愛知工業大学大学院経営情報科学研究科

修 士 論 文

セ キュア オ ー バ ー レ イ ネット ワ ー ク システムにおけるサービス拡張性実現手法に関する研究

Study on Methods for Achieving Service Extensibility in Secure Overlay Network Systems

2024年3月

B22707　後 藤 廉

GOTO Ren

指導教員　内 藤 克 浩　教 授

目 次

第1章　序論

1.1　背景

インターネットは一般消費者向けから産業分野まで時代とともに浸透してきた. 元来, ネットワークサービスはメールの送受信や情報の検索, Social Networking Service (SNS) の利用, 交通情報やニュース記事の収集に用いられてきた. インターネットが普及した当初, アクセス元として主にパソコンや携帯端末等のコンシューマ向け機器が大半を占めていた. 一方, 昨今ではタブレット端末やテレビ, 家庭用ゲーム機等をはじめ, ネットワークノードの種類が多様化している [1,2]. また, 2000 年頃から登場した Internet of Things (IoT) も, 現在では産業界を中心に普及が進み, 様々なモノに通信機能を持たせてインターネットに接続する「モノのインターネット」が一般化しつつある. IoT では, 情報を得るための各種センサや, センサが捉えたデータを蓄積して分析するためのコンピュータ, 処理結果を外部に出力するための装置まで, 様々なモノがインターネットに接続される. その結果, 機器をインターネット上で相互に接続及び連携することによってヘテロジニアスなサービスを実現することが可能になった [3,4]. また, 開発者は構成する機器を自由に選択することで, IoT を活用した豊富なソリューションを作り出すことが可能なため, サービスの種類は益々多様化することが予想される [5].

IoT の普及により, 多くの労力や時間を要していた仕事の効率化が見込める [6]. IoT によって接続するモノは, 機器やセンサをはじめ様々であるが, その接続方法やデータ処理の仕組みはフレームワークとして提供することで, 他のサービスにも転用可能である. これは, 高効率, 高信頼性, 低コスト等の優れた IoT ソリューションの仕組みを利用して, 他のサービスを最適化できる可能性があることを示唆している. 独自の IoT をゼロから構築するのは, 労力面やコスト面で大きな負担となるが, 定型化された IoT ソリューションを活用すれば, 負担が軽減され, 短期間での導入が可能となる. その結果, 企業や組織は IoT を自社のサービスに益々導入し易くなり, 今後も爆発的に普及することが予想される [7,8].

また, IoT はクラウドサービスとの親和性が高い点もメリットとして挙げられる. IoT はインターネットを基盤としているため, ネットワーク網に展開されるサーバやストレージ等のクラウドサービスと連携することが容易である [9]. 近年では, IoT とクラウドサービスを連携するソリューションとしてデジタルツインが注目されている. デジタルツインとは, 「デジタルの双子」とも呼ばれ, フィジカル空間に存在する製品や製造設備の情報, 及びそれらのオペレーションデータをリアルタイムに収集してサイバー空間に送り, サイバー空間上にフィジカル空間と同じ状態・状況を再構築して, 仮想モデルを用いた高度なシミュレーションを行う概念である [10,11]. デジタルツインを作り出すには, フィジカル空間からサイバー空間へと送られる膨大なデータが必要となる. IoT は, デジタルツインを実現する基盤技術として機能し, エッジデータを収集する有効な手段として用いられる [12–14]. また, IoT 端末を数多く導入した大規模なシステムを実現する仕組みとして, Cyber Physical System (CPS) が挙げられる. CPS は, IoT センサからデータを収集する CPS 端末と, 収集したデータの解析を行う CPS クラウドに大別される. CPS 端末は, 他の CPS 端末と分散・協調処理を行うことで, エッジ領域のデータを収集する. 収集されたデータはクラウドサービスを通して解析を行うことで, 情報, 価値を創出し, 産業の活性化や社会問題を解決することが期待される [15,16].

一方, IoT 機器やそれらを用いたサービスが多数登場する中で, 特有のセキュリティリスクやトラフィック増大に伴うネットワーク帯域の逼迫が課題として浮き彫りになっている [17–20]. IoT 端末を用いたサービスでは, モノが所有する情報がネットワークを介して収集・分析・フィードバックされ, 様々な形で活用

される. 例えば, Society 5.0 の先行的な実現の場である, スマートシティは, 街中に設置された IoT センサや防犯カメラから収集したデータを都市 Operating System (OS) と呼ばれる連携基盤を介して分析・活用・流通させることで実現される [21]. 従って, インターネットを介して様々な機器と接続することや, プライバシに関わる機微な情報を外部とやり取りすること, クラウドサービスを活用することを前提としており, 適切なセキュリティ対策が必要である. 他方, インターネットに接続するセンサやカメラ等の IoT 機器は散在しており, 多様なデータが流通しているため, 常にサイバー攻撃のリスクに晒される可能性がある [22]. これまで主流であったファイアウォールや Access Control List (ACL) を用いたネットワーク境界でのセキュリティ対策は, 端末が散在する場面では非常に複雑化することから有効に機能しない場合がある [23]. また, 分散ネットワークにおいて, 拠点間 Virtual Private Network (VPN) やリモートアクセス VPN のような, 従来の集約型 VPN ソリューションはリモートアクセス制御において管理が煩雑化する懸念がある [24].

さらに, Ericsson, Inc. はインターネットサービスの普及やデジタル化に伴い, 2023 年末の時点で約 130 エクサバイト/月に達した世界中のインターネットトラフィックは, 2029 年には 3 倍の約 403 エクサバイト/月に達すると予測している [25]. また, Cisco Systems, Inc. は 2018 年の時点で 184 億台であった全世界のネットワーク端末数は, 2023 年までに 293 億台を超えると見込んでおり, IoT サービスの中でも特に割合を占める Machine-to-Machine (M2M) 通信は, ネットワークトラフィック全体の半数を占めると発表している [26]. エッジ端末が取得及び生成したデータの多くはデータセンターやクラウドサービスに集約される. その結果, IoT サービスを実現するために必要となる管理サーバにはトラフィックが集中することで, ソリューションのボトルネックとなることが懸念される. さらに, サービスの拡充とともにサーバの負荷が増大する傾向にあるため, 障害や可用性の担保を考慮して, 組織は運用に伴いクラウドサービスに莫大な資金を投じる必要がある.

前述の背景を踏まえ, 近年では IoT 端末をクラウドサービスで管理し, 通信データはエンド端末間で直接送受信するハイブリット Peer-to-Peer (P2P) モデルが注目を集めている [27, 28]. ハイブリット P2P モデルとは, 中央のサーバがネットワーク情報や端末を管理する Client-to-Server (C2S) モデルと, 端末間で直接データを送受信する P2P モデルを組み合わせたネットワークモデルである [29]. C2S モデルは, 中央のサーバが全ての端末を管理するため, 認証や制御が比較的容易に施しやすく, セキュリティや管理の面で有用なネットワークモデルである. その結果, 現在の IoT サービス提供基盤の主流なモデルとして世の中に普及している. 一方, C2S モデルでは送受信データが常に中央のサーバを経由するため, 冗長なトラフィックや遅延が発生する [30, 31]. また, 端末の増加に伴い, トラフィック量も増大するため, クラウドサービスの増強が必要となることから, コスト面においても課題が残る. 他方, P2P モデルは, 端末間で直接データを送受信するため, ネットワーク経路の冗長化や通信遅延を抑制可能である. しかし, P2P のアーキテクチャでは個々の端末を認証・制御する管理サーバが存在しないため, セキュリティのリスクが生ずる [32]. 故に, C2S, P2P におけるメリット, デメリットを踏まえ, 双方の利点を取り入れたハイブリット P2P モデルを採用することで IoT 端末をセキュアに保護するとともに, 効率的な通信を実現可能となる.

一方で, ハイブリット P2P モデルを用いたサービスを構築する場合, 端末を管理する中央のサーバは, 本来の提供サービスを制御する機能に加え, P2P に基づいた端末間直接通信を行うためにネットワーク情報の収集や, 端末への経路指示といった機能を準備する必要がある. 従って, 機能要件に比べて, ネットワークやセキュリティに関する非機能要件が大幅に増加する可能性がある. その結果, 開発者はクラウドサービスを増強することで, C2S モデルを IoT サービスの基盤として引き続き採用し続けることを余儀なくされる.

また, 次世代のセキュリティ対策手法としてゼロトラストセキュリティモデルが注目を集めている. ゼロトラストセキュリティモデルは, 各エンドポイントに権限を付与し, 通信毎に安全な端末であるかを確認することで, 端末間のセキュアな通信を実現するセキュリティモデルである. 組織のネットワーク内外を問わずに各エンドポイントに認証, 認可, セキュリティ構成の妥当性確認を要求する [33]. その結果, ネッ

トワーク境界のセキュリティ対策では保護することが困難であった, 情報や資源のセキュリティを確保することが可能である. また, ゼロトラストセキュリティモデルは個々の端末を対象としてセキュリティ対策を適用する. そのため, 一つのプライベートネットワークを保護する従来の境界型セキュリティモデルと比較して, 散在する IoT 端末に対しても有効に機能することが期待される [34–36].

現在では, 多くの組織がクラウドサービスを使用してデータを処理していることから, 一般的なゼロトラストセキュリティ製品では, 中央のサーバが各エンドポイントを管理する. そのため, ゼロトラストセキュリティの実現は, 提供されるクラウドサービスに依存する [37]. また, IoT 端末の開発者は, セキュリティモデルよりもサービスの実現を意識する傾向にあるため, セキュリティの優先順位が, 本来のサービス機能に比べて低くなりがちである [38]. サービスの利便性とセキュリティ対策による安全性の確保はトレード・オフの関係にあるため, 提供サービスとの共生を図りながら効果的にセキュリティ対策を導入していくことが重要となる. 従って, IoT サービスの普及に伴い, 開発者のサービス開発プロセスを支援可能なセキュアな通信フレームワークが必要である.

各 IoT 端末は分散ネットワークに存在するため, フレームワークは Network Address Port Translation (NAPT) や Internet Protocol (IP) バージョンの差異に影響されることなくエンド間通信を実現する必要がある. インターネットを介した相互通信は IP を用いて実現されており, ネットワークに接続される端末はインターネット上で一意となる IP アドレスを保持する [39, 40]. しかし, IP アドレスの管理を行う Internet Assigned Numbers Authority (IANA) は 2011 年 2 月頃, 使用可能な全ての IPv4 アドレスを Regional Internet Registry (RIR) に割り当てたと発表しており, 事実上グローバル IPv4 アドレスを全ての端末に一意に割り当てることが不可能となった [41]. そこで, 現在のインターネットには IP アドレスを全ての端末に割り当てるために, NAPT と IPv6 が導入されている. NAPT は, 広域ネットワークで利用されるグローバル IP アドレスと, Local Area Network (LAN) 等のローカルネットワークで利用されるプライベート IP アドレスを相互に変換する技術である. NAPT は, 一つのグローバル IP アドレスと複数のプライベート IP アドレスを 1 対多で変換することにより, アドレスの消費を削減するとともに, LAN 内に存在する複数の端末を同時にネットワークに接続することを可能にする [42, 43]. 一方で, NAPT は, 広域ネットワークに存在する端末から NAPT 配下の端末に対するインバウンドトラフィックを遮断する. これは NAPT 越え問題として広く知られており, インターネットにおける相互接続性の課題となっている [44–46]. また, IPv6 は, 割り当て可能なアドレス数が 2^{128} 個であり, 膨大なアドレス空間を保持しているため, 全ての端末に対して一意に IP アドレスを割り当てることが可能である [47]. しかし, IPv6 と IPv4 はパケットの構造が異なることから互換性がないため, 双方のバージョン間で相互通信を行うことは極めて困難である [48]. 一般に, NAPT 越え問題や, IPv4-IPv6 の非互換による相互運用の問題は通信接続問題と呼ばれ, IoT 端末間の直接通信を妨げる原因となっている. これまで, NAPT トラバーサルや IPv4-IPv6 間の相互通信を実現する通信接続技術が多数提案されてきたが, それぞれの技術には課題が残されている [49–51]. さらに, これら通信接続技術の詳細を開発者が理解し, サービスに組み込む必要があるため, IoT ソリューションの構築が複雑化する懸念もある.

また, IoT 端末はサービス状況に応じて, 無線アクセスポイントやセルラ網を切り替えて通信を行うことが予測される. そのため, IoT 端末のエンド間通信を実現するフレームワークは, アクセスネットワークの切り替えによらず, 継続的な通信を実現する必要がある [52, 53]. しかし, IP では, 端末の移動について考慮されていないため, 通信中に端末が移動しアクセスネットワークが切り替わると, 保持する IP アドレスが変化し, 通信が切断される [54]. 端末のネットワーク間移動を隠蔽することで, コネクションの切断を防ぐ技術を移動透過技術と呼ぶ. 代表的な移動透過技術として, Mobile IP (MIP) が存在する [55]. しかし, MIP は NAPT 越えや IP バージョンの互換性を考慮したデュアルスタックへの検討が不十分である. また, 通信に伴い, パケットの転送処理が発生するため, オーバーヘッドは増大し, 通信経路が冗長となる課題も残されている [56, 57]. 従って, 通信フレームワークは通信接続性及び移動透過性を兼ね備え, ゼロトラストセキュリティに基づいたセキュアな P2P 通信機構を包括的に提供する必要がある.

著者らの研究グループは, 通信接続性及び移動透過性を兼ね備えたセキュアな端末間接続を実現する通信

フレームワークとして CYber PHysical Overlay Network over Internet Communication (CYPHONIC) を提案している [58–62]. CYPHONIC は, NAPT トラバーサルや IPv4-IPv6 間の相互接続をサポートし, 端末移動時にも継続な通信を実現する. CYPHONIC を導入する端末は, 仮想的な Network Interface Card (NIC) を介して通信を行うことで, TCP/IP を用いるあらゆるアプリケーションに対してオーバーレイネットワークサービスを提供可能である. また, CYPHONIC はゼロトラストセキュリティの概念に基づき, 利用端末の認証と, 通信を行う双方の端末のみが復号可能な共通鍵を用いることで, エンドノード及びそれらの間で送受信されるデータを保護する. その結果, 開発者は CYPHONIC を用いることで, IoT サービス本来の機能開発に集中するとともに, セキュアなエンド間通信サービスを組み込むことが可能となる.

CYPHONIC は, CYPHONIC Daemon と呼ばれるクライアントプログラムを搭載した CYPHONIC ノードと, CYPHONIC ノードのネットワーク情報や通信を管理する 3 種類のクラウドサービスで構成される. CYPHONIC ノードにおける CYPHONIC Daemon は端末内で常駐動作し, 仮想 NIC の生成と仮想 IP アドレスの割り当て, シグナリングメッセージの交換及び処理, アプリケーションデータのカプセル化及び暗号化等の機能を提供する. また, CYPHONIC クラウドは, Authentication Service (AS), Node Management Service (NMS), Tunnel Relay Service (TRS) から成り立つ. これらのサービスは提供する機能や, アクセス頻度に応じて分離され, マイクロサービスとして運用されている. AS は, CYPHONIC ノードの認証機能を担い, オーバーレイネットワーク通信に必要な仮想 IP アドレスや端末識別子である Fully Qualified Domain Name (FQDN) を割り当てる. NMS は, CYPHONIC ノードの所属ネットワーク情報を管理し, 新規通信を検知した際は, 通信を試みる双方のノードに対して経路情報を提供する. TRS は, CYPHONIC ノードが何れも NAPT 配下に存在する場合に, 中継サービスとして機能することで, P2P トンネルの確立をサポートする. また, トランスレータとしての機能を兼ね備えており, IPv4-IPv6 間の相互通信にも対応する. トランスレータとは, エンド端末で使用可能な IP バージョンが限定されている場合に, 中継サーバが IP ヘッダを変換することで通信接続性を実現する技術である.

CYPHONIC ノードは通信を確立する前に, CYPHONIC クラウドと所定のシグナリングメッセージを交換することで相手端末との P2P トンネルを確立する. はじめに, 通信を開始する CYPHONIC ノードは, 所望の相手ノードを FQDN によって識別し, 通信経路をクラウドサービスに問い合わせる. そして, クラウドサービスから, 相手ノードと通信するためのネットワーク経路を取得し, CYPHONIC ノード間で共通鍵を直接交換する. データ送受信の際は, 双方の CYPHONIC ノードのみが知り得る共通鍵を用いてメッセージを暗号化することで, セキュアなエンド間通信を実現する. また, 端末の移動に伴い, 物理 NIC に紐付く実 IP アドレスが変化した場合も, 新たなトンネル通信を確立するとともに, 仮想 IP アドレスに従って通信を行うことで, コネクションを維持する. CYPHONIC ノードは, 常に仮想 IP アドレスに基づいて通信を行うが, 実際の通信では実 IP アドレスを用いて全てのパケットをカプセル化し, User Datagram Protocol (UDP) トンネルによる通信を行う.

また, 一般的な IoT 端末や組み込み機器, 産業用の機械は工場出荷後に既存のプログラムを改良することが困難な場合がある. さらに, 特定用途の専用サービスサーバも, 安定性を重視して稼働していることから, 機能の異なる新規プログラムの追加は, しばしば嫌悪感を抱かれ避けられる傾向にある [63]. これらの端末をサポートするべく, CYPHONIC では, CYPHONIC Daemon が担う処理機能を代行するアダプタ端末として CYPHONIC アダプタが提案されている [64,65]. 既存製品の改良が困難な端末 (以下, 一般ノード と表記) は, 隣接設置される CYPHONIC アダプタに接続することで, CYPHONIC Daemon をインストールすることなく, 我々のオーバーレイネットワークを利用可能である. CYPHONIC アダプタは, TCP/IP で用いられる標準的なプロトコルと CYPHONIC のシグナリングパターンを連携することで, 一般ノードをサポートする. IoT ソリューションの開発者は, CYPHONIC Daemon を端末にインストールし, 通信確立に伴う処理を CYPHONIC に移譲することで, P2P モデルを自身のサービスに容易に組み込むことが可能である. また, 必要に応じて CYPHONIC アダプタを用いることで, 多様な IoT 端末にも対応可能である.

一方で, 既存の CYPHONIC アダプタは一般ノードと 1 対 1 対応を想定した設計となっているため, 一般ノード一台につき, CYPHONIC アダプタを一台準備する必要がある．また, Adapter Daemon と呼ばれる CYPHONIC アダプタのプログラムはシングルスレッドでパケットを処理する簡便なシステム設計であるため, ネットワーク性能の低下が懸念される．近年では, ローカルエッジコンピューティングをはじめ, データ加工や分析等の一部の処理をネットワーク末端の IoT 端末や, その周辺領域に配置したサーバで行い, 加工されたデータのみをクラウドサービスに送信する処理形態が普及している．その結果, 一つの区画に複数の端末や産業機器を設置した IoT サービスも多数登場することが予想される．また, スマートシティや商業店舗内の監視カメラソリューションも, 複数の IoT カメラを用いる可能性があると考えられる [66–68]．従って, CYPHONIC アダプタにおいて, 複数の一般ノードの同時接続対応が求められる．さらに, 多数の一般ノードをサポートする場合, 複数の処理を同時並行的に実行する高度な並行処理を要するため, アダプタにおける各機能のマルチスレッド化が必要となる．

さらに, CYPHONIC アダプタが複数の一般ノードに対応すると, CYPHONIC を利用可能なノードの幅はさらに拡大するため, 利用者やエンド端末の台数が急増することが予測される．しかし, 既存の CYPHONIC クラウドは単一インスタンスのみでの稼働となっており, ノードの増加を想定した柔軟なスケール機能は未搭載である．CYPHONIC クラウドが過負荷や障害によって停止した場合, CYPHONIC ノードは P2P トンネルを確立することが不可能となり, 結果として IoT サービス本来の機能にも影響が及ぶ．これまで, CYPHONIC ノードのネットワーク情報管理や通信経路指示を担う NMS のみ, フェールオーバ機能を搭載していた．しかし, フェールオーバによるマスター・スレーブ方式によるサーバの切り替えは, 障害を前提とした対策手法であり, ダウンタイムが発生する．さらに, NMS の冗長化に伴い, マスターサーバ, 及び全てのスレーブサーバ情報を CYPHONIC ノードに通知するため, パケットサイズが肥大化する問題や, 複数の NMS に対して同様のシグナリングを実行するため, 処理が煩雑になる問題がある．また, 認証サービスである AS や, 中継サービスとして機能する TRS は障害許容性もとい可用性を担保する仕組みが未実装である．特に, TRS は直接通信が困難なネットワーク環境では, トラフィックを常に中継し続ける必要があるため, 負荷が他のクラウドサービスと比較して高くなることが予測される．従って, CYPHONIC クラウドの障害を未然に防ぐ仕組みや, クラウドサービス内部で発生した障害をノードに対して隠蔽する仕組みが必要である．また, CYPHONIC クラウドは, 機能毎に負荷の傾向が異なるため, 各サービスのリクエストに応じた適切なスケールアウト機能が必要である．

1.2　本研究の目的

本研究では, CYPHONIC における前述の課題に対応するべく, エンドノード, 及びクラウドサービスのそれぞれに着目し, CYPHONIC の拡張性設計を提案する．エンドノード側では, CYPHONIC アダプタの処理機能をマルチスレッド化するとともに, 複数一般ノードの同時接続をサポートする．従来の CYPHONIC アダプタは予め決められた単一の一般ノードのみを管理する．そのため, 複数の一般ノードを管理する機能や, 複数のコネクションを同時に確立して通信パケットを処理する機能は未実装である．従って, 本提案では, 複数の一般ノードをサポートするべく, 一般ノード情報を管理するための内部キャッシュを準備する．また, CYPHONIC アダプタは, 標準プロトコルと, CYPHONIC で用いるシグナリングを同時に処理するため, Adapter Daemon 内部の処理機能をマルチスレッド化する．CYPHONIC アダプタは, オーバーレイネットワーク通信の際に, 一般ノードから取得したアプリケーションデータに対してカプセル化と暗号化を施す．また, 後方通信の場合は, 送信時と逆の処理手順によって, 受信データを復号し, デカプセル化した後, 一般ノードに転送する．この時, パケット送受信時の暗号化/復号処理は, CYPHONIC アダプタの機能において, 特に処理負荷が高く, プログラムのボトルネックとなることが懸念される．そのため, 提案方式では, カプセルメッセージの処理機能に専用のワーカースレッドを割り当て, 並行処理を導入する．また, 各ワーカースレッドは非同期で実行され, それらの処理時間は OS や言語

ランタイムの負荷状態に依存する．その結果, 一般ノードから受信したパケットと, 相手ノードに送信するパケットの順序にずれが生ずるため, フラグメントデータを正しく処理できない可能性がある．例えば, TCP の場合, パケットの順序ずれによって再送処理が発生し, スループットが極端に低下することはよく知られている．また, UDP の場合は ”ゆらぎ” が発生することで, サービス品質に影響が顕著に現れる可能性がある．この問題に対処するため, 本提案ではパケット処理に関する機能を分離し, 送受信機能と処理機能を分離したパケット逐次処理スキームを導入する．検証及び評価として, CYPHONIC アダプタが複数の一般ノードを同時にサポート可能であることを確認し, 一般ノードにおける通信スループットや通信遅延から処理性能を評価する．

また, CYPHONIC クラウドにおいて, 各サービスサーバの多重化と負荷分散機能を導入したスケーラブルなインフラストラクチャ設計を提案する．既存のクラウドサービスが単一インスタンスで稼働するのに対して, 本提案では, コンテナオーケストレータを用いて, 各サービスサーバの台数を多重化したクラスタを用いる．クラスタを構成する各サーバの CPU やメモリの使用率といったメトリクスを一定間隔で収集することにより, 処理負荷に応じたオートスケーリングを実現する．さらに, クラスタの前段にロードバランサを設置することで, 個々のサーバに掛かる負荷を平準化する．一般的な Software Load Balancer (SLB) では, クライアント端末がロードバランサの IP アドレスに通信を開始し, ロードバランサが各サーバにリクエストを振り分けることで負荷分散を実現する [69–72]．この時, サーバに到達するリクエストの送信元 IP アドレスは SLB の IP アドレスとなるため, クライアント端末本来の IP アドレスを特定することが困難となる Source Network Address Translation (SNAT) が生ずる．CYPHONIC では, クラウドサービスがエンド端末から取得したネットワーク環境情報に基づいて通信経路を指示する．そのため, 送信元 IP アドレスが正しく取得されない場合, 通信経路を適切に指示することが極めて困難となる．そこで, 本提案では, クラスタの外部に Border Gateway Protocol (BGP) ルータを設け, Equal Cost Multi Path (ECMP) による等コストパスを用いた負荷分散方式を導入する．BGP ルータとクラスタを構成する各ワーカーノードは internal BGP (iBGP) としてピアリングされるため, SNAT を回避した上で負荷分散機構を実現可能である．検証として, リクエスト数に応じた各サービスサーバのオートスケーリングや, 障害発生時におけるオートヒーリングを確認する．また, ECMP による各ワーカーノードへの負荷分散と, CYPHONIC ノードが適切に P2P トンネルを確立可能であることを確認し, 提案する CYPHONIC クラウドの規模拡張性設計が有用であることを示す．

1.3　本論文の構成

本論文は次の章立てで構成される．第 2 章では, 本研究に関連する既存技術を取り上げて説明する．第 3 章では, 著者らの研究グループが提案する CYPHONIC について説明する．第 4 章では, CYPHONIC アダプタのマルチノード設計, 及び CYPHONIC クラウドの規模拡張性設計について提案手法を説明する．第 5 章では, 提案手法を採用した CYPHONIC アダプタの実装詳細と, 規模拡張性を考慮した CYPHONIC クラウドのインフラストラクチャ設計の実装について説明する．また, 実装を行なった CYPHONIC アダプタ, 及び CYPHONIC クラウドについて検証・評価を実施し, その結果について考察する．最後に, 第 6 章において, 本論文のまとめと, 今後の課題について説明する．

第2章　関連技術

2.1 Internet Protocol

Internet Protocol (IP) はパケット交換方式のコンピュータ通信ネットワークで相互接続しているシステムにおいて, データ通信の方法を定めた規約である. インターネットに接続される機器は, IP が提供する機構によって相互に通信を行う. IP は, データグラムと呼ばれるデータブロックを送信元から宛先へ転送することで通信を行う. しかし, IP はコネクションレス型のプロトコルであるため, 制御情報のオーバーヘッドが少ない一方で, 宛先へ確実にデータが到達する保証はない. コネクションレスとは, 事前にコンピュータ間でコネクションを確立せずにデータ伝送を行う通信方式である. さらに, ネットワーク通信に際して, 通信速度が向上する一方で, 十分な品質を保証しないベストエフォート特性を持つ. 故に, IP は信頼性の低いプロトコルであるため, 確実な送受信を行うためには, IP 層より上位のトランスポート層プロトコルによって通信を制御する必要がある.

また, IP の通信では IP アドレスが使用される. IP アドレスとは, インターネットにおいて端末を一意に特定するための識別子であり, ネットワーク上の位置情報と通信識別子を同時に表現している. 元来, インターネットでは Internet Protocol version 4 (IPv4) が広く利用されてきた. しかし, IPv4 アドレスは 2 進数 32 桁で表現されるため, 割り当て可能なアドレス数は 2^{32} 個に限定される. そのため, 世界人口が 70 億人を超える現在では, 全ての端末に対して一意にアドレスを割り当てることが不可能である. IPv4 の枯渇問題に伴い, 1994 年頃から次世代プロトコルとして Internet Protocol version 6 (IPv6) が検討されはじめた. IPv6 アドレスは, 2 進数 128 桁で表現されるため, 割り当て可能なアドレス数は 2^{128} 個に及ぶ. 従って, 全ての端末に対して一意にアドレスを割り当てることが可能であり, 枯渇の心配もない. しかし, IPv6 への完全な移行には時間を要するため, 当面の間, IPv4 アドレスと IPv6 アドレスが混在したデュアルスタック環境となる. 図 2.1 に, IPv4 と IPv6 のパケット構造の違いを示す. 図に示すように, IPv4 と IPv6 ではパケットの構造が異なることから互換性がないため, 相互運用性の問題が存在する.

2.1.1 Transmission Control Protocol

Transmission Control Protocol (TCP) は, インターネットプロトコルスイートの中核をなす通信プロトコルの 1 つであり, IP 層より上位のトランスポート層プロトコルとして信頼性の高いデータ通信を提供する. TCP は, Three-way handshaking と呼ばれる手続きによって, 通信前に相手端末とコネクションを確立してからデータ送受信を開始するため, 到達性が保証される. TCP 通信では, パケットをセグメントと呼ばれる小さな塊に分割し, それぞれのパケットにシーケンス番号を付与することで, 送信側と受信側がデータの送受信状況を把握する. 送信側はフラグメントと呼ばれる手続きによってパケットを受信側に分割して送信し, 受信側はパケットをデフラグメントと呼ばれる手続きによって再構築することで元のデータを復元する. その結果, パケットサイズが規定のサイズを超える場合においても, フラグメントとシーケンシャルな送受信機構によってデータの生合成を維持した通信が可能である.

また, TCP は信頼性の確保に重点を置いており, パケットの到達確認や再送処理を行う. 受信側は受信したパケットに対して確認応答を返し, 送信側は確認応答を受け取ることでパケットの到達を確認する. パケットが何らかの理由により, 到達しなかった場合や順序が入れ替わった場合には, 送信側で再送処理

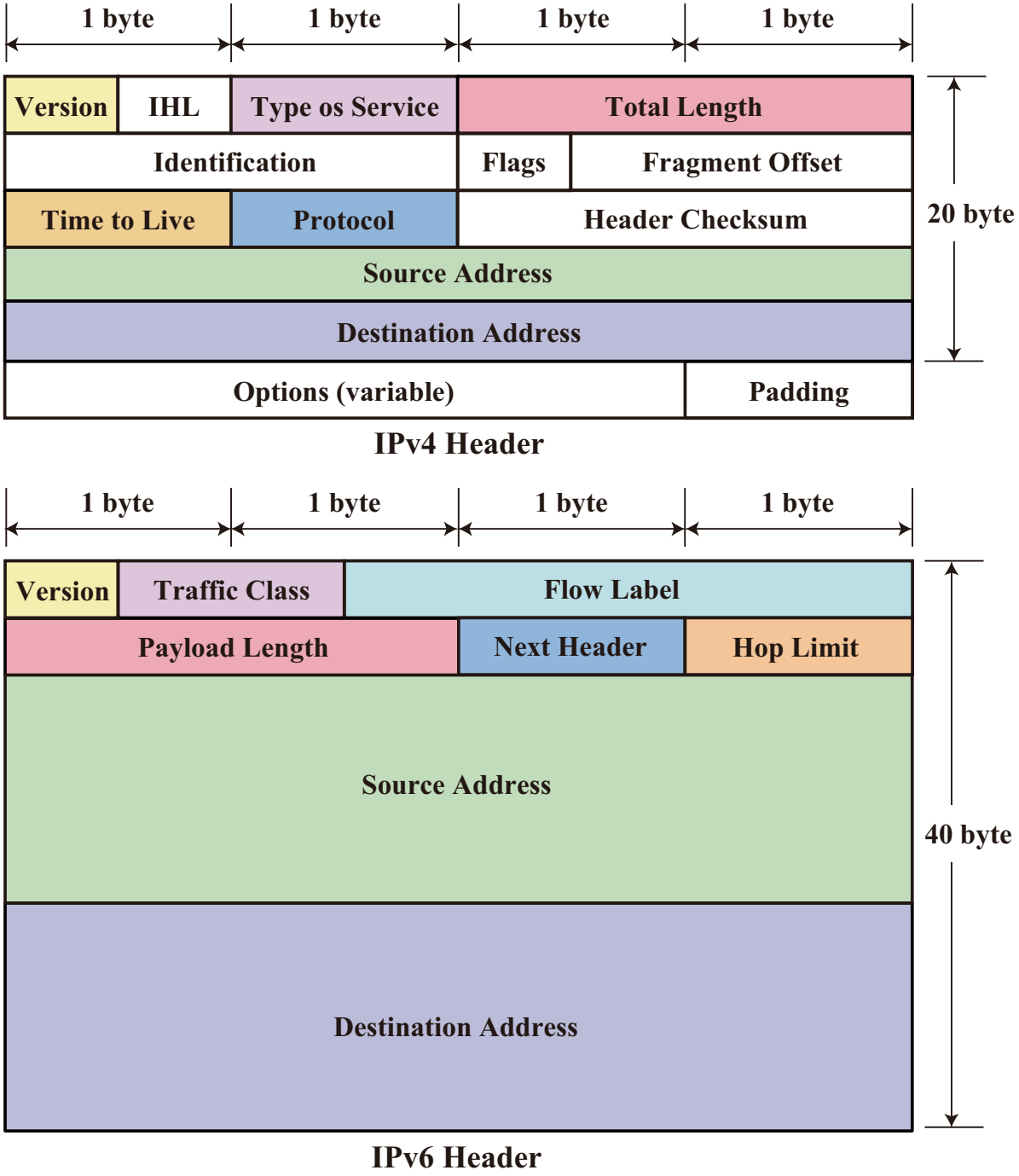


図 2.1: Protocol header: IPv4 and IPv6

を行う. 加えて, TCP はフロー制御や輻輳制御の機能を備えることで, ネットワーク帯域の状況に応じてパケットサイズを柔軟に変更する. フロー制御とは, 送信速度を受信側が処理できる速度に制御することで, ネットワークの過負荷を防ぐ仕組みである. また, 輻輳制御とは, ネットワークの輻輳を避けるために送信速度を通信環境に応じて調整する機能である. 輻輳制御機構は様々存在するが, 1988 年頃に登場した Tahoe をはじめ, Linux カーネル version 2.6.19 以降及び現在では CUBIC, その他 Reno, New Reno 等, Loss-based アルゴリズムが標準搭載される. TCP は一般的なウェブサイトの閲覧に用いる Hyper Text Transfer Protocol (HTTP) や, 電子メールの送受信に用いられる Simple Mail Transfer Protocol (SMTP), ファイル転送に用いられる File Transfer Protocol (FTP) 等, 多くの通信の信頼性が要されるインターネットアプリケーションで広く利用される.

