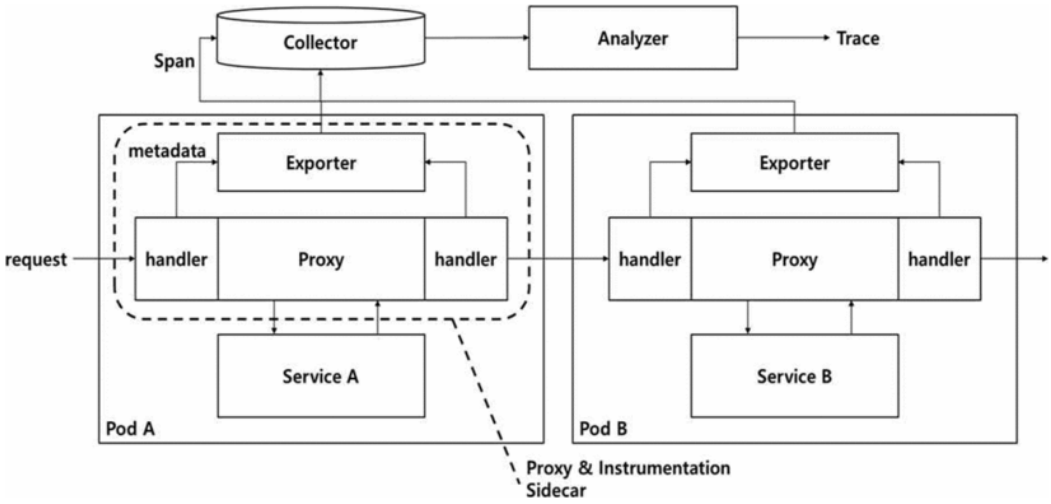


図 3.1.3 提案構造[22]

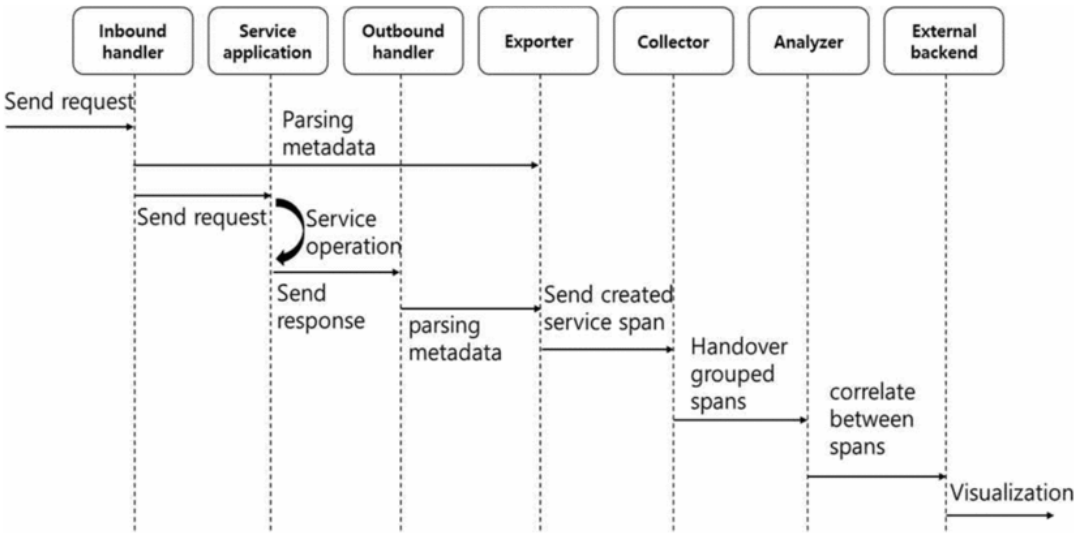


図 3.1.4 トレースのフロー[22]

一方，サイドカー内で得られる情報は，サービスネットワークに対応する箇所からトレースを行うため，トレース範囲に違いが生じる．サービス内部にライブラリを挿入する既存のトレーシングでは，システム内の観測が可能であるが，提案手法[22]では，サービス間の通信の観測に限定される．既存の分散トレーシングとの違いは表 3.1.1 に示す．

表 3.1.1 ライブラリベースと提案手法の比較[22]

	Built-in library-based tracing system	**Proposed Architecture**
Trace scope	Service code level	Service flow level
Pre-work internal service	Required	Unnecessary
Scalability	Limited	Unlimited
Impact of service change	Affected	Non-affected
Error detection	Function level	Service level

Viperprobe [23]では，オブザーバビリティデータの収集の際のオーバーヘッドに関して言及されている．

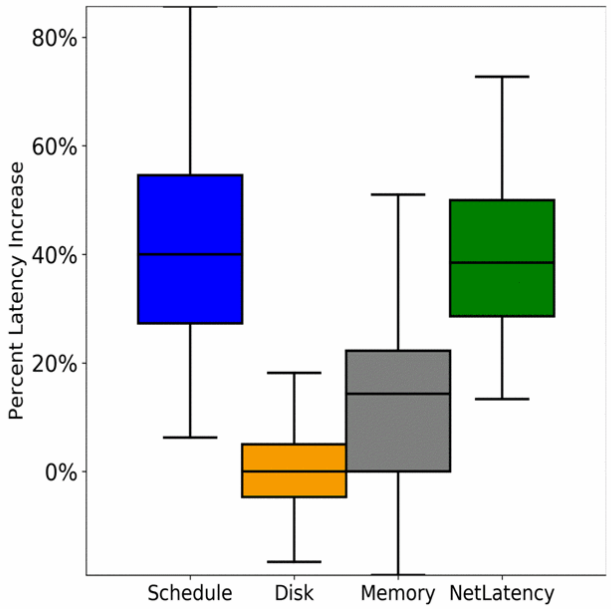


図 3.1.3 収集するメトリクスのオーバーヘッド[23]

図 3.1.3 に収集するメトリクスによってオーバーヘッドが異なることが示されている．そのため，常時収集する必要がないメトリクスを判別するアルゴリズムをオフライン上で構築し，重要メトリクスを判別することでオーバーヘッドを削減している．

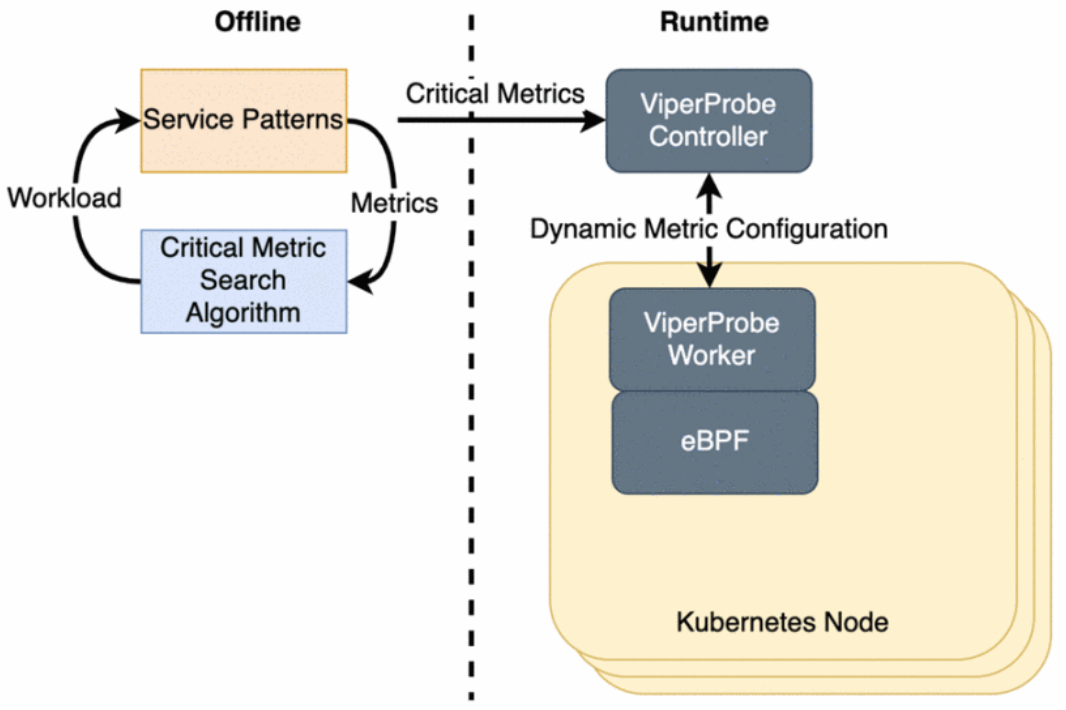


図 3.1.4 Viperprobe[23]

上述のようにデータを効率的に収集する研究は盛んに行われている一方で，通信状況を把握するために，収集したデータをどのように可視化するのがよいのかという観点での研究はあまりなされていない．

3.2 Cefore

ICN/CCN ソフトウェアプラットフォームの 1 つとして情報通信研究機構（NICT）が開発している Cefore[24][25]がある．V. Jacobson らによって提唱された論文[18]から，開発された CCNx に準拠している．特徴としては，「基本機能と拡張機能の分離」と「拡張機能開発の容易性」が挙げられている[24].

3.2.1 基本構成

Cefore の概要を以下の表 3.2.1.1 に示す

表 3.2.1.1 Cefore 概要

言語	C
OS	Linux, MacOS, Raspbian, Android

Cefore の実装は C 言語を用いているが，Cefore アプリ開発用に cefpyco という python パッケージも付属しており，python でも記述することができる．

図 3.2.1.1 に Cefore の機能構成を示す.

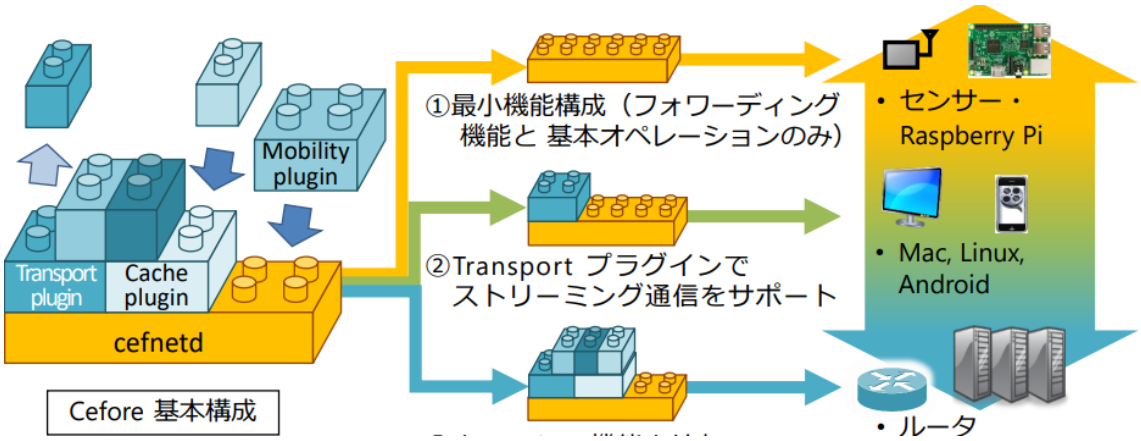


図 3.2.1.1 Cefore 構成[25]

Cefore は cefnetd というシンプルなフォワーディングデーモンと csmgrd というキャッシュ機能に分離されている.