2.1.2 User Datagram Protocol

User Datagram Protocol (UDP) は, TCP と同様にインターネットプロトコルスイートの中核をなす通信プロトコルの 1 つであり, IP 層より上位のトランスポート層プロトコルとして高速なデータ通信機構を提供する. UDP は, 信頼性よりも速度や効率性を重視したデータ通信を提供する. また, UDP は, TCP とは異なり, パケットの到達確認や再送処理機能を搭載してない. そのため, データの信頼性や整合性を保証する機能が存在しない. また, UDP はベストエフォート型の通信を行うため, 送信したデータの到達性について保証されていない.

一方で, UDP は信頼性や到達性に要する複雑な処理機構を除くことで, データ通信の速度向上や効率性を高めることが可能である. そのため, UDP はビデオストリーミングやオンラインゲームをはじめ, リアルタイム性が求められるアプリケーションやトラフィックが比較的重くなる傾向にあるサービスにおいて使用される. これらのサービスでは, 一部データの欠落が許容できる一方で, 遅延が問題視されるため, 即座に新しいデータを送信することが重要となるため, 速度や効率性に優れている UDP が用いられる. また, UDP は IP 電話や Domain Name System (DNS) 等の, データの到達性よりも通信の高速化や効率化が重視されるアプリケーションにも広く利用されている. さらに, UDP Hole Punching と呼ばれるベストエフォート特性を用いた UDP 通信技術により, 後述する Network Address Port Translation (NAPT) に介在する通信接続問題を解決することが可能である.

2.2 アドレス変換技術

従来, IP のバージョンとして IPv4 が広く普及してきた. しかし, IPv4 アドレスは数に限りがあるため, 現在ではインターネットに接続する全ての端末に一意なアドレスを割り当てることが不可能である. そのため, 1990 年代後半から 2000 年代初頭にかけて, IPv4 アドレスを節約するために, Network Address Translation (NAT) 及び Network Address and Port Translation (NAPT) が提案された. 以下に, NAT 及び NAPT について詳述する.

2.2.1 Network Address Translation

Network Address Translation (NAT) とは, プライベート IP アドレスをグローバル IP アドレスに変換する技術である. NAT の概要を図 2.2 に示す. IP アドレスの枯渇問題を受け, 現在のインターネットは, グローバルネットワークと複数のプライベートネットワークに分離することで, アドレスの消費を削減する仕組みとなっている. グローバルネットワークで用いられるアドレスはグローバル IP アドレスと呼ばれ, 広域ネットワークで利用される. また, プライベートネットワークで用いられるアドレスはプライベート IP アドレスと呼ばれ, 主に家庭内や社内で用いられる. プライベート IP アドレスは同一プラ

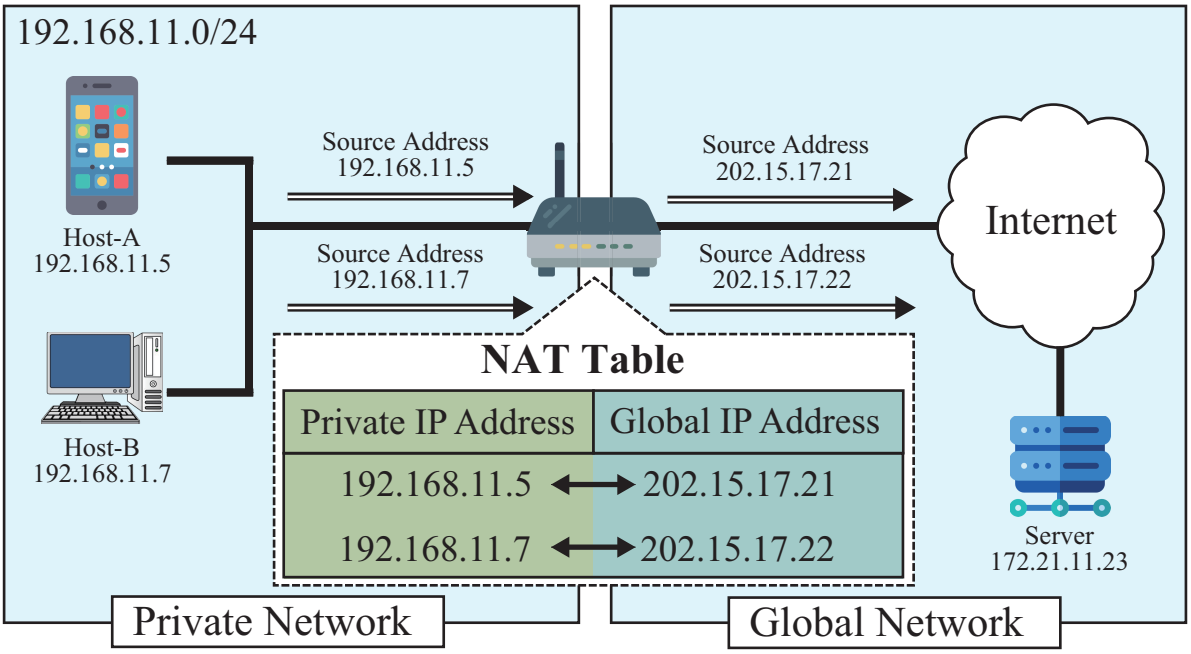


図 2.2: Overview of NAT

イベートネットワーク上でのみ一意に定まる. 通常, 端末は Local Area Network (LAN) と呼ばれるプライベートネットワークに接続され, プライベート IP アドレスを取得する. そしてインターネットにアクセスする際に, NAT を用いて一意なグローバル IP アドレスに変換することでネットワーク接続を可能にしている. 一方, グローバル IP アドレスは一つしかないため, プライベート IP アドレスを使用する端末がネットワーク内に複数ある場合, それらの端末は同時にインターネットに接続することが困難である.

2.2.2　Network Address Port Translation

Network Address Port Translation (NAPT) は, アドレス変換の際, 同時に端末が利用するポート番号も変換する技術である. NAPT の概要を図 2.3 に示す. NAT は, グローバル IP アドレスが一つしかないため, プライベート IP アドレスを使用する端末がネットワーク内に複数ある場合, それらの端末が同時にインターネットに接続することは困難である. また, NAT はグローバル IP アドレスとプライベートアドレスを 1 対 1 で変換するため, IP アドレスの枯渇問題は本質的に解決されていない. NAPT はアドレス変換の際, IP アドレスに加え, 端末が利用する Transmission Control Protocol (TCP) や User Datagram Protocol (UDP) のポート番号を同時に変換する. そのため, NAPT は単一のグローバル IP アドレスを複数のプライベート IP アドレスで併用することが可能であり, プライベートネットワーク内の複数の端末は, 同時にインターネットへの接続が可能となる. LAN に存在する端末がインターネットに向けて通信を開始すると, ゲートウェイとして介在する NAPT ルータがその通信パケット内の送信元 TCP/UDP ポート番号をユニークなエフェメラルポート番号に書き換える. そして, 送信元のプライベート IP アドレスをグローバル IP アドレスに変換し, ポート番号と一緒にインターネットへ送り出す. 今日では, NAPT の機能が一般のルータに組み込まれており, 広く利用されている.

また, NAPT は IPv4 アドレス削減の他, グローバルネットワークから隠蔽されたプライベートネットワークを作成するため, セキュリティが向上するというメリットがある. さらに, IP フィルタリングやポートフィルタリング等を設定できることから, ファイアフォールとしても機能する. 一方で, グローバルネットワークに存在する端末は, NAPT による変換前のポート番号を知る術がないため, プライベートネット

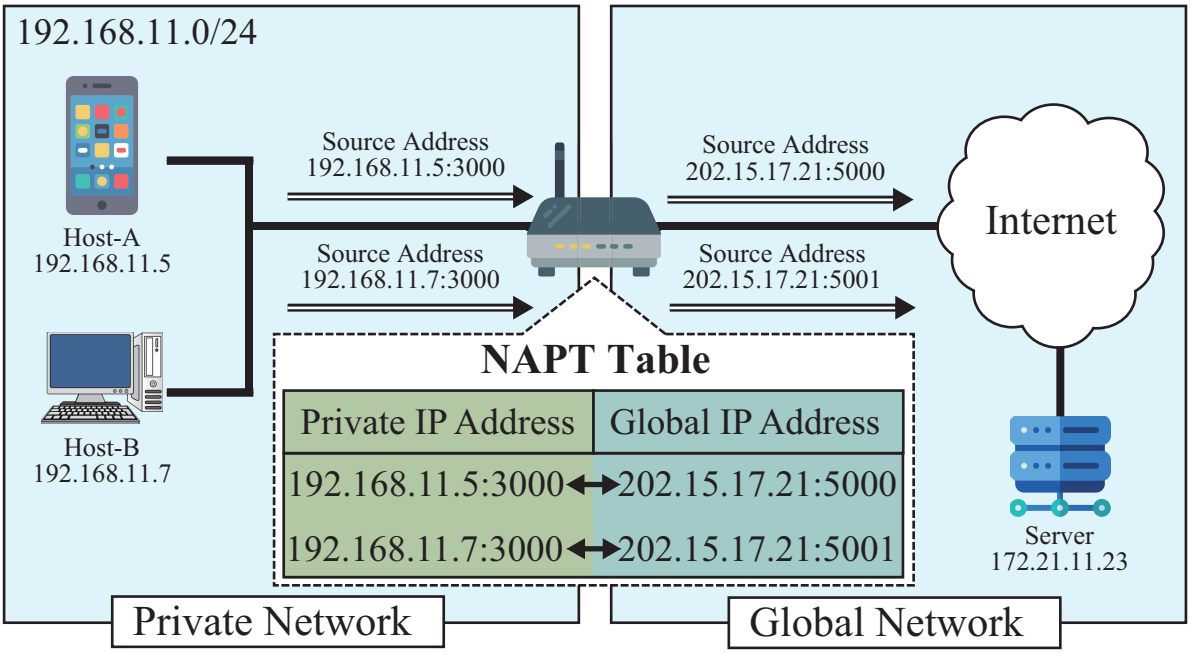


図 2.3: Overview of NAPT

ワーク内の端末へ通信を開始することが極めて困難になる．これは, NAPT 越え問題と呼ばれ, アドレスとポート番号を同時に変換する NAPT において, 避けることは困難である．NAPT 越え問題は, 外出先等, LAN 外からアクセスする場合や, 端末の NAPT を跨いだ通信時に障害となる．

NAPT は, 変換の際のマッピングとフィルタリングの規則より Full Cone NAPT, Restricted NAPT, Port Restricted NAPT, Symmetric NAPT の 4 種類に分けられる．マッピングの規則とは, NAPT 配下の端末が NAPT 外部と通信した際にマッピングテーブルに保持されるエントリの変換規則である．フィルタリングの規則とは, NAPT 外部から NAPT 配下の端末に宛てられたパケットのフィルタリング規則である．Cone 型 NAPT では, 通信を行う外部端末の宛先に関わらず, 単一のエントリを作成する．一方, Symmetric 型 NAPT では, 外部端末の宛先毎にエントリを生成するため, エントリが存在する宛先からのアクセスのみを受け入れる．

2.3　通信接続技術

現在のインターネットには, NAPT 配下の端末に対して, 外部ネットワークから通信を開始することが極めて困難となる, NAPT 越え問題が存在する．また, IPv4 アドレスと, 導入が進む IPv6 アドレスは双方のパケット構造が異なるために互換性がなく, 相互運用が困難な問題が存在する．これらは通信接続問題と呼ばれる．通信接続性を担保するためには, NAPT 越え問題を解決する技術, 及びデュアルスタック環境において相互通信を可能にする技術が必要となる．NAPT 越え問題を解決する技術は, 一般に NAPT Traversal と呼ばれる．

代表的な NAPT Traversal として UDP Hole Punching, Session Traversal Utilities for NAPT (STUN), Traversal Using Relays around NAPT (TURN), Interactive Connectivity Establishment (ICE) が存在する． また, デュアルスタック環境において相互通信を可能にする技術として DualStack, Tunneling, Translator が存在する．以下セクションにて, これら通信接続技術の詳細を説明する．

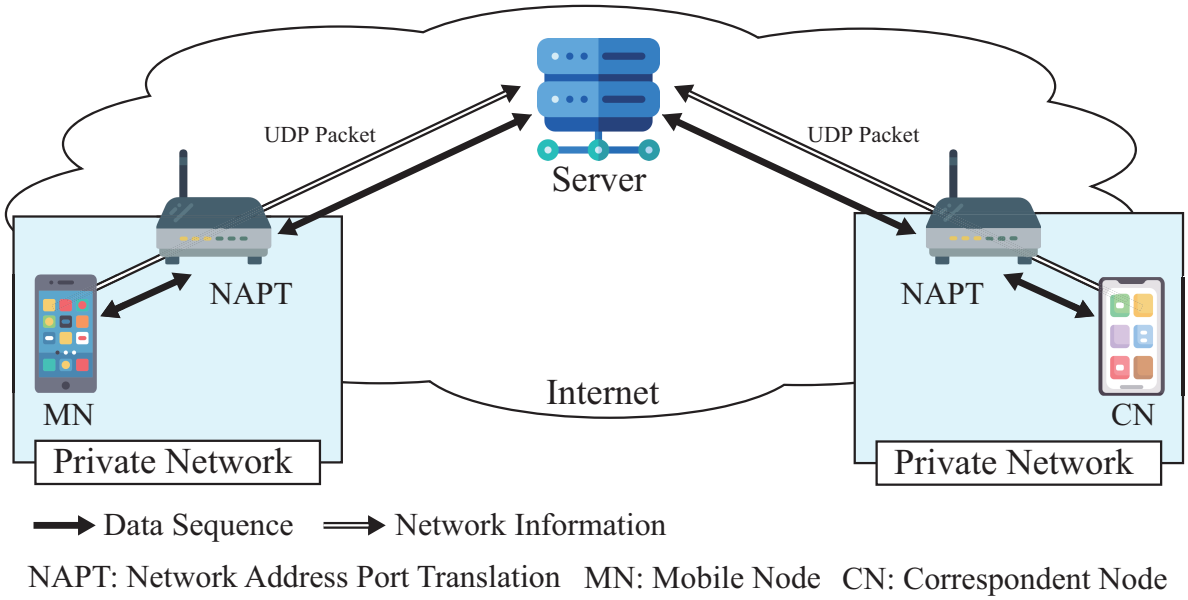


図 2.4: Overview of UDP Hole Punching

2.3.1　UDP Hole Punching

UDP Hole Punching の動作概要を図 2.4 に示す. UDP Hole Punching では, 内部から通信を行なった後, 一定期間のみ外部ネットワークからの通信を許可するという NAPT の性質を利用する. NAPT 配下に存在する端末が外部ネットワークへ通信を開始する際, NAPT はパケットの IP アドレスとポート番号を変換する. その際, NAPT は相手ノードからの応答を配下の端末へ転送可能とするため, 変換した IP アドレスとポート番号の情報をマッピングテーブルに保持する. マッピングテーブルに存在する IP アドレスとポート番号に宛てたパケットを受信した場合, NAPT は変換情報に基づき, パケットを NAPT 配下の端末へ転送する. しかし, マッピングテーブルは一定期間通信が行われない場合削除されるため, UDP Hole Punching は一定間隔で UDP パケットを送信し続けることでマッピングテーブルを維持する. これにより, 外部に存在する端末から通信を開始することが可能となり, 通信接続性を実現する. しかし, 両端末が NAPT 配下に存在する場合, UDP Hole Punching を行なっても通信接続性を実現することは困難である. これは, NAPT 配下の端末は, NAPT が生成した変換後のポート番号を互いに知ることが極めて困難なためである.

2.3.2　Session Traversal Utilities for NAPT

UDP Hole Punching では両端末が NAPT 配下に存在する場合, 通信接続性を実現することが極めて困難である. これは, NAPT によって生成された変換情報を, 互いの端末は知る術がないためである. Session Traversal Utilities for NAPT (STUN) は STUN サーバをグローバルネットワークに設置して, 双方の変換情報を記録し, 相手端末に通知することで通信接続性を実現する軽量なクライアントサーバ型プロトコルである. STUN の動作概要を図 2.5 に示す. STUN をサポートするクライアント (以下, STUN クライアント と表記) は一般に, Voice over Internet Protocol (VoIP) やインスタントメッセージ等のアプリケーションが有するプロトコルライブラリに含まれる. STUN クライアントは NAPT による IP マスカレードが行われるプライベートネットワーク内で動作し, 通信を開始する前に STUN サーバに STUN Binding Request を UDP で送信する. STUN サーバは双方の STUN クライアントのそれぞれのリクエストに対し, 他方のグローバル IP アドレスとポート番号を STUN Binding Response によって返す. こ

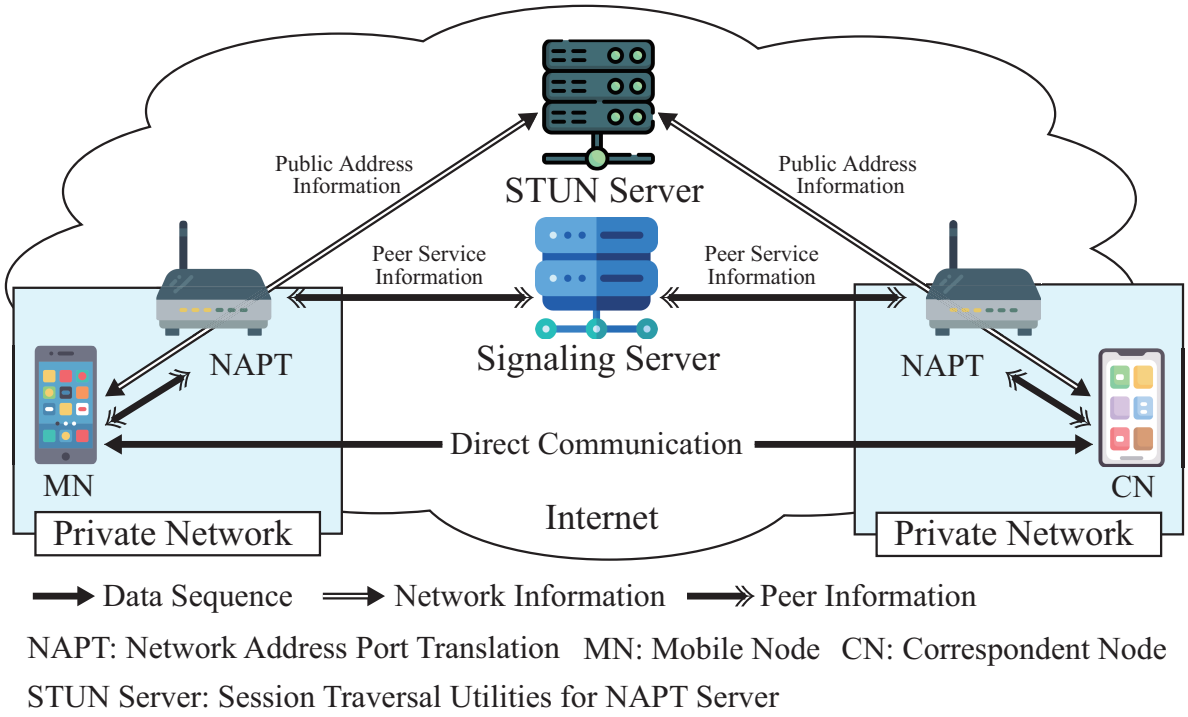


図 2.5: Overview of STUN

れにより, STUN クライアントが通信相手の変換情報を相互に得るため, 両端末が NAPT 配下に存在する場合でも通信接続性の実現が可能となる. しかし, STUN において, Symmetric 型 NAPT では STUN サーバに変換情報を通知しても, STUN サーバからのアクセスのみを許可する. そのため, 何れかの端末が Symmetric 型 NAPT 配下に存在する場合, STUN では通信接続性を実現することが困難である.

2.3.3　Traversal Using Relays around NAPT

STUN では, 両端末のどちらかが Symmetric 型 NAPT 配下に存在する場合, 通信接続性を実現することが極めて困難である. この問題を解決し, Symmetric 型 NAPT が存在する場合でも, NAPT 越えを実現する NAPT Traversal として Traversal Using Relays around NAPT (TURN) が提案されている. TURN の動作概要を図 2.6 に示す. TURN はグローバルネットワークに存在する TURN サーバが, 端末間の通信を中継することで Symmetric 型 NAPT 配下に存在する端末との通信をサポートする. TURN をサポートするクライアント (以下, TURN クライアント と表記) は TURN Allocate Request により TURN サーバに対して認証を行う. 認証に成功すると TURN Allocate Success Response により, TURN サーバが割り当てたリレーに用いるための IP アドレスとポート番号を XOR-RELAYED-ADDRESS として付与し, NAPT のグローバル IP アドレスを XOR-MAPPED-ADDRESS として付与する. TURN では TURN クライアントと TURN サーバ間で送受信するデータはポート番号 3478 の通信の中でカプセル化される. TURN の Send and Data メカニズムでは, クライアントは Create Permission Request により, 相手端末との通信開始を TURN サーバに対して依頼する. 相手との通信が許可されると Create Permission Response にて, 相手のグローバル IP アドレスとポート番号が通知される. TURN サーバから Create Permission Response を受けると XOR-PEER-ADDRESS に相手のグローバル IP アドレスとポート番号を含めて TURN サーバに Send Indication を送信する. TURN サーバがクライアントから Send Indication を受信すると, 相手先とのリレーに用いる XOR-RELAYED-ADDRESS を送信元として相互に UDP パケットを転送する. 両端末が前述の手順を相互に行うことで, TURN サーバを介し

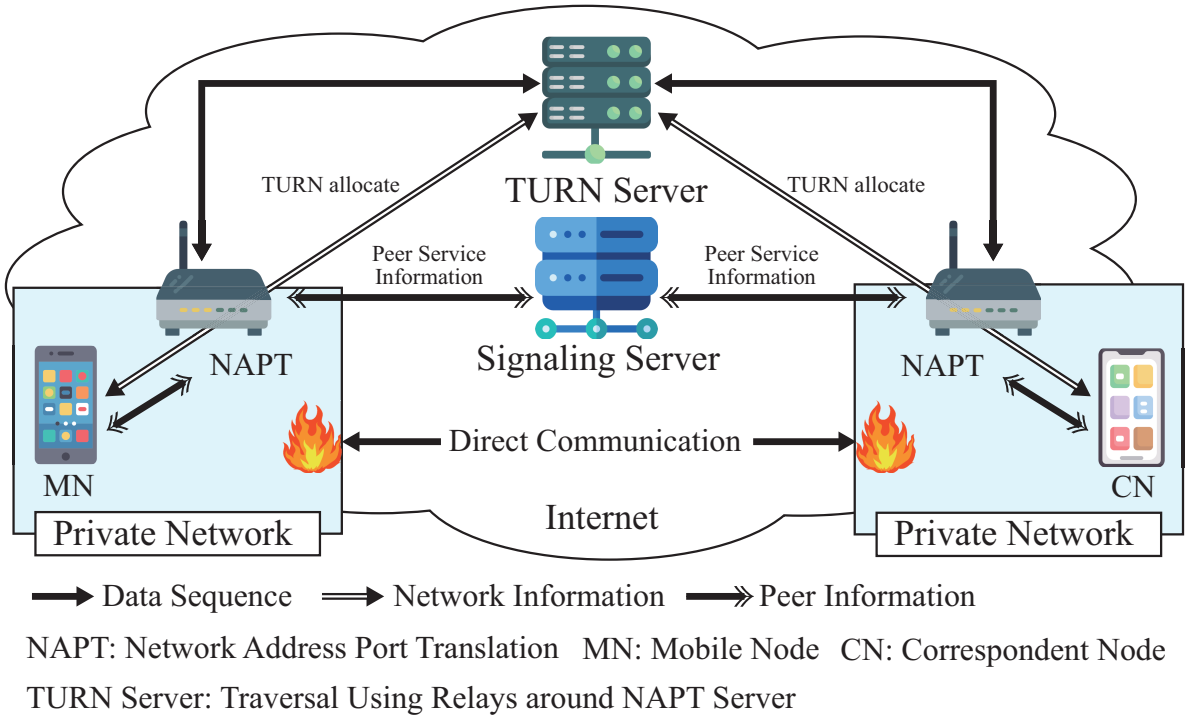


図 2.6: Overview of TURN

たリレー通信によって, Symmetric 型 NAPT が存在する場合でも通信接続性を実現可能である. しかし, TURN は通信の際に, 常に TURN サーバを経由した通信が必要となるため, 経路冗長化問題を抱える.

2.3.4　Interactive Connectivity Establishment

TURN では, TURN サーバが両者の間で中継処理を行うことで通信接続性を実現可能であるが, 通信経路が冗長になるという問題が存在した. Interactive Connectivity Establishment (ICE) は STUN や TURN といった複数の NAPT Traversal を包括した通信接続技術であり, 端末が存在するネットワーク環境に応じて適切な NAPT 越えを実現する. ICE の動作概要を図 2.7 に示す. ICE をサポートするクライアント (以下, ICE クライアント と表記) は TURN サーバに TURN Allocate Request を送信し, Candidate と呼ばれる通信相手の候補と, 自身のネットワーク情報を収集する. ICE クライアントは収集した Candidate から Session Description Protocol (SDP) と呼ばれる, ストリーミング送信を開始する際に, 必要なパラメータを伝達するためのデータを生成する. 両者が Candidate から SDP を生成すると, Session Initiation Protocol (SIP) , Web Real Time Communication (WebRTC) , eXtensible Messaging and Presence Protocol (XMPP) 等のプロトコルによって双方で Candidate の交換を行う. クライアントが通信相手から SDP を受信すると, 取得した全ての Candidate に対して STUN Binding Update を送信して接続確認を行う. これにより, 双方の端末は相手端末と通信を行う際に, NAPT の種類に応じて TURN サーバを経由して通信するか, 直接通信が可能かを判別可能となる. 従って, 端末が存在するネットワークにおいて, 最適な NAPT 越えの実現が可能となる. しかし, ICE は複数の NAPT Traversal を用いるため, 各技術の実行に伴う通信時間等のオーバーヘッドやシグナリングコストが増大する. 加えて, ICE では後に述べる, 移動透過への検討がされていないため, モバイル端末での実用には向いていない.

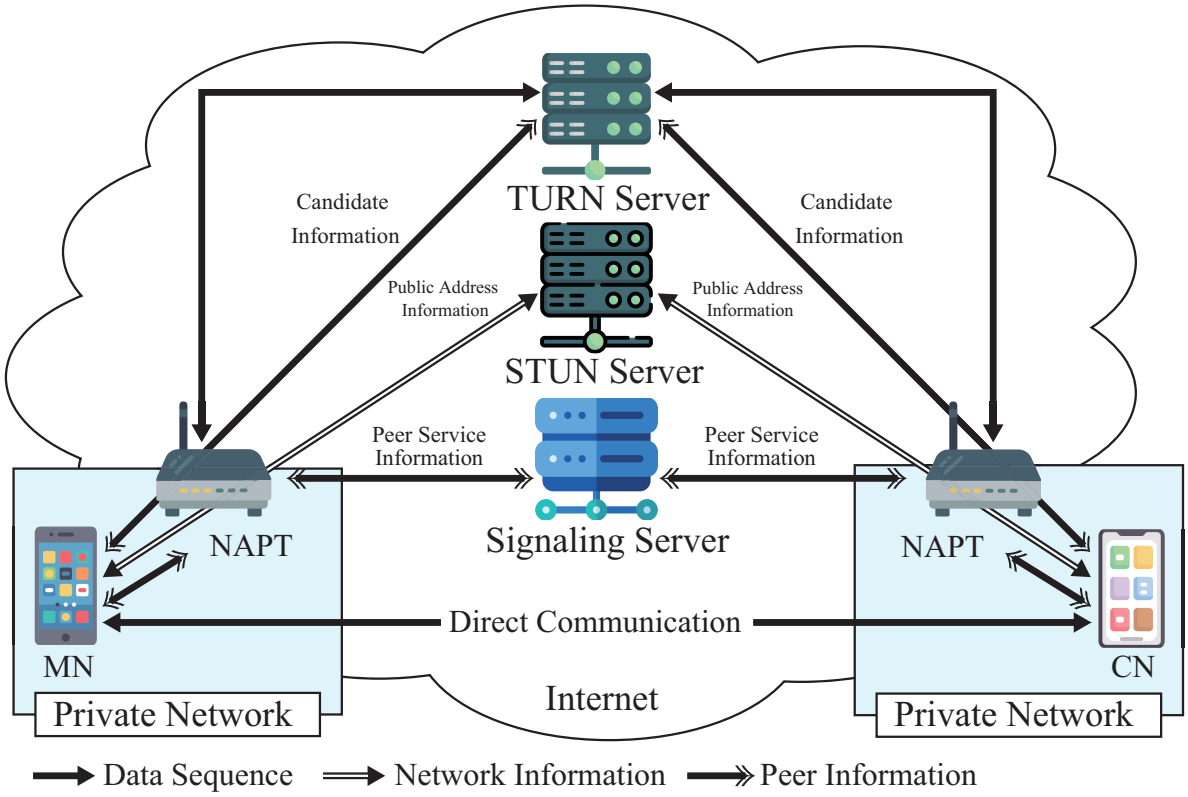


図 2.7: Overview of ICE

2.3.5　DualStack

IPv4 アドレスの枯渇問題に伴い, IPv6 アドレスの導入が進む現在, 端末は他のあらゆる端末と相互に通信するため, IPv4 アドレスと IPv6 アドレスの両方に対応する必要がある. DualStack の概要図を図 2.8 に示す. DualStack は単一機器に IPv4 と IPv6 の, 仕様の異なる 2 つのプロトコルスタックを共存させる技術である. 端末が IPv4/IPv6 の両バージョンに対応するため, それぞれの通信を行うソフトウェアスタックを用意することで, 通信相手の IP バージョンに応じた通信を行うことが可能となる. 従って, 端末を IPv4 と IPv6 の両方のアドレスに対応させることで, 通信接続性を実現し, IP バージョンの異なる双方のネットワークに存在する端末は相互に接続が可能である. しかし, DualStack では, IPv4 と IPv6 の両 IP バージョンへの対応を行うために, 端末に特定の改良が必要となる. 加えて, 一部の端末では特定の IP バージョンに対応することが困難であり, DualStack の実装が困難な問題が存在する.

2.3.6　Translator

DualStack では, 一部の端末で IPv4/IPv6 の両 IP バージョンへの対応が困難な問題が存在した. Translator は端末が DualStack を実装しておらず, 特定の IP バージョンのみしか扱えない場合に, 中継サーバが IP ヘッダを変換することで通信接続性を実現する技術である. Translator の概要図を図 2.9 に示す. 例えば, IPv4 のみを扱う端末から IPv6 のみを扱う端末へ通信を開始する場合, 中継サーバが事前

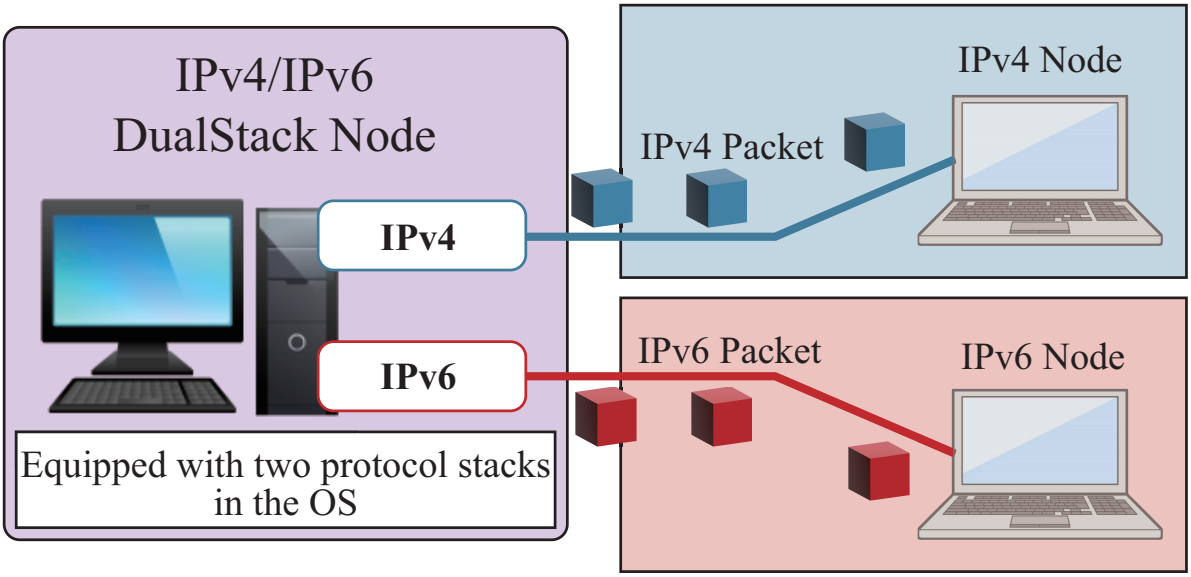


図 2.8: Overview of DualStack

に端末の IPv4 アドレスと IPv6 アドレスの経路情報を保持しておくことで, IPv4 ノードは中継サーバを経由して, IPv6 ノードとの通信が可能となる. 中継サーバは主に Proxy 方式, Network Address Port Translation-Protocol Translation (NAPT-PT) 方式, Transport Relay Translator (TRT) 方式によって実装される. Proxy 方式は Translator がアプリケーション毎に送信元の代理となって通信を行う方式である. NAPT-PT 方式は, Domain Name System-Application Layer Gateway (DNS-ALG) 機能により, DNS と連携し, プロトコルのアドレスやポート番号を動的に変換することで IPv4 と IPv6 間を相互通信する方式である. TRT 方式は, トランスポート層でセッションを傍受し, TCP や UDP 間通信の代理となって各ネットワークの送信先へ通信する方式である. Translator は, あらゆる端末を短期間で IPv6 に対応させることが可能である一方で, Security Architecture for Internet Protocol (IPsec) や SIP 等の一部のプロトコルは IP ヘッダを変換することが困難な問題が存在する.