3.2.2 cefnetd

cefnetd はフォワーディングデーモンであり，FIB と PIT を利用したパケット転送をサポートする．基本的には全ノードで稼働する土台となすシステムである．プラグインインターフェイスをを持つため，cefnetd の実装自体を変更することなく経路制御等のプラグインを開発することができる．

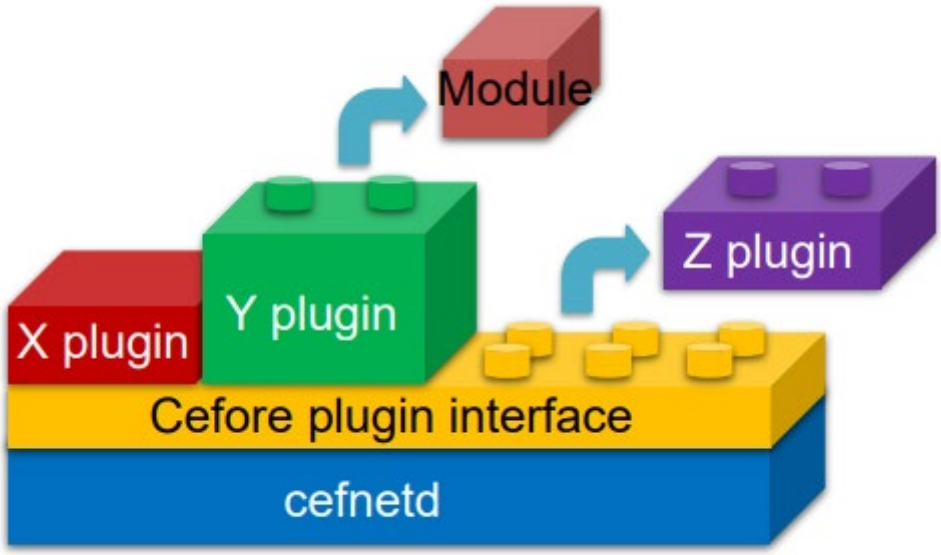


図 3.2.2.1 cefnetd 構成[26]

3.2.3 csmgrd

csmgrd は高負荷なキャッシュ機能である．cefnetd と csmgrd はローカルソケットもしくは TCP で接続される．

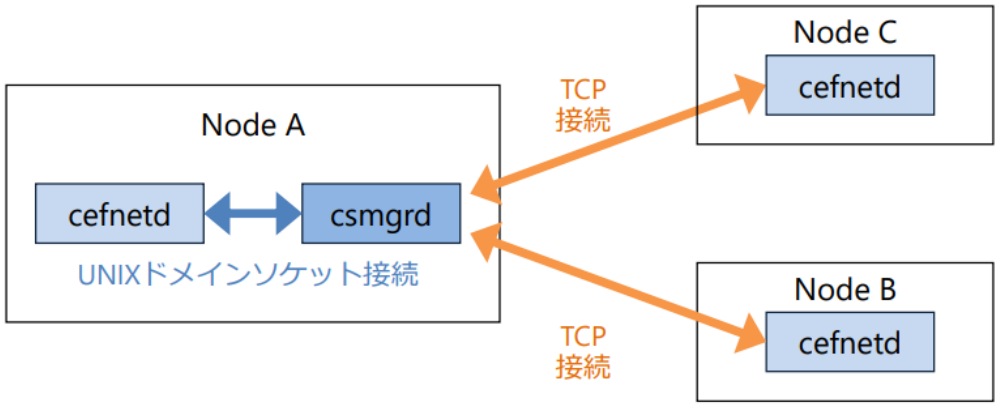


図 3.2.2.2 cefnetd と csmgrd の接続[26]

3.2.4 ヘッダー

Cefore に用いられているパケットは CCNx-1.0 のパケットフォーマットに準拠している [27].

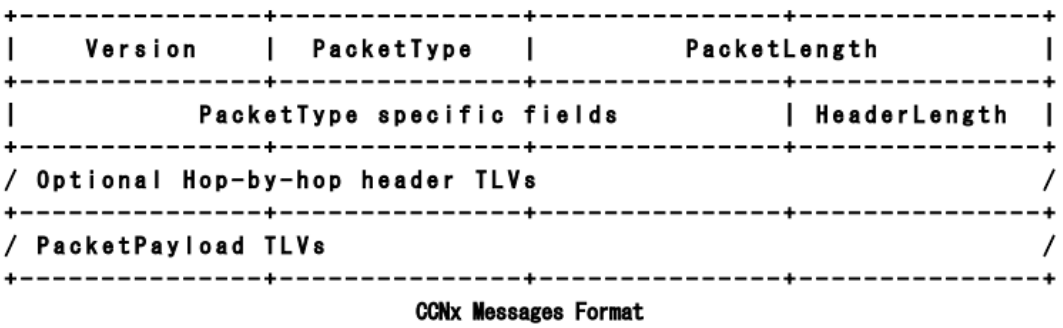


図 3.2.3.1 パケットフォーマット[27]

3.3 SMI(Symbolic Interest)

3.3.1 概要

ICN におけるデータ要求パケットは Interest と呼ばれる．Interest でデータの一意の名前を指定し，通信を開始することになるが，大きなデータを送信する場合はデータを一定容量で分割し，シーケンス番号を割り当てて送信することになる．つまり，その分 Interest も複数送信しなければならない．Interest を複数送信することによって，オーバーヘッドが生じることが考えられる．

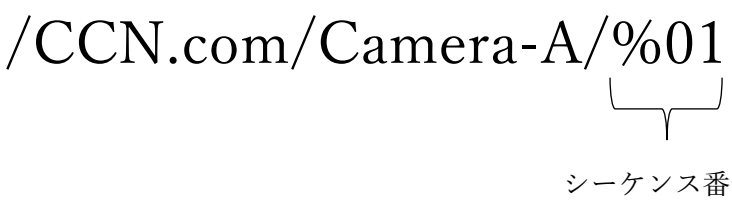


図 3.3.1 シーケンス番号の例

Interest 送信とデータ転送を効率的に行う研究として Pipeline Interest[28]，Any Next Packet[29]や Symbolic Interest(SMI)[25]等がある．Cefore では SMI が実装されているため，SMI の概要について紹介する．SMI に対して通常の Interest を RGI(Regular Interest)と呼ぶ．

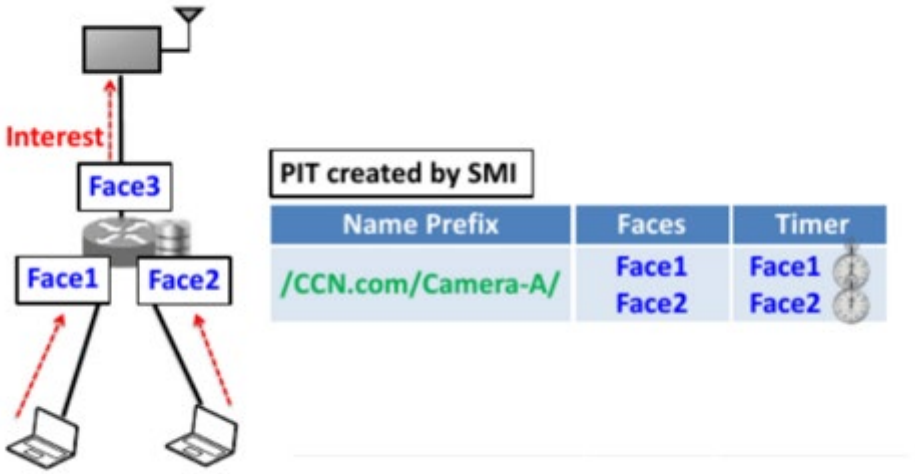


図 3.3.2 SMI 概要[19]

SMI では，シーケンス番号を省略することで Interest 送信の削減を行う概念である．RGI では 1 つの Interest に対して，1 つの対応したデータパケットを送信する．しかし，SMI では，Interest を一度受信すると一定期間 PIT を保持する（図 3.3.2）．PIT を保持している間に同一名の Interest を受信した場合，そのまま継続的に同じ経路を参照して転送する．また，Interest ヘッダーの中にフラグを挿入することで，RGI と SMI を区別する．一方，SMI を継続再送信することによって，データサイズに依存せずに通信状態を維持することができるため，ストリーミング送受信にも利点がある．

3.3.2 性能評価

[19]では，映像データを低遅延で取得するシミュレーションを行っている．性能指標として IPR(Interest Packet Rate)，DRD(Data Reception Delay)，PITnum(PIT エントリー数)を用いている．

IPR は中間ノードにおける Interest トラフィックのオーバーヘッドを指す．DRD はパブリッシャーのデータ送信時間と，コンシューマーのデータ取得時間の差である，PITnum は，中間ノードでの転送時間の指標とする．実験のトポロジーを図 3.3.2.1 に示す．
実験シナリオとして，コンシューマーの総数を 24,48,72 と変化させている．シミュレータ結果を図 3.3.2.2～図 3.2.2.3 に示す．

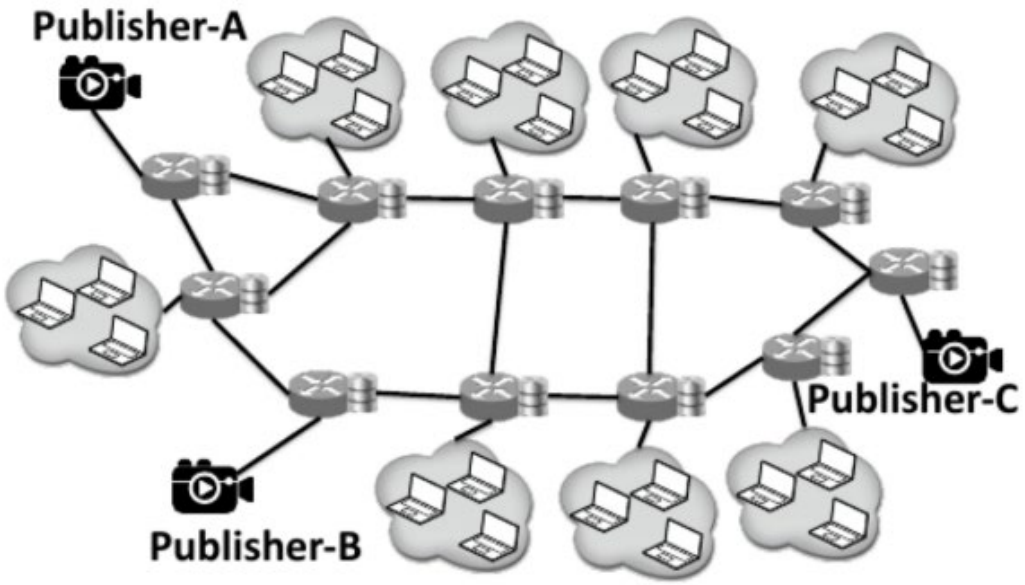


図 3.3.2.1 実験トポロジー[19]

実験結果から，SMI は IPR と DRD において高い性能が示すことがわかる．一方，SMI の注意点として，コンシューマー側での受信レート制御ができないため，輻輳によって性能が低下する可能性を指摘している．

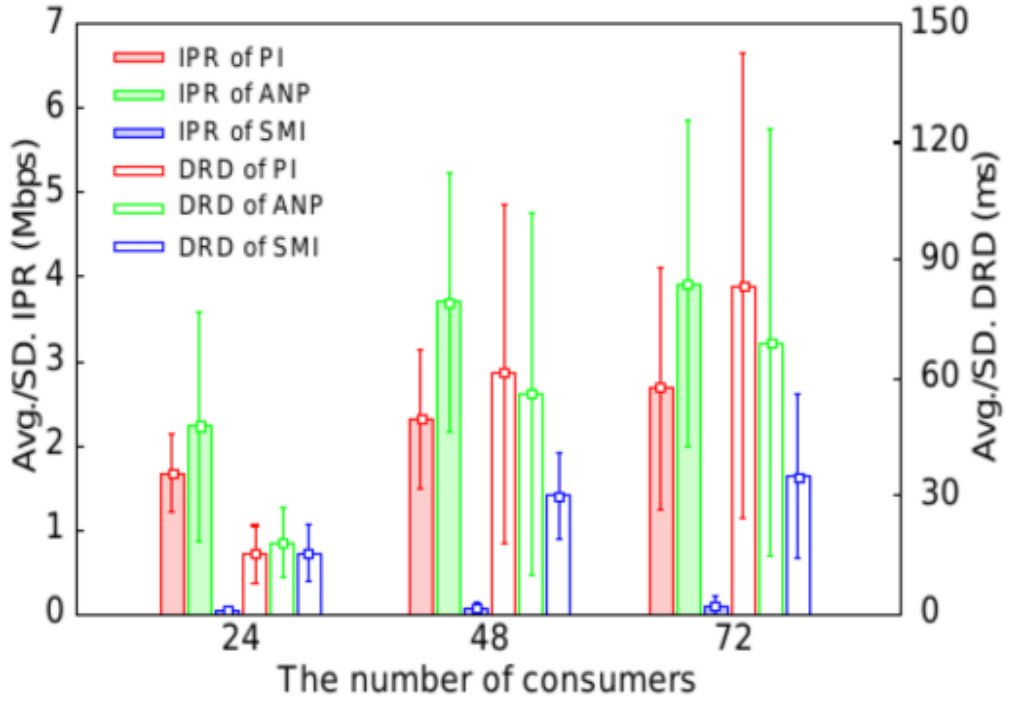


図 3.3.2.2 実験結果[19]