2.3.7　Tunneling

Tunneling は両端末が同一の IP バージョンで通信が可能である一方, ネットワーク経路上に存在するルータが特定の IP バージョンのみしか扱えない場合において, カプセル化を行うことで通信接続性を実現する技術である. カプセル化とは, パケット全体を別の通信プロトコルに埋込んで通信する技術である. Tunneling の概要図を図 2.10 に示す. 例えば, 両端末は IPv6 でデータの送受信を行いたいが, 経路上に存在するルータが IPv4 パケットしか扱えない場合, IPv6 パケットを IPv4 パケットにカプセル化して IPv4 ネットワーク上で IPv6 を使用する, IPv6 over IPv4 通信を行う. また, 両端末が IPv4 でデータの送受信を行いたいが, 経路上に存在するルータが IPv6 パケットしか扱えない場合は, IPv4 パケットを IPv6 パケットにカプセル化して IPv6 ネットワーク上で IPv4 を使用する, IPv4 over IPv6 通信を行う. これにより, 宛先でパケットがデカプセル化される際に, 中身のデータを取り出すことで, 送信元端末から発信された本来の IP バージョンでパケットの送受信を行うこと可能となる. しかし, Tunneling は NAPT 越え問題を考慮していないため, IPv6 ネットワーク上の端末から IPv4 プライベートネットワーク上の端末へ通信を開始することが困難である.

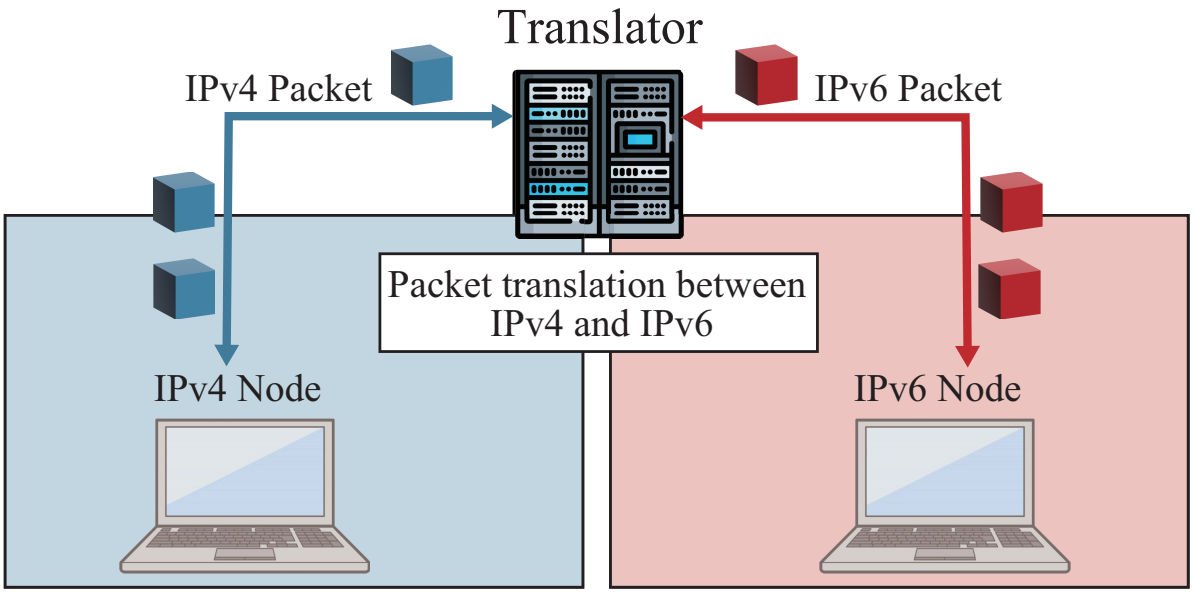


図 2.9: Overview of Translator

2.4　移動透過技術

近年の端末は複数の無線インターフェースを実装しており, ネットワークにアクセスする際にインターフェースを切替えて利用することが可能である. セルラ通信は, Wi-Fi よりも通信範囲が広く, 建物等の障害物にも強い. 一方で, 通信可能容量が決められており, 利用量が増えると帯域制限がかかるというデメリットがある. そのため, 屋外ではセルラ通信, 屋内では Wi-Fi といった通信手段を使い分けて利用することで, セルラ通信のトラフィックを分散させる, トラフィックオフローディングが必要である. 端末は, トラフィックオフローディングに伴い, 接続ネットワークを切り変えて利用することが予想される. しかし, IP では, 端末の移動について考慮されておらず, 通信中にネットワークの切り替えが発生すると, 既存の通信セッションを維持することが困難となる. これは, IP アドレスが, 端末の通信識別子とネットワーク上の位置情報を同時に管理しているためである. IP は, IP アドレスを用いて通信相手を識別するため, 端末の移動に伴い IP アドレスが変化すると, 通信フローを識別不能となり, 通信が切断される. また, IP は特性上, データの到達性を保証しないため, IP 層より上位のトランスポート層によってコネクションを確立して通信を行う. しかし, トランスポート層のプロトコルである TCP や UDP は, IP アドレスを用いて接続を管理するため, 通信中に移動が発生すると, コネクションが切断される. このような, 端末の移動に伴う IP アドレスの変化が要因で, コネクションを継続し続けることが困難な問題を移動透過問題と呼ぶ. 移動透過問題を受け, ネットワーク切り替えに伴うコネクションの切断を防ぐ移動透過技術がこれまでに多数提案されてきた.

代表的な移動透過技術として Mobile IP (MIP) が存在する. MIP は主に IPv4 ネットワーク用の Mobile IPv4 (MIPv4) と, IPv6 ネットワーク用の Mobile IPv6 (MIPv6) が検討されている. さらに, IPv4/IPv6 ネットワークを同時に利用可能な Dual Stack Mobile IPv6 (DSMIPv6) や Proxy Mobile IPv6 (PMIPv6) も提案されている. また, 近年では SIP を応用した SIP Mobility と呼ばれる移動透過技術や, ネットワークアドレスに変わる通信識別子としてコネクション ID によってセッションフローを管理する Quick UDP Internet Connections (QUIC) が登場している. 以下セクションにて, これら移動透過技術の詳細を説明する.

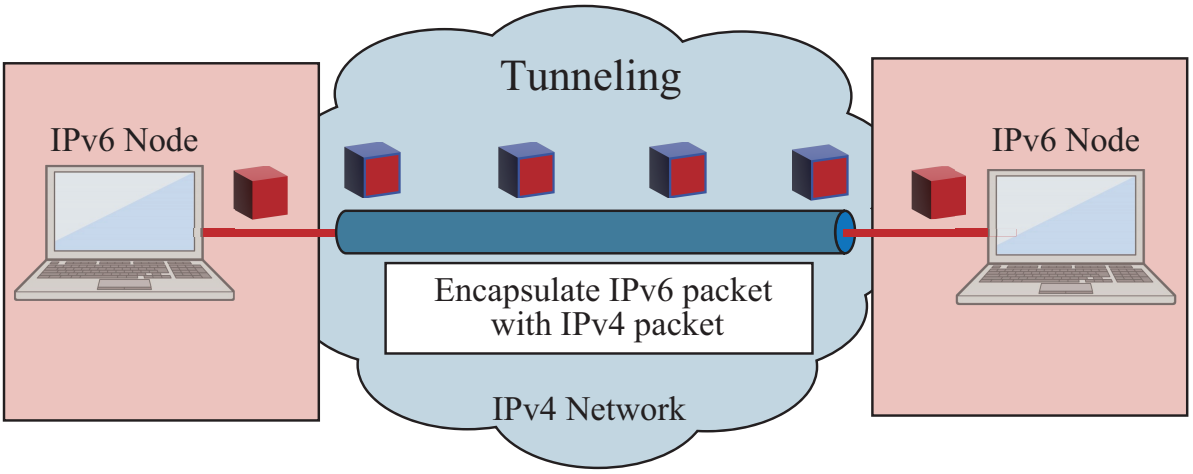
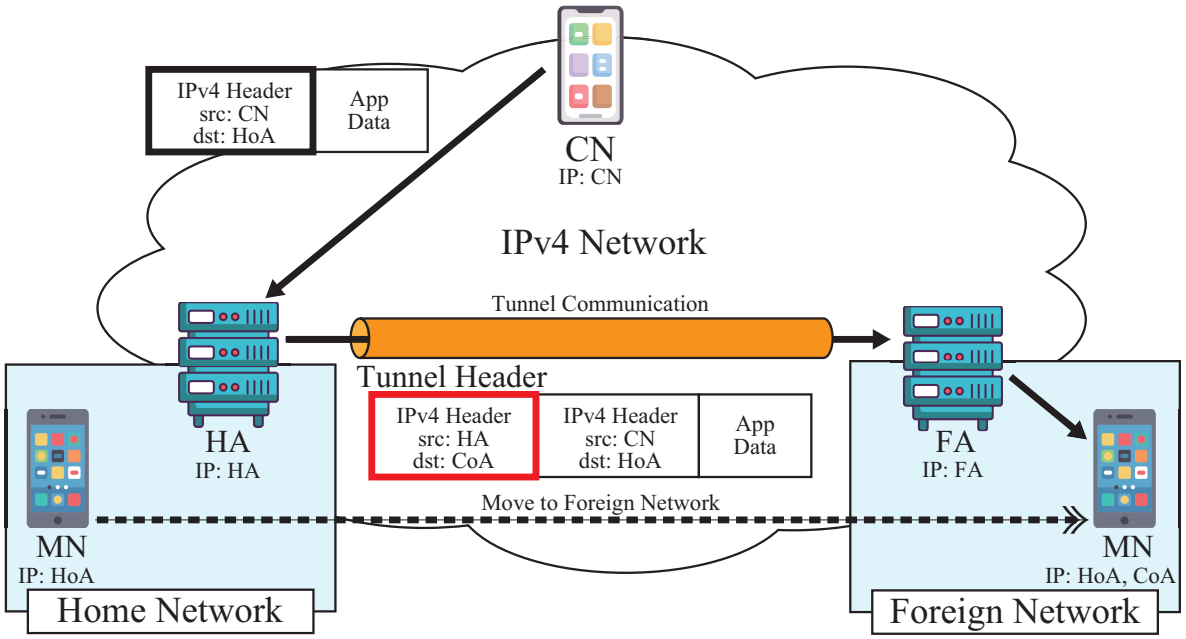


図 2.10: Overview of Tunneling

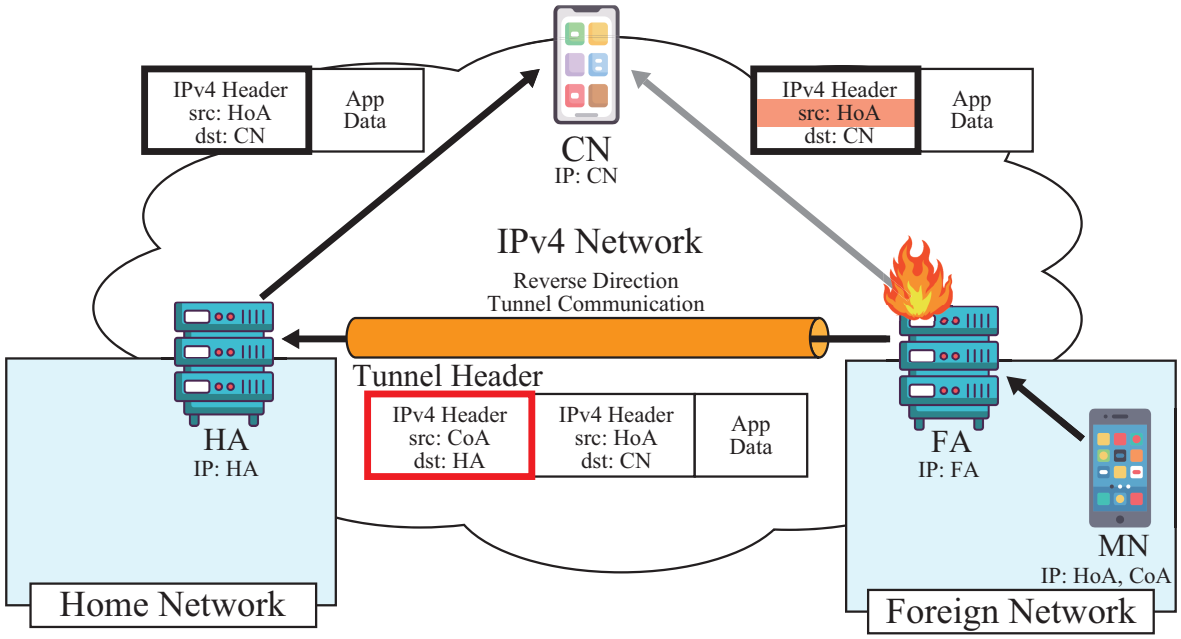
2.4.1　Mobile IPv4

MIPv4 の概要図を図 2.11 に示す．MIPv4 ではネットワーク上に Home Agent (HA), Foreign Agent (FA) と呼ばれる機器を設置し, それらを介して移動ノードである Mobile Node (MN) とその相手ノードである Correspondent Node (CN) が通信を行う．IP アドレスの位置情報と通信識別子が同時に表現されることは, ネットワーク移動に伴うコネクション切断につながる．MIP では IP が持つ位置情報と通信識別子をそれぞれ, 移動に伴って変化する Care-of Address (CoA) と不変な Home Address (HoA) に分離することで, 移動に伴う IP アドレスの変化を隠蔽し, 移動透過性を実現する．MIPv4 において, MN はホームネットワーク内に設置される HA から HoA を取得し, CN に通知する．MN はホームネットワークに存在するとき, HoA を送信元 IP アドレスとして通信を行い, CN は HoA を宛先 IP アドレスとして通信を行う．MN が移動先ネットワーク (以下, 訪問先ネットワーク と表記) に移動すると, FA から CoA を取得する．次に, 元居たホームネットワークの HA に対して, HoA と訪問先ネットワークで取得した CoA の対応関係を通知するため, Binding Update Message を送信する．HA は Binding Update Message を受信すると, Binding Cache テーブルに MN の訪問先ネットワーク情報を登録する．これにより, HA は MN の訪問先ネットワーク情報を把握することが可能となる．その後, CN から MN へのパケットは, HA へ送信された後, 送信元 IP アドレスを HA, 宛先 IP アドレスを CoA とするパケットでカプセル化され, トンネル通信によって訪問先ネットワークに存在する MN へ転送される．MN はパケットを受信するとトンネルヘッダをデカプセル化して CN が MN の HoA 宛に送信したパケットを取得可能である．これにより, CN は移動前と同様に MN のアドレスを HoA として認識可能なため, 通信フローを識別してコネクションを継続することが可能となる．

しかし, MN が保持している HoA は本来, 訪問先ネットワークで有効なものではない．そのため, MN が訪問先ネットワークから CN にパケットを直接送信する場合, FA のイングレスフィルタリングによって破棄される可能性がある．イングレスフィルタリングとは, 内部ネットワークからルータに入ってくるパケットに対するフィルタリングであり, 不正なパケットをブロックする目的で設定される．また, CoA を送信元 IP アドレスとする場合, CN が認識するアドレスが変化するため, 移動透過性を実現することは不可能である．これらの問題を解決するため, 逆方向トンネルを用いて, 一度 HA へパケットを送信し, CN へ中継転送する必要がある．その結果, MIPv4 では常に HA を介した通信を行うため, 経路冗長化問題を抱える．



(a) Send Packet: CN to MN



(b) Send Packet: MN to CN (Reverse direction)

MN: Mobile Node　CN: Correspondent Node　HA: Home Agent　FA: Foreign Agent
HoA: Home Address　CoA: Care-of Address

図 2.11: Overview of Mobile IPv4

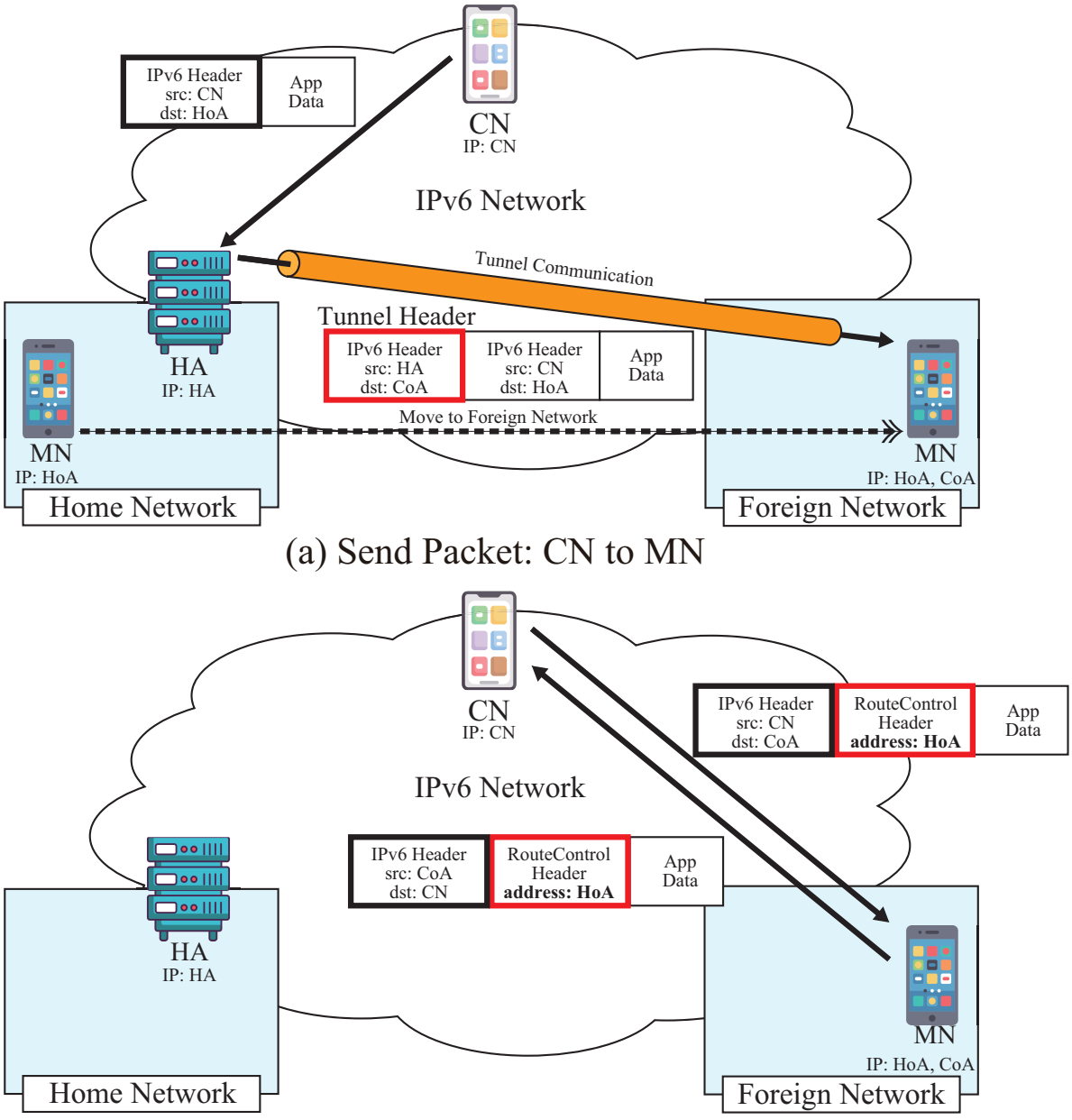
2.4.2 Mobile IPv6

MIPv4 では MN と CN が通信を行う際に, 常に HA を介する必要があるため, 経路冗長化問題が存在した. MIPv6 は MIPv4 を IPv6 ネットワークに対応させた移動透過技術であり, 経路最適化が検討されているため, 冗長な経路を排除することが可能である. MIPv6 の概要図を図 2.12 に示す. MIPv6 は MIPv4 をベースとしているため, MN が保持する HoA を用いた通信方法や, HA を経由したトンネル通信はそのまま引き継がれている. 一方で, IPv6 の場合, 全てのアドレスがグローバル IP アドレスを所持可能なため, NAPT の概念は存在しない. 故に, MN の訪問先ネットワークには FA が存在しないため, CoA は Stateless Address Autoconfiguration (SLAAC) 等の IPv6 アドレス自動生成機能, または Dynamic Host Configuration Protocol for IPv6 (DHCPv6) によって取得される. MN は移動後に, Binding Update Message を送信すると HA との間に双方向トンネルが構築され, HA を経由した送受信を行う. また, イングレスフィルタリングによってパケットが破棄されてしまう心配はないため, MN から CN へのパケットは直接送信することが可能である. 加えて, Binding Update Message を HA の他, CN に対しても送信することで, CN 自身が MN のホームネットワーク, 及び訪問先ネットワーク情報を取得可能である. これにより, 経路最適化を行うことが可能なため, MN と CN 間の通信は IPv6 の拡張ヘッダとして定義されている経路制御ヘッダを用いて, 直接通信を行うことが可能となる.

一方で, CN は宛先 IP アドレスとして MN の CoA を指定するため, Binding Cache テーブルに不正な CoA を登録された場合に, セッションハイジャックが発生する恐れがある. MIPv6 ではセッションハイジャックを防ぐために, 経路最適化を行う際, Return Routability と呼ばれる手続きを行う. Return Routability のシーケンスを図 2.13 に示す. Return Routability を行う際, MN は HoA を送信元とした Home Test Init パケットを HA 経由で CN へ送信し, CoA を送信元とした Care-of Test Init パケットを直接 CN へ送信する. CN は Home Test Init パケット, 及び Care-of Test Init パケットを受け取ると, Home Test パケットを HoA 宛に HA 経由で送信し, Care-of Test パケットを CoA 宛に直接送信する. MN は Home Test パケット, 及び Care-of Test パケットに含まれる Home Keygen Token と Care-of Keygen Token を元に認証鍵を生成し, Binding Update をハッシュ化して CN へ送信する. CN は受信した認証データを検証し, 完全性が確保されている場合に ACK を MN へ応答する. 以後, MN と CN 間の通信は宛先 IP アドレス CN, CoA をそれぞれ指定してセキュアな送受信が可能となる. MIPv6 は, 冗長な経路を排除可能であるが, 端末が IPv6 に対応していることを前提としている. そのため, 元来使用されている IPv4 には対応しておらず, MIPv4 と MIPv6 では, 双方独立した技術となっている.

2.4.3 Dual Stack Mobile IPv6

MIPv4 は移動透過性を実現可能であるが, 経路が冗長になるという問題があった. また, MIPv6 では冗長な経路を排除可能であるが, 端末は IPv6 に対応している必要があった. DSMIPv6 は IPv4 と IPv6 が混在するネットワークに向けて提案された移動透過技術であり, MIPv4 と同様の手法で IPv6 に対応するように拡張した技術である. DSMIPv6 の概要図を図 2.14 に示す. 前提条件として MN, CN はデュアルスタックネットワークに存在しなければならない. DSMIPv6 は MIPv4 を拡張しているため, MIPv4 と同様に HA を介した通信となる. MN が CN と異なる IP バージョンのネットワークに移動した場合, Binding Update Message に移動前に保持していた HoA, 及び訪問先ネットワークで割り当てられた CoA を含めて HA へ送信する. MN はホームネットワークに存在するとき, HA によって HoAv4, 及び HoAv6 を取得する. MN が IPv4 ネットワークに移動した場合, CN からのパケットは IPv4 ヘッダでカプセル化され, HA と MN との間に構築された IPv6 over IPv4 トンネルを通して転送する. また, IPv6 ネットワークに移動した場合, CN からのパケットは IPv6 ヘッダでカプセル化され, HA と MN との間に構築された IPv6 トンネルを通して転送する. この時, CN が MN の HoAv4 に対して通信を開始した場合, MN が IPv6 ネットワークに移動した際は, IPv4 パケットを IPv6 パケットでカプセル化し, IPv4 over



MN: Mobile Node　CN: Correspondent Node　HA: Home Agent　FA: Foreign Agent
HoA: Home Address　CoA: Care-of Address

図 2.12: Overview of Mobile IPv6

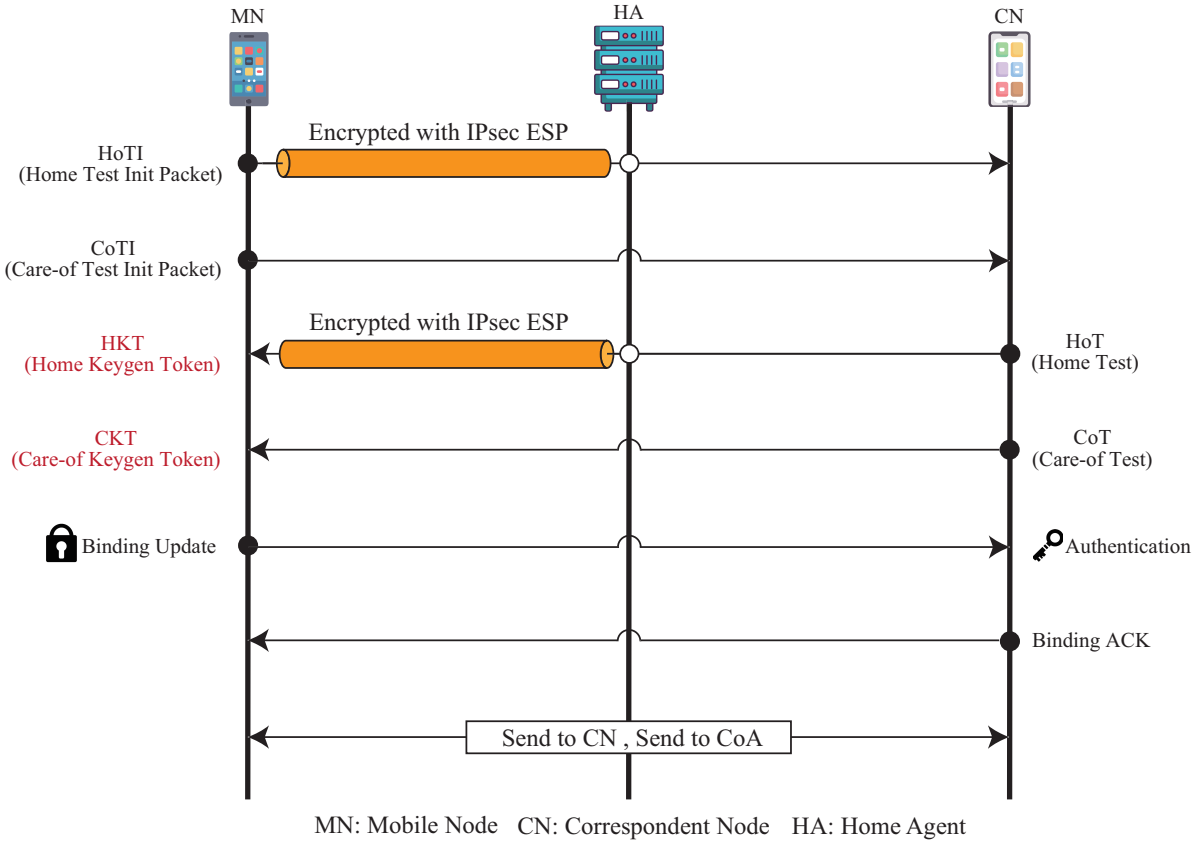


図 2.13: Overview of Return Routability

IPv6 トンネルを通して転送する．また, MN が IPv4 ネットワークに移動した際は, HA と MN との間に構築された IPv4 トンネルを通して転送する．従って, CN は HA と MN との間に構築されたトンネルを用いて通信を行うことにより, 異なる IP バージョンにおいても移動透過性を実現することが可能となる．しかし, DSMIPv6 では IP バージョンが異なる場合, パケットのカプセル化処理が必要になる．そのため, 通信の際は HA を経由する必要があり, 経路冗長化問題を引き継ぐ．

2.4.4　Proxy Mobile IPv6

上述してきた MIP は, モバイル端末等の MN が訪問先ネットワークで取得した CoA を HA に通知することで, 移動透過性を実現する技術であった．従って, MN が移動支援機能を持つためには, 特定のソフトウェアをインストールしたり, Operating System (OS) のカーネルにプロトコルを実装したりする必要がある．そこで, 携帯電話システムを IP 化する議論の中で, ネットワーク側に移動支援機能を実装することで移動透過性を実現する方式が検討された．端末に手を加えることなく, 移動透過性を実現する代表的な技術として PMIPv6 が存在する．PMIPv6 の概要図を図 2.15 に示す．PMIPv6 では, ネットワークに一つの Local Mobility Anchor (LMA) と, 複数の Mobile Access Gateway (MAG) と呼ばれる機器を設置する．LMA, MAG は, いずれも DualStack ノードとして動作する．LMA は MIPv6 における HA の役割を担う装置であり, MN の移動を管理したり, MN 宛のパケットを代理で受信して転送したりする機能を持つ．MAG は MIPv6 における MN 側に実装していた移動支援機能を有しており, MN に代わって LMA へ訪問先ネットワーク情報を通知する．PMIPv6 が適用されるネットワークを PMIPv6 ドメインと呼び, ドメイン内の移動であれば移動透過性を実現することが可能である．なお, MAG と MN 間は

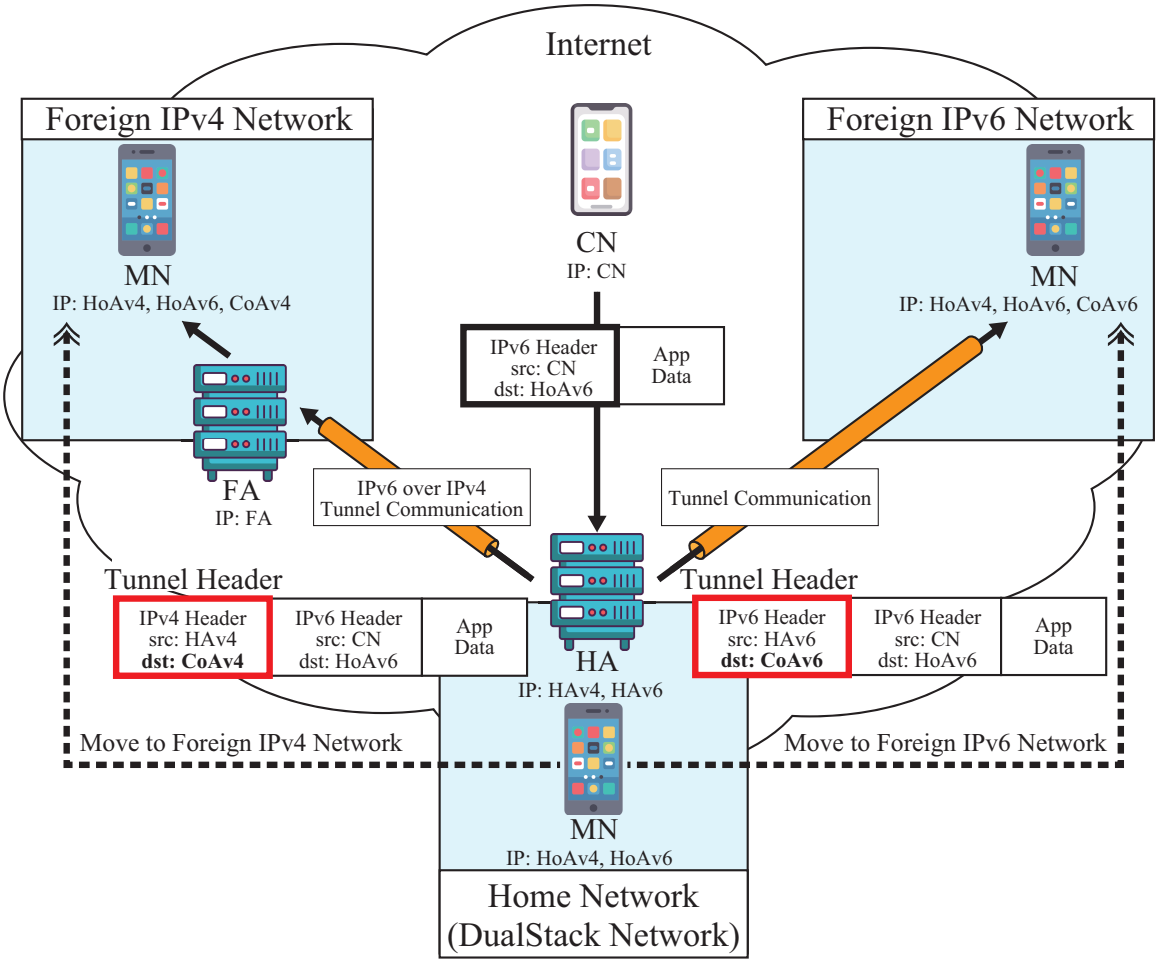


図 2.14: Overview of Dual Stack Mobile IPv6

Point-to-Point Protocol (PPP) で接続される.