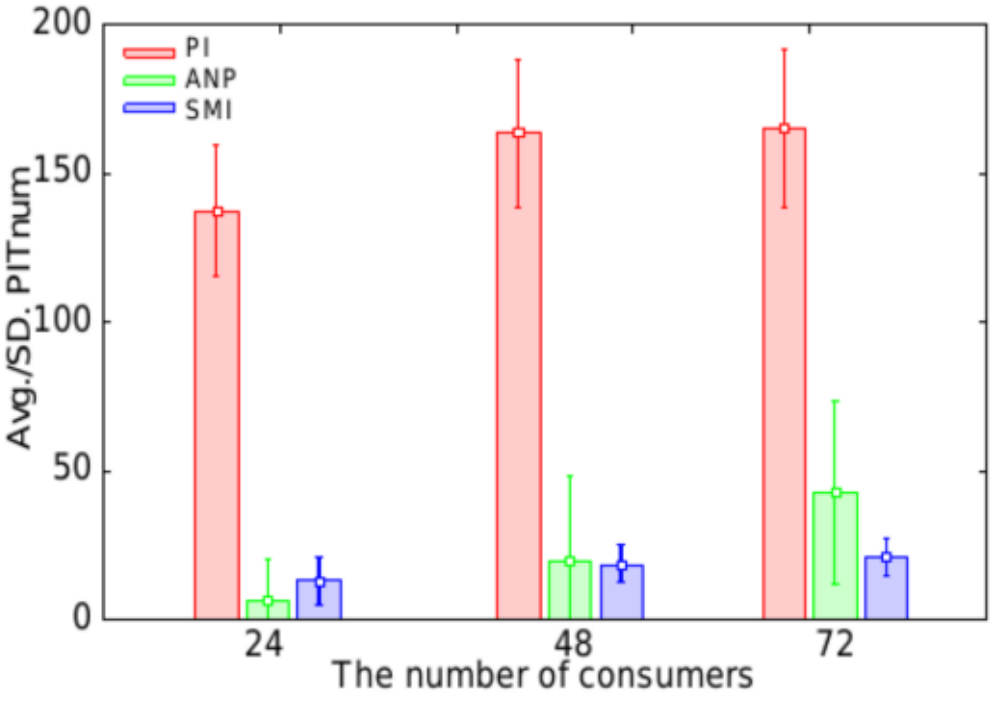


図 3.3.2.3 実験結果[19]

3.4 ICN の応用

エッジコンピューティングの発展及びコンピューティング資源の分散化に伴い，ユーザが求める処理機能を，分散配置された資源を利用して提供する考え方が広まっている．2.3節で説明した通り，従来 ICN の概念は，コンテンツ取得のために考案されたものである．しかし，上述したネットワーク技術及びコンピューティング技術の発展に伴い，各ルータが持つキャッシュ機能部分を処理機能として考え，ユーザとネットワーク内の処理機能を繋ぐ通信部分に ICN を適用する研究が多くなされている．

NFN(Named Function Networking)[30]では，ユーザが Interest を利用してデータ名と処理関数名を指定し，それを受信した NFN capable router が Interest 名を解釈して各関数（処理）をできるルータに転送することで，データに目的の処理を施す機構が提案されている．計算処理を(a)既に過去に処理された可能性のある処理（b)処理されたことがない，あるいは処理されたことがあっても現在使用できない処理（c)ポリシー等で他サイトに処理を push する必要がある処理の 3 つ分類し，転送している（図 3.4.1)．

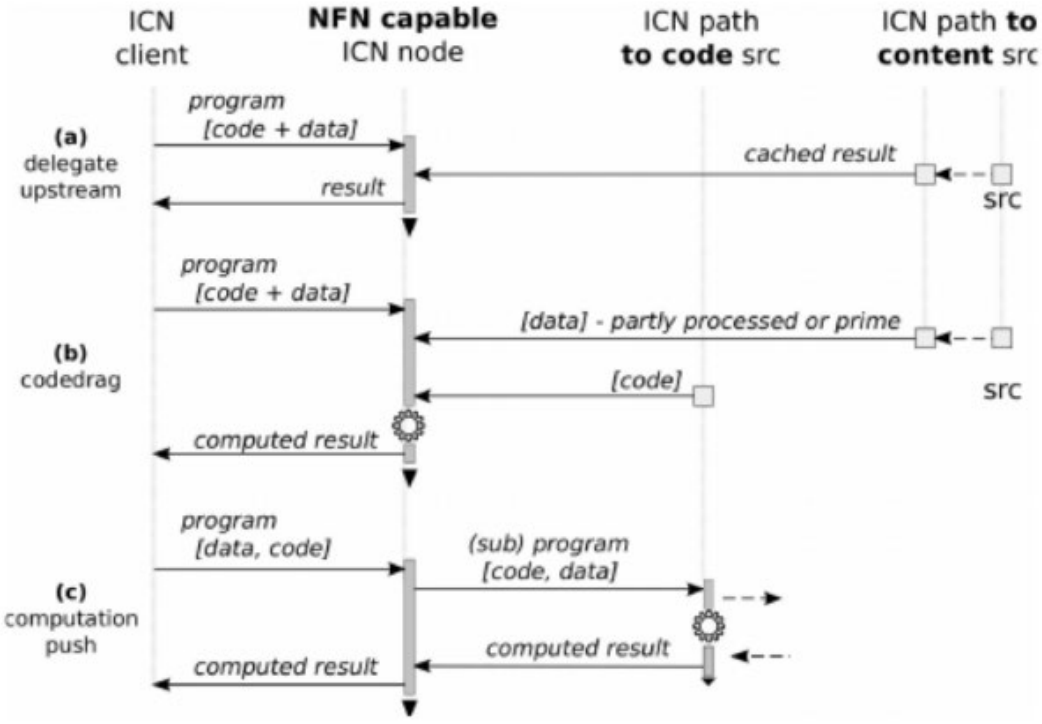


図 3.4.1 NFN が扱うシナリオ[30]

また，ICN-FC[31]では，関数処理を行う順番に並べて名前付けされた Interest 利用し，それに沿ってデータ処理を行う手法を提案している．データを関数 A→関数 B の順に処理する場合は，“関数 B/関数 A/データ名”といった Interest を発行する．Interest は適切にルーティングされ，関数 B を持つルータに到着する．そのルータは Interest の関数 B 部分を削除し”関数 A/データ名“と Interest を再定義して転送する．これを繰り返すことで，Interest は目的の処理前データを持つルータに到着できる．そこから Interest が辿ってきた経路を逆方向にデータを転送し，道中の各ルータで目的の処理（関数 A,関数 B）を行うことで目的を達成する．

第 4 章 情報指向サービスメッシュ

4.1 概要

第 2 章及び第 3 章で示した通り，クラウドネイティブの考え方に基づき，マイクロサービスアーキテクチャ及びサービスメッシュ構成は，ネットワーク環境に拡張性や柔軟性をもたらすといえる．また，計算資源が分散された環境下でサービス間通信に ICN を用いてデータの送受信を行うことは，様々なメリットがある．そこで，[32]によって提案されている通信部のサイドカーに ICN を用いたサービスメッシュを利用し，データ送受信の検証及び通信状況の可視化を行った．本章では，実験で使用したサービスメッシュの機能・構成を示す．

4.2 基本構成

本実験で使用したサービスメッシュのアーキテクチャおよび実装されている機能について説明する．サービス全体構成を図 4.2.1 に示す．

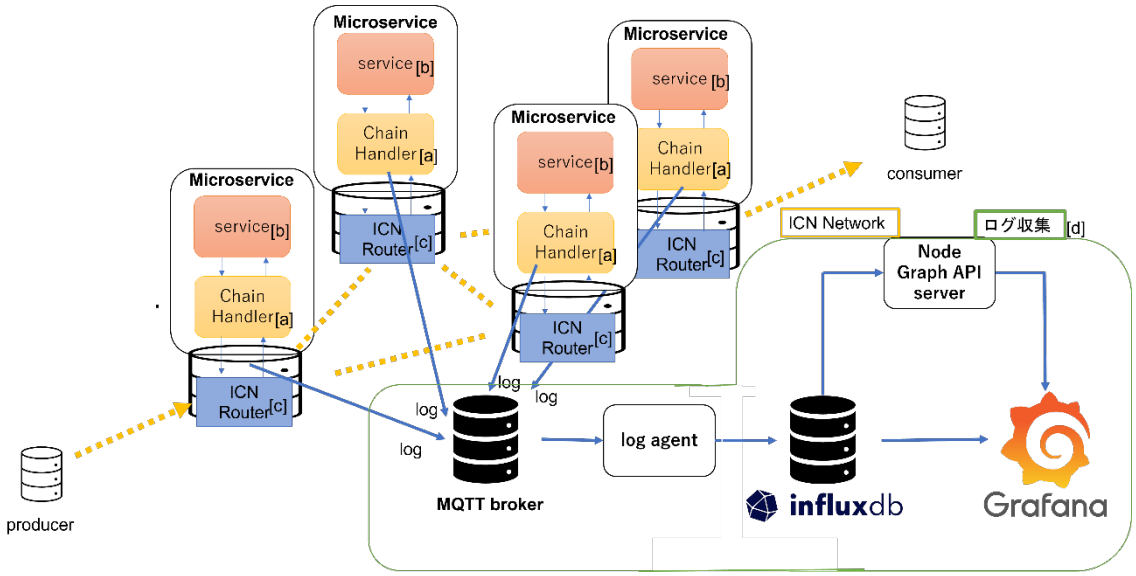


図 4.2.1 サービスの全体構成

[a]チェインハンドラー

ICN のルータ機能と処理機能（サービスファンクション）を中継するミドルウェアである．サービスチェインの機能を提供しており，Interest Parser, Branch Handler, Pseudo Consumer, Merge Handler, Service Handler, Pseudo Producer の 6 つの機能が実装されている．チェインハンドラー内で疑似的にコンシューマーとプロデューサーを生成し，ICN ルータとサービス処理とのデータを受け渡す構成となっている.Interest の種類として[25]で提案されている SMI を用いる．各機能の役割を簡単に説明する．

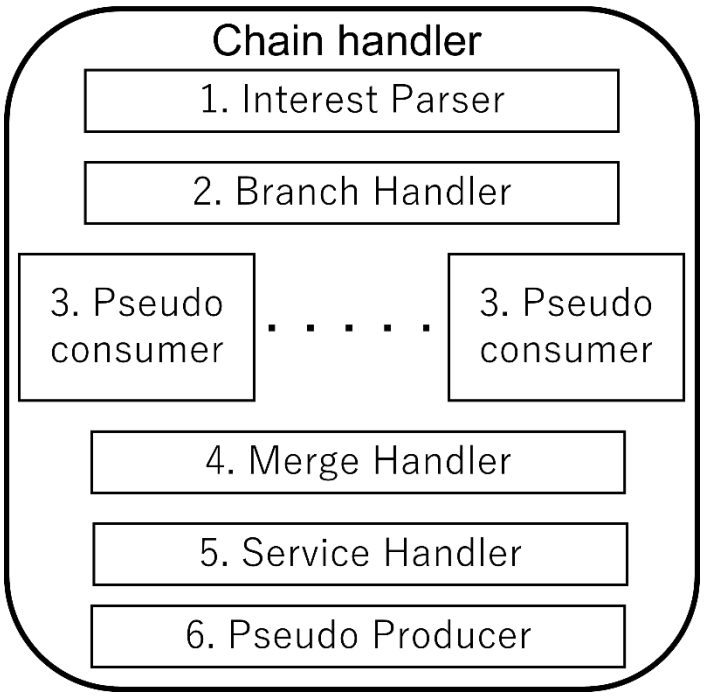


図 4.2.1 チェインハンドラー構成

1.Interest Parser

コンシューマー，もしくは，他のマイクロサービス内の疑似コンシューマーで生成された Interest の要求を解釈する．

2.Branch Handler

次のマイクロサービスに転送して目的のデータを取得するための新しい Interest を生成する．また，要求データの個数に応じて，複数の疑似コンシューマーを動的に作成する．これらのコンシューマーは非同期で並列に生成される．

3.Pseudo consumer

Branch Handler で生成された Interest を送信し，データを受信するまで待機する．

一方，Interest の名前付けは ICN-FC[31]の方法を踏襲している．例えば,relay1 という名前のマイクロサービスが”ccnx:/relay1/relay2/data”という Interest を Interest Parser で受信したとする．この場合，Branch Handler で，新たな Interest として” ccnx:/relay2/data” を作成し，Pseudo consumer を生成する．Pseudo consumer から relay2 に到達するように Interest を送信し，データを受信するまで待機する．

4.Merge Handler

受信したコンテンツのバッファリングを行う．

5.Service Handler

サービスが要求するコンテンツのすべてを Pseudo consumer で受信したことを確認後，サービスへデータを送信し，サービスでの処理結果が返ってくるまで待機する．

6.Pseudo Producer

処理結果をコンシューマー，もしくは他のマイクロサービス内の疑似コンシューマーに送信する．

[b]処理機能

任意の処理機能を実装できる．後述する実験では，入力データをコピーして出力するサービスを使用している．

[c]ログ処理機能

各サービスから可視化に必要な情報を受信し，必要な処理を行う．ログ処理機能の詳細は後述する．

4.3 ログ処理機能

4.3.1 概要

通信状態の管理及び可視化のために，ICN サイドカー（チェインハンドラー）から必要情報を抽出しデータベースに保存している．その抽出したデータの転送及び保存のためのアーキテクチャ構成と可視化までの手順を説明する．