MAG は, MN の接続を検知すると LMA に Proxy Binding Update Message と呼ばれる, MN を識別するための情報を送信する. LMA は, メッセージの送信元から取得した Proxy-CoA と呼ばれる MAG の IP アドレスと対応付けて, Binding Cache テーブルに登録する. その後, LMA は HoA の生成に必要な IPv6 ネットワークプレフィックスを記載した Proxy Binding ACK を応答し, MAG との間に双方向トンネルを構築する. MAG は, LMA によって付与された IPv6 ネットワークプレフィックスを含めて Router Advertise (RA) パケットを生成し, MN へ通知する. MN は, MAG から受信したプレフィックスを元に, IPv6 アドレス自動生成機能を使用して, HoA を生成する. CN が, MN の HoA 宛にパケットを送信すると, LMA が代理で受信する. 次に, CN からのパケットは, LMA と MAG との間に構築された双方向トンネルを用いて MAG へ転送され, MAG から MN へ送信される. また, MN が PMIPv6 ドメイン内を移動して MAG に接続する度に, MAG は LMA への通知, 及び双方向トンネルの構築を行う. 従って, MN は PMIPv6 ドメイン内であれば, MAG からプレフィックスを受け取ることが可能なため, 同じ HoA を使用し続けることが可能である. LMA と MAG 間はトンネル通信を行うため, IPv6, IPv4 のいずれでも許容される. また, MN に割り当てる HoA も IPv4, IPv6, または両方でも良いため, 柔軟な運用が可能である. しかし, 端末は PMIPv6 ドメインを超えて移動透過支援を受けることは仕様上困難であるため, 移

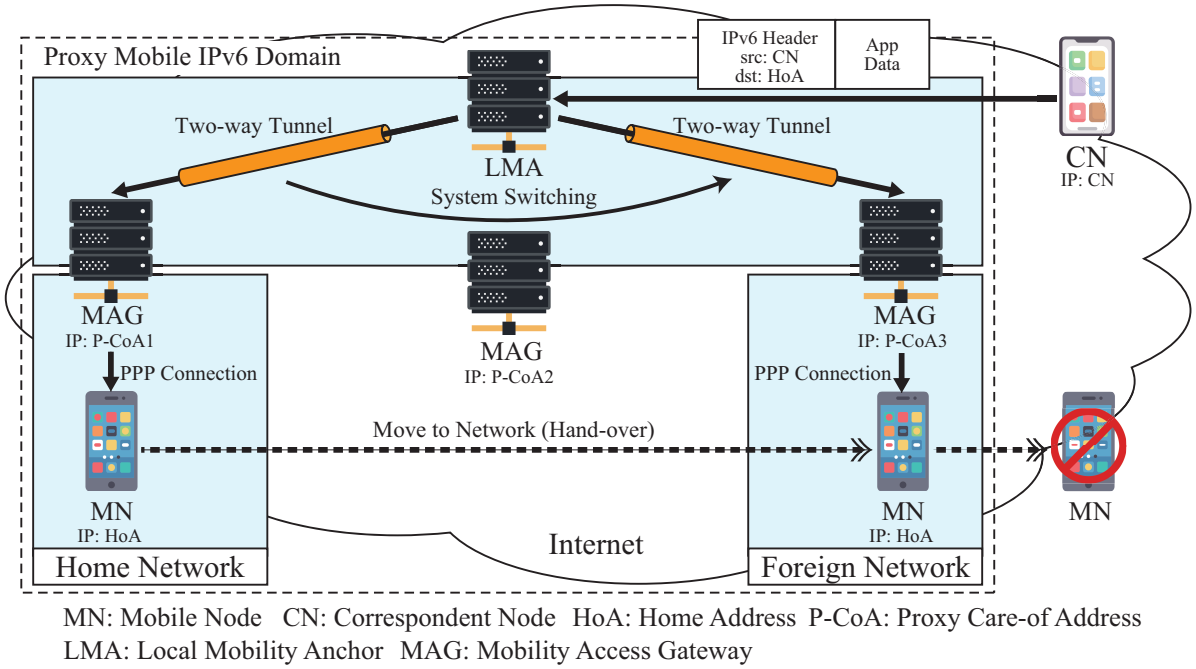


図 2.15: Overview of Proxy Mobile IPv6

動の範囲が限定されるという制約の課題が存在する.

2.4.5　Session Initiation Protocol Mobility

移動透過性を実現する手法として, SIP Mobility がある. SIP Mobility とは SIP を拡張することにより, 移動透過性を実現する移動透過技術である. SIP とは UDP ベースで設計された Internet Engineering Task Force (IETF) 標準通信プロトコルであり, 2 つ以上のクライアント間でセッションを確立する際に利用される. SIP Mobility の概要図を図 2.16 に示す. SIP を用いる端末は, Uniform Resource Identifier (URI) と呼ばれる一意のアドレスを持つ. 端末は SIP サーバに対して, URI と IP アドレス, 及びポート番号の登録処理を行う. 端末は通信する相手端末の URI を SIP サーバに問い合わせるか, または通知を受け取る. 相手端末は端末の IP アドレスを取得して, 直接, 通信相手とのセッションを確立することが可能となる. 従って, MIP のように HA を介した通信を行う必要がないため, 経路冗長化問題は発生しない. また, 端末がネットワークを移動した場合, 端末は IP アドレスの変化を SIP サーバに対して通知し, 更新処理を行う. その後, SIP サーバから通信相手端末に向けて通信を行い, 端末の IP アドレスを取得する. そのため, 直接, 相手端末とのセッションを再構築することが可能である. さらに, SIP サーバはセッション情報を管理しているため, IP アドレスの変更に伴いセッション情報を更新する. 従って, UDP 等のコネクションレスなセッションであれば, 端末が通信中に異なるネットワーク間を移動した場合にも, 既存のセッションを維持し続けることが可能である. その結果, コネクションレス通信では移動透過性と通信接続性を同時に実現することが可能である. しかし, TCP をはじめコネクションを確立した通信を行う際は, 端末が移動した場合, IP アドレスが変化するため, セッションを継続することが困難である. また, IPv6 ネットワークでの利用は十分に検討されていない.

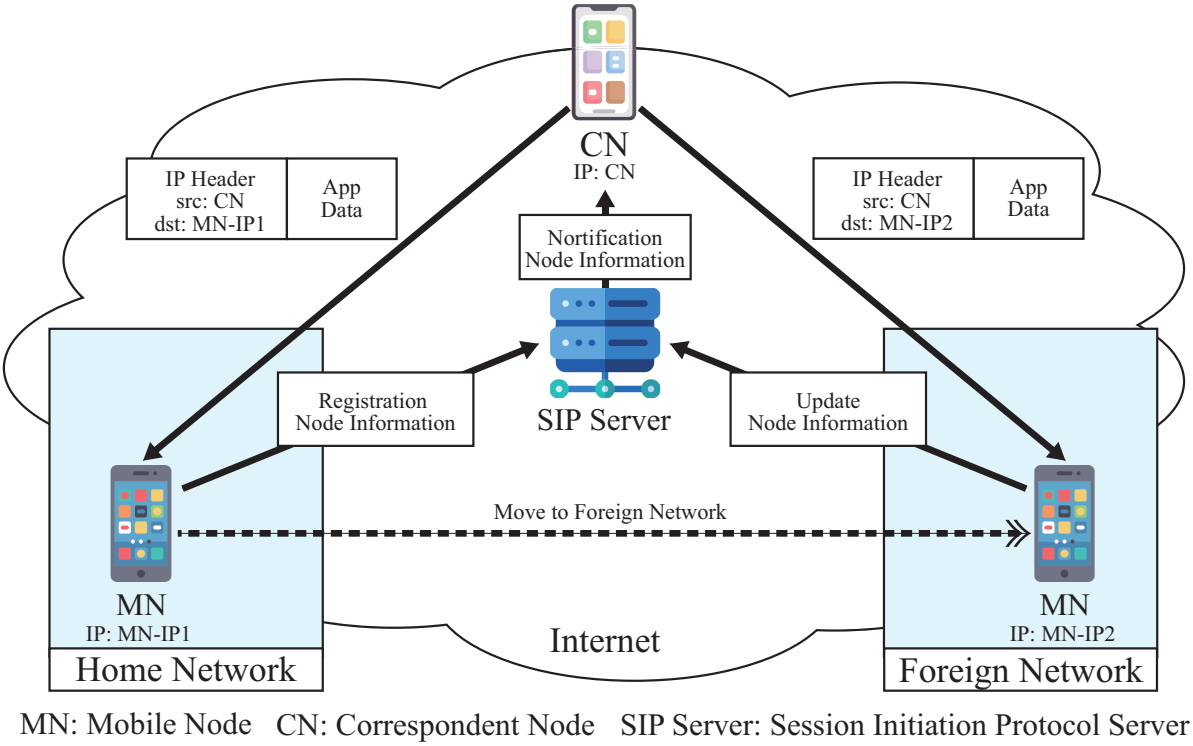


図 2.16: Overview of Session Initiation Protocol Mobility

2.4.6　Quick UDP Internet Connections

Quick UDP Internet Connections (QUIC) は, インターネット上で高速かつ安全なデータ転送を実現するために設計されたプロトコルであり, IETF によって標準化が進められている. QUIC は UDP を基盤としており, TCP よりも高速なデータ通信を提供する. また, QUIC では TCP における Three-way handshaking の手続きや暗号化処理等を効率化し, 従来の TCP よりも低遅延での通信が可能である. さらに, 複数のデータストリームを同時に処理する多重化技術としてマルチプレクシング機能も備えている. マルチプレクシングとは, 複数のデータストリームを 1 つの通信路や回線上で同時に伝送する仕組みである. 一般的に通信路や回線は, 帯域幅に限りがあるため, 効率的に利用するために複数のデータストリームを同時に伝送することが重要となる. QUIC は, マルチプレクシングにおいて, データストリームを区別するための識別子として, ストリーム ID を用いることで, 複数のストリームを 1 つの QUIC 接続内で一意に識別可能である. これにより, 複数のデータストリームを同時に処理し, 効率的な通信を実現する. ストリーム ID は, 送信側から受信側に向けてのデータストリームの識別に使用される他, マルチプレクシングにおけるデータのフラグメントやデフラグメントの際にも有効に機能する.

また, QUIC はコネクション指向プロトコルであるため, 通信の際は, 端末間でコネクションを確立した上でデータの送受信を行う. 各々の接続はコネクション ID と呼ばれる識別子によって区別される. 通信を行う双方の端末がそれぞれがコネクション ID を生成し, 自身が生成するものを送信元コネクション ID (Source Connection ID), 相手端末側が生成するものを送信先コネクション ID (Destination Connection ID) と呼ぶ. QUIC では, コネクション ID によって通信フローを識別可能なため, TCP と異なり IP アドレスやポート番号が変化した場合でも既存のセッションを維持可能である. そのため, モバイル端末において通信中のネットワーク切り替えが発生した場合においても, コネクションを継続可能であるため移動透過性を実現する. 一般に, IP アドレスやポート番号の変化に伴う処理をコネクションマイグレーションと呼ぶ.

一方で, QUIC は NAPT 配下の端末に対してグローバルネットワークから通信を開始することが極めて困難となる NAPT 越え問題を抱える. これは, 通信フローを識別するコネクション ID が, 端末の保持する IP アドレスとポート番号によって生成されるためである. NAPT はグローバルネットワークからのインバウンドトラフィックを遮断するため, QUIC を利用してグローバルネットワークからプライベートネットワークへ通信を開始することは困難であり, ポートフォワーディングや STUN, TURN を用いた NAPT Traversal 技術を別途実装する必要がある. そのため, QUIC は従来のクライアントサーバモデルにおいて通信をより高速かつ効率的に行う目的で導入される. 現在では, QUIC は Chrome ウェブブラウザから Google のサーバへの全コネクションの半分以上で利用されている. また, W3Techs の統計調査では, 約 25.5 % の Web サイトが当面の間 HTTP-over-QUIC (HTTP/3) を使用すると予想されている [73, 74].

2.5　Client-to-Server モデル

クライアントサーバモデルは, サーバがクライアントからのリクエストを受信して処理を行い, 結果をクライアントに返送する中央集権に基づいたアーキテクチャである. クライアントサーバモデルは 1980 年代頃から普及し, Web サーバと Web ブラウザの組み合わせをはじめ, 現在に至るまで多くのサービスで採用されてきた. クライアントサーバモデルでは, クライアントがサーバに対してリクエストを送信し, サーバがそのリクエストを処理する役割を担うため, 双方で機能が分離されている. 中央のサーバで一括して端末の管理及び処理を行うことで, クライアント情報の管理や制御が容易であり, データやリソースの一貫性, セキュリティを確保することが可能である. また, サーバは通常, 高性能なハードウェアや冗長化されたシステムを利用しており, 信頼性が高い通信環境を提供する. さらに, サーバ側でのデータバックアップ, アクセス制御やデータ暗号化等のセキュリティ対策を実施することで, セキュリティポリシの適用や管理のハードルを下げることも可能である.

一方で, サーバは全てのリクエスト処理を担う必要があるため, 単一障害点の問題を抱える. 一般に, クライアントサーバモデルではサーバが何らかの理由によって停止した場合には, サービスの利用が困難となる. また, サーバの負荷が増大するとスケーラビリティの問題が生じるため, 新たにサーバを追加したり, 負荷分散機構を準備したりする必要がある. その結果, 追加の管理コストやサーバの増強に伴う費用の増大が課題となる. さらに, クライアントは通信に際して常にサーバを介して相手端末とデータの送受信を行うため, ネットワーク帯域の浪費や遅延が発生する懸念もある.

2.6　Peer-to-Peer モデル

Peer-to-Peer (P2P) モデルは, サーバを介さずにピアと呼ばれる端末同士で直接データの送受信を行い, 双方の端末がそれぞれ処理を行う分散システムに基づいたアーキテクチャである. P2P モデルでは情報を一元管理するサーバが存在せず, ネットワークに接続している端末間でデータの検索や転送を直接行う. 情報を所持している端末がサーバの役割を担い, 情報を必要としている端末がクライアントとして, サーバ役の端末にリクエストの送信を行う. そのため, 1 つの端末がクライアントとサーバの両方の役割を担う. P2P は, 複数の端末で処理を分散するため, クライアントサーバモデルの課題であった単一障害点の問題を解決することが可能である. また, 直接通信を行うことで, トラフィック量を削減し, 通信遅延を改善することが期待される. そのため, 現在ではビデオ会議やストリーミングメディアの配信, オンライン対戦ゲーム等, リアルタイム通信が要求される場面で利用される. 近年では, ブロックチェーン技術を用いた分散型データベースや仮想通貨をはじめ, 一部のアプリケーションの開発においても利用されつつある. さらに, 端末の処理性能が年々向上していることからも, 今後幅広く普及すると予想される.

一方で, P2P は中央集権的なサポートが欠如しており, 端末が分散していることから, 一つのサービスを実現するにあたり端末情報の管理が複雑化する可能性がある. また, クライアントサーバモデルでは, 境界型ネットワークを利用したセキュリティ対策が容易である一方, P2P は, 端末が分散しているため, 個々の端末をセキュアに保護するセキュリティ対策が必要となる. 既存のセキュリティソリューションの多くは, クライアントサーバモデルによって, 中央のクラウドサービスが端末のセキュリティを確保する. そのため, P2P モデルは, セキュリティの観点でクライアントサーバモデルに依存することが多く, P2P モデルのメリットを活かしきれない課題が存在する. 現在では, P2P モデルの形態は大きく 3 つに分類されており, サーバを利用した構築方式も存在する. 以下に, P2P モデルの形態として, ピュア P2P, ハイブリット P2P, スーパーノード P2P について詳述する.

2.6.1　ピュア Peer-to-Peer

ピュア P2P とは, 中央にサーバを設置せず, 各ノード間で直接ファイルの検索と転送を行う完全な P2P 形態である. サーバを利用しないピュア P2P では, 端末同士でデータや情報を分散して持ち, 不足しているデータを補い合いながらネットワークを構築する. 具体的には, 情報を必要としている端末がクライアントとなり, 情報を所持している他の端末に対して検索要求を行うことで情報を取得する. ピュア P2P は中央のサーバを必要せず, 相互にデータを共有し合うため, 分散型ネットワークの構築が容易である. また, ピアの追加や削除が容易であり, ネットワーク全体のリソースや帯域幅の増加はピアの追加によって行われる. そのため, サーバ障害によるサービス停止の懸念がなく, システムの性能やネットワークのスケーラビリティも容易に向上させることが可能である.

一方で, ピュア P2P では, ピア同士が直接通信するため, セキュリティリスクが高まる. また, ピュア P2P では, 各ピアの信頼性が課題となっており, ネットワークに信頼できないピアが含まれている場合, データの完全性や可用性の点で脆弱性を抱える. 特に, ファイル共有やストリーミングサービス等において大量のデータを共有する場合, 信頼性の問題が深刻化する.

2.6.2　ハイブリット Peer-to-Peer

ハイブリッド P2P とは, 中央のサーバを用いて P2P モデルを構築する形態である. ハイブリッド P2P では, 各ピアが持つ情報のインデックスをサーバで管理し, 実際のデータ送受信をピア間で直接行う. 分散型の P2P モデルと, 中央集権型のクライアントサーバモデルの利点を組み合わせることにより, 信頼性の向上や負荷分散等のメリットを同時に享受することが可能である. また, ピアの追加や削除によって, ネットワークスケーラビリティを向上させるとともに, 必要に応じてサーバリソースを増やすことでサービスの拡張を容易に行うことが可能である. そのため, ハイブリッド P2P は, 異なるアプリケーションやユースケースに対応する柔軟性を持つ.

一方で, ハイブリッド P2P は, ピュア P2P とクライアントサーバモデルを組み合わせた複雑なアーキテクチャである. そのため, サービス設計や管理が複雑になり, 実装や運用のコストが増加する可能性がある. また, ピア同士の直接通信に伴う情報をサーバが管理するため, 中央のサーバがサイバー攻撃を受けた場合, 端末間の通信もセキュリティリスクに晒される. さらに, サーバを用いて P2P モデルを構築するため, サーバの停止によって新規に端末間通信を確立することが困難となる課題も存在する.

2.6.3　スーパーノード Peer-to-Peer

スーパーノード P2P とは, ピアのインデックス情報をスーパーノードと呼ばれる特別な端末が管理して P2P モデルを構築する形態である. スーパーノードは, 処理能力が高く通信回線が安定している端末

が複数選出される．スーパーノードとして選ばれた端末は, ネットワークに参加しているピアの情報を管理し, 各ピアからの要求に応じて必要な情報の検索と配布を行う．また, スーパーノードは下位のピアから受信したリクエストを他のピアに転送する役割も担う．スーパーノードがネットワーク全体の管理や調整を行うことで, 通信の安定性や効率性が向上し, ピア間の効率的なデータ送受信が可能である．さらに, スーパーノード P2P は, ネットワーク内の信頼性の高いノードを選択して構成されるため, スーパーノードを介した通信は, 信頼性が高く, データの完全性や可用性も向上する．

一方で, スーパーノード P2P では, スーパーノードを用いることで, クライアントサーバモデルに存在する中央集権化のリスクを引き継いでいる．そのため, スーパーノードの障害や攻撃により, ネットワーク全体の信頼性が低下するリスクが存在する．また, 単一のスーパーノードの性能やリソースには限界があるため, リクエストの集中によってネットワークのパフォーマンスが低下する可能性がある．さらに, スーパーノードを介して通信を行うピアは, 接続元のスーパーノードが離脱することでコネクションの継続が困難になるため, 通信が一時的に中断される懸念もある．

2.7 Virtual Private Network

インターネット上には様々な脅威が存在するため, 端末間の接続及び通信に際してセキュリティの確保は重要な課題である．近年では, リモートワークやクラウドサービスも普及したことで, 端末やサーバ機器はしばしば異なるネットワークに分散して配置される．異なるプライベートネットワークに存在する端末をセキュアに接続する技術として, Virtual Private Network (VPN) 技術がある．VPN は, 物理ネットワーク上に仮想的なトンネル経路を確立することで, 地理的な離れた端末間を同一ネットワークに存在するかのように接続することが可能である．そのため, 広域ネットワークに存在する脅威から送受信データや端末を保護することが可能である．また, VPN を利用することで, 自身の IP アドレスや位置情報を秘匿して外部端末と通信を行うことが可能なため, プライバシを保護することが可能である．近年では, 場所を問わずネットワークにアクセスをするインターネット利用形態が増えており, VPN を用いた安全かつセキュアなリモート接続は, 組織や企業において必要不可欠となっている．一般に VPN は, 集約型 VPN と分散型 VPN に大別される．以下に, それぞれの VPN 接続形態の概要と, それらを用いた VPN ソリューションについて詳述する．

2.7.1 集約型 VPN

従来, 端末は一つのネットワークに存在することから, リモートアクセスの際は, 単一ネットワークを対象として VPN による拠点間接続を行うことで, 内部に存在する複数の端末を保護していた．このように, 一つのネットワークに集約された端末を対象とした VPN ソリューションを集約型 VPN と呼ぶ．集約型 VPN では, ネットワーク境界に設置されるルータやハードウェアボックススイッチに VPN のサーバソフトウェアをインストールして VPN サーバとして構築する．また, 遠隔地に存在する端末は, VPN のクライアントソフトウェアをインストールして, VPN サーバに接続することでプライベートネットワーク間をセキュアに接続する．集約型 VPN では, 単一ネットワークを代表して VPN サーバが接続口となるため, アクセス先の端末は VPN ソフトウェアをインストールする必要はない．集約型 VPN の代表的なソリューションとして OpenVPN をベースとした SoftEther が挙げられる．

SoftEther は, 集約型の VPN ソリューションであり, 端末とネットワークを接続する VPN 接続形態を提供する．SoftEther は, VPN サーバと VPN クライアントの両方のソフトウェアを提供しており, Microsoft Windows, Linux, macOS, FreeBSD 等のさまざまなプラットフォームで利用することが可能である．また, 通信プロトコルとして OpenVPN, Layer 2 Tunneling Protocol/Security Architecture for Internet Protocol (L2TP/IPsec), Secure Socket Tunneling Protocol (SSTP), EtherIP, Layer 2 Tunneling

Protocol version 3 (L2TPv3), Cisco VPN 等, 豊富な選択肢が提供されている. 組織は, ネットワーク境界に SoftEther をインストールした VPN サーバを設置し, 利用者は自身の端末に VPN クライアントを実装することで, 遠隔地からプライベートネットワーク内の端末にセキュアにアクセスすることが可能である. SoftEther は, Web ベースの管理コンソールを提供しており, ユーザアカウントの管理やアクセス制御を柔軟に設定することが容易である.

一方で, VPN サーバの構築に際して所属ネットワークに適した設定を行う必要があるため, SoftEther の導入のためには専門的な知識を要する. また, 集約型 VPN ソリューションは, 端末が分散ネットワークに散在する場面では, 設定や管理が複雑化することから有効的でない. 加えて, 複数の端末を VPN サーバが一括して管理するため, 端末数の増加によってサーバの負荷が増加し, 通信遅延の増大や VPN サービスそのものが停止する懸念がある.

2.7.2　分散型 VPN

集約型 VPN に対し, 分散ネットワークに配置された端末間をセキュアに接続する VPN ソリューションを分散型 VPN と呼ぶ. 分散型 VPN は, 通信を行う双方の端末のそれぞれに VPN ソフトウェアをインストールする. そのため, 端末自身が VPN サーバと VPN クライアントの双方の役割を果たすため, P2P モデルに基づいた VPN トンネルによる端末間の直接通信が可能である. 近年では, ゼロトラストセキュリティが注目されていることから, 分散型 VPN の需要はさらに急増することが予測される. ゼロトラストセキュリティとは, 全ての端末を信頼せず, 全ての通信フェーズにおいて認証・認可, 暗号化を施すことで, 性悪説に基づきインターネット上に存在するあらゆる脅威から保護するセキュリティ対策手法である. 分散型 VPN を実現する代表的なソリューションとして Wireguard をベースとした Tailscale が挙げられる.

Tailscale は, 分散型の VPN ソリューションであり, 通信を行う端末同士を直接接続する VPN 接続形態を提供する. そのため, ユーザは Tailscale を導入することで, リモート端末をローカル端末と同様に扱うことでき, プライベートネットワークを跨いだセキュアかつシームレスな端末間通信を利用可能である. また, Tailscale は, ゼロコンフィグレーションの VPN サービスとしてシンプルなネットワーク接続を構築可能である. ゼロコンフィグレーションとは, ユーザの特別な介入がなく, 自動的に構成されるシステム及びプロセスを指す. ユーザは, Tailscale のクライアントソフトウェアをインストールし, アカウントにログインするだけで, ネットワーク接続及び相手端末とのトンネル経路が自動的に設定される. そのため, ネットワークの知識が無くても, 誰でも容易に VPN を構築することが可能である. さらに, 通信に際して, Multi-Factor Authentication (MFA) による認証をサポートしており, 相手端末の真正性を保証するとともに, 全ての送受信データを暗号化によって保護することで, 中間者攻撃やデータの盗聴を防止する. また, 管理者は Tailscale が提供するクラウドベースの管理コンソールを用いることで, ネットワークの設定や端末の管理を中央集権的に行うことが可能である.

一方で, データの管理がクラウドサービスに依存していることから, 単一障害点の観点でサービスの可用性に影響を与える可能性がある. また, セルフホスティングに際しては, クラウドベースの運用や管理に関連するコストが増大する懸念がある. 加えて, 一部のユーザはクラウドサービスにデータを委ねることに対するプライバシ上の懸念を持つ場合がある. 従って, クラウドサービスプロバイダは, ユーザデータを適切に保護・管理することが求められる.

2.8　Domain Name System

Domain Name System (DNS) は, インターネット上のリソースを識別するための階層的な名前空間を管理し, ドメイン名と IP アドレスの対応付けを行うシステムである. DNS は, ユーザーが覚えやすいド

メイン名を対応する IP アドレスに変換することで, ウェブサイトやサービスへのアクセスを容易にする. インターネットに接続されている全てのコンピュータは, 固有の IP アドレスを持ち, インターネット上の他のコンピュータにアクセスするためには, そのコンピュータに割り当てられた IP アドレスを知る必要がある. しかし, IP アドレスは IPv4 の場合, 0 から 255 までの数値 4 オクテットで表現されるため, 人間には記憶しにくいという問題がある. そのため, IP アドレスを文字列で扱うことができるような機構が必要となり, インターネットドメイン名が考案された. DNS において, 一般にドメイン名から IP アドレスを引き出す機能を正引きと呼び, DNS の代表的な機能である. また, 近年では, DNS を用いた負荷分散機構も多数提案されており, DNS-Round Robin (DNS-RR) が代表的な例として挙げられる. DNS-RR とは, ドメイン名と IP アドレスを対応付ける DNS を利用して, 一つのホスト名に複数の IP アドレスを対応付けてサーバの負荷分散を行う手法である. ユーザは, 単一のドメインにアクセスすることで, DNS-RR が複数の IP アドレスを解決するため, 自動的にトラフィックを分散させることが可能である.

また, DNS は階層的な構造を持ち, Top-Level Domain (TLS), Second-Level Domain (SLD), その他, サブドメイン等の階層に分かれている. これにより, ドメイン名の管理が効率的に行われる. DNS のプロセスは, クライアントがドメイン名を指定して名前解決を要求し, ローカル DNS サーバがルート DNS サーバや TLD サーバ等と連携して解決を行う. ドメイン名が解決されると, クライアントに IP アドレスが返送され, 通信が開始される. 一般に通信の際にはスタブリゾルバと呼ばれるクライアントサイドで動作する DNS ソフトウェアが優先的に用いられる. スタブリゾルバは, ユーザが指定したドメイン名を解決するために, ローカルにキャッシュされた情報や, DNS サーバへの問い合わせを利用して名前解決を行う. 通常, OS の一部として提供され, アプリケーションが名前解決を必要とする際に呼び出される.

DNS サービスはセルフホスティングすることが可能であり, 独自ドメインを用いたネットワークを構築することが可能である. DNS サーバを提供する代表的なソフトウェアとして, Berkeley Internet Name Domain (BIND), 及び CoreDNS が存在する. 以下に, それぞれの概要について詳述する.

2.8.1　Berkeley Internet Name Domain

Berkeley Internet Name Domain (BIND) はカリフォルニア大学バークレー校で開発された DNS サーバソフトウェアであり, 現在は, Internet Software Consortium (ISC) によって開発・配布が行われている. BIND は, 1988 年に開発が開始されてから現在に至るまでデファクトスタンダードとして広く普及し, 昨今動作する DNS サーバの多くで利用されている. BIND は, DNS サーバとして必要な機能が全て揃っており, 適切に設定することで, 個人用途から大規模向けのサーバ, コンテンツサーバ, キャッシュサーバの何としても動作します. これまで, BIND4, BIND8 がリリースされてきた. しかし, セキュリティの問題や DNS Security Extensions (DNSSEC) と呼ばれる仕組みへの対応が必要となり, ソースコードが完全に書き直された BIND9 が現在の最新リリースとなっている. DNSSEC とは, DNS キャッシュポイズニングや DNS キャッシュを介した攻撃等, DNS のセキュリティ上の脅威に対処するために開発された DNS の拡張機構であり, ユーザが信頼できる DNS データにアクセス可能とするためのセキュリティ対策手法である. 一方で, BIND 及び BIND9 は, 設定や管理が複雑であり, 導入にあたりコストが増大する可能性がある. また, リソースを比較的多く消費するため, ハードウェア要件が高くなる傾向にある.