4.3.2 手順

ICN サイドカーからログデータを送信し，可視化ツールである Grafana に転送するまでの過程を順に説明する．

①ICN サイドカーから，サービスに関する情報を取得する．

サイドカー部分はサービス自体とは切り離されて実装されているため，サービス自体の実装を変更することなく，ログ情報を収集できる．ログ情報は json ファイルに格納する．図 4.3.2.1 に一部抜粋したログデータの例を示す．各マイクロサービスは，収集したログ情報を MQTT ブローカーへ publish する．これによって各マイクロサービスから得たログ情報を MQTT ブローカーに集約することができる．

```
"log": {
  "timestamp": {
    "input": "2023-01-16T18:06:01.964300Z",
    "output": "2023-01-16T18:06:01.966569Z"
  },
  "data size": {
    "input": 38196,
    "output": 19271
  },
  "proc time": {
    "input": "null",
    "output": "null",
    "service": 0.002
  },
  "service log": {
    "state": "dummy",
    "log": "null"
  },
  "rx queue size": 0
}
```


図 4.3.2.1　ログデータの例（一部抜粋）

②MQTT ブローカーから log 収集エージェントに送信し，そこからデータベースに格納する．データベースは，オープンソースの時系列データベースである influxDB[33]を用いる．InfluxDB は，ソースコードが Go で記述されており，高速読み書きが可能である．加えて，クエリ言語を使用することができるため，一定期間のデータの取り出しやソートを容易に行うことができる．図 4.3.2.2 のようにログデータを成形し influxDB に格納する．格納したデータは図 4.3.2.3 のような形式で取り扱うことができる．

```
data_to_db = [{
  "measurement":"depgraph_data",
  "fields": {
              "origin_app": in_name,
              'target': out_name,
              },

}]
print (data_to_db)
result = influx_client.write_points(data_to_db)
```

図 4.3.2.2 influxDB への書き込み

Time	origin_app	target
2023-01-16 02:18:43	relay3	relay1
2023-01-16 02:18:43	relay2	relay1

図 4.3.2.3 influxDB 内のデータの例

③Grafana でノードグラフを用いた可視化をするツールとして，node graph api[34]/service dependency graph[35]というプラグインを用いる．各ツールに応じた方法で influxDB 内のデータを取り出し，成形する．

(i)node graph api を用いる場合は，　REST API 処理を行うための node graph api server を構築し，必要データを influxDB から取り出す．

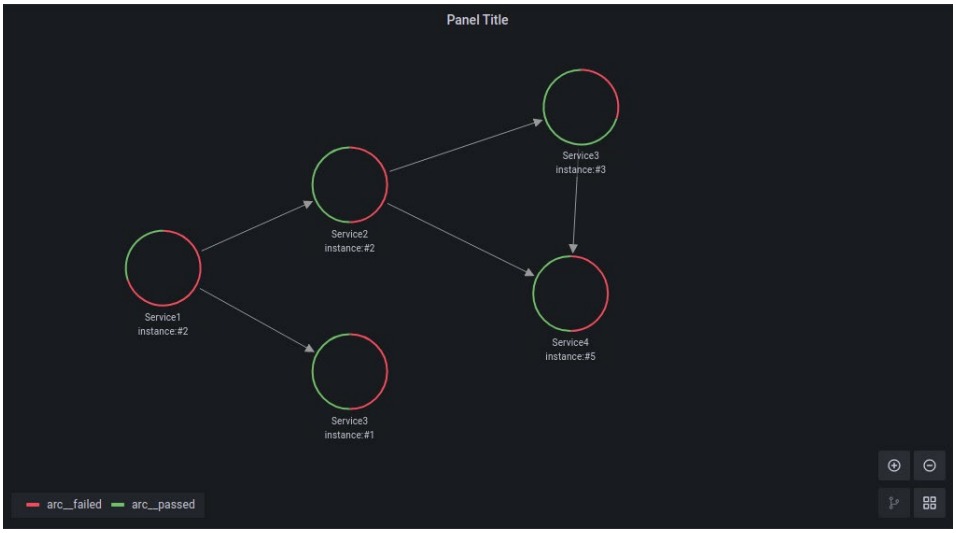


図 4.3.2.4 ノードグラフの例[35]

Node graph API での可視化に必要なデータ形式を図 4.3.2.5 に示す．データ形式は”node”と”edge”で構成される．edge はノード間のつながりを示す．Grafana のダッシュボード上から/api/graph/data に GET メソッドを送ることでその時点での，node と edge を可視化できる．また図 4.3.2.5 に示されるように edge の”mainstat”に値を格納することで，サービス間通信の任意のメトリクスをグラフ上に表示できる．

```
"edges": [
    {
        "id": "1",
        "mainStat": "53/s",
        "source": "1",
        "target": "2"
    }
],
"nodes": [
    {
        "arc__failed": 0.7,
        "arc__passed": 0.3,
        "detail__zone": "load",
        "id": "1",
        "subTitle": "instance:#2",
        "title": "Service1"
    }
]
```


図 4.3.2.5 node graph api で用いるデータ構造

(ii) service dependency graph を用いる場合は，Grafana 側からデータベースとして influxDB を直接指定することで可視化できる．必要なデータ構造を図 4.3.2.7 に示す．

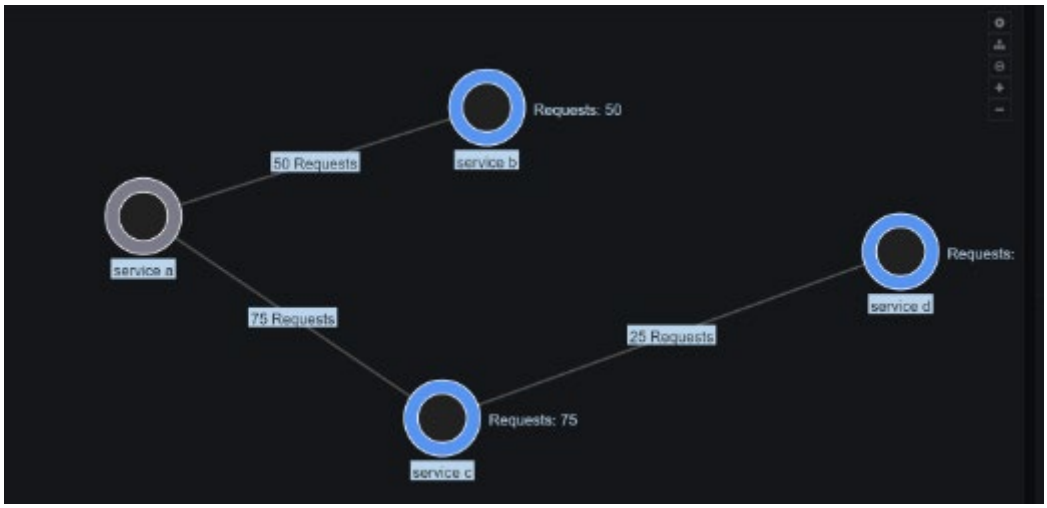


図 4.3.2.6 ノードグラフの例[36]

```
{
 "origin_app": in_name,
 'target': out_name,
 'req_rate': serve_proc_time,
 'resp_time':resp_time
 },
```

図 4.3.2.7 service dependency graph で用いるデータ構造

④ また，Grafana で，influxDB 内のデータを直接取得することで，Grafana 上に任意のメトリクスやグラフ等をダッシュボード上に表示できる．

第 5 章 評価実験

5.1 実験環境

ICN サービスメッシュのシミュレーションを行い，そのデータフローを可視化する実験を行った．シミュレーション実験のパラメータを以下の表 5.1.1 に示す

表 5.1.1 実験パラメータ

環境	Local Kubernetes Cluster
送信データ	画像データ(約 15kB)
マイクロサービス	入力をコピーして出力
Producer	6
consumer	1
Service function	10
通信手法	ICN(SMI)

5.2 実験内容

実験を行ったサービスメッシュの構成図を図 5.1.1 に示す.

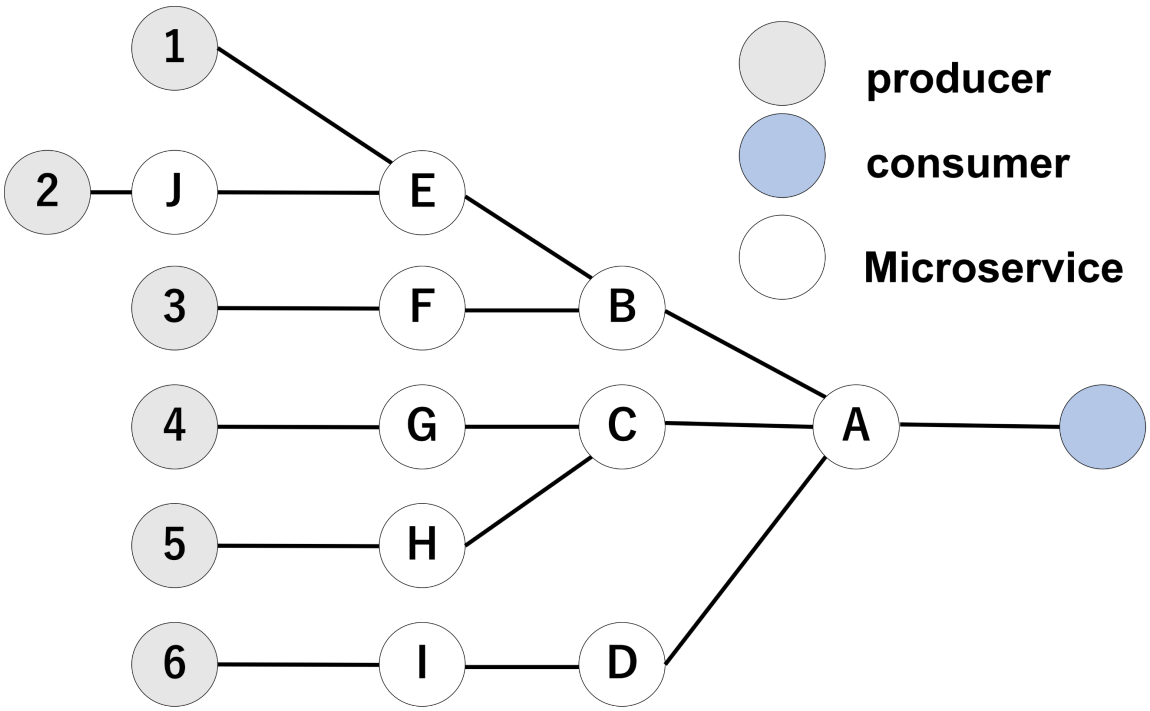


図 5.2.1　実験トポロジー図

図 5.1.1 の consumer から Interest を送信し，各サービスメッシュを通過点として，1〜6 の consumer まで転送する．consumer では Interest を受信次第，Data を同一ルートで送り返す．各マイクロサービスは戻ってきたデータにサービス処理を加え，次ノードに転送する．各サービス間は ICN サービスメッシュで連結されている．各ノードに名前付けを行い，コンテンツ取得のための Interest を「ccnx:/ルータ名/・・・・/ルータ名/プロデューサー名」と定義する．各ノードの名前付けを表 5.2.1，　producer①(lena)に到達する Interest を例に Interest の解析手順の説明を図 5.2.2 に示す.

表 5.2.1 各ノードの名前付け

Microservice A～J	Relay1～10
Producer 1	lena
Producer 2	test
Producer 3	hoge
Producer 4	john
Producer 5	aaaa
Producer 6	eeee

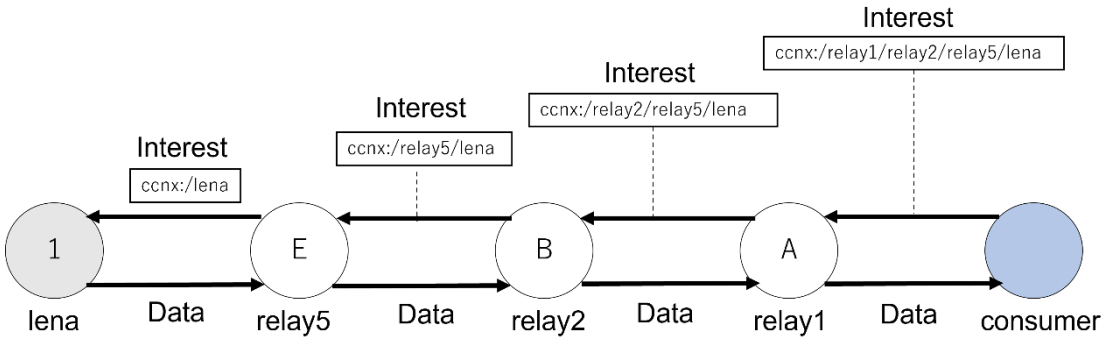


図 5.2.2 Interest コンテンツ名の遷移

5.3 ログデータ加工

第 4 章で記述した構成を用いて抽出したログデータを influxDB に保存する．Interest の送受信とデータの送受信をグラフ化するために，それぞれの入出力タイムスタンプを抽出する．一方，Interest とデータの送信時間はミリセカンドのオーダーとなる．これを同じスケールのままノードグラフで可視化した場合，サービス間データフローは非常に短期間しか表示されず，正しいデータフローを目視で認識することができなくなる．

そこで，一定時間の送受信時間のみを抽出し，各データフローの基準時間（シミュレーション開始時間）からの経過時間を定数倍する．本実験では，シミュレーション時間として 30 秒，倍率を 1000 倍としてシミュレーションを行った．InfluxDB に登録した生データを図 5.3.1 に示す．ログデータの加工は，influxDB で使用できるクエリ言語を用いて容易に行うことができる．

Time	id	source	target	in_time	out_time
2023-01-20 07:07:29	aaaa_relay8	aaaa	relay8	1674132763	1674133649
2023-01-20 07:07:29	john_relay7	john	relay7	1674132761	1674133649
2023-01-20 07:07:29	eeee_relay9	eeee	relay9	1674132765	1674133649
2023-01-20 07:07:29	hoge_relay6	hoge	relay6	1674132759	1674133649
2023-01-20 07:07:29	relay9_relay4	relay9	relay4	1674133649	1674133649
2023-01-20 07:07:29	relay7_relay3	relay7	relay3	1674133649	1674133649
2023-01-20 07:07:29	relay8_relay3	relay8	relay3	1674133649	1674133649
2023-01-20 07:07:29	test_relay10	test	relay10	1674132757	1674133650
2023-01-20 07:07:29	lena_relay5	lena	relay5	1674132755	1674133650

図 5.3.1 実験データ

5.4 結果

5.4.1 node graph api での可視化

5.4.1.1 結果

Node graph api での可視化したデータフローを時系列に沿って，図 5.4.1.1〜図 5.4.1.8 に示す．本実装では，ノード間のつながりをあらかじめ線で結び，Interest/Data が受信をするたびに，その受信したノードの円周上の色を変化させて，データフローを可視化した．データを待機している状態の時には円周上を緑色とし，データを受信しているときは赤色として表示した．矢印の向きに関してはデータの方向性とは一致しない．