2.8.2　CoreDNS

CoreDNS は, コンテナ化されたクラウドネイティブ環境向けに設計されたオープンソースの DNS サーバソフトウェアとして開発され, 現在では Cloud Native Computing Foundation (CNCF) によりホスティングされている. 近年では, 次世代のインフラ基盤を担うとされている Kubernetes クラスタ内の DNS サーバにおいて, kube-dns としても採用されており, ネットワークリソースの名前解決やサービスディ

スカバリのために使用される．また, CoreDNS は, カスタマイズ性に優れており, 柔軟なプラグインアーキテクチャを持ち合わせることで, 高度な柔軟性と拡張性を利用することができ, DNS リクエストに対してカスタムロジックや転送ルールを記述して実行することが可能である．さらに, プライベート DNS ゾーンを設定することが容易であり, 内部ネットワーク内のリソースに名前解決を提供可能である．同時に, カスタムドメインの設定もサポートしていることから, CoreDNS を用いてカスタムドメイン名の設定をサポートし, ドメイン名と IP アドレスのマッピングを制御可能である．これらの機能により, 豊富なユースケースに適用することが期待される．また, 今日ではプラグインとして, DNS-over-TLS (DoT) 及び DNS-over-HTTPS (DoH) 等のサポートが開始され Encrypted DNS を用いたセキュアな DNS 通信も利用することが可能となった．CoreDNS は, SkyDNS とも互換性があり, etcd をバックエンドとした DNS サーバとして動作する．その他, dnsmasq 及び hosts ファイルをベースに名前解決も可能であり, 従来の DNS サーバに存在した機能も利用することが可能である．CoreDNS は軽量かつモジュール化された設計とクラウドネイティブな機能を提供していることから, 次世代の DNS サーバとして今後, 益々普及することが予想される．

2.9　暗号化技術

インターネットの普及やデジタル化の進展に伴い, データの送受信が増加し, それに伴う情報漏洩や不正アクセスのリスクが増大している．このような状況下で, 暗号化技術は, データの保護とプライバシの確保において極めて重要な役割を担っている．暗号化技術は, データを不正なアクセスから保護するために, 情報を変換し, 意味のある形から不可解な形式に変換する．これにより, 第三者がデータを盗み見たり改ざんしたりすることを防止する．また, 暗号化技術を利用してデータを暗号化することで, 改ざんや破損の検知が容易になるため, 信頼性の高い通信を実現することが可能である．以下に, 代表的な暗号化技術を取り上げて詳述する．

2.9.1　Transport Layer Security

TLS とは, セキュアな通信を実現するためのプロトコルである．TLS は公開鍵認証方式を用いたサーバの認証や共通鍵暗号方式を用いた暗号化通信, ハッシュを用いた改ざん検知を規定している．公開鍵認証とは, 公開鍵と秘密鍵からなるキーペアを用いて, ノードの真正性を確認する方式である．TLS を使用する場合, サーバは秘密鍵を用いて電子署名を作成し, クライアントへ送信する．この際使用される電子署名は, サーバ証明書とも呼ばれる．電子署名を受信したノードは, サーバの公開鍵からその署名の妥当性を確認する．署名が正しいと確認された際には, その署名を作成したサーバは正しい秘密鍵を所有していることから, 正規のサーバであると判断される．TLS は, オプションとして, サーバとクライアントの役割を入れ替えクライアントの真正性を確認する, クライアント認証と呼ばれる機能を有している．また, TLS は認証プロセスにおける通信を経て共通鍵を交換し, 以降の通信を暗号化する．さらに, 暗号化通信に加え, 一方向ハッシュ関数によって生成されたハッシュ値を通信データに付加することで, 改ざん検知を行う．

2.9.2　OpenSSL

OpenSSL は, Secure Sockets Layer (SSL) プロトコル及び Transport Layer Security (TLS) プロトコルの実装を提供するオープンソースの暗号化ライブラリである．クライアントとサーバ間でデータを暗号化して送受信を行う際には, 暗号化と復号に同じ鍵を使用する共通鍵暗号方式を使用する．また, 作成した共通鍵をクライアントとサーバで共有する際には, 暗号化と復号に異なる鍵を使用する公開鍵暗号

方式を使用する．サーバが暗号用の鍵と復号用の鍵の組み合わせを作成し, 復号用の鍵は秘密鍵としてサーバが保持し, 暗号化用の鍵は公開鍵としてクライアントに付与する．OpenSSL の代表的な使用例として Hyper Text Transfer Protocol Secure (HTTPS) が挙げられる．HTTPS は Web ブラウザの閲覧時等に利用される HTTP を OpenSSL の機構を用いて暗号化したネットワークプロトコルである．また, OpenSSL は暗号化を実装するための crypto ライブラリが含まれる．OpenSSL は, ライブラリに含まれる関数をコマンドラインから実行可能なコマンドラインツールが含まれる．コマンドラインツールには, Rivest Shamir Adleman (RSA) 等の秘密鍵の作成, SSL サーバ証明書の Certificate Signing Request (CSR) の作成, SSL/TLS クライアントとサーバのテスト等, 様々な機能が実装されている．

2.9.3　Advanced Encryption Standard

Advanced Encryption Standard (AES) は, 現在最も一般的に使用されているブロック暗号の一つである．ブロック暗号とは, 固定サイズのブロックにデータを分割し, 鍵を使用してそれぞれのブロックを暗号化または復号する暗号化方式である．AES は, 対称鍵暗号アルゴリズムであり, 同じ鍵を暗号化と復号に使用する共通鍵暗号方式を使用する．AES は, 1997 年に米国の National Institute of Standards and Technology (NIST) によって暗号化標準の選定プロセスが開始され, 2001 年に AES として採用された．AES は, 主に, 128 ビット, 192 ビット, 256 ビットの 3 つの鍵長をサポートしており, 現在では 256 ビットの鍵長が最も一般的に使用される．また, Substitution-Permutation Network (SPN) 構造を採用しており, セキュリティマージンが低く抑えられているという特長を持つ．AES は, 前述の OpenSSL においても利用されている．

2.9.4　Hash-based Message Authentication Code

Hash-based Message Authentication Code (HMAC) は, 秘密鍵を用いてメッセージの整合性と認証を確認するためのハッシュ関数を利用した認証コードである．HMAC は, メッセージに対して秘密鍵を組み合わせて計算される Message Authentication Code (MAC) の一種であり, 安全な認証タグを生成する．HMAC は, 2 回ハッシュ関数を用いた認証子の生成や, 鍵付きハッシュ関数として利用されることがある．HMAC は Secure Shell (SSH) プロトコルや SSL プロトコルにおける鍵生成の際に利用されている．また, IP レイヤでのセキュリティを担保するプロトコルである IPsec のうち, Internet Key Exchage (IKE) においても key derivation と呼ばれる手続きに HMAC が利用される．

2.10　認証技術

インターネットを介した通信において暗号化を行うことで第三者からの盗聴を防止することが可能である．しかし, 暗号化通信では, 相手端末や通信データが信用できるものであるか否かを保証する仕組みが提供されていない．そのため, 相手端末と通信データの認証を行うことで真正性や完全性の確認を行う必要がある．認証技術は, ユーザやシステムが正当なものであることを確認するための技術であり, 情報システムやネットワークのセキュリティを確保する上で欠かせない要素である．以下に, 通信における代表的な認証技術として, メッセージ認証及びデジタル署名を取り上げて詳述する．また, メッセージ認証及びデジタル署名における課題を解決するための公開鍵暗号基盤である Public Key Infrastructure (PKI) や, 認証・認可基盤として標準化されている OpenID Connect (OIDC) を取り上げて詳述する．

2.10.1　メッセージ認証

メッセージ認証は共通鍵暗号方式を利用した認証技術であり, 相手端末と通信データが正しいことを保証する. メッセージ認証では Message Authentication Code (MAC) を利用することでメッセージの認証を行う. MAC は任意長のメッセージと共通鍵を基に固定長の文字列である MAC 値を計算する関数である. 送信者は送信メッセージの MAC 値を送信メッセージに添付したパケットを送信する. 受信者は受信メッセージから MAC 値を計算し, 添付された MAC 値と照合することでデータの完全性と送信者の真正性を検証する. しかし, メッセージ認証では暗号化と復号を同じ鍵で行う共通鍵を利用することから, 鍵の漏洩によって第三者による盗聴のリスクが存在する.

2.10.2　デジタル署名

デジタル署名では, 認証に公開鍵暗号方式を用いる. 公開鍵暗号方式では, 暗号化と復号に異なる鍵を使用する. まず, デジタル署名では, 任意長のデータのハッシュ値を秘密鍵で暗号化することでデジタル署名を生成し, メッセージに添付して送信する. 受信者は送信者の公開鍵でデジタル署名を復号し, 受信データのハッシュ値と照合することでメッセージの完全性と送信者の真正性を検証する. デジタル署名は秘密鍵を持つ本人のみ生成可能であるため, 本人以外が成り済ますことは極めて困難である. しかし, 受信者が取得した公開鍵が, 送信者本人が配布したものであるかを保証する仕組みは存在しない.

2.10.3　Public Key Infrastructure

Public Key Infrastructure (PKI) は公開鍵暗号方式に基づいたセキュリティ基盤であり, 公開鍵の配布と検証を安全に行うためのインフラストラクチャを提供する. 公開鍵暗号方式では, 公開鍵が本人のものであるかを保証することが困難であるという課題が存在した. PKI では, Certification Authority (CA) と称される, 信頼のある共通の第三者を通じて, 通信相手の真正性を検証することで, 公開鍵の信頼性を保証する.

PKI は主に次の要素で構成される.

- 認証局 (CA: Certification Authority)
  公開鍵暗号方式に基づく暗号化通信における信頼性を確保するための中心的な役割を果たす組織であり, 絶対的な信頼を確立している. CA は, 証明書所有者に対して, 公開鍵とそれに対応する秘密鍵の所有者の証明書 (公開鍵証明書) を発行する. 発行した証明書の信頼性が失われた場合は, その証明書を失効させ, 証明書失効リスト (CRL: Certificate Revocation List) を発行する.

- 登録局 (RA: Registration Authority)
  ユーザからの証明書申請が発生した場合に本人確認を実施する. また, CA に対して証明書の発行や失効を要求する.

- リポジトリ (Repository)
  証明書リスト及び CRL を格納してユーザへ公開する.

- アーカイブ (Archive)
  証明書の長期保存や秘密鍵のバックアップを実施する.

- 証明書所有者 (Certificate Holder)
  証明書及び対応する秘密鍵を用いて, 電子文書へのデジタル署名や暗号文の復号を実施する. 証明書に対応する秘密鍵を安全に保持し, デジタル署名や暗号文の復号に使用する.

- 証明書利用者 (Relying Party)
  証明書所有者の証明書を入手してデジタル署名の検証や文書の暗号化を実施する．リポジトリから証明書や CRL を取得し, それらの有効性を検証する．また, 取得した証明書を用いて, 署名検証や文書の暗号化を実施する．

一般的なクライアントサーバモデルにおいて, クライアントが PKI を利用した接続先サーバの信頼性を保証する手続きを次に説明する．

まず, サーバ側 (証明書所有者) で公開鍵と秘密鍵のペア鍵を生成する．次に, 公開鍵と各種証明書を CA に送信し, デジタル証明書の発行を申請する．CA は, 厳正な審査の上, デジタル証明書を発行してサーバに付与する．この時, デジタル証明書は CA の秘密鍵によってデジタル署名され, 暗号化される．また, 認証局の公開鍵が含まれたデジタル証明書をルート証明書としてクライアントに送信する．以上により, サーバの信頼性を第三者である CA が保証し, サーバが保持する秘密鍵は CA が発行した公開鍵によってのみ復号可能な状態となる．

クライアント側 (証明書利用者) は, サーバに接続要求を送信すると, サーバはデジタル証明書を送信する．次にクライアントは, サーバから送信されてきたデジタル証明書が認証局から発行されたものであるかを, CA の公開鍵によって検証する．サーバから付与されたデジタル証明書が, CA の公開鍵によって復号可能な場合, 接続先サーバの真正性が保証される．以上の手続きにより, PKI を通じて通信相手を保証することが可能である．現在のインターネットでは, CA 自体をさらに上位の CA が保証することで, 公開鍵を用いた認証基盤が連なっており, 通信相手を保証する仕組みが整備されている．

2.10.4　OpenID Connect

OpenID Connect (OIDC) とは, OAuth 2.0 プロトコルをベースにした認証及び認可のための標準プロトコルである．認証 (Authentication) とは, ユーザやシステムが自分自身を証明するプロセスであり, システムにログインする際や, システムが別のシステムと通信をする際に, アイデンティティを確認する場面で使用される．一般に, パスワード認証や, 指紋, 顔認識等の生体認証, 証明書やトークンによる認証方式が用いられる．認証が成立すると, ユーザやシステムは正当なアクセス権を取得し, 他のシステムやリソースにアクセスすることが可能となる．また, 認可 (Authorization) とは, ユーザやシステムがアクセスできるリソースや機能を制御するプロセスである．認証が成功した後, システムはユーザやクライアントに対して, どのような操作が許可されているのかを決定する．認可プロセスは, ユーザやクライアントがリソースにアクセスする際に, そのアクションが許可されているかどうかを判断するために使用される．一般的な認可の方法として, ロールベースのアクセス制御 (RBAC: Role Based Access Control), アクセス制御リスト (ACL: Access Control List), 属性ベースのアクセス制御 (ABAC: Attribute Based Access Control) が挙げられる．

OIDC は主に次の要素で構成される．

- 認可エンドポイント (Authorization Endpoint)
  認証プロセスの開始地点であり, ユーザが認証されるウェブページである．クライアントアプリケーションは, このエンドポイントを通じてユーザ認証を開始し, リダイレクト URI を指定してリクエストを送信する．

- トークンエンドポイント (Token Endpoint)
  アクセストークンや ID トークンを要求するためのエンドポイントである．認証コードグラントタイプやリフレッシュトークン等の認可フローにおいて使用される．

- ユーザ情報エンドポイント (UserInfo Endpoint)
  認証されたユーザに関する情報を取得するためのエンドポイントである．ID トークンに含まれる情

報に加えて, 追加のユーザ属性情報を取得することが可能である. 一般に, ユーザ情報エンドポイントはリソースサーバとも呼ばれる.

- クライアント (Client)
  OIDC を使用して認証を行うアプリケーションやサービスそのものを指す. 一般に, ウェブブラウザが OIDC クライアントの代表的な例として挙げられる. ユーザを認証するために認証エンドポイントにリクエストを送信し, トークンエンドポイントからトークンを取得する.

- ID プロバイダ (ID Provider)
  OIDC のプロバイダであり, 認証エンドポイントやトークンエンドポイント等のエンドポイントを提供する. また, ユーザの認証と情報提供を行い, ID トークンを発行する.

OIDC を利用した認証・認可のフローを次に説明する.

まず, クライアントは ID プロバイダが提供するトークンエンドポイントに対して ID トークン, 及びアクセストークンの発行を依頼する. ID トークンは, ユーザ及びアプリケーションが認証された事実や, 登録情報が捏造されていないことを証明するために利用される. また, アクセストークンは, クライアントが任意のアプリケーションにおいて, 実際にリソースにアクセスする際に使用するトークンである. アクセストークンは, 一般に HTTP リクエストヘッダフィールドを活用した JSON Web Token (JWT) 認証で利用される.

ID プロバイダは, クライアントから ID トークンの発行リクエストを受信すると, クライアントに対して, ID トークンの発行に必要となる本人確認情報及び属性情報の提示を要求する. クライアントは, ID プロバイダから本人確認情報の提示を要求されると, 自身の必要事項を記載した情報を認可エンドポイントに向けて送信する. 次に, ID プロバイダは, クライアントから受信した情報を検証して認可を決定し, リソースに接続するためのアクセストークンと ID トークンを発行する. 最終的に, クライアントは, 任意のサービスが提供するユーザ情報エンドポイント及びリソースサーバに対してアクセストークンを含めたリクエストを送信する. リソースサーバは, ID プロバイダが発行するアクセストークンを検証することで, 認可に応じたリソースを提供する. 以後, クライアントは ID プロバイダから発行された ID トークンを利用することで, サービス利用に伴う認証の手続きを省略することが可能となる. 以上の手続きにより Single Sign On (SSO) 機構及び ID トークンを用いることで, ユーザは一度の認証で, 他の OIDC 対応のサービスやアプリケーションに容易に接続することが可能なため, セキュリティの向上やサービスの認証・認可プロセスの効率化が見込める.

OIDC は, 従来の ID・パスワードを用いたベーシック認証や, 証明書認証と比較して, ユーザの認証情報や証明書情報を直接サーバに送信しないため, 中間者攻撃や盗聴のリスクを排除することが可能である. また, OAuth 2.0 プロトコルをベースとしていることから, セキュアな認証プロトコルを使用するとともに, 短命なトークンベースの認証を行うことで強固なセキュリティを確保する. ID トークンは, 定期的にリフレッシュトークンと呼ばれる特殊なトークンを使用して更新されるため, 継続的にセキュアなアクセスを確保可能である. OIDC サービスの例として, Google 社が提供する Firebase Authentication が挙げられる. Firebase Authentication は, Google, Facebook, Twitter, GitHub 等の多くの ID プロバイダをサポートしており, フェデレーションによってサービス連携をセキュアに行うことが可能である. これにより, 元来, 異なるサービスにおいてパスワードを使い回すことで生じていたセキュリティリスクを排除することが可能である. 近年では, Identity as a Service (IDaaS) の登場や, パブリッククラウドサービスでの利用も一般化されつつあり, 今後, OIDC 連携による認証・認可はさらに普及していくと予想される.

2.11　並行処理技術

特定の処理を行うサーバにおいて, あるクライアントが通信を行なっている間に他のクライアントが接続した場合, 別クライアントの接続も確立されリクエストの送信が可能となる. しかし, 最初のクライアントとの通信を終えるまでサーバは後続のクライアントのデータをエコーバックすることはない. この種のソケットアプリケーションは反復サーバと呼ばれる. 反復サーバは各クライアントからの要求が小さく, サーバの作業量が限られるアプリケーションでは効率的に機能する. 一方で, 反復サーバは一つのクライアントの処理に負荷が掛かり過ぎると, 他の全てのクライアントが接続するまでに掛かる時間が許容限度を超えることがある. UNIX, Windows 等の OS には, この問題を解決するための方法が用意されている.

これらの OS は一般にマルチタスク OS と呼ばれ, プロセスやスレッドを利用することで一つのクライアントの処理を, 独立して動作するサーバのコピーに任せることが可能である. プロセスとは, OS とユーザインターフェースにおける処理実行の基本単位である. また, プロセス間のメモリ空間は分離される. メモリ空間はプロセスから OS に要求することで, 空きがある場合に, 追加取得することが可能である. 一方, スレッドとは, プロセスのコンテキスト内で実行されるひとまとまりの命令である. 単一プロセス内で動作するスレッドはメモリ空間, 及び処理に必要なリソースを共有する. スレッドはプログラムを実行する際のコンテキスト情報が最小で済むため, コンテキストスイッチ動作が比較的高速である. コンテキストスイッチとは, CPU が現在実行しているプロセスやスレッドを一時停止し, 他のプロセスやスレッドに切り替えて実行を再開することである. スレッドには, プログラム内のコードにより管理することが可能なユーザスレッドと, OS カーネルが管理するカーネルスレッドが存在し, いずれも CPU コアに対してマッピングされる. 一般に, カーネルスレッドは, ユーザスレッドと比較してオーバーヘッドが増大する傾向にある. これは, カーネルスレッドの管理が OS に委ねられるため, スレッドの切り替えに関連するコンテキストスイッチやシステムコールが都度発生するためである. 一方, ユーザスレッドはアプリケーションが自己管理するため, オーバーヘッドが比較的低い傾向にある.

また, 並行処理とは, スレッドを利用することで単一のプロセスを用いてクライアントや処理毎に並行して作業を行う処理方式である. 高スループットが求められるミドルウェアや, 複数クライアントからの接続を受け入れるサーバは, 並行処理方式を用いることでタスクを効率的に実行することが可能となる. 以下セクションにて, アプリケーションのタスクに対して, プロセスが管理するスレッドを複数に分けることで並行処理を実現するマルチスレッドと, Go 言語のランタイムによって管理される複数の軽量スレッドを用いて並行処理を実現する Goroutine について説明する.

2.11.1　マルチスレッド

マルチスレッドとは, 一つのプロセスを複数のスレッドに分けて処理することにより, 処理速度の向上を実現する並行処理方式である. 各プロセスは最低一つのスレッドを保持する. 従って, 複数のプロセスを使用するプログラムを実行した場合は, 複数のスレッドを使用する. 複数のプロセスに存在するスレッドを切り替える場合, メモリ空間, 及びアドレス空間の切り替えが必要となるため, コンテキストスイッチが発生し, 処理に掛かる時間が増加する. また, 一つのプロセスは一つのアドレス空間でもあるため, プロセスを生成するためには, 新たなアドレス空間を確保する必要があり, メモリを浪費する. 一方で, スレッドの生成は, 現在のプロセスのアドレス空間を使用するため, システムへの負荷が小さくなる. 加えて, プロセスの切り替えにはアドレス空間の切り替えが伴うが, 単一プロセス上で動作するスレッドの場合, アドレス空間の切り替えは発生しない. 従って, スレッドの切り替えに要する時間は, プロセスの切り替えに要する時間に比べて短い. 通常, シングルコアの場合, CPU は一度に一つの処理しか実行することができないため, マルチスレッドを実装した場合でも, シングルスレッドに比べて一つの処理に掛かる時間が長くなることがある. 図 2.17 にシングルコア CPU とマルチコア CPU におけるマルチスレッドの処理時間の比較を示す. マルチスレッドは, 実行コアが複数存在するマルチコア CPU において, スレッドを用い

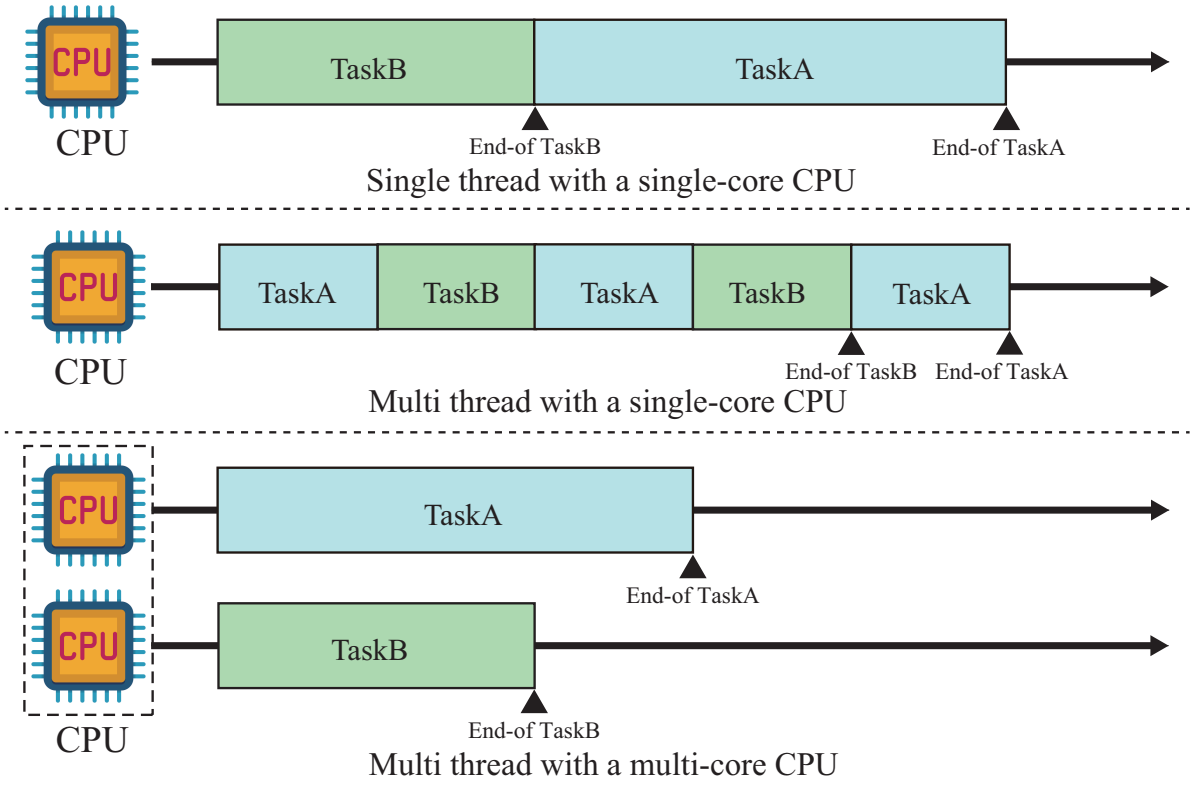


図 2.17: Overview of Multi-Thread processing time

ることによりアプリケーション内で必要に応じて多数の処理を並行して進められるため, 処理速度と精度が飛躍的に向上する. また, 現代の CPU のほとんどはマルチコアを実装しており, 複数のコアで一度に並行処理を実現することが可能である. 通常, プロセスの作成は OS によって行われ, メモリ, スタック, ファイルディスクリプタ, 及びソケットディスクリプタ等を割り当てることで特定の処理を実行することが可能となる. しかし, 接続されるクライアント毎にプロセスを割り当てる場合, 親プロセスの全ての状態をコピーする必要があるため, コストが高くなるという問題がある. 一方, スレッドは同一プロセスの中で複数のタスクを処理できるため, コストを削減することが可能である. また, 新しく生成されたスレッドは親スレッドと同じアドレス空間を共有するため, 親プロセスの状態を複製する必要がない. OS のスレッドスケジューラは, 一つのプロセス上で動作するプログラムにおいて, 各スレッドを順番に短時間ずつ処理を再開する. その際, 順番やタイムスライスはスレッド毎に設定されている優先度によって決定され, ランキューと呼ばれるリストに入っている実行予定のスレッドに対して, 公平に処理が回るようにスケジューリングされる. タイムスライスとは, スレッドが一回に実行する実行時間を指す. 一方, スレッドを生成, 及び破棄する場合には, OS に対して都度, 要求を送信する必要があるため, オーバーヘッドが発生する. また, スレッド間のコンテキストスイッチにはプログラムカウンタやメモリ参照場所の切り替えが必要となるため, 少なからず計算コストが掛かる.

2.11.2　Goroutine

Goroutine は, Go 言語プログラムにおいて最も基本的な構成単位であり, 軽量スレッドを用いて処理をすることで処理速度の向上を実現する並行処理方式である. Go 言語で記述されたプログラムが動作すると, Main Goroutine と呼ばれる Goroutine が最低一つ起動する. Main Goroutine は, プロセスが開始する際に自動的に生成され, 起動される. 生成された他の Goroutine 間の処理に優先度はなく, 排他制御を

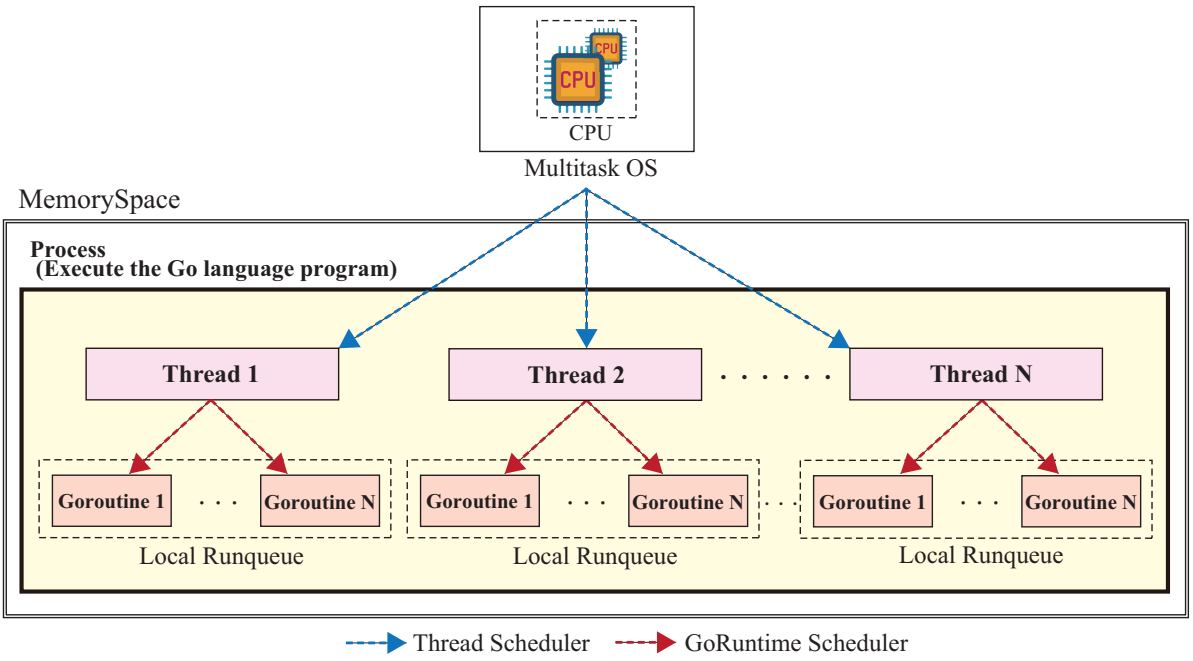


図 2.18: Relationship between thread and goroutine

実装しない限り, 全ての Goroutine は非同期で実行される. スレッドと Goroutine の位置付けを図 2.18 に示す. スレッドの場合, CPU コアに対してマッピングされ, OS によって管理されるが, Goroutine は OS のスレッドに対してマッピングされ, Go ランタイムによって管理される. Goroutine は通常の OS スレッドの ID と異なり, 指定しない限りどのスレッドにマッピングされるかは決定されていない. Go ランタイムは生成された Goroutine に対して以下の処理を実行する.

- カーネルから割り当てられたメモリを分割し, 必要に応じて割り当てる.
- ガベージコレクタを動かす.
- Goroutine のスケジューリングを行う.

Go ランタイムの概要図を図 2.19 に示す. Go ランタイムの内部は, 主にマシンスレッド, Goroutine, Go ランタイムスケジューラから構成される. マシンスレッドは Linux カーネルにおける物理 CPU コアに対応する. Goroutine はプロセスに相当し, Go ランタイムスケジューラは, OS が提供するスレッドスケジューラと同等の機能を提供する. 現代のカーネルは CPU の M 個のコア数に対し, 同時に N 個の複数の処理が可能な $M{:}N$ スケジューラを実装しており, ハイブリッドスレッディングを提供する. Go ランタイムスケジューラも同様に, N 個のカーネルスレッドに, M 個のユーザスレッドを対応させたものにスケジューリングされるため, 複数の CPU コアを扱うことが可能であり, $M{:}N$ スケジューラを実現する. Go ランタイムは sysmon と呼ばれる特殊なマシンスレッドが, プログラムの実行においてボトルネックとなる処理を常に監視する. その際, sysmon はスケジューラによって実行が止められることがないよう, sysmon が稼働しているマシンスレッドは, 特定の OS スケジューラから分離される. Go ランタイムスケジューラはランキュー等のスレッドが行う作業を束ね, マシンスレッド毎にタスクとして Goroutine のリストを保持する. 各スケジューラはマシンスレッドを保持しており, 実行可能な Goroutine をグローバルキューから取り出して, ローカルキューに追加する. タスクは Goroutine が実行される際, ローカルキューから Go ランタイムスケジューラによって取り出され, マシンスレッド上で実行される. また, G0 はマシンスレッドに割り当てて実行する Goroutine とは別に割り当てた, 特別な Goroutine であり,

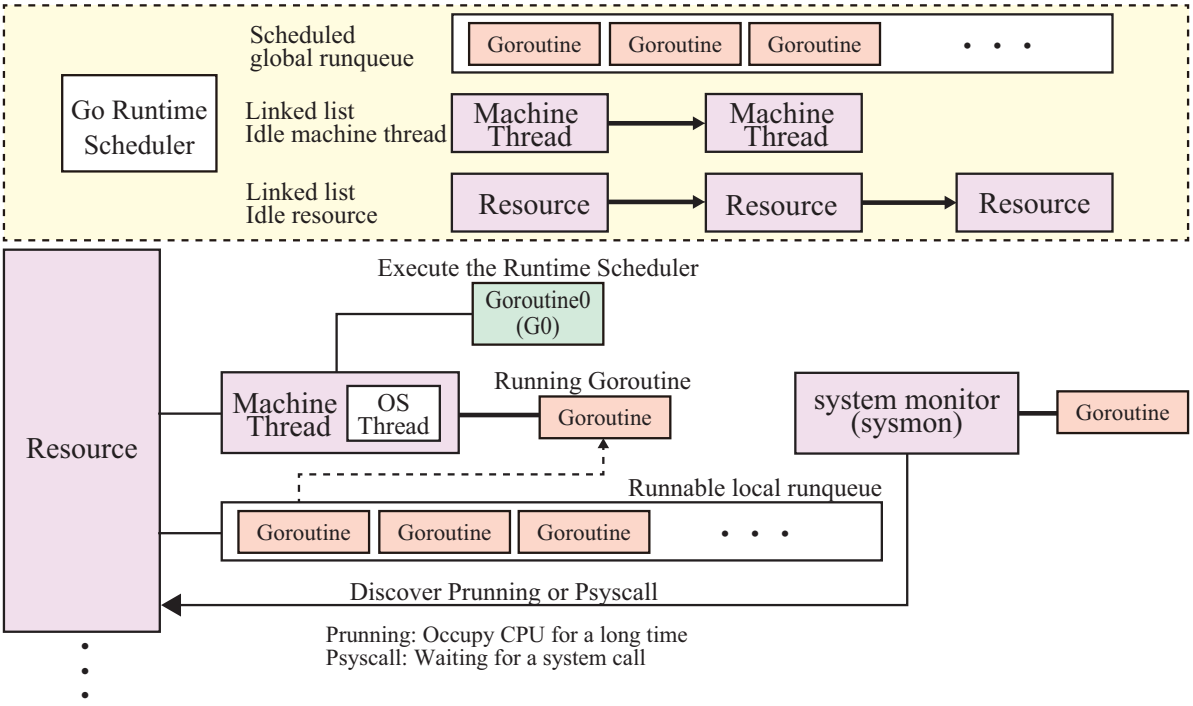


図 2.19: Overview of Go Runtime

実行中の Goroutine が待ち状態, もしくはブロック状態になると起動する. さらに, Go ランタイムスケジューラを動かすことの他に, Goroutine によって割り当てられたスタックの増減処理や, ガベージコレクションを独自に実行する. ガベージコレクションとは, プログラムが動的に確保したメモリ領域のうち, 断片化したメモリ領域, 及び不要になったメモリ領域を自動的に解放する機能である. Go ランタイムは常に一つの Goroutine を実行させることはなく, 適度に実行する Goroutine を取り替えることにより, プログラム実行の効率化を図る. 実行中の Goroutine を一旦中断する処理は, プリエンプションと呼ばれ, sysmon がボトルネックとなっている Goroutine を見つけると実行される. 実行のボトルネックになっている Goroutine とは, CPU を長時間占有している Goroutine や, システムコール待ち状態の Goroutine を指す. 以上より, Go は内部で Go ランタイムスケジューラを動作させることにより, OS カーネルが提供するスレッドのスケジューラとは分離して管理され, 異なる Goroutine を実行する際も OS スレッドの切り替えを伴わない仕組みを持つ. そのため, マシンスレッド上で実行されている Goroutine が Go ランタイムスケジューラによって切り替えられた場合にも, OS がコンテキストスイッチを認識することはない. つまり, Goroutine の切り替えは, カーネルスレッドの切り替えのように, プログラムカウンタやメモリ参照場所の切り替え処理が発生しない. 加えて, Goroutine は生成と破棄に関する操作が非常に低コストであるため, 生成, 破棄のたびに OS に要求を行うカーネルスレッドと比較して, 実行時間が極めて短縮され, 軽量に動作することが可能である.

2.12　ソフトウェアアーキテクチャ

現代のクラウドサービスにおいて, 様々なサービスを提供するミドルウェアは, 主に 2 つのアーキテクチャの何れかで設計される. 1 つ目は, Apache で利用されているスレッド駆動型アーキテクチャである. スレッド駆動型アーキテクチャは, マルチスレッド方式のソフトウェアアーキテクチャであり, 受信した通信リクエスト一つに対して専用のスレッドを生成し, 処理を行う. 2 つ目は, Nginx で利用されているイ

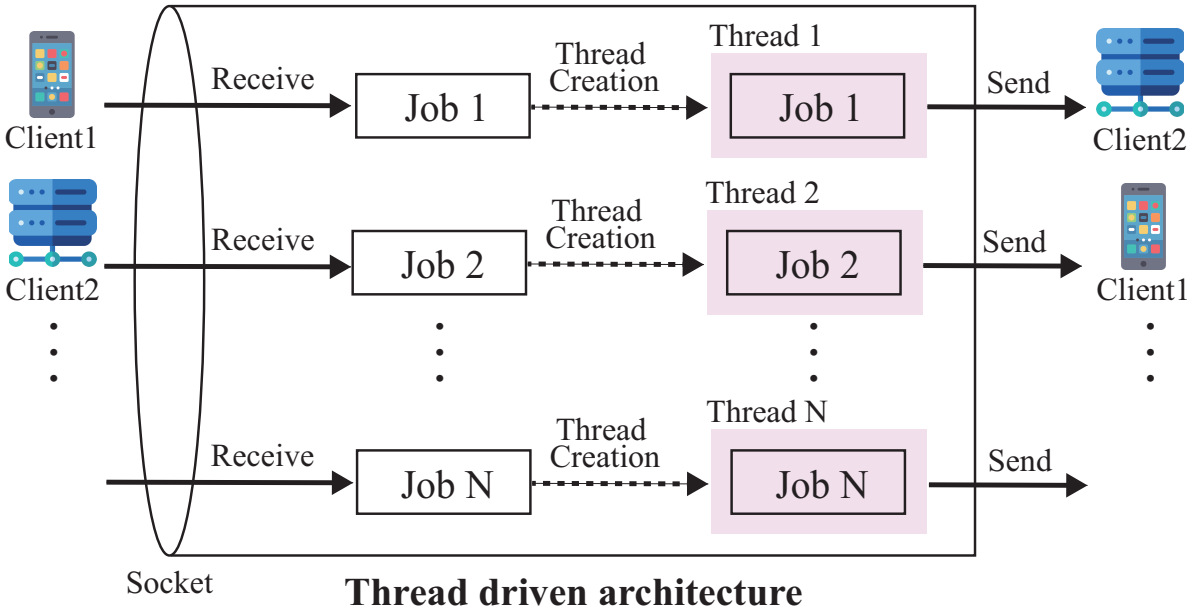


図 2.20: System Model of Thread driven architecture

ベント駆動型アーキテクチャである．イベント駆動型アーキテクチャは，受信した通信リクエストをイベントキューに格納し，事前に生成された複数のワーカースレッドに引き渡すことで，ワーカースレッドが処理を行う．以下に，スレッド駆動型アーキテクチャ，及びイベント駆動型アーキテクチャについて詳述する．

2.12.1　スレッド駆動型アーキテクチャ

スレッド駆動型アーキテクチャは主に Apache で利用されているアーキテクチャである．スレッド駆動型アーキテクチャを図 2.20 に示す．スレッド駆動型アーキテクチャは，マルチスレッド方式により処理を行うアーキテクチャであり，リクエストから生成されるジョブ毎に専用のスレッドを生成する．そのため，単一ジョブの計算量が多い高負荷リクエストの処理を得意とする．まず，スレッド駆動型アーキテクチャでは，クライアントからのリクエストジョブを受け取ると，ジョブを処理する専用のスレッドを生成する．その後，生成されたスレッドが処理を行うことで，レスポンスを生成して送信する．

スレッド駆動型アーキテクチャは，ジョブ毎に処理用スレッドを生成するため，一つ一つのリクエストの計算量が多い場合でも，他の処理に影響を与えにくい．一方で，同時接続数が増加した場合，スレッドの生成が都度発生することから，処理遅延や，C10K 問題が懸念される．C10K 問題とは，ハードウェアやネットワーク性能に問題がない場合でも，クライアントの同時接続数がある一定数を超えるとリソース不足により，サーバが処理をしきれなくなる問題である．

2.12.2　イベント駆動型アーキテクチャ

イベント駆動型アーキテクチャは主に Nginx で利用されているアーキテクチャである．イベント駆動型アーキテクチャを図 2.21 に示す．イベント駆動型アーキテクチャは事前に生成したワーカースレッドで処理を行うため，スレッドの生成や待機による処理遅延はほとんど発生しない．そのため，同時発生するリクエストの処理を得意とする．まず，イベント駆動型アーキテクチャでは，クライアントからのリクエストジョブを受け取ると，ジョブを親スレッドに格納する．次に，子スレッドはジョブ処理用のワーカー

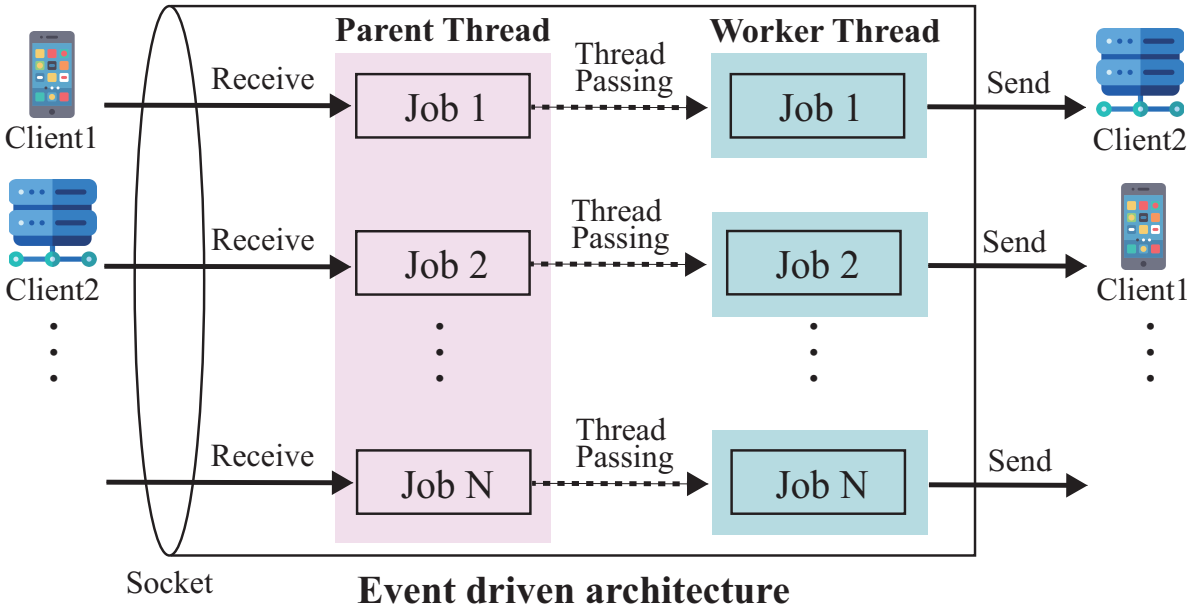


図 2.21: System Model of Event driven architecture

スレッドの状態を確認し, ワーカースレッドが処理可能な状態であれば, ジョブをワーカースレッドに引き渡す. その後, ジョブを引き渡されたワーカースレッドが処理を行うことで, レスポンスを生成して, 送信を行う. 一般に, ワーカースレッドは, CPU の論理コア数に応じて生成される.

マルチスレッド方式を採用するスレッド駆動型アーキテクチャに対し, イベント駆動型アーキテクチャではコンテキストスイッチを避けるために, シングルスレッドで処理を行う. また, 非同期なイベント駆動と, I/O の多重化により, ワーカースレッドにてリクエストが完全に処理されることを待たずに次の処理を実行可能である. そのため, 同時に発生した多数のリクエストを処理可能であり, C10K 問題にも対応可能である. 一方で, 膨大な CPU リソースを必要とする高負荷処理が発生する場面では, プロセスのブロックにより, 処理性能が劣化する可能性がある. また, ワーカースレッドによる非同期処理の制御が非常に複雑なことから, 実装において高度な技術力が要求される.

2.13　ネットワークソケット

ネットワークソケットとはアプリケーションがインターネットを介してデータを送受信するための仕組みを抽象化したものである. また, Socket Application Programming Interface (Socket API) はアプリケーションに対して, 異なるホスト間での通信をサポートするためのトランスポート層以下の機能を提供するプログラムインタフェースである. 一般的なネットワークソケットを用いたクライアント・サーバ間におけるメッセージの送受信の様子を図 2.22 に示す. データの送受信は, アプリケーションがディスクリプタと呼ばれるファイルを開き, 内蔵ディスクに対してデータを読み書きするのと同じ原理で行われる. アプリケーションはソケットを使ってネットワークに接続することで, 同じくネットワークに接続されている別のアプリケーションと通信を行うことが可能となる. 一方のマシンでアプリケーションがソケットに書き込んだ情報を, 相手のマシンで動作するアプリケーションが読み取ることで, インターネットを通じた相互通信が成立する. ソケットの種類として, 一般的に利用される TCP/IP プロトコルファミリのソケットにストリームソケットとデータグラムソケットが存在する. また, 既存のプロトコルを利用しつつ, 新たなプロトコルを実装する際に利用される Raw ソケットが存在する. 以下に, スタンダードソケット, 及び Raw ソケットについて詳述する.

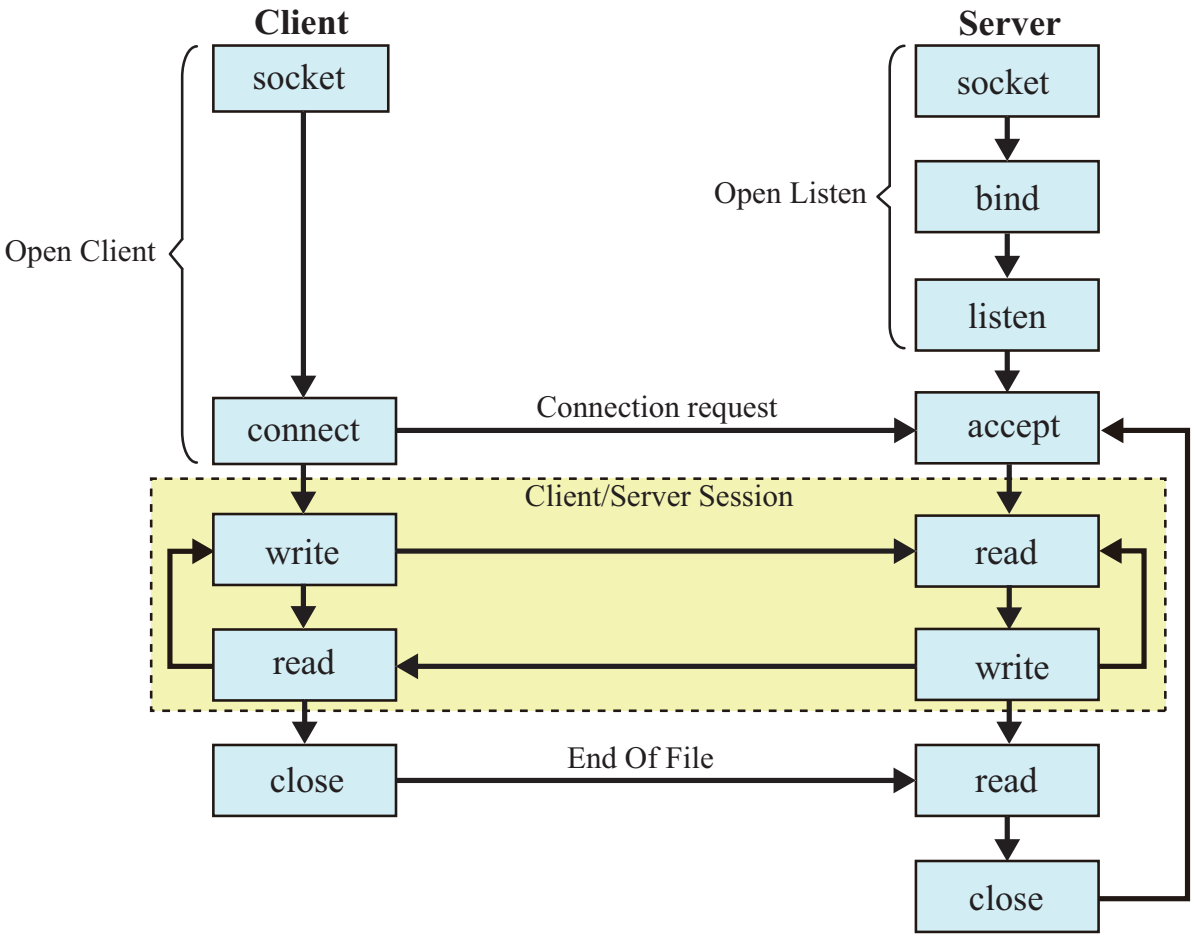


図 2.22: Overview of Socket API

2.13.1　スタンダードソケット

スタンダードソケットのうち, ストリームソケットは IP の上位層でエンドツーエンドプロトコルとして TCP を使用するため, 信頼性の高いストリームサービスが利用可能である. また, 同じくスタンダードソケットで定義されるデータグラムソケットは IP の上位プロトコルとして UDP を使用するため, ベストエフォート型のデータグラムサービスを利用することが可能である. ストリームソケットは確実な通信を実現する一方で, 通信制御に伴うオーバーヘッドが大きくなる. データグラムソケットは高速な通信を実現可能である. また, 最大で一度に約 65,500 バイトものメッセージを送信可能であるが, データ到達の確実性については保証されない. ストリームソケット, データグラムソケット等の一般的な標準ソケットにおいて, 何れも送信するペイロードはトランスポート層プロトコルによってカプセル化される.

2.13.2　Raw ソケット

Raw ソケットは, 通常処理されてしまう IP ヘッダや TCP/UDP ヘッダを含んだ Raw パケットを送受信する. 一般的に用いられる標準ソケットと Raw ソケットの送受信するパケットの違いを図 2.23 に示す. Raw ソケットは, ユーザ空間で新たなプロトコル層を実装することが可能であるため, 通信プロトコルの開発や既存プロトコルにアクセスする際に使用される. Raw ソケットを使用するプログラムにおいて, 生成したソケットからパケット操作を行う関数の動作を図 2.24 を用いて示す. Raw ソケットの場合, ソケットインターフェースは *PF_PACKET* を指定することにより, 変更が一切加えられていない受信直

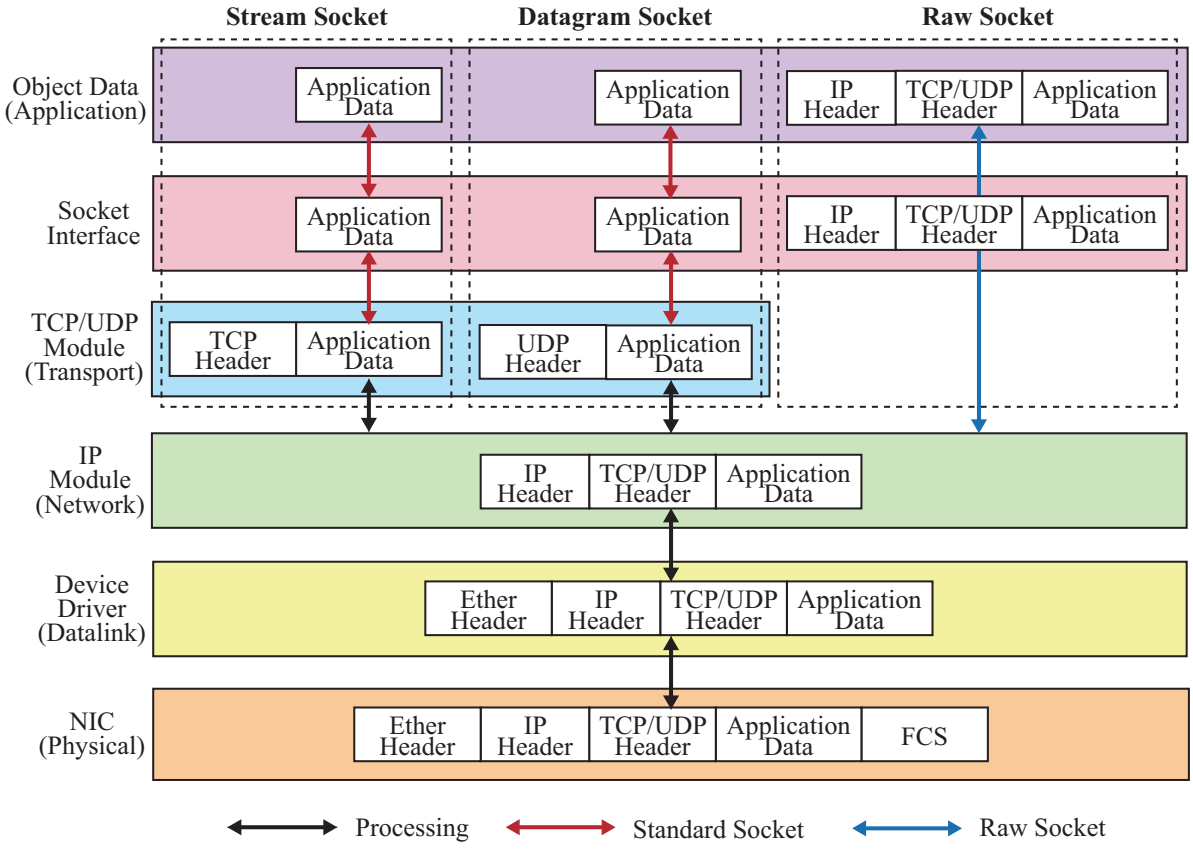


図 2.23: Packet handling using sockets

後の Raw パケットを, そのままユーザ空間のアプリケーションで受け取ることが可能となる. また, ソケットオプションとして *SOCK_DGRAM* を指定した場合, IP 層からアプリケーション層までのデータを取得する. また, *SOCK_RAW* を指定した場合, ソケットは IEEE 802.3 フレームの IEEE 802.2 Logic Link Control (LLC) ヘッダの生成や解析を行うことが可能なため, データリンク層からデータの取得が可能となる. 一方で, Raw ソケットを用いた場合にも, 自身の Media Access Control address (MAC) アドレス以外の宛先が指定されたパケットを受信した場合, 受信ソケットで読み出し可能なデータになる前に破棄される. そのため, ルータやブリッジ, リピータ等のネットワーク中継機を実装する機器では, 自身のアドレスとは無関係の宛先が指定されているパケットを受信可能とする, プロミスキャスモードが実装される.

2.14　ネットワークパケットフィルタリング

ネットワーク機器におけるパケットフィルタリング機能は, 通信機器やコンピュータの持つネットワーク制御機能の一つであり, 外部から受信したデータを管理者が設定した一定の基準に従って処理, または破棄するために適用される. また, ルータや L3 スイッチをはじめ, ネットワーク中継装置はパケットの転送時に, 特定のルールが適用されることが一般的である. パケットフィルタリング機能は, 受信した IP パケットやその中の TCP パケット, または UDP データグラムのヘッダ部分を解析して, 送信元 IP アドレス, 及び宛先 IP アドレス, 送信元ポート番号, 及び宛先ポート番号, またプロトコルの種類等の情報を取得する. そして, 得られた情報を元に予め設定された条件と比較して, パケットを通過させるか破棄するかを判断する. また, ファイアウォールとして機能することで, 端末に流入した悪意のあるパケットを処

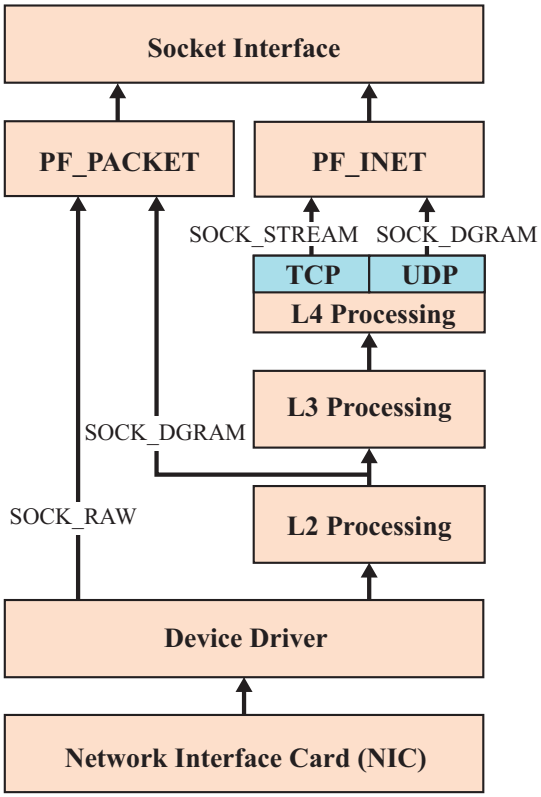


図 2.24: Overview of Socket Function

理する前に破棄することが可能である．一般に，フィルタリング条件の指定には，許可される条件に適合するリクエストのみを許可し，それ以外の全てをブロックするホワイトリスト方式と，特定の条件に一致するリクエストをブロックし，それ以外の全てを許可するブラックリスト方式が存在する．

ホワイトリスト方式は，許可されていないネットワーク通信を全てシャットアウトするため，高いセキュリティ性能を誇る．一方で，ファイアウォールを運用するためには，必要な通信を毎度設定しなければならないため，運用コストの増大や，専門的な知識が無ければホワイトリストを準備することが極めて困難となる．

他方，ブラックリストは不要な通信をピックアップして，リストに拒否する通信を増やす．現在は，ファイアウォールに限らずセキュリティ対策ツールの多くがブラックリスト方式を採用している．ブラックリスト方式では，システムのアップデートの度にブラックリストが更新されていくため，月日が経つごとにセキュリティ強度は高くなる傾向にある．一方で，ブラックリスト方式は同じ脅威を侵入させない対策として非常に有効であるが，未知の脅威に対してはセキュリティポリシを適用することが不可能である．そのため，今日ではセキュリティの要件，リスク管理戦略，システムの用途によって適切なフィルタリングルールを適用することが重要となる．以下に，パケットフィルタリング機能を提供する，netfileter，及び iptables を取り上げて詳述する．

2.14.1　netfilter

netfilter は，Linux カーネル内のネットワークパケットのフィルタリング，変更，制御を担当するカーネル空間のサブシステムである．特に，後述する Berkeley Packet Filter (BPF) 等により下位レイヤでネットワークパケットをフィルタリングした後，netfilter を用いて，それらに対するアクションを実行する．netfilter は，カーネル空間においてパケットがネットワークスタックを通過する際に，特定のフックポ

イントからカーネルにフック処理を行うことで, パケットの制御を行う. 具体的には, netfilter は, パケットのフィルタリング, NAPT, コネクショントラッキング, IP マスカレードや Destination NAT (DNAT), リストリクション等の機能を提供する. これにより, ネットワークセキュリティやルーティングポリシの実装が可能である. また, カーネル空間上でパケットフローを制御するため, 非常に高速かつ効率的なパケット処理を実現する. 一方で, netfilter はカーネルコードを制御する必要があることから, フィルタリングルールの適用が複雑化するという課題が存在する.

2.14.2 iptables

iptables は, Linux システムにおいて, ネットワークトラフィックの制御, フィルタリング, 転送等を管理するためのユーザ空間ソフトウェアである. netfilter は, カーネル空間で動作することで, 高速なパケット処理を実現する一方で, 制御が複雑であるという課題が存在した. 一方で, iptables は, ユーザ空間上で動作し, コマンドラインツールから操作をすることが可能なため, フィルタリングルールの設定が容易である. iptables の設定は, ルールセットを作成し, chain と呼ばれるカテゴリを適用することで, netfilter を制御する. また, chain の永続性を保つために, 通常は設定ファイルに保存して起動時に読み込むことで, フィルタリングルールを適用する. 現在では, フィルタリングルールに基づいたパケット処理をカーネル空間に存在する netfilter が担い, netfilter の制御をユーザ空間に存在する iptables が担うことで, Linux におけるパケットフィルタリングの適用が効率化されている.

2.14.3 Berkeley Packet Filter

Berkeley Packet Filter (BPF) は, 特定の OS 上において特にネットワークトラフィックの解析に必要なプログラムで使われる技術である. BPF は, UNIX 系 OS である Berkeley Software Distribution (BSD) の一部として開発され, 元来ネットワークトラフィックを効率的にフィルタリングするために使用されてきた. また, BPF を使うことで, カーネル内で受信パケットをフィルタリングしたり, パケットの統計情報を記録したりすることが可能となる. 昨今では, ネットワークパケットのキャプチャ, ファイアウォールルールの適用, ネットワークトラフィックの分析等に用いられている. BPF は, プログラム可能なフィルタリング機能を持ち合わせており, ユーザ空間プログラムを用いて特定のパケットを通過させたり, 破棄したり, ログに記録したりすることが可能である. また, BPF は, 低レベルなカーネルモードで実行するため, 非常に高速に動作する. しかし, BPF は主にカーネルで実行されるため, ユーザ空間プログラムから直接制御されるのではなく, カーネル内での実行が前提となっている. そのため, BPF の実行に伴い, ユーザ空間とカーネル空間で命令をやり取りする場合に, オーバーヘッドが発生し, パフォーマンスに影響を与える懸念がある.

2.14.4 extended Berkeley Packet Filter

BPF は, 低レベルなカーネルモードで実行され, ユーザ空間プログラムを通じて制御することで, 高速なパケット処理が可能な一方で, カーネル空間とユーザ空間を跨いで制御する場合に, オーバーヘッドが増大するという課題が存在した. 近年では, これらの課題を解決するため, 従来の BPF を拡張した, extended Berkeley Packet Filter (eBPF) が登場し, Linux カーネル内での様々なプログラム可能なタスクに使用されている. eBPF は, ユーザ空間とカーネル空間の間でプログラムを実行可能となるように拡張されており, ユーザ空間プログラムがカーネルに与える影響を制限することで, カーネルの安定性や信頼性を確保する. これにより, ユーザ空間のアプリケーションがカーネル空間の機能にアクセスし, 動的なプログラムを実行するとともに, カーネル機能を最小限のオーバーヘッドで容易に呼び出すことが可能になる. そのため, eBPF は BPF よりも柔軟性が高く, ユーザ空間からより多くの制御を行うことが可能となっている.

eBPF は, ネットワーク分野において, ファイアウォール, ロードバランサ, パケットフィルタリング等のネットワークの既存機能をカスタマイズするために用いられる. また, ネットワーク以外の分野でのカーネル内プログラムの実行にも適しており, セキュリティ, パフォーマンス分析, トレーシング, カスタムプログラミング等, 多岐に渡って用いることが可能である. 具体的には, eBPF は, Linux カーネル内で実行されるプログラムを許可し, カーネル内の様々なイベントやデータに容易にアクセスすることが可能となっている. そのため, ネットワークトラフィック, システムコール, パフォーマンスメトリクス等をリアルタイムで監視, 制御, 変更することが可能である. また, eBPF を使用して特定のシステムコールの呼び出しをトレースし, セキュリティインシデントを検出することも可能であるため, セキュリティ対策の一環としても利用される. 加えて, パフォーマンスのボトルネックを特定するために, CPU 使用率, メモリアクセス, ディスク I/O 等のメトリクスをリアルタイムで収集及び分析することで, システムを最適化することも可能である.

eBPF は, C 言語のサブセットを使用してプログラムを記述し, その後, Low Level Virtual Machine (LLVM) コンパイラを使用してバイトコードにコンパイルされる. ここで, C 言語のサブセットとは, 特定のプロジェクトや目的に合わせて一部の機能を限定して呼び出すことである. そのため, eBPF は安全かつ効率的なカーネルプログラムの実行が実現されている. また, 動的にプログラムをロード及びアンロードすることで, システムのランタイムで柔軟に変更することが可能である.

2.15　ネットワークホスト構成技術

インターネットを介して通信を行う全ての端末は, IP アドレスを一意に取得する必要がある. しかし, 所持する端末の全ての機器に手動で IP アドレスやデフォルトゲートウェイ, DNS サーバ情報等を設定するのは非常に手間であり, ネットワークに関する専門知識も要する. そのため, 現在では, コンピューターネットワーク上でホストが通信するための設定や構成を自動的に管理するための技術として, ネットワークホスト構成技術が普及している. ネットワークホスト構成技術を用いることで, 端末はアクセスポイントや有線で接続することで, サーバから自動的にネットワーク接続に必要となるアドレス情報を受け取ることが可能である. また, ネットワークが成長するにつれて, ホストの追加や変更が必要になる場合においても, ネットワークホスト構成技術を使用することでスケーラビリティを確保し, 柔軟にネットワークを拡張することが可能である. ネットワークホスト構成技術は, 特定の端末を指定してネットワークに接続したり, 重要なアプリケーションやサービスに優先順位を設けて通信帯域を制御したりする等, Quality of Service (QoS) を向上させる目的で利用される場合もある.

現在のインターネット環境は, IPv4 と IPv6 が混在したデュアルスタックネットワークとなっている. そのため, IPv4 のみをサポートする端末, また IPv6 のみをサポートする端末が存在する. ネットワークホスト構成技術において, アドレスを割り当てる重要なプロトコルとして Dynamic Host Configuration Protocol (DHCP) が挙げられる. DHCP は IPv4 アドレスを割り当てる Dynamic Host Configuration Protocol for IPv4 (DHCPv4) と, IPv6 アドレスを割り当てる Dynamic Host Configuration Protocol for IPv6 (DHCPv6) が存在する. 以下に, それぞれのアドレス割り当てに伴うシーケンスについて詳述する.

また, 一般にインターネットにアクセスするためには, IP アドレスから最終的に一意となる端末を識別する必要がある. TCP/IP では, IP アドレスから端末及び端末が使用するネットワークインターフェースを識別する技術として Address Resolution Protocol (ARP) が用いられている. ARP は主に, IP アドレスから Media Access Control address (MAC) アドレスと呼ばれる端末固有の識別子を解決するために, IPv4 ネットワークにおいて利用されるリンクレイヤプロトコルである. また, IPv6 では, ARP におけるいくつかの課題を解決し, ネットワークレイヤにおいて, アドレス解決を行うために Neighbor Discovery Protocol (NDP) が用いられている. 以下では, これらアドレス解決技術についても合わせて詳述する.

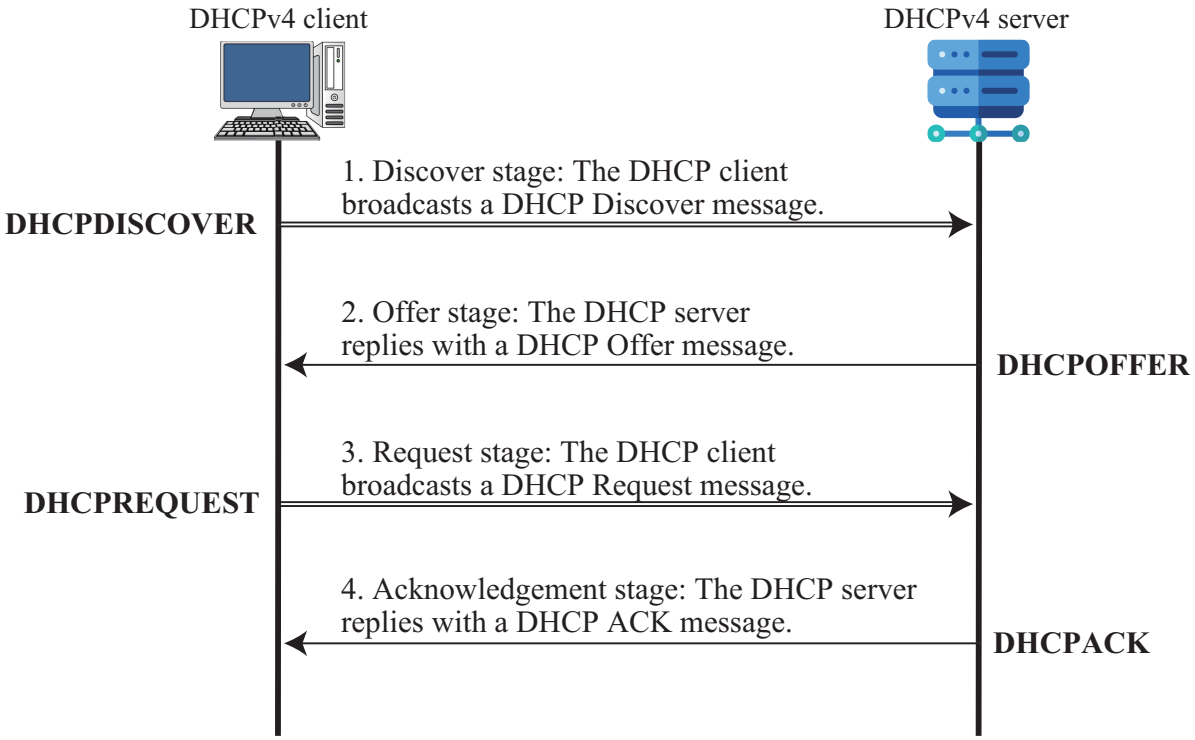


図 2.25: DHCPv4 Process

2.15.1　Dynamic Host Configuration Protocol for IPv4

Dynamic Host Configuration Protocol for IPv4 (DHCPv4) は, IPv4 ネットワークにおいてインターネットに接続されるホストの基本的な構成を自動的に行うためのプロトコルである. 一般にインターネットに接続するホストは, ネットワークを介した通信を行うため, 自身の IP アドレス, サブネットマスク, デフォルトゲートウェイ, DNS サーバのアドレス情報を設定する必要がある. ユーザは, DHCP サービスを用いることにより, 個々のホストに対して, 構成情報を手動で設定することなく, インターネットへ接続することが可能となる. ホストが DHCPv4 サーバによってアドレスを取得する際のシーケンスを図 2.25 に示す.