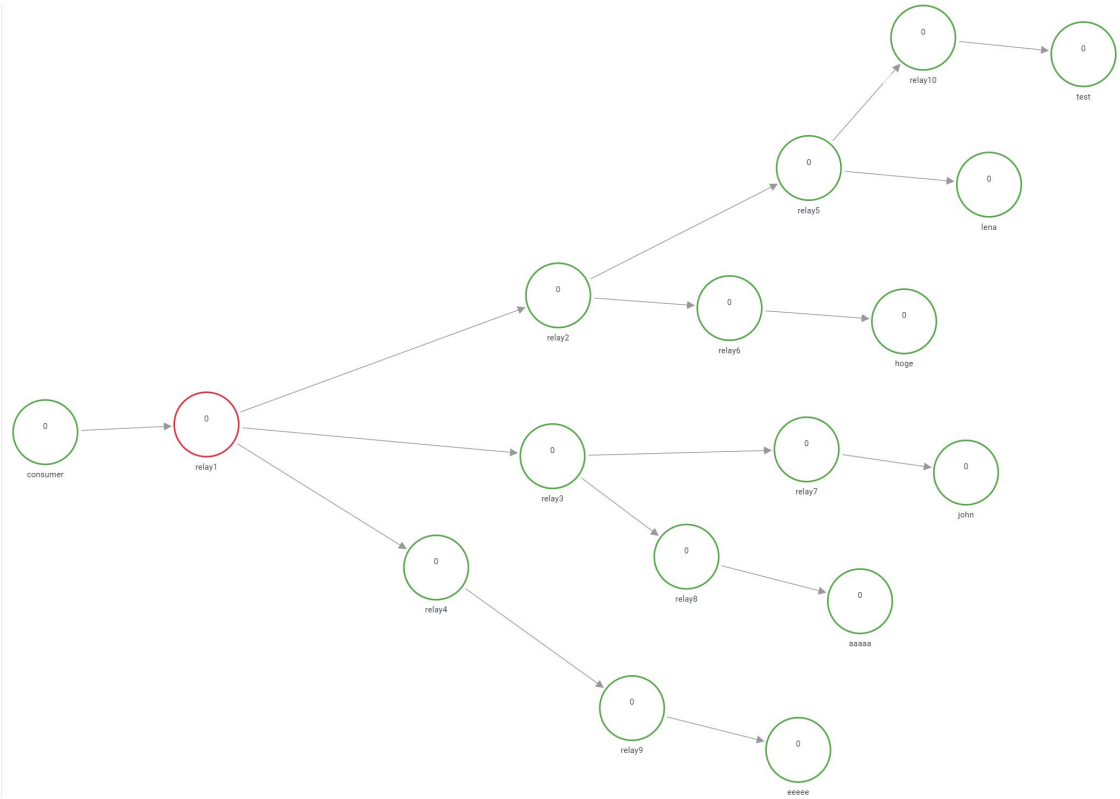


図 5.4.1.1 Interest フロー①

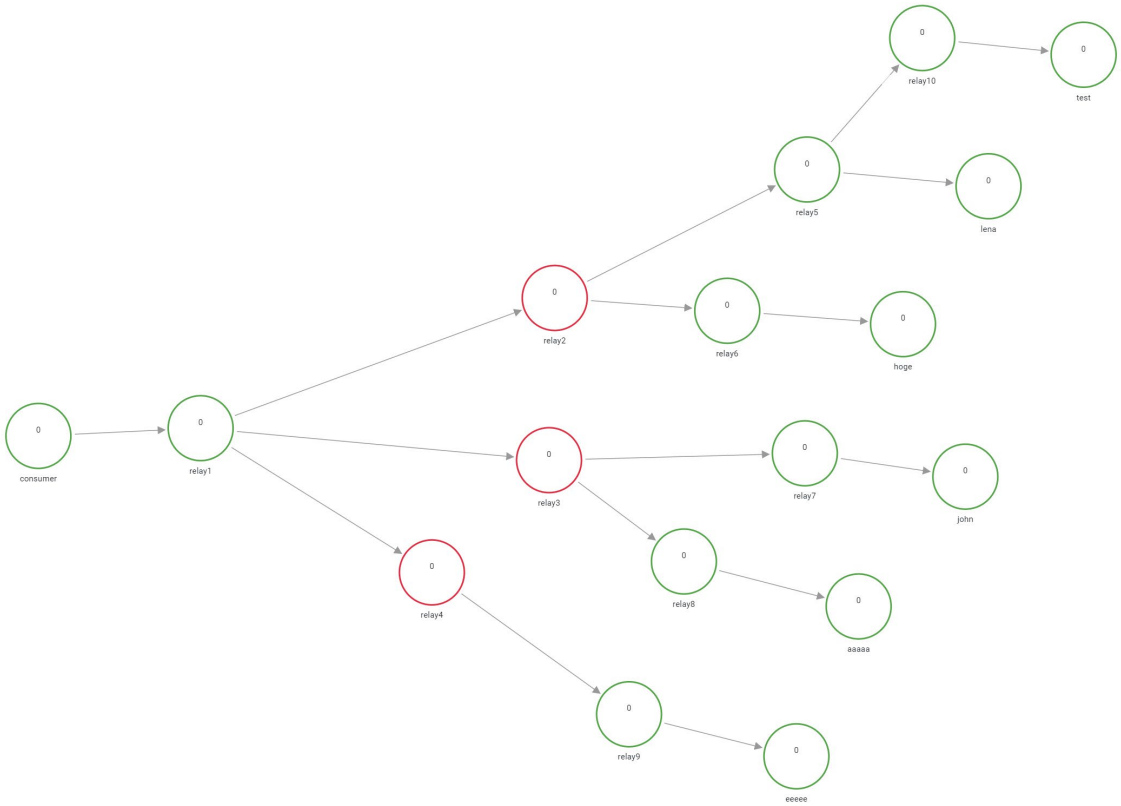


図 5.4.1.2 Interest フロー②

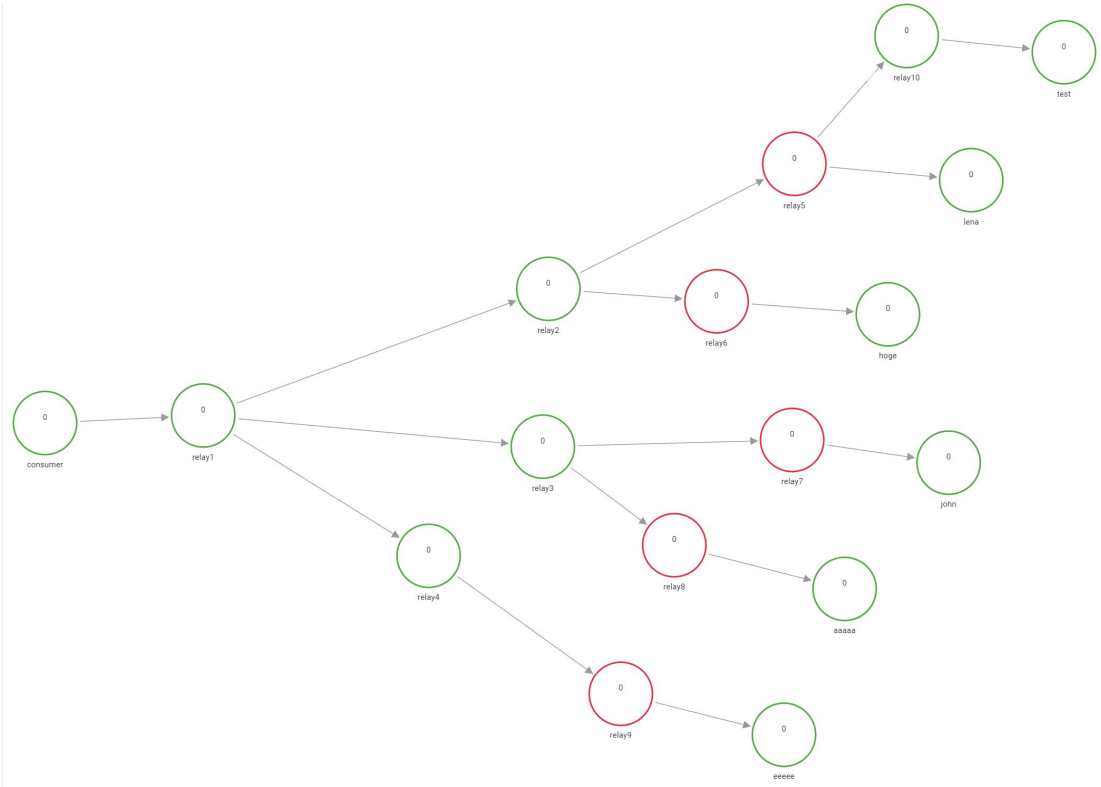


図 5.4.1.3 Interest フロー③

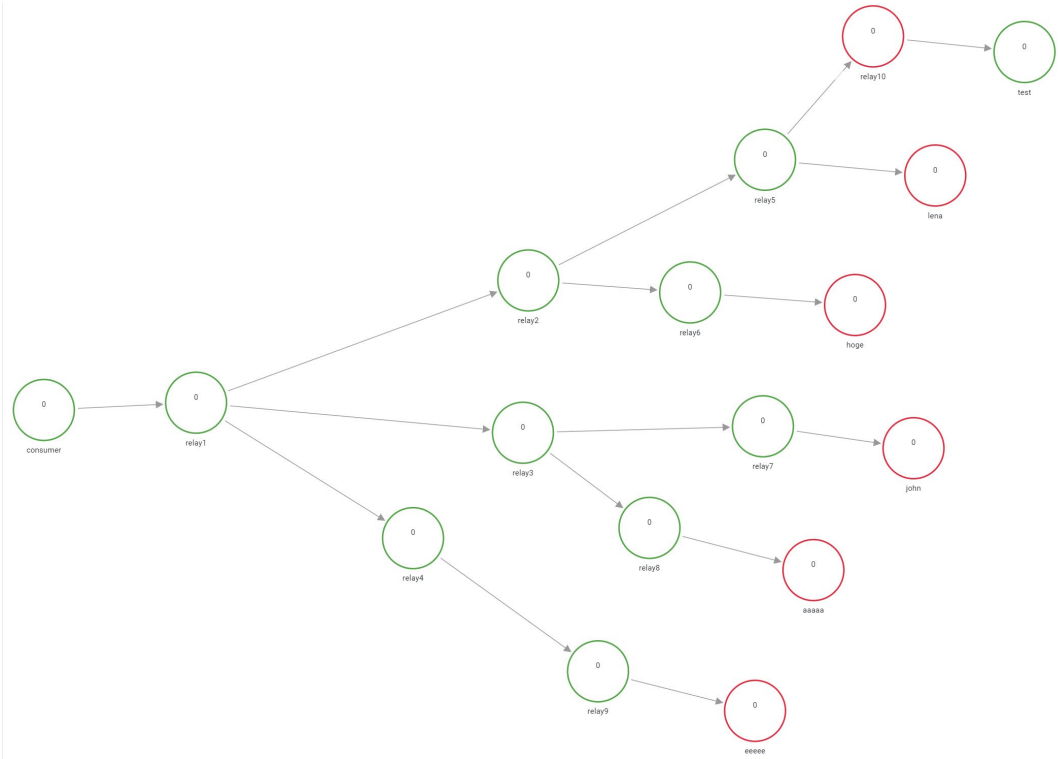


図 5.4.1.4 Interest フロー④

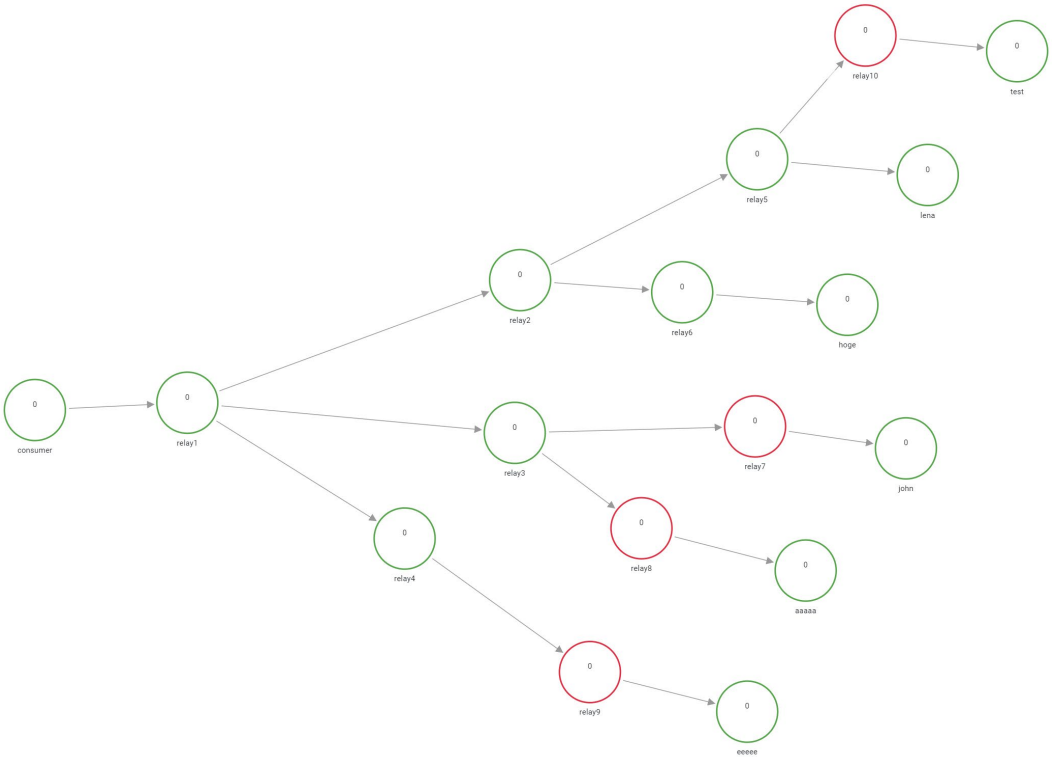


図 5.4.1.5 data フロー①

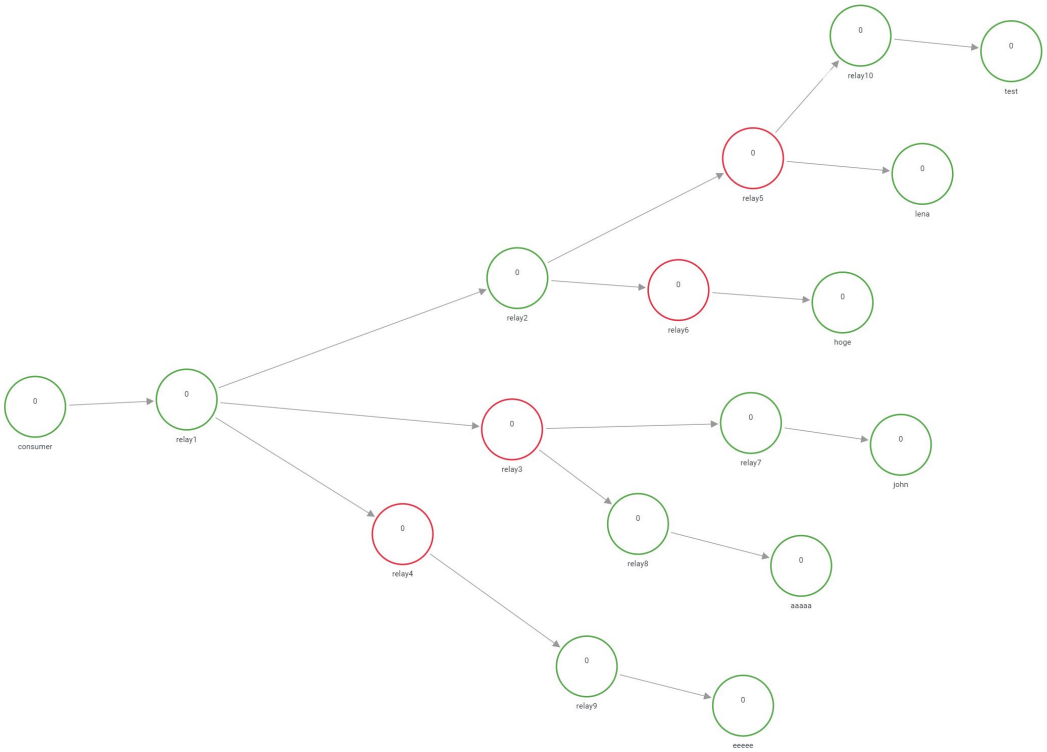


図 5.4.1.6 data フロー②

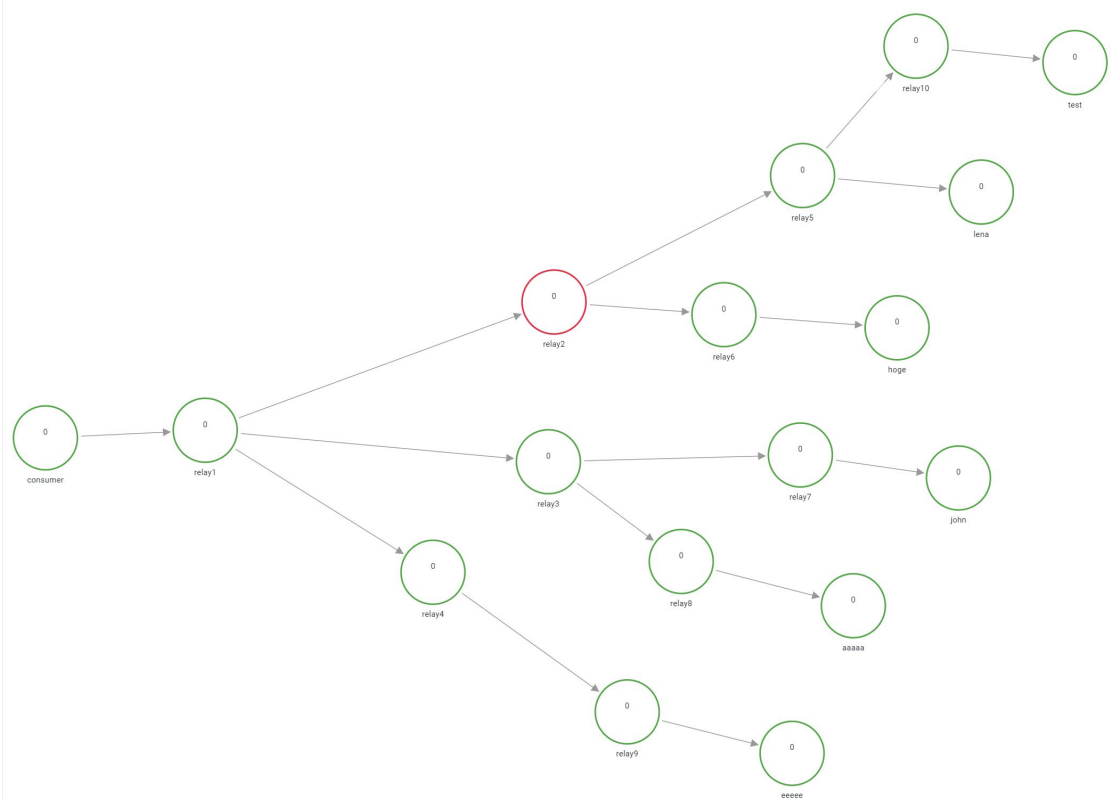


図 5.4.1.7 data フロー③

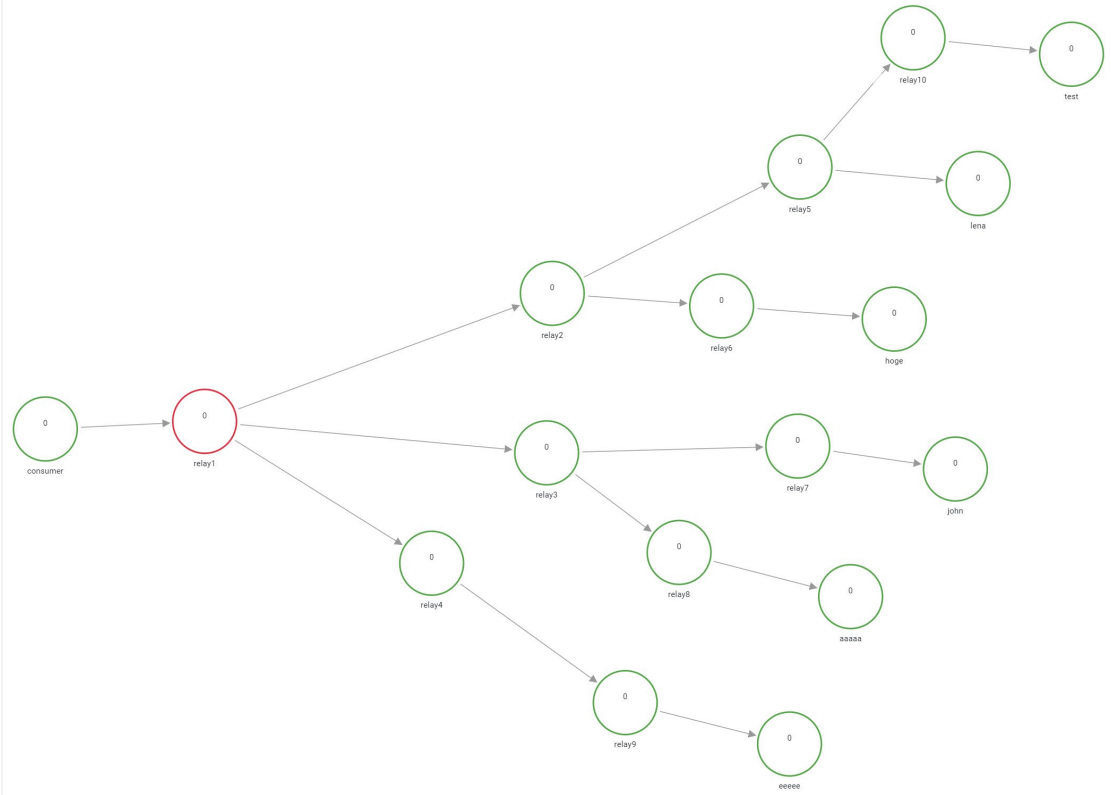


図 5.4.1.8 data フロー④

この可視化によって，Interest が中間ノードを経由して各 producer に転送され，producer に到着後，同一ルートを逆向きに data が送信され，relay1 まで転送されていることが確認できる．

5.4.1.2 評価

Node graph api ではノードの色変化を利用してデータフローを可視化した．今回のように，階層的にデータが転送される場合は，その送受信タイミングが見やすい．しかし，この手法の問題点は，ノード間のつながりを示す矢印を使用しないため，階層化されていない複雑な構造になった場合見づらくなることや，データの向きを表示できないため直感的にデータの動きを追うのが難しいことが挙げられる．

5.4.2 service dependency graph での可視化

5.4.2.1 結果

service dependency graph での可視化したデータフローを時系列に沿って，一部切り取った結果を図 5.4.2.1～図 5.4.2.5 に示す.本実装では，ノード間の矢印上でパケットを模した長方形が行き来することでデータの動きを示す．データ量に応じて，行き来するパケットの数が自動で変化する．また，現在受信を行っているノードの円周上を赤く表示させた.