インターネット接続を試みるホストが起動すると, ネットワーク上の DHCPv4 サーバを探索するために, DHCPDISCOVER メッセージをブロードキャストする. DHCPv4 サーバは, DHCPDISCOVER メッセージを受信すると, 割り当て可能な IPv4 アドレスを予約し, ホストへ提示するために, DHCPOFFER メッセージをユニキャストで送信する. 次に, ホストは, DHCPOFFER メッセージによって DHCPv4 サーバから提示された IPv4 アドレスを使用するために, DHCPREQUEST メッセージをブロードキャストする. なお, LAN 内に複数の DHCPv4 サーバがある場合, クライアントは複数の DHCPOFFER メッセージを受け取る場合がある. そのため, ホストは受け入れる IPv4 アドレス情報を明示的に示す DHCPREQUEST メッセージを送信する. また, 同一ネットワーク内の DHCPv4 サーバによって以前にアドレスが割り当てられていた場合, 通常同じ IPv4 アドレスを再度要求する. ホストは同一のアドレスを要求する際に, ClientIPAddr フィールドに以前のアドレスを含めて DHCPREQUEST メッセージを送信する. DHCPv4 サーバは, DHCPREQUEST メッセージを受信すると, 提示した IP アドレスをホストに割り当てるために, DHCPACK メッセージをユニキャストで送信する. ホストは, DHCPv4 サーバから DHCPACK メッセージを受信すると, DHCPv4 プロセスを終了する.

DHCPv4 では, 主に 2 つの設定方式によってホストへアドレスを割り当てる. 1 つは, アドレスプールを用いて, 一定範囲から自動的に割り当てる, 動的アドレス割り当て方式である. 動的アドレス割り当て方

式では, ホストに割り当てるためのアドレスを予め一定数確保しておき, ホストが接続された際に動的にアドレスを割り当てる. ホストに割り当てるアドレスには, リース期間が設定されているため, 常に一つのアドレスを単一のホストが独占することがない仕組みとなっている. 動的アドレス割り当て方式は, 不特定多数のユーザを対象としてアドレス割り当てを行う場合に適している. 2 つ目は, 各ホストの識別子に対応した静的 IP アドレスを割り当てる, 静的アドレス割り当て方式である. ユーザは, 事前にホストの MAC アドレスを DHCP サーバに登録しておき, DHCP サーバは MAC アドレスを用いて各ホストを識別することで, 静的アドレスの割り当てを行う. 静的アドレス割り当て方式では, IP アドレスは MAC アドレスに紐付けられるため, リース期間が終了した場合にも, DHCP サーバは再度同じアドレスを割り当てる. これにより, 各ホストは常に同じ IP アドレスを保持することが可能である. 静的アドレス割り当て方式は, 特定のホスト, あるいは登録されたホストのみを対象としたアドレス割り当てを行う場合に適している. 現在では, 一部のアドレス帯域を動的アドレス割り当て方式によって払い出し, 任意の IP アドレスのみを静的アドレス割り当てに用いるために予約しておくことが一般的である.

2.15.2　Dynamic Host Configuration Protocol for IPv6

Dynamic Host Configuration Protocol for IPv6 (DHCPv6) は, IPv6 ネットワークにおいてインターネットに接続されるホストの基本的な構成を自動的に行うためのプロトコルである. DHCPv4 では, ホストが DHCPv4 サーバと DHCP メッセージをやり取りすることによって, 自動的にホストを構成する仕組みとなっていた. 一方で, DHCPv6 では, DHCPv6 サーバを用いたアドレス割り当て方式の他に, 自身で IPv6 アドレスを生成することが可能である. また, ルータと DHCPv6 サーバが連携して IPv6 アドレスを割り当てる方式も存在する. 一般に, DHCPv6 では, 接続先ルータから受信する Neighbor Discovery Protocol (NDP) パケットのうち, Router Advertisement (RA) メッセージに基づいてアドレス割り当て方式を決定する. NDP は, Internet Control Message Protocol for IPv6 (ICMPv6) メッセージとして定義されているネットワーク層のプロトコルである. ホストは, ルータが送信する RA に含まれる Managed address configuration (M) フラグ及び Other configuration (O) フラグを検証することで, 後続の処理方式を変更する. DHCPv6 におけるアドレス割り当て方式は, 大きく 3 つに分類される.

1 つは, Stateless Address Autoconfiguration (SLAAC) と呼ばれる, 端末自身でのアドレス自動設定方式である. SLAAC は, ネットワークアドレスを管理する DHCPv6 サーバが存在せず, ホスト自身がアドレスを生成する方式である. まず IPv6 をサポートするホストが起動すると, 近隣のルータに対して Router Solicitation (RS) と呼ばれるメッセージがマルチキャストで送信される. この時, 送信される RS は, 全ルータマルチキャストアドレス (All Router Multicast Address) と呼ばれる特殊な宛先 IP アドレスに対してマルチキャストで送信される. 近隣のルータは, 任意のホストから RS を受信すると, M フラグ, O フラグを設定した RA 返送する. この時, SLAAC として構成する場合, M フラグと O フラグは, 何れも false となる. 次に, ホストは RA を受信すると, 自身のネットワークインターフェース情報等を元に, アドレスを自動的に生成する. 通常, アドレスの自動生成には, Extended Unique Identifier-64bit (EUI-64) や, ランダム生成が用いられる. EUI-64 とは, EUI-64 プロセスによって 48 ビットの MAC アドレスを使用してインターフェイス ID を生成する機能である. また, ランダム生成は, ホストの OS 機構によって決定される乱数からインターフェイス ID を生成する機能である. ホストはアドレスを生成すると, Duplicate Address Detection (DAD) と呼ばれる手続きによって, 自身のアドレスが他ホストと重複しないことを確認する. DAD には, 同じく NDP を用いる. まず, DAD を実行するホストは, Neighbor Solicitation (NS) と呼ばれるメッセージを近隣ノードに対してマルチキャストで送信する. この時, 重複するアドレスを持つ近隣ノードが存在する場合, Neighbor Advertisement (NA) と呼ばれるメッセージを自身のアドレスを含めて応答する. ホストは, NS を送信した後, NA が受信されないことを確認すると, そのアドレスを自身のアドレスとして割り当てる. また, 一般にデフォルトゲートウェイや DNS サーバの情報には近隣ルータのアドレスが設定される. SLAAC は, DHCPv6 が存在しない環境においても, ホ

ストを自動的に構成することが可能である．しかし, 生成された IPv6 アドレスをネットワーク管理者が都度把握することは困難であるため, 管理や制御の点で課題が存在する．

2 つ目は, Stateless DHCPv6 と呼ばれる, ルータと DHCPv6 サーバが連携することアドレスを割り当てるアドレス割り当て方式である．Stateless DHCPv6 では, アドレス情報の取得に RA を使用し, 追加の設定パラメータは DHCPv6 サーバから入手するようにホストに通知される．まず, ホストは SLAAC と同様に, RS を近隣ルータにマルチキャストで送信する．Stateless DHCPv6 によるアドレス構成を想定した IPv6 ネットワーク環境では, ルータは M フラグを false, O フラグを true として RA を応答する．次に, RA に含まれる情報を元に 64 ビットプレフィクスと EUI-64 プロセスまたはランダムに生成されたインターフェイス ID を組みわせて IPv6 グローバルユニキャストアドレスを生成する．IPv6 グローバルユニキャストアドレスとは, IPv6 ネットワーク上で一意であり, グローバルにルーティング可能な IPv6 アドレスである．一般に, 配布されたプレフィックを元に, アドレスを生成する機能を DHCPv6-Prefix Delegation (DHCPv6-PD) と呼ぶ．ホストは, PD からアドレスを生成し, 上述した DAD の手続きによって, アドレス重複を確認する．また, Stateless DHCPv6 では, RA によって取得されなかった, DNS サーバの IPv6 アドレスリスト情報や Maximum Transmission Unit (MTU) に関する情報等, 一部のパラメータはアドレス生成とは別に DHCPv6 サーバから取得する．この時, DHCPv6 INFORMATION-REQUEST と呼ばれる, DHCPv6 プロトコルに準拠したメッセージが DHCPv6 サーバに向けて送信される．Stateless DHCPv6 は, アドレス割り当てに関する処理をルータに移譲し, DNS サーバやネットワーク通信に伴う一部の情報のみを DHCPv6 サーバで管理する．そのため, 複雑なアドレス割り当て方式を DHCPv6 サーバで管理する必要はなく, 最小限の情報のみを扱うことで効率的なアドレス割り当てと管理が実現可能である．一般に, Stateless DHCPv6 はアドレス管理の必要がない一方で, 通信の際に所定の DNS サーバにアクセスする必要がある場合に使用される．

3 つ目は, Stateful DHCPv6 と呼ばれる, DHCPv6 サーバから IPv6 アドレスを含む, 全てのネットワーク接続情報を受け取るアドレス割り当て方式である．Stateful DHCPv6 は, ルータとのやり取りを除き, 基本的な動作が DHCPv4 と同様である．まず, 前述の割り当て方式と同様に IPv6 をサポートするホストが起動すると RS を近隣ルータに送信する．近隣ルータは M フラグ及び O フラグをともに true として RA を応答する．ホストは, これらのフラグが立てられた RA を受信すると, DHCPv6 クライアントのプロセスを起動し, DHCPv6 サーバに全ての情報を要求する．最初に, DHCPv6 SOLICITATION と呼ばれるメッセージを DHCPv6 サーバに送信する．DHCPv6 サーバは, DHCPv6 SOLICITATION に対して, DHCPv6 ADVERTISEMENT をユニキャストで返送する．ホストは, DHCPv4 時と同様に, 複数の DHCPv6 ADVERTISEMENT を受信する可能性があるため, 任意の DHCPv6 サーバを選出して DHCPv6 REQUEST を特定の DHCPv6 サーバに向けてユニキャストで送信する．DHCPv6 サーバは, ホストから DHCPv6 REQUEST を受信したことを確認すると, 最終的に, DHCPv6 REPLY メッセージをユニキャスト送信する．ホストは, DHCPv6 REPLY に含まれるアドレスを自身のアドレスとして使用する．Stateful DHCPv6 は, 従来から利用される DHCPv4 に準拠したシーケンスで扱うことが可能であるため, IPv6 の複雑な手続きの影響を受けにくいというメリットがある．また, 割り当てるアドレスを制御可能なため, 管理の点においても他の 2 つの割り当て方式と比較して優れている．IPv6 では, 通常全てのアドレスがグローバル IP アドレスとなる．このアドレスは一般に, グローバルユニキャストアドレスと呼ばれる．しかし, 一部の要件では, IPv4 のようにプラベート IP アドレスを使用した LAN を構成する場合がある．Stateful DHCPv6 では, グローバルユニキャストアドレスの他, 特定のリンクで利用可能な ユニークローカルアドレスを割り当てることも可能である．IPv6 ユニークローカルアドレスは, IPv4 ネットワークにおける, プライベート IP アドレスに相当し, 直接グローバルネットワークにルーティングされることはない．従って, セキュリティの観点からも, 一部の端末を保護する目的で使用される．

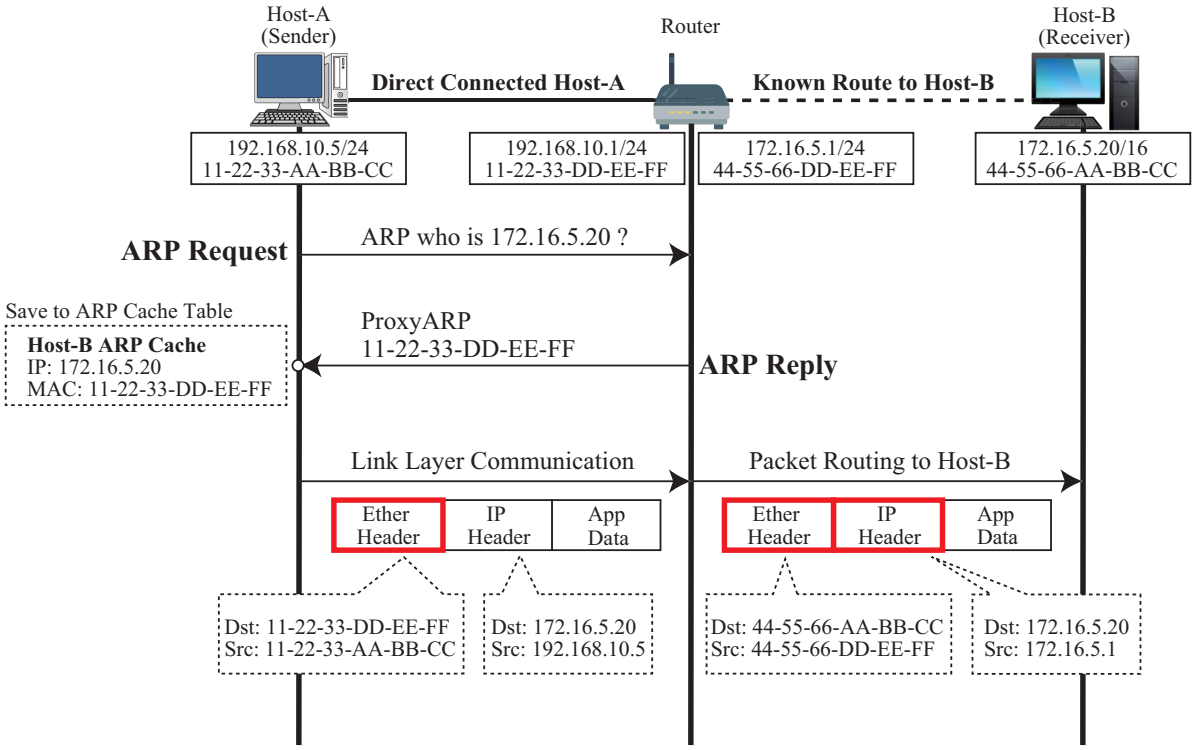


図 2.26: Overview of Proxy ARP

2.15.3　Address Resolution Protocol

Address Resolution Protocol (ARP) は, ホストの IP アドレスから MAC アドレスを動的に取得するためのアドレス解決プロトコルであり, 主に IPv4 ネットワークで用いられる. ARP は, ネットワークに接続するホストがイーサフレームを送信する際, 宛先ホストの IP アドレスを基に, 宛先ホストの MAC アドレスを取得するために使用される. MAC アドレスとは, コンピュータ機器を一意に識別するための識別子であり, Network Interface Card (NIC) に紐づくハードウェア固有の番号のことである. コンピュータネットワーク上で送受信される IP パケットは, イーサフレームにカプセル化されて伝送される. そのため, 最終的に宛先ホストへ IP パケットを送り届けるためには, 宛先ホストの MAC アドレスを指定する必要がある. 送信元ホストは宛先ホストの MAC アドレスを解決するために, まず, IP パケットに自身の IP アドレスと MAC アドレス, 宛先ホストの IP アドレスを記載した ARP Request をネットワーク上にブロードキャストする. ネットワーク上に存在する宛先ホストは, 自身の IP アドレス宛に送信された ARP Request を受信すると, 自身の MAC アドレスを送信元に ARP Reply パケットを生成して応答する. 次に, 送信元ホストは宛先ホストの MAC アドレスを取得すると, 宛先ホストの IP アドレスと対応付けて ARP テーブルへキャッシュとして保存する. 以後, ホスト間は, ARP テーブルを参照することで, リンクレイヤ通信を行う. また, MAC アドレスに割り振られた IP アドレスは変更されることがあるため, ARP キャッシュへマッピングされた情報は一定時間経過すると消去される. そのため, ホストは ARP キャッシュが消去された場合, 必要に応じて再度 ARP Request を送信する. なお, ARP キャッシュのクリア時間は, OS によって異なる.

また, 宛先ホストが LAN 外に存在する場合, ネットワーク境界に存在するルータが Proxy ARP を実行する. Proxy ARP とは, ルータが自身以外に対する ARP Request を受信した際に, ルートを知っている IP アドレスであれば, 宛先ホストの代理として自身の MAC アドレスを応答する機能である. Proxy ARP の概要を図 2.26 に示す. 一般に LAN 外に存在するホストと通信を行う場合, ルータによって IP パ

ケットをルーティングする必要がある．Proxy ARP では, 宛先ホストに代わり, ルータが自身の MAC アドレスを応答する．そのため, ARP テーブルには宛先ホストの IP アドレスと, ルータの MAC アドレスがマッピングされて保存される．故に, 送信元ホストが LAN 外に存在する宛先ホストへ通信を開始する場合, イーサフレームはルータに送信されるため, ルーティングを実行することが可能である．Proxy ARP はサブネット化されたネットワーク環境において, サブネットマスクの認識が困難な旧式ホストが通信をする際に, 一時的に使用されていた機構である．一方で, 現在はクラスレスアドレスでインターネットが構成されており, サブネットを設定できない旧式ホストはほとんど存在しない．また, サブネットを適正に実装することが可能となっていることからも, 昨今では, Proxy ARP の実装例は少なく, 単に ARP の代理応答という観点で用いるケースが一般的である．

2.15.4　Neighbor Discovery Protocol

IPv4 ではリンクレイヤプロトコルである ARP を用いて, ARP Request をブロードキャストすることで, 相手端末の MAC アドレスを解決する．一方, IPv6 は多くの処理がネットワーク層で実装されており, アドレス解決を行うためには Neighbor Discovery Protocol (NDP) が用いられる, NDP は, Internet Control Message Protocol for IPv6 (ICMPv6) として実装されており, ネットワーク層のプロトコルとして機能する．IPv6 は膨大な端末を集約することを前提に設計されたため, ブロードキャスト通信を排除することで, 不要なパケットが特定のネットワーク全域に伝搬されるのを防ぐ仕組みとなっている．そのため, 近隣解決プロセスにおいてブロードキャストは存在せず, IPv6 リンクローカルアドレスをベースに, マルチキャストメッセージ, またはユニキャストメッセージを用いて近隣端末の MAC アドレスを解決する．IPv6 では, 端末のネットワークインターフェースに IPv6 アドレスが設定されると, その IPv6 アドレスのインターフェース ID に対応する要請ノードマルチキャストアドレス (Solicited Node Multicast Address) のグループに自動的に参加する．要請ノードマルチキャストアドレスとは, IPv6 ネットワークにおいて, 近隣端末の MAC アドレスの解決を効率的に行うために使用される特別なマルチキャストアドレスでる．また, 要請ノードマルチキャストアドレスは, 対象の IPv6 ユニキャストアドレスの下位 24 ビットを取り出し, 先頭に ff02::1:ff を追加して構成される．IPv6 では, 同一リンク上に接続された端末に対して, マルチキャストを用いて, Neighbor Solicitation (NS) と呼ばれる MAC アドレスを解決するための ICMPv6 メッセージを送信する．該当する IPv6 アドレスが割り当てられた端末は NS を受信すると, 自身の MAC アドレスを含めて, Neighbor Advertisement (NA) と呼ばれるメッセージを生成し, 送信元の端末に ICMPv6 メッセージとして返送する．NS を送信した端末は NA を受信することにより, その端末に対して IPv6 アドレスを用いた通信を開始することが可能となる．

また, NDP における代理応答機能として, IPv4 における Proxy ARP に相当する機構として Neighbor Discovery Proxy (ND Proxy) が準備されている．ND Proxy は, NDP を使用して, ローカルネットワーク上のホストに対する IPv6 が割り当てらた近隣端末のリクエストを代理で処理する機能である．通常, IPv6 における近隣端末の MAC アドレスの解決は, ネットワーク上のホスト同士が, マルチキャストメッセージ, 及びユニキャストメッセージを直接交換することによって行われる．しかし, 一部のネットワーク環境では, 通信を中継するネットワーク機器等のミドルボックスが IPv6 通信の途中に介入する場合がある．このようなケースでは, NS メッセージが, 介在するミドルボックスによって阻まれることで, 相手端末の MAC アドレスを正しく解決することが困難となる場合がある．ND Proxy は, このような環境下において, IPv6 の近隣解決を補助する目的で使用される．ND Proxy では, 介在するミドルボックスが, NS メッセージを受信し, 相手端末の代わりに NA メッセージを応答する．この時, NA メッセージでは, 自身の MAC アドレスと相手端末の IPv6 アドレスをマッピングする．その結果, ND Proxy を用いることで, IPv6 通信が中継されるネットワーク環境において, 近隣解決のプロセスを効率的に行うことが可能となる．

2.16　コンテナ仮想化技術

コンテナ仮想化技術は, アプリケーションやサービスを環境から分離し, 独立した実行環境で動作させるための技術である. これにより, 異なる環境間でアプリケーションを移植する際の依存関係や互換性の問題を軽減し, 運用に伴うコストが軽減されるメリットがある. また, 冪等性を保証することにより水平スケーリングが容易となる. コンテナ仮想化技術は, 従来の Kernel-based Virtual Machine (KVM) や Hyper-V をはじめとしたカーネルベースの仮想化技術よりも軽量で素早いデプロイメントが可能である. 従来の仮想マシンは, ハイパーバイザによって仮想的なハードウェアを提供し, その上で完全な OS が実行されるため, 起動や停止に時間を要し, リソースの使用効率が低いという課題が存在した. このような背景を踏まえ, コンテナ仮想化では, ホスト OS を共有し, アプリケーション毎に必要なライブラリやランタイムを含む軽量な仮想化環境を提供することで, 仮想化における従来の課題を解決する.

コンテナ仮想化技術の普及は, Docker や Kubernetes 等のツールやプラットフォームの台頭によって, 近年, 益々加速している. 特に Docker は, コンテナイメージの作成や配布, 実行を容易にすることで, 開発者や運用者にとって利便性の高い解決策を提供した. また, Kubernetes はコンテナの自動化されたデプロイメントやスケーリング, 管理を可能にし, 大規模なコンテナベースのアプリケーションの運用を支援する. コンテナ仮想化技術は現代のソフトウェア開発やクラウドサービスの運用において不可欠な要素であり, マイクロサービスアーキテクチャの普及と相まって, 広範囲に採用されている. 以下に, 代表的なコンテナ仮想化技術である, Docker と, 多数のコンテナを分散, オーケストレーションすることで大規模なインフラストラクチャを構築可能な Kubernetes を取り上げて詳述する.

2.16.1　Docker

Docker は, アプリケーションの開発, 配布, 実行, 管理を容易にするためのプラットフォームであり, コンテナ仮想化技術の代表的なソフトウェアである. Docker は, 軽量な仮想環境としてコンテナと呼ばれる単位でアプリケーションをパッケージ化する. また, Docker エンジンは, このプロセスを支える中核となるコンポーネントであり, コンテナの作成や管理を行う. アプリケーションは Docker イメージとしてパッケージ化されることで, コードの依存関係, 実行に必要な全ての要素をシングルバイナリで保持する. これらのイメージは, Docker イメージと呼ばれ, 通常 Docker Hub をはじめとする, インターネット上に設けられたコンテナイメージレジストリで共有, 管理される.

また, 多数の Docker コンテナを束ねる技術として Docker Compose が存在する. 通常, アプリケーションは, フロントサービスからバックエンドサービス, ミドルウェアを含め, 複数のコンテナから構成される. そのため, コンテナを個別に管理して, それらを接続すること困難である. そこで, Docker Compose を用いてアプリケーションの管理を簡素化することで, コンテナベースのアプリケーションをひとまとまりにすることが可能である. Docker Compose は, 一般に YAML Ain't a Markup Language (YAML) や JavaScript Object Notation (JSON) を使用してアプリケーションの構成を記述し, 一括して起動, 停止, 削除等の操作を行うことが可能である. Docker は, 開発者や運用者にとってアプリケーションの配布と実行を簡素化し, 環境間の整合性や移植性の問題を解決する強力なツールとして広く使用されている.

Docker Compose は 単一ホスト上で動作する複数のコンテナアプリケーションを管理することが容易である. しかし, 複数のホストに跨った運用や, クラスタ上でのスケーリングや冗長性の確保が極めて困難な課題がある. また, 自動的な障害復旧やスケーリング等の高度な機能は提供されていない. そのため, ローカルホスト上でのサービス開発が容易な一方で, 実運用には不向きである.

2.16.2　Kubernetes

Kubernetes は, クラウドネイティブアプリケーションを管理するための分散コンテナ実行基盤である. Docker Compose は単一ホスト上でのサービス開発に特化している一方で, 実運用には向かないという課題があった. Kubernetes は, 複数のホストやクラスタ上でコンテナ化されたアプリケーションを管理, 自動的なスケーリング, 障害復旧, 負荷分散等の機能を提供する. また, Kubernetes は, コンテナオーケストレーションの標準として広く採用されており, 大規模なクラウド環境でのアプリケーションの展開と運用に適している. コンテナオーケストレーションとは, コンテナ化されたアプリケーションの管理を自動化するプロセスであり, 大規模なクラウド環境において, 複数のコンテナを効率的にデプロイ, スケーリング, 管理するためには欠かせない仕組みである. 『2019 Docker Usage Report』によれば, Kubernetes は, コンテナオーケストレーションエンジンのシェアのうち 約 77 % を占めており, 他の OpenShift 約 9 %, Docker Swarm 約 5 % と比較して圧倒的な普及率を誇る [75].

2.16.2.1　Kubernetes の発展

Kubernetes は元々, Google が管理する複雑なインフラストラクチャを効率的に動作させるために提案された. 当初, Borg と呼ばれる内部プロジェクトによって発足し, 大規模なクラスタ管理システムとして 2003 年頃から開発が進められた. Borg は, Google のサービスやジョブを数千台以上のマシンで効率的に実行し, スケジューリング, ロードバランシング, 障害復旧等の機能を提供した. 2014 年, Google Cloud Platform は, Google の社内で使われているツール類が全て分散コンテナ実行基盤上で動作していることを明らかにしており, 毎週 20 億個以上のコンテナを起動していると発表した. 従って, 今日, 我々が使用する Google のソリューションは, そのほとんどがコンテナによって提供されている [76].

Borg の成功を受け, Google はそのアーキテクチャやアイデアをベースにしたオープンソースのプロジェクトを開発し, 現在では Kubernetes と称して世の中に普及している. 2014 年に Google は Kubernetes を公式に発表し, Apache 2.0 ライセンスの下でオープンソースとしてリリースした. Kubernetes は, Borg のアーキテクチャや概念を引き継ぎながら, クラウド環境でのコンテナ化されたアプリケーションの管理を容易にするよう拡張された.

Kubernetes は 2015 年 7 月頃に バージョン 1.0 として General Availability (GA) を迎えるとともに, Google は Linux Foundation と共同で Cloud Native Computing Foundation (CNCF) を設立し, Kubernetes を主となる技術として提供した. 現在では, CNCF がオープンソースとして Kubernetes を管理しており, 業界全体での開発と普及が促進している. さらに, 近年では Google Cloud Platform (GCP), Amazon Web Services (AWS), Microsoft Azure 等のパブリッククラウドがマネージド Kubernetes エンジンとして, それぞれ Google Kubernetes Engine (GKE), Elastic Kubernetes Service (EKS), Azure Kubernetes Service (AKS) を提供している. パブリッククラウドのサービスでは Kubernetes の複雑な仕組みが抽象化されているため, 容易に扱うことが可能である.

2.16.2.2　Kubernetes の特長

Kubernetes は, 宣言的設定に基づき, 常にあるべき状態にインフラストラクチャリソースを保つように,「監視」「差分の取得」「制御」を行う. 宣言的設定とは, 目標の状態を定義するのみであり, 手動での状態管理や手作業の構成を最小限に抑えることで管理を容易にする管理方式のことである. また, Kubernetes における宣言的設定においてマニフェストと呼ばれる設定はコードによって定義することが可能である. 一般に, インフラストラクチャの定義をコードで管理することを Infrastructure as Code (IaC) と呼ぶ.

通常, Kubernetes において, 宣言的設定に基き, インフラストラクチャリソースの状態を維持する処理を調整ループ (Reconciliation Loop) と呼ぶ. Kubernetes は調整ループを主要な機能として, さらに以下の機能を提供する.

- 自動的なロールアウト及びロールバック (Automated rollouts and rollbacks)
  Kubernetes を利用するアプリケーションは, デプロイメントにおいて自動化されたロールアウトとロールバックを利用可能である. これにより, 新しいバージョンのアプリケーションを安全かつ効率的に導入し, 問題が発生した場合に, 前の安定したバージョンに容易に切り戻すことが可能である.

- ストレージオーケストレーション (Storage orchestration)
  Kubernetes はストレージのオーケストレーションをサポートし, 永続化されたデータの管理を容易にする. これにより, データベースやその他の永続化されたアプリケーションをクラウドネイティブな環境で実行可能である.

- コンテナ配置の自動化 (Automatic bin packing)
  Kubernetes はリソースを効率的に利用するため, コンテナをクラスタ内のワーカーノードに自動的に配置する. これにより, リソースの最適な利用とコストの最小化が実現される.

- デュアルスタックサポート (IPv4/IPv6 dual-stack)
  Kubernetes は IPv4 と IPv6 のデュアルスタックをサポートしており, ネットワークの柔軟性と拡張性を提供する. これにより, IPv4, 及び IPv6 の何のネットワークに所属する端末からのリクエストも受け付けることが可能である.

- 障害からの自動復旧 (Self-healing)
  Kubernetes はコンテナやワーカーノードそのもので発生した障害に対処するために, 自己修復機能を提供する. サーバが何らかの原因によって停止した場合, 自動的に影響を受けたコンテナを再起動し, 他のワーカーノードにコンテナを移動させることで, アプリケーションの可用性を確保する.

- サービスディスカバリと負荷分散 (Service discovery and load balancing)
  Kubernetes はサービスのディスカバリとロードバランシングを自動的に管理する. サービスのディスカバリは, 複数のサーバを DNS サービスを用いて探索することである. これにより, クライアントは DNS を用いて特定のサービスにアクセスすることが可能である. また, DNS-Round Robin (DNS-RR) を用いることにより, 個々のコンテナに掛かる負荷が平準化される. 現在, ではサービスディスカバリを実現する kube-dns と呼ばれる DNS 機構は, CoreDNS によって実現されている.

- 機密情報と構成管理 (Secret and configuration management)
  Kubernetes はシークレットや設定情報の管理を提供し, アプリケーションに安全に秘密情報を渡すことが可能である. これにより, デプロイメント時のセキュリティを強化し, 機密情報の漏洩を防止する.

- バッチ実行 (Batch execution)
  Kubernetes は, バッチジョブの実行をサポートしており, クラスタのリソースを効率的に活用する. これにより, 定期的なジョブの実行やデータ処理等のタスクを自動化することが可能である.

- 水平オートスケーリング (Horizontal scaling)
  Kubernetes はアプリケーションの負荷に応じてコンテナの数を自動的に水平スケーリングする機能を提供する. これにより, アプリケーションのパフォーマンスを維持しながら, リソースの効率的な利用が可能になる. また, リクエストに応じてコンテナの台数を増やすことで, 単一コンテナに掛かる負荷を軽減することが可能である.

- 拡張性を考慮した設計 (Designed for extensibility)
  Kubernetes は, 拡張性を重視して設計されており, カスタムリソースやカスタムコントローラを追加して, 独自の機能を実装することが可能である. これにより, Kubernetes の機能をカスタマイズして, 特定のニーズや要件に適用することが可能である.

2.16.2.3　Kubernetes の構成要素

Kubernetes をデプロイすると, クラスタが展開される. Kubernetes クラスタは, コンテナ化されたアプリケーションを実行するデータプレーンと, データプレーンを制御するコントロールプレーンから構成される. また, 全てのクラスタには少なくとも 1 つのワーカーノードが存在する. ワーカーノード, 及びデータプレーンは, アプリケーションのコンポーネントである Pod を最小単位としてホストする. マスターノード, 及びコントロールプレーンは, クラスタ内のワーカーノードと Pod を制御する中核機能を担う. また, 実際の運用では複数のマスターノードを使用することで, クラスタのフェイルオーバと高可用性を実現する.

コントロールプレーンコンポーネントは，クラスタに関する全体的な決定を行う. また, クラスタイベントの検出及び応答を行う. 以下に, コントロールプレーンコンポーネントの構成要素を説明する.

- kube-apiserver
  kube-apiserver は, Kubernetes API を外部に提供する. kube-apiserver は Kubernetes コントロールプレーンのフロントエンドとして機能し, マニフェストと呼ばれる宣言的な構成ファイルを受信する. また, 水平スケールが可能であり, 複数の kube-apiserver インスタンスを実行することで, インスタンス間のトラフィックを分散させることが可能である.

- etcd
  etcd は, 一貫性, 高可用性を持つ Key-Value Store (KVS) であり, Kubernetes クラスタ内の全ての情報を保管する.

- kube-scheduler
  kube-scheduler は, ワーカーノードのリソース状態やリクエスト頻度を都度確認して Pod のスケジュールを決定する. スケジューリングの決定は, Pod または Pod 群のリソース要求量, ハードウェア, ソフトウェア, その他 ポリシによる制約, アフィニティ及びアンチアフィニティの指定, データの局所性, ワークロード間の干渉, 有効期限等を考慮して行われる. アフィニティとは, Kubernetes において特定のポッドが特定のワーカーノードにスケジュールされる条件を指定する機能である. また, アンチアフィニティは, 特定のポッドが特定のワーカーノードと同じワーカーノードにスケジュールされないようにする条件を指定する機能である.

- kube-controller-manager
  kube-controller-manager は, コントロールプレーンにおける他のコンポーネントの管理及び制御を行う. 論理的には, 各コントローラは個別のプロセスとして動作するが, 複雑さの観点から一つの実行ファイルにまとめてコンパイルされ, 単一のプロセスとして稼働する.

- cloud-controller-manager
  cloud-controller-manager は, パブリッククラウドサービスを制御するための制御コンポーネントである. クラスタをクラウドプロバイダの API とリンクし, クラスタのみで相互作用するコンポーネントからクラウドプラットフォームで相互作用するコンポーネントを分離する役割を担う. cloud-controller-manager は, クラウドプロバイダ固有のコントローラのみを実行するため, セルフホスティングされた Kubernetes に, 当コンポーネントは存在しない.

また, データプレーンコンポーネントは全てのワーカーノードで実行され, 稼働中の Pod の管理や Kubernetes そのものの実行環境を提供する. 以下に, データプレーンコンポーネントの構成要素を説明する.