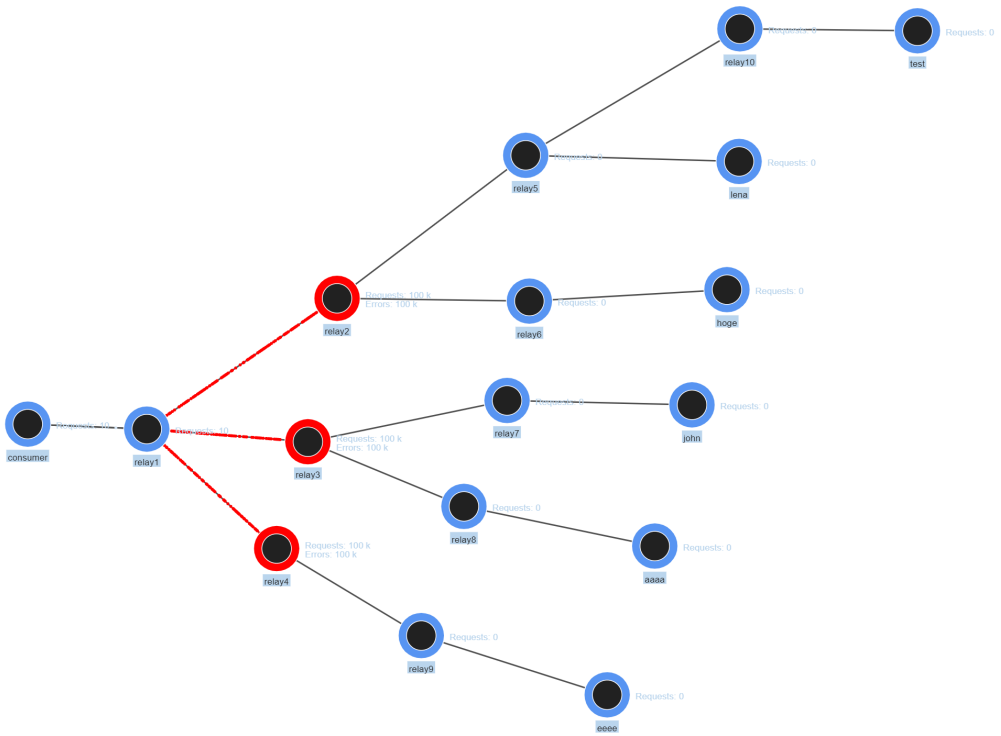


図 5.4.2.1 Interest フロー①

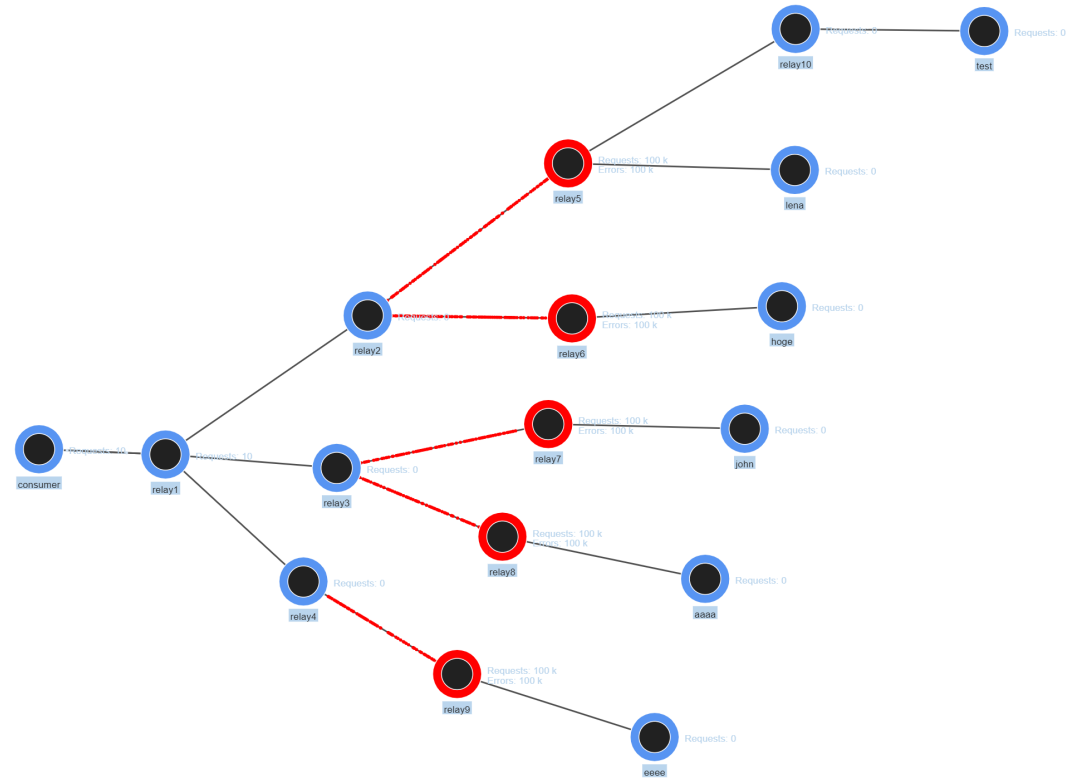


図 5.4.2.2 Interest フロー②

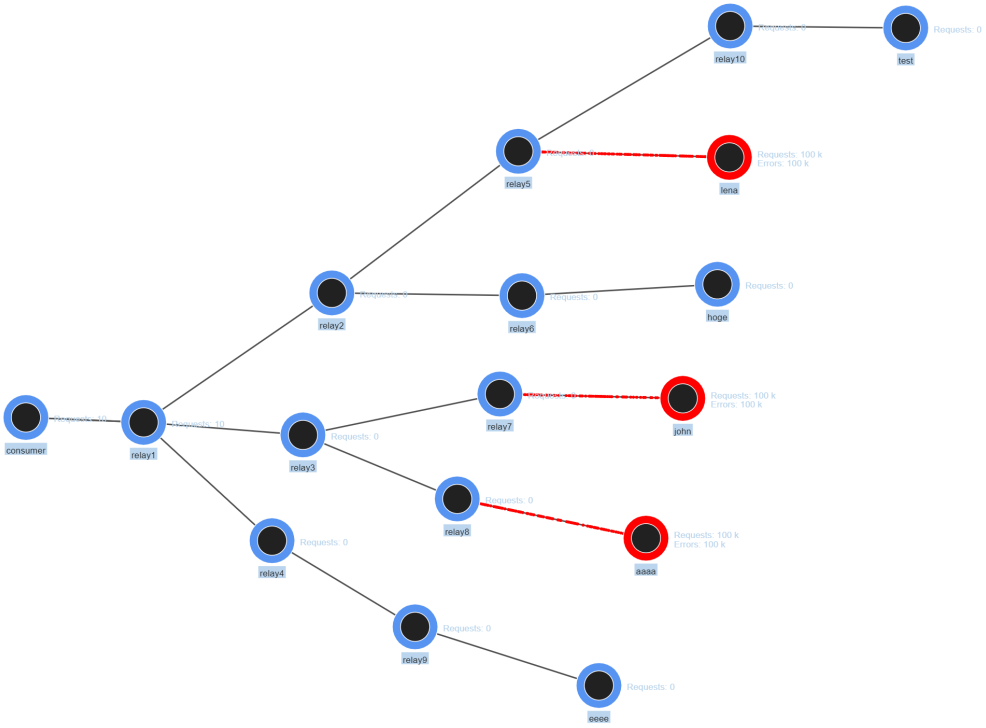


図 5.4.2.3 Interest フロー③

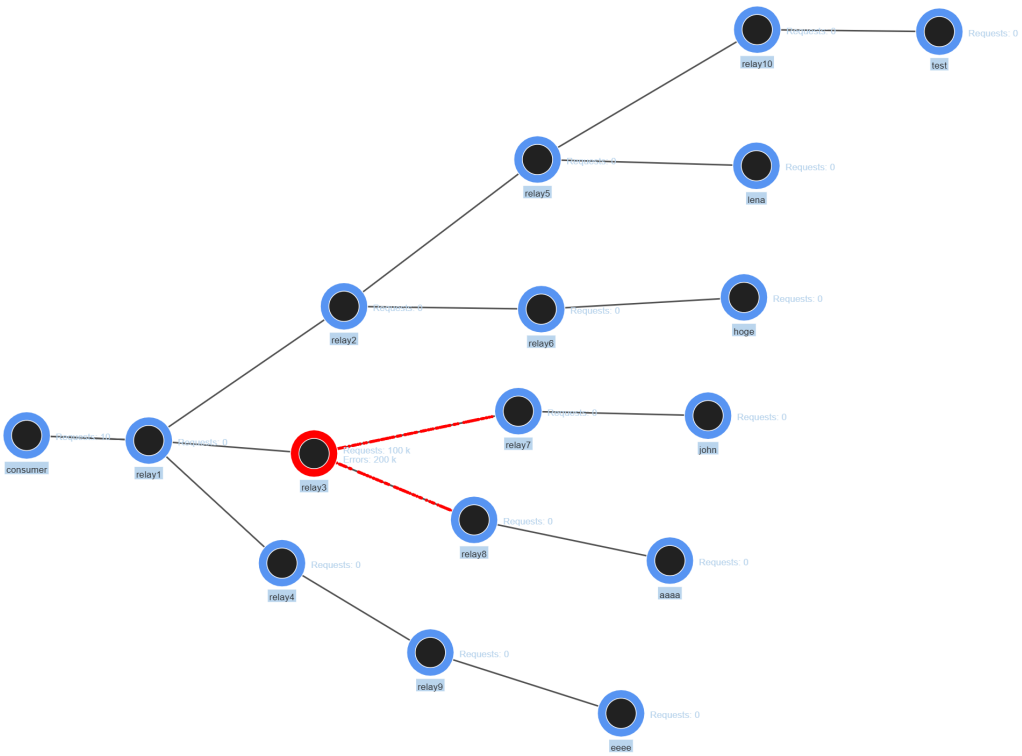


図 5.4.2.4 data フロー①

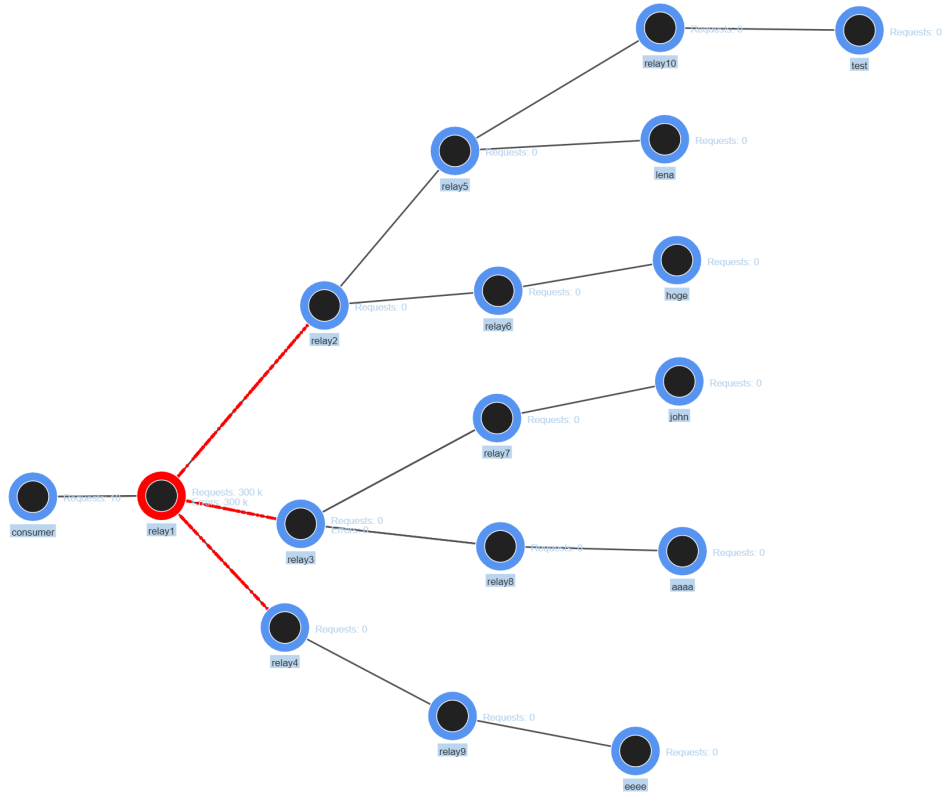


図 5.4.2.5 data フロー②

5.4.2.2 評価

Service dependency graph ではノードの色変化に加えて，ノード間の線上にもパケットの行き来を表示することができる．そのため，グラフが複雑になった場合も目視で確認しやすい．更にダッシュボード上でノードをドロップすることで自由に配置できるため，より直感的にデータフローを確認できた．

5.5 妥当性検証

Grafana では influxDB に格納されているデータを容易にグラフ化できる．そこで，可視化のためにモニターし抽出したデータと，consumer で測定したデータを比較し，収集したログが同等の性能を示しているかを評価する．評価のためのトポロジーとして図 5.3.1 を使用した．

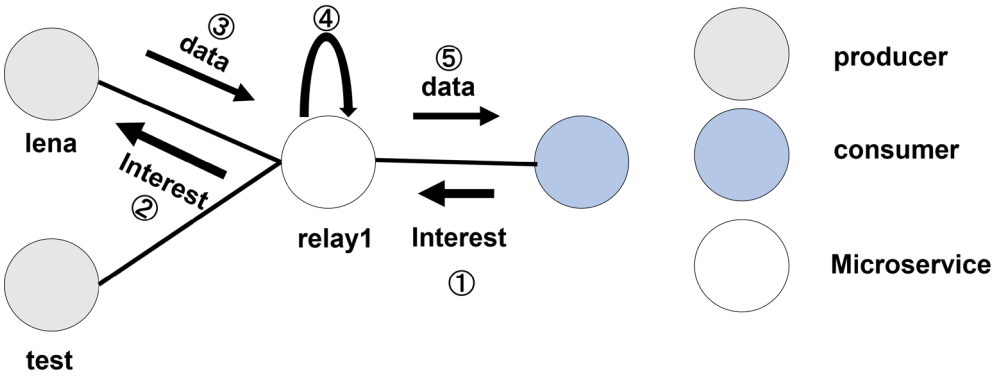


図 5.5.1 実験トポロジー図

図のように consumer が producer である lena からデータを取得し relay1 で処理されたものを受信するための Interest を発行する．Consumer が Interest を送信してから目的のデータを受信するまでの過程は①Interest を relay1 に送信，②relay1 が lena に Interest を転送，③lena が relay1 にデータを送信，④relay1 が受信したデータをサービス処理，⑤relay1 が処理後のデータを consumer に転送する，の 5 段階である．①～⑤の処理時間をそれぞれリアルタイムのログ情報として，influxDB に格納し合計する．一方実際の性能として，consumer 上で Interest を送信してからデータを受信するまでの End to End 遅延を測定する．①～⑤のデータの和と end-to-end 遅延を重ね合わせたグラフを図 5.5.2 に示す．

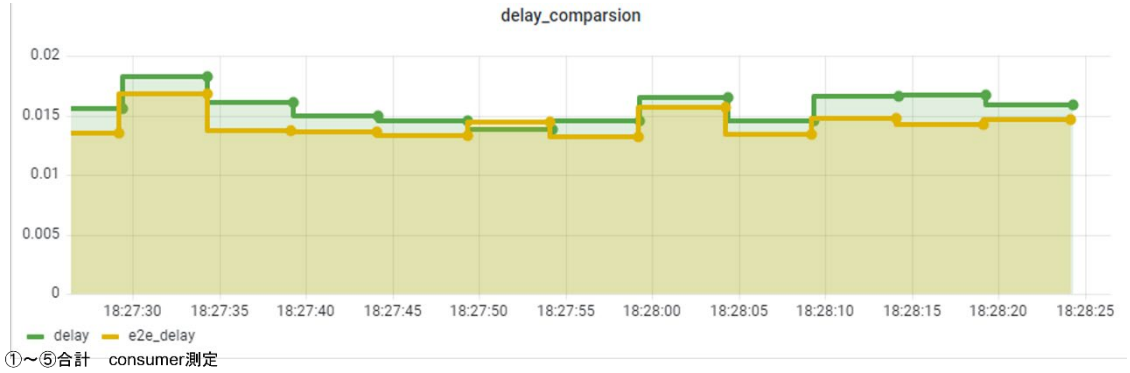


図 5.5.2 各ノード遅延の合計と e2e 遅延の比較

上記の図の緑が①～⑤の合計であり，黄色が consumer での end-to-end 遅延である．リアルタイムで収集したログ情報と実際の性能には大きな差がなく，通信状況を妥当な範囲で可視化できているといえる．

第 6 章 総括

6.1 まとめ

本稿では，新たなネットワークアーキテクチャである ICN を通信方法としたサービスメッシュを構築し，可視化を目的としたログ情報の収集機構の構築，及び，目視で判断しやすい可視化手法を検討した．ネットワークアプリケーションの種類や目的は多様化しており，より多くのマイクロサービスを通信させアプリケーションを構築していく環境が想定される．そのような環境下で，各ノードから得られるログ情報を，時間をかけて分析するよりも，ノードグラフを用いて，データフローをリアルタイムに可視化することは，サービスが正しい挙動を示しているかどうかを把握するためには有用である．Grafana や influxDB といったオープンソースのツール及び node graph api や service dependency graph といったプラグインを用いて可視化を行ったが，収集したログ情報から得られるサービス性能と実際のサービス性能と比較し，大きな差はなく，十分に可視化できていることが確認できた．

6.2 今後の展望

ノードグラフを他のデータ（トレースデータ等）と結び付ける等を行うことで，更に直感的にサービス状況を把握するための工夫に取り組みたい．また，可視化のために必要なデータを新たに取得する際には，そのデータを効率的に取得できるようなアーキテクチャについても十分に検討したい．

謝辞

本研究を行うにあたり，様々なご指導や助言を頂きました甲藤二郎教授に心より感謝申し上げます．また，研究を進める上で多くの貴重なアドバイスをくださった金井謙治さんをはじめ，甲藤研究室の皆様に御礼申し上げます．

最後に，様々な場面において支えてくださった家族や友人，多くの方々に深く感謝いたします．

2023 年 1 月 2 3 日

飯塚　真梧

参考文献

[1] 総務省, ”令和 4 年版情報通信白書”, 2022.

[2] Cloud Native Computing Foundation, “CNCF Cloud Native Definition v1.0”, 2018.

[3] J. Thönes, "Microservices," in IEEE Software, vol. 32, no. 1, pp. 116-116, Jan.-Feb. 2015, doi: 10.1109/MS.2015.11.

[4] Istio, “Simplify observability, traffic management, security, and policy with the leading service mesh.”. [online]: https://istio.io/ (accessed January 18, 2023)

[5] J. Halpern and C. Pignataro, "Service function chaining (SFC) architecture", 2015.

[6] Bhamare D., Jain R., Samaka M., Erbad A., “A survey on service function chaining”, J. Netw. Comput. Appl., 75, 2016

[7] M. Usman, S. Ferlin, A. Brunstrom and J. Taheri, "A Survey on Observability of Distributed Edge & Container-Based Microservices," in IEEE Access, vol. 10, pp. 86904-86919, 2022, doi: 10.1109/ACCESS.2022.3193102.

[8] JAEGER, “Jaeger: open source, end-to-end distributed tracing”, [online]: https://www.jaegertracing.io/ (accessed January 18, 2023)

[9] ZIPKIN, “Zipkin”, [online]: https://zipkin.io/(accessed January 18, 2023)

[10] OpenTelemetry, “OpenTelemetry -High-quality, ubiquitous, and portable telemetry to enable effective observability” [online]: https://opentelemetry.io/(accessed January 18, 2023)

[11] L. Andrey, O. Festor, A. Lahmadi, A. Pras, and J. Schönwälder. Survey of SNMP performance analysis studies. International journal of network management, 19 (6):527--548, 2009.

[12] Song, H., Qin, F., Martinez-Julia, P., Ciavaglia, L., and A. Wang, "Network Telemetry Framework", RFC 9232, DOI 10.17487/RFC9232, 2022.

[13] Sock Shop: Microservices Demo, [online]: https://microservices-demo.github.io/ (accessed January 22, 2023)

[14] Sabharwal, N., Pandey, P, “Deploying Containerized Applications with Google GKE. In: Pro Google Kubernetes Engine “, 2020.

[15] G. Xylomenos et al., "A Survey of Information-Centric Networking Research," in IEEE Communications Surveys & Tutorials, vol. 16, no. 2, pp. 1024-1049, Second Quarter 2014, doi: 10.1109/SURV.2013.070813.00063.

[16] 朝枝 仁 松園和久 大岡 睦, “6-2 情報指向型ネットワーク技術 6-2 Information-/ Content-Centric Networking”, 情報通信研究機構研究報告 Vol. 64 No. 2 pp. 93-102,2018.

[17] M. Amadeo et al., "Information-centric networking for the internet of things: challenges

and opportunities," in IEEE Network, vol. 30, no. 2, pp. 92-100, March-April 2016, doi: 10.1109/MNET.2016.7437030.
[18] V. Jacobson, et al., Networking Named Content, ACM CoNEXT, 2009.
[19] K. Matsuzono, D. Nguyen and H. Asaeda, "Content Request Handling for Application-Oriented Transport Control," in IEEE Communications Magazine, vol. 57, no. 6, pp. 14-19, June 2019, doi: 10.1109/MCOM.2019.1800717.
[20] N. Marie-Magdelaine, T. Ahmed and G. Astruc-Amato, "Demonstration of an Observability Framework for Cloud Native Microservices," 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Arlington, VA, USA, 2019, pp. 722-724.
[21] Li, B., Peng, X., Xiang, Q. et al. Enjoy your observability: an industrial survey of microservice tracing and analysis. Empir Software Eng 27, 25 (2022).
[22] D. Cha and Y. Kim, "Service Mesh Based Distributed Tracing System," 2021 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Korea, Republic of, 2021, pp. 1464-1466, doi: 10.1109/ICTC52510.2021.9620968.
[23] J. Levin and T. A. Benson, "ViperProbe: Rethinking Microservice Observability with eBPF," 2020 IEEE 9th International Conference on Cloud Networking (CloudNet), Piscataway, NJ, USA, 2020, pp. 1-8, doi: 10.1109/CloudNet51028.2020.9335808.
[24]ICN 電子情報学会情報指向ネットワーク技術特別研究専門委員会, “ICN/CCN 通信を実現するソフトウェアプラットフォーム Cefore”, [online]: https://www.ieice.org/~icn/?page_id=1208, (accessed January 18, 2023)
[25] H. Asaeda, A. Ooka, K. Matsuzono and R. Li, "Cefore: Software platform enabling content-centric networking and beyond", vol. E102-B, no. 9, pp. 1792-1803, 2019.
[26]NICT, “Cefore の基本機能と導入手順”, [online]: https://www.ieice.org/~icn/wp-content/uploads/2018/08/hands_on_01_Cefore.pdf , (accessed January 18, 2023)
[27] M. Mosko, I. Solis, Christopher A. Wood, “Content-Centric Networking (CCNx) Messages in TLV Format”, Internet Research Task Force (IRTF),2019.
[28] Jacobson, V.; Smetters, D. K.; Briggs, N.; Plass, M. F.; Stewart, P.; Thornton, J. D.; Braynard, R. VoCCN: voice over content-centric networks. Proceedings of the 2009 Workshop on Re-architecting the Internet (ReArch 2009); 2009 December 1; Rome, Italy. NY: ACM; 2009; 1-6.
[29] Jiachen Chen, M. Arumaithurai, Xiaoming Fu, K. Ramakrishnan, “SAID: A Control Protocol for Scalable and Adaptive Information Dissemination in ICN”, Proceedings of the 3rd {ACM} Conference on Information-Centric Networking, {ICN} '16, Kyoto, Japan, September 26-28, 2016.
[30] M. Sifalakis, et al., “An information centric network for computing the

distribution of computations," ACM ICN, 2014, pp. 137–146.
[31] L. Liu et al., "ICN-FC: An Information-Centric Networking based framework for efficient functional chaining," 2017 IEEE International Conference on Communications (ICC), Paris, France, 2017, pp. 1-7, doi: 10.1109/ICC.2017.7996572.
[32] Kenji Kanai, Toshitaka Tsuda, Hidenori Nakazato, Jiro Katto, "Information-centric service mesh for autonomous in-network computing", ICN '22: Proceedings of the 9th ACM Conference on Information-Centric Networking, 2022.
[33] influxdata, "influxDB", [online]: https://www.influxdata.com/
[34] Node Graph API, [online]: https://grafana.com/grafana/plugins/hamedkarbasi93-nodegraphapi-datasource/
[35] Service Dependency Graph, [online]: https://grafana.com/grafana/plugins/novatec-sdg-panel/