- kubelet
  クラスタ内の各ノードで実行される Kubernetes クライアントのデーモンプログラムであり, コントロールプレーンとの接続に用いられる. コントロールプレーンは, kubelet を通じて, 各ワーカーノードの状態を把握し, Pod のスケジュールを行う. また, その他データプレーンの制御は全て kubelet が担う.

- kube-proxy
  kube-proxy は, クラスタ内の各ワーカーノードで動作するネットワークプロキシである. kube-proxy は, ワーカーノードの netfileter, iptables を利用することで, リクエストを特定の Pod, あるいはコンテナに転送する. また, ロードバランス機構を兼ね備えており, Pod セットに対して送信されるリクエストが平準化されるように調整する.

- コンテナランタイム
  コンテナランタイムは, ワーカーノード内でコンテナを実行するためのソフトウェアである. 現在では, コンテナランタイムとして Docker エンジンにも使用される containerd がデファクトスタンダードとなっている.

2.17　負荷分散技術

インターネットでは, サービスを提供するインフラストラクチャにおいて, 多数のサーバを用いることで単一サーバの処理負荷を軽減する負荷分散機構が用いられることがほとんである. また, クラスタ化されたサーバの前段にロードバランサが設置されることで, ネットワークトラフィックを複数のサーバに振り分ける. これにより, 各サーバに掛かる負担が軽減するとともに, 単一サーバあたりの処理効率が向上し, パフォーマンスの高速化やレイテンシの低減が可能となる. そのため, 負荷分散技術は, アプリケーションサービスが正常に機能するために必要不可欠である.

一般にロードバランサは, トランスポート層をサポートする L4 ロードバランサと, アプリケーション層をサポートする L7 ロードバランサに分類される.

- Load Balancer for Layer 4 (L4LB)
  L4LB は, トランスポートレベルでの負荷分散を行う. 通常, クライアントの IP アドレスやポート番号に基づいてトラフィックをバックエンドサーバに分散する. 代表的な製品として, BIG-IP, Citrix NetScaler, Citrix ADC が挙げられる.

- Load Balancer for Layer 7 (L7LB)
  L7LB はアプリケーションレベルでの負荷分散を行う. HTTP ヘッダや URL, Cookie 等の情報に基づいてトラフィックをバックエンドサーバに分散する. L7LB を用いることにより, アプリケーションの特定の要求に基づいて負荷分散を行うことが可能性である. 代表的な製品として, NGINX Reverse Proxy, NGINX Ingress Controller, HAProxy が挙げられる.

ロードバランサは後段に位置するサーバの構成に応じていくつかの設置方式及びアーキテクチャを選定して設けられる. 以下に, 一般的に用いられるロードバランサ構成として, Avtive-Active 構成, Active-Standby 構成, Direct Server Return (DSR) 構成を取り上げて詳述する. また, セルフホストされたベアメタル Kubernetes において, ロードバランス機構を提供するソフトウェアとして MetalLB を取り上げて詳述する.

2.17.1　Active-Active 構成

Avtive-Avtive 構成とは, 複数のロードバランサを同時にアクティブな状態で動作させる構成のことである. Avtive-Avtive 構成では, 全てのロードバランサがトラフィックを処理し, クライアントからのリクエストを均等に分散する. そのため, 負荷を均等に分散するとともに, ロードバランサ自体の冗長性も確保することが可能となっている. また, 何れかのロードバランサ間でハートビートを使用して相互監視を行い, フェイルオーバを実現するため, シームレスなスケーラビリティと高可用性を提供する. 一方で, 複数のロードバランサを管理する必要があるため, 構成の同期や監視等の管理タスクが増加する. また, 複数のロードバランサ間で常にトラフィックの同期を行う必要があるため, ネットワークのオーバーヘッドが伴う. 加えて, ロードバランサ間のトラフィック同期や相互監視により, 一部のリソースが消費され, パフォーマンスが低下する可能性がある.

2.17.2　Active-Standby 構成

Active-Standby 構成とは, 1 つのロードバランサのみをアクティブとし, 他のロードバランサをスタンバイとして待機させる構成のことである. アクティブなロードバランサが何らかの理由により停止した場合, スタンバイ状態のロードバランサが自動的にアクティブに切り替わることで, ロードバランス機能を引き継ぐ. そのため, Avtive-Avtive 構成と比較してリソースの無駄を最小限に抑えるとともに, 可用性を確保することが可能である. また, シンプルなロードバランス構成であるため, 設定や管理が容易になるというメリットがある. 一方で, アクティブ・スタンバイの切り替え及びフェースオーバに伴い, 少なからずダウンタイムが発生する可能性がある. また, スタンバイのロードバランサがアクティブになるまでに, 一定の期間を要する場合があり, 一時的なサービスの中断に繋がる可能性がある.

2.17.3　Direct Server Return 構成

Direct Server Return (DSR) 構成とは, ロードバランサがクライアントからのリクエストを受け取ってバックエンドサーバに転送した後, サーバが直接クライアントにレスポンスを返す構成である. つまり, レスポンスパケットはロードバランサを経由することなく, 直接サーバからクライアントに返送される. DSR 構成において, ロードバランサはリクエストパケットの転送処理に集中することが可能である. そのため, レスポンストラフィックを制御する必要がなく, ロードバランサ自体の処理負荷を軽減することが可能である. 前述の, Active-Active 構成及び Active-Standby 構成が, 物理的なロードバランサを複数台設置することで, 処理負荷の軽減と可用性を実現する一方で, DSR 構成は, トラフィックの流入経路そのものを変えることで可用性や処理効率の向上を実現する. また, サーバが直接クライアントに応答するため, レスポンスタイムが短くなり, スループットが向上するメリットもある. さらに, ロードバランサとサーバ間の通信が最小限に抑えられるため, パケット転送に伴うネットワーク遅延も軽減される. 一方で, サーバはクライアントに直接レスポンスパケットを返送するため, クライアントからのリクエストに応じてリダイレクトやリライアビリティに関する追加の設定が必要となる. また, サーバがクライアントに直接応答するため, サーバのネットワーク設定が複雑となり, セキュリティの問題が生じる懸念もある.

2.17.4　MetalLB

Kubernetes では, Service や Ingress と呼ばれるリソースを定義することにより, 自動的に Software Load Balancer (SLB) が追加され, ワーカーノード及び Pod に対するリクエストを分散することが可能である. 通常, Kubernetes では Service リソースを定義することで, L4LB による外部トラフィックを Pod にルーティングする. また, Ingress リソースを定義することで, L7LB を追加することが可能であり, ア

プリケーションにおける HTTP パスルーティングを実現する．しかし，これらのリソースやロードバランシングの機能は標準の Kubernetes には搭載されていない．そのため，GCP や AWS，Microsoft Azure 等のパブリッククラウドでは，ロードバランサを設置するためのマニフェストを準備することで，マネージドな SLB を払い出して利用する．

一方で，セルフホストされたベアメタル Kubernetes では，ロードバランサを自前で準備する必要がある．しかし，既存のロードバランサソリューションを Kubernetes クラスタに統合する場合，構成が複雑化し，管理コストも増大する課題が存在する．このような背景を受け，Cloud Native Computing Foundation (CNCF) は，Kubernetes ネイティブな SLB ソリューションとして MetalLB を提供している．MetalLB は，Kubernetes クラスタ内で外部からのトラフィックを受け取り，サービス間の負荷を分散するネットワークロードバランサを実現する．MetalLB は，Service リソースが *type: LoadBalancer* として構成された場合にのみ動作し，Service に割り当てられた外部 IP アドレスをワーカーノードに割り当てる．一般に外部 IP アドレスは，Dynamic Host Configuration Protocol (DHCP) 機構によって所属ネットワークの DHCP サーバから取得する．また，MetalLB はクラスタ内のリソースの使用量や外部 IP アドレスの利用可能性等を考慮して，IP アドレスのプールを効率的に管理する．そのため，カスタム設定や拡張性の高いアーキテクチャを提供し，Kubernetes クラスタ内のアプリケーションに対して信頼性と柔軟性を担保する．

クライアントからのリクエストが，Service に割り当てられた IP アドレスに到達すると，MetalLB はクラスタ内の Pod にトラフィックをルーティングする．MetalLB は，Layer 2 及び Layer 3 のロードバランシングをサポートしており，ネットワークのトポロジに応じて適切なモードを選択することが可能である．Layer 2 モードでは，Address Resolution Protocol (ARP) プロトコルを使用し，Layer 3 モードでは Border Gateway Protocol (BGP) プロトコルを使用する．前者は，構成管理が容易である一方で，トラフィックの分散先として単一のワーカーノードが選出される．そのため，一つのワーカーノードが一括して全てのリクエストを受信することで，パフォーマンスの低下を誘発する懸念がある．また，後者を使用する場合，クラスタワイドな負荷分散を実現可能であり，全てのワーカーノード及び Pod に均等にトラフィックを分散することが可能である．さらに，送信元 IP アドレスの細やかな制御も可能であり，特定のニーズに対応することが可能である．しかし，Layer 3 モードでは BGP プロトコルを使用することから，クラスタの前段に別途 BGP ルータを準備する必要があり，インフラストラクチャ構成が複雑化する課題がある．

2.18　分散ストレージシステム

分散ストレージシステムは，データを複数のストレージノードに分散し，冗長性を持たせることで信頼性や可用性を高める技術である．分散ストレージはデータが複数のノードに分散して保存されるため，一つのノードが故障しても他のノードからデータにアクセスすることが可能である．そのため，ノードの故障や障害が発生してもデータへのアクセスが継続して可能となり，システム全体の可用性が向上する．また，スケーラビリティが高く，新しいノードを追加することでストレージ容量を増やすことが容易である．データが複数の場所に複製されることで，一部のノードやディスクが故障してもデータが失われるリスクは低減されるため，単一ストレージシステムと比較して信頼性が向上する．また，システム全体のパフォーマンスが向上し，大規模なデータ処理やアクセスパターンの変化にも柔軟に対応可能である．さらに，地理的に離れた場所にデータを分散して保存しておくことで，災害復旧や地域間のデータレプリケーションが可能である．

代表的な分散ストレージシステムを構築するソフトウェアとして Ceph が挙げられる．また，Kubernetes において Ceph を扱うためのオペレータとして Rook が存在する．以下に，Ceph 及び Rook について詳述する．

2.18.1　Ceph

Ceph はオープンソースの分散ストレージプラットフォームであり, 障害許容性及び規模拡張性を提供する. Ceph は, オブジェクトストレージ, ブロックストレージ, ファイルストレージの 3 つの主要なストレージモードを統合する. Ceph は, Reliable Autonomic Distributed Object Store (RADOS) と呼ばれるオブジェクトストレージクラスタをベースとしており, データはクラスタ内の複数のノードに分散される. RADOS はデータの冗長コピーを保持し, ノードの故障やディスクの障害に対応するための自己修復機能を備えている. Ceph は柔軟なスケーラビリティを持つため, 新規ノードの追加や削除が容易であり, 大規模なデータセンターやクラウド環境においても運用が可能である. Ceph は, NASA 米国国家航空宇宙局 や CERN 欧州原子核研究機構 等の膨大なデータを管理する組織においても導入実績がある.

2.18.2　Rook

Rook は Kubernetes 上で Ceph をはじめとする分散ストレージシステムを管理するためのオペレータである. Kubernetes オペレータとは, Kubernetes クラスタ内のアプリケーションやサービスをデプロイ・運用, 及び管理を自動化するためのカスタムコントローラである. Ceph は堅牢な分散ストレージシステムを構築可能な一方で, コンポーネントが非常に多く管理や運用が非常に複雑化する. そのため, Kubernetes 上で Ceph を構築することは容易でない. CNCF は, このような課題を受け, Ceph コンポーネントを Kubernetes クラスタに統合する Kubernetes オペレータとして Rook の提供を開始した. Rook は, Kubernetes のカスタムコントローラを使用してストレージクラスタのデプロイ, スケーリング, 及び運用を自動化する. また, Rook は Ceph 以外のストレージプラグインもサポートしており, Kubernetes クラスタ内で様々なストレージ要件に対応することが可能である.

第3章　CYPHONIC

3.1　概要

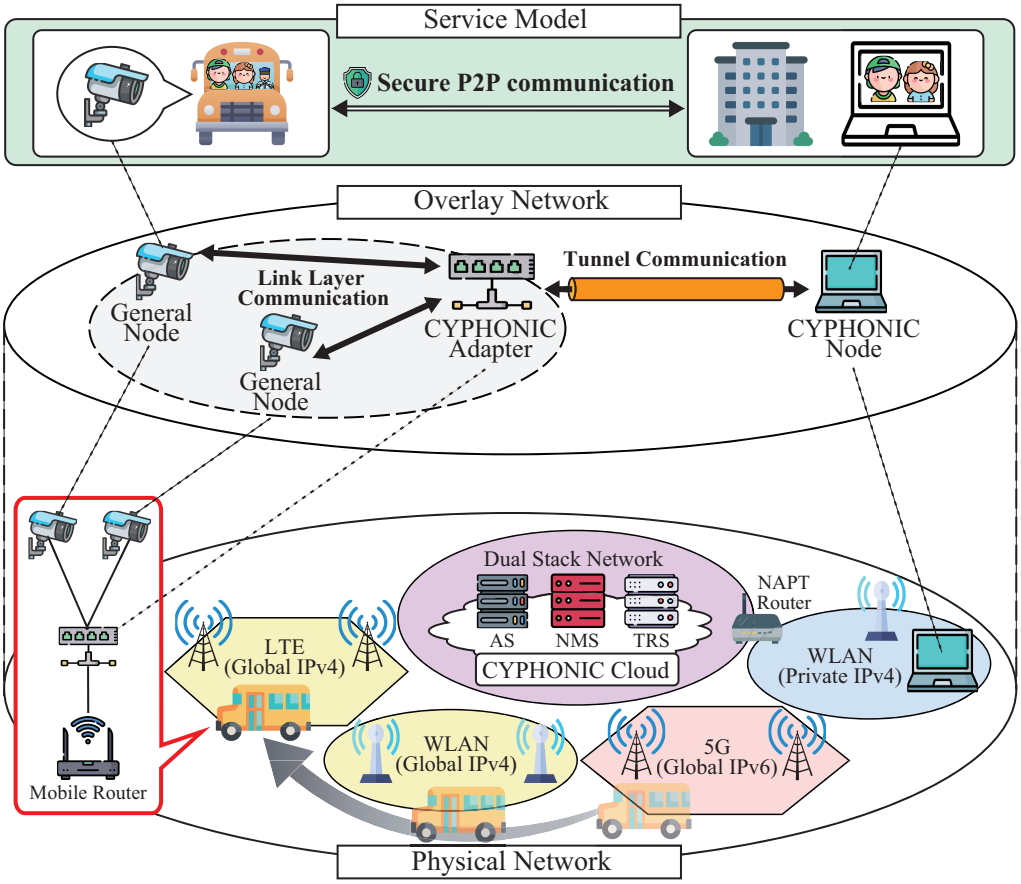
図 3.1 に CYPHONIC の全体概要を示す．CYber PHysical Overlay Network over Internet Communication (CYPHONIC) は, 通信接続性と移動透過性を兼ね備えたオーバーレイネットワーク技術であり, セキュアな端末間通信を実現する．CYPHONIC は, 端末に仮想 Internet Protocol (IP) アドレスを付与してオーバーレイネットワークを構築することで, 物理ネットワーク環境における接続性の課題をアプリケーションから隠蔽する．その結果, Network Address Port Translation (NAPT) や, 端末の IP バージョンに影響せず, 通信中にネットワーク切り替えが発生した場合においてもコネクションの継続が可能である．また, CYPHONIC 上で通信を行う端末は, クラウドサービスで認証を行い, 双方の暗号鍵を端末間で直接交換することにより送受信データの暗号化を行う．従って, 通信を行う端末のみが知り得る共通暗号鍵を用いることで, 第三者からの盗聴や改ざんを防止する．

CYPHONIC 上で通信を行う端末は, 識別子として一意な Fully Qualified Domain Name (FQDN) と, ネットワークの移動に依らず不変な仮想 IP アドレスを保持する．アプリケーションは FQDN を用いることで, 相手端末を識別し, 仮想 IP アドレスを用いてオーバーレイネットワークを構築する．そのため, 端末の移動に伴い, 実 IP アドレスが変化した場合も, 端末間の通信が継続可能であり, 移動透過性を実現する．また, 端末はクラウド上に存在する管理サービスに対して定期的に通信を継続することで, ネットワーク環境の変化を随時通知する．通信の際は, 端末が管理サービスに相手端末の FQDN を問い合わせると, 管理サービスが双方のネットワーク環境から適切な経路を選択する．端末は管理サービスから通知された通信経路を用いることで, NAPT が存在する場合や, 通信を行う双方の端末の IP バージョンが異なる場合でも Peer-to-Peer (P2P) トンネルを確立可能である．また, 管理サービスにより通信経路が決定すると, 通信を行う端末は送受信データを暗号化するための暗号鍵を生成する．その後, 生成した暗号鍵を相手端末と直接交換し, 構築されたトンネルを用いて暗号化データの送受信を行うことで, セキュアな端末間通信を実現する．端末は, 仮想 IP アドレスに基づいて通信を行うが, 実際の通信では実 IP アドレスを用いて全てのパケットをカプセル化し, User Datagram Protocol (UDP) トンネルによる通信を行う．

3.2　CYPHONIC の構成要素

CYPHONIC は, 通信を管理するクラウドサービス (以下, CYPHONIC クラウド と表記) と, 実際に CYPHONIC 上で通信を行う端末 (以下, CYPHONIC ノード と表記) で構成される．また, IoT 端末や専用サービスサーバ等の既存機器の改良が困難なノード (以下, 一般ノード と表記) をサポートする CYPHONIC アダプタが必要に応じて利用される．

CYPHONIC クラウドには, Authentication Service (AS), Node Management Service (NMS), Tunnel Relay Service (TRS) の 3 種類のサービスが存在する．AS, NMS, TRS は何れも IPv4 と IPv6 ネットワークからの通信リクエストを処理するため, デュアルスタックネットワークに配置される．AS は, 認証サービスであり, CYPHONIC ノードが正規のユーザであることを証明するための認証処理を行う．NMS は, CYPHONIC ノードの管理サービスであり, CYPHONIC ノードが存在するネットワーク情報を収集して, 適切な経路の選択及び指示を行う．TRS は, 通信中継サービスであり, CYPHONIC ノード間での



AS: Authentication Service NMS: Node Management Service TRS: Tunnel Relay Service
NAPT: Network Address Port Translation

図 3.1: Overview of CYPHONIC

直接通信が不可能な場合のみに, 端末間の送受信データを中継することでトンネル通信をサポートする. CYPHONIC ノードは CYPHONIC クラウドの各サービスと連携することで, 相手ノードとの間にオーバーレイネットワークを構築し, 直接通信を行なう. CYPHONIC アダプタは, TCP/IP における標準プロトコルと CYPHONIC で用いるシグナリングを連携することで, 一般ノードをサポートする, オーバーレイネットワーク用のゲートウェイ装置である. 一般ノードは隣接設置される CYPHONIC アダプタに接続することで CYPHONIC の通信サービスを利用可能となる. 以下に CYPHONIC クラウドの各サービスと, CYPHONIC ノード及び CYPHONIC アダプタの詳細を説明する.

- Authentication Service (AS)
  AS は, 全ての CYPHONIC ノード及び CYPHONIC アダプタ, 一般ノードを含む端末を認証するサービスである. CYPHONIC サービスを利用するノードは端末起動時に AS に対して認証要求を送信する. AS は, 認証要求に含まれる端末情報, または証明書情報に基づいて認証を行う. 認証が許可されると, AS は CYPHONIC ノードと NMS 間の通信を暗号化するための共通鍵を生成し, 認証応答パケットに含める. また, ノードを一意に識別するための FQDN, 及びオーバーレイネットワーク通信に用いる仮想 IP アドレスを生成し, 合わせて認証応答パケットに含める. なお, AS とノード間のシグナリングは Transmission Control Protocol (TCP) コネクションを確立した後, Transport Layer Security (TLS) version 1.3 に基づいて実行される.

- Node Management Service (NMS)
  NMS は, CYPHONIC ノード及び CYPHONIC アダプタ, 一般ノードを含む端末のネットワーク環境情報を把握・管理するサービスである. さらに, ノードが所望のノードと P2P トンネルを確立しようと試みる場合に, 両端末が存在するネットワーク情報から最適な通信経路を選択する. CYPHONIC ノード, 及び CYPHONIC アダプタは, AS での認証処理が完了すると, NMS に対して所属ネットワークの登録を行う. また, 一般ノードは CYPHONIC アダプタを介してシグナリングプロセスを実行する. NMS は, シグナリングを受信することにより, ノードから受信するパケットを用いて, NAPT の有無や IP バージョンの情報を収集する. この時, 双方の端末が何れも NAPT 配下に存在する場合, 後述する TRS に中継処理を依頼し, 各 ノードに TRS を中継する通信経路を指示する. なお, NMS とノード間の通信は UDP コネクションを確立して行い, 送受信される全てのシグナリングメッセージは, 認証処理の際に AS から付与された共通鍵で暗号化される. また, NMS とノード間は UDP メッセージを一定間隔で送受信することで Keep Alive を行い, コネクションを維持する. UDP ベースのシグナリングに用いることで, NAPT のマッピング情報を維持可能であり, 端末移動時にもネットワーク環境情報を再登録することが可能となる.

- Tunnel Relay Service (TRS)
  TRS は, 異なる IP バージョン間の通信や, 両ノードが NAPT 配下に存在する場合に, 通信を中継するサービスである. TRS は, 経路選択処理の際に, NMS から通信中継の指示を受けることで, ノード間で通信を行う際に, 送受信されるデータの中継処理を行う. ノード間で直接通信が不可能な場合は, 送受信データを暗号化するための暗号鍵も, TRS を中継して交換する必要がある. この時, ノード間で送受信データを暗号化するために用いる暗号鍵は第三者に渡してはならない. そのため, TRS による中継処理によってノード間で暗号鍵を交換する場合, 暗号鍵を TRS が復号不可能な別の暗号鍵を用いて暗号化する. これにより, ノード間で使用する暗号鍵情報を TRS から隠蔽したまま中継処理を行えるため, セキュアな P2P 通信を維持可能である. なお, TRS とノード間のシグナリングも UDP ベースで実行され, 送受信される全てのシグナリングメッセージは, 経路選択処理の際に NMS から付与される共通鍵で暗号化される.

- CYPHONIC ノード
  CYPHONIC ノードは, CYPHONIC のクライアントプログラムを搭載した端末である. CYPHONIC ノードは, DeviceID・パスワードによるベーシック認証, または証明書を用いて AS で認証処理を行う. 認証が成立すると, AS から CYPHONIC ノードの識別子である FQDN と, オーバーレイネットワーク通信に用いる仮想 IP アドレス, NMS との通信で利用する共通鍵を取得する. 認証処理後, CYPHONIC ノードは NMS と通信を行い, 所属ネットワークの登録処理を行う.

  オーバーレイネットワーク通信を開始する場合, CYPHONIC ノードは, 相手ノードを FQDN によって識別し, NMS に所望の FQDN への通信開始を依頼する. その後, NMS からの経路指示を受けることにより, 所望のノードに対するトンネル通信を確立して CYPHONIC ノード間で共通鍵を交換する. 最後に, 仮想 IP アドレスを用いて通信を行うとともに, 送受信データの暗号化を行うことでオーバーレイネットワーク上でセキュアな P2P 通信を実現する. また, CYPHONIC ノードに移動が発生し, 実 IP アドレスが変化した場合も, 新たなトンネル通信を確立するとともに, 仮想 IP アドレスを用いることで, 継続的な通信を実現する.

- CYPHONIC アダプタ
  CYPHONIC アダプタは一般ノードに対して通信機能を提供する CYPHONIC のゲートウェイ端末である. 一般ノードは, 既存のプログラムを変更したり, 新たなプログラムを追加したりすることが極めて困難である. そのため, CYPHONIC アダプタは CYPHONIC ノードと同様の機能を一般ノードに提供するため, オーバーレイネットワーク構築に関わる全ての処理機能を代行する.

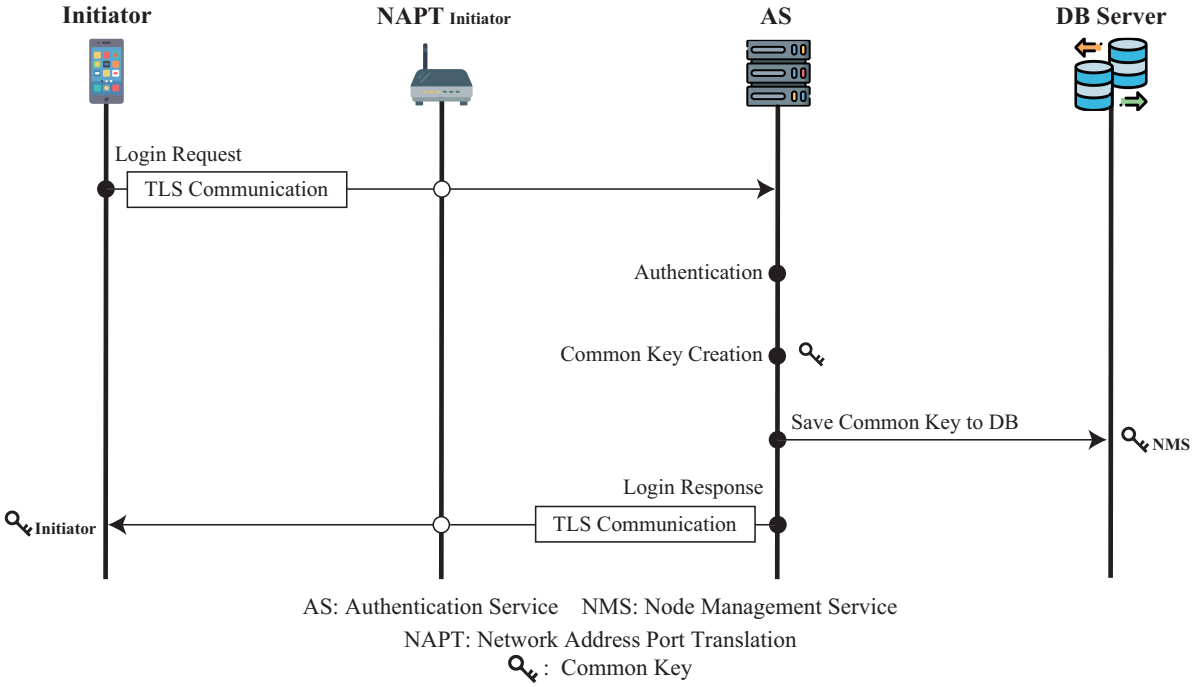


図 3.2: Authentication process

CYPHONIC アダプタは物理 Network Interface Card (NIC) を 2 枚搭載しており, 片方を一般ノードとブリッジ接続し, もう片方をインターネットに接続して利用する. CYPHONIC アダプタは, TCP/IP の標準プロトコルと, CYPHONIC で用いるシグナリングを連携することで一般ノードをサポートする.

一般ノードに割り当てる仮想 IP アドレスは, CYPHONIC 上では仮想 IP アドレスとして使用するが, 一般ノード上では実 IP アドレスとして機能する. また, 一般ノードがオーバーレイネットワーク通信を行う場合, CYPHONIC アダプタはリンクレイヤ通信によってイーサフレームを取得する. その後, 取得したペイロードデータを CYPHONIC で用いる所定のパケットフォーマットへ変換し, 相手ノードと交換した共通鍵によって暗号化する. 後方通信の場合は, 復号とデカプセル化のために逆の手順で処理を行い, 一般ノードに転送する. 一般ノードは CYPHONIC アダプタを介して通信を行うことで, CYPHONIC のセキュアな P2P 通信サービスを利用可能である.

3.3　各プロセスの通信シーケンス

CYPHONIC では, P2P 通信を実現するために, 認証処理, 位置情報登録処理, 経路選択処理, トンネル確立処理, データ通信処理の 5 つの通信プロセスを行う. また, 両 CYPHONIC ノードが何れも NAPT 配下に存在する場合, 直接通信を実現するための通信プロセスとして経路最適化処理を行う. 以下に, 各通信プロセスにおける処理を詳述する. なお, 以後の説明において, 通信を行うノードを区別するために, 通信を開始するノードを Initiator, 通信を待ち受けるノードを Responder として説明する.

3.3.1　認証処理

認証処理は, CYPHONIC ノードの認証を行うための処理であり, オーバーレイネットワークにおいてセキュアな通信を実現するために行われる. CYPHONIC ノードは初回起動時に, AS に対して認証を行

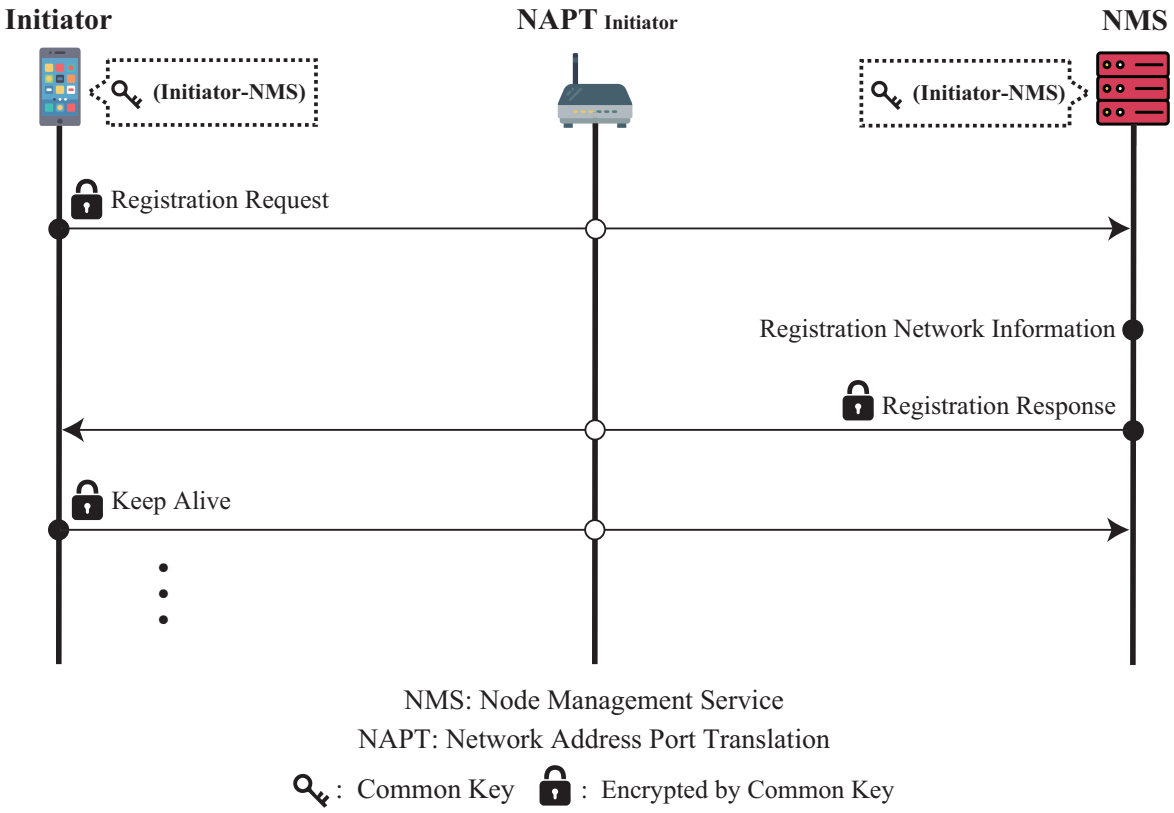


図 3.3: Registration process

う．その際, CYPHONIC ノードは事前に取得したルート証明書を用いて通信相手の AS が正規のサービスであるかを確認する．また, CYPHONIC ノードと AS 間の通信は TLS による暗号化通信を行う．なお, 一般ノードの場合は, CYPHONIC アダプタを用いることで, 認証処理を行い, 認証に必要となる情報は CYPHONIC アダプタが管理する．

図 3.2 に CYPHONIC ノードと AS による認証処理の通信シーケンスを示す．認証のはじめに, CYPHONIC ノードは AS に対し, Login Request メッセージを送信する．Login Request メッセージには, CYPHONIC ノードを利用する DeviceID やパスワード等の端末情報, もしくはクライアント証明書の情報が含まれる．AS が Login Request メッセージを受信し, 認証が完了すると, CYPHONIC ノードと NMS 間の通信に利用する共通鍵を生成し, データベースに保存する．また, CYPHONIC ノードの FQDN 及び仮想 IP アドレスを生成してデータベースに保存する．最後に, AS は Login Response メッセージを CYPHONIC ノードに応答すると, 認証処理が完了となる．Login Response メッセージには, CYPHONIC ノードを一意に識別するための FQDN, 通信に用いるための仮想 IP アドレス, 及び NMS との通信に使用する共通鍵が含まれる．

3.3.2　位置情報登録処理

位置情報登録処理は, CYPHONIC ノードが所属するネットワークの環境情報を NMS に登録する処理である．位置情報登録処理は, AS での認証処理後に行われる．また, CYPHONIC ノードが移動し, ネットワーク環境が変化した際にも, 再度, 位置情報登録処理が行われる．CYPHONIC ノードと NMS 間の通信は, AS が生成した共通鍵を用いて暗号化される．なお, CYPHONIC ノードが使用する共通鍵は, Login Response メッセージから取得し, NMS はデータベースから取得する．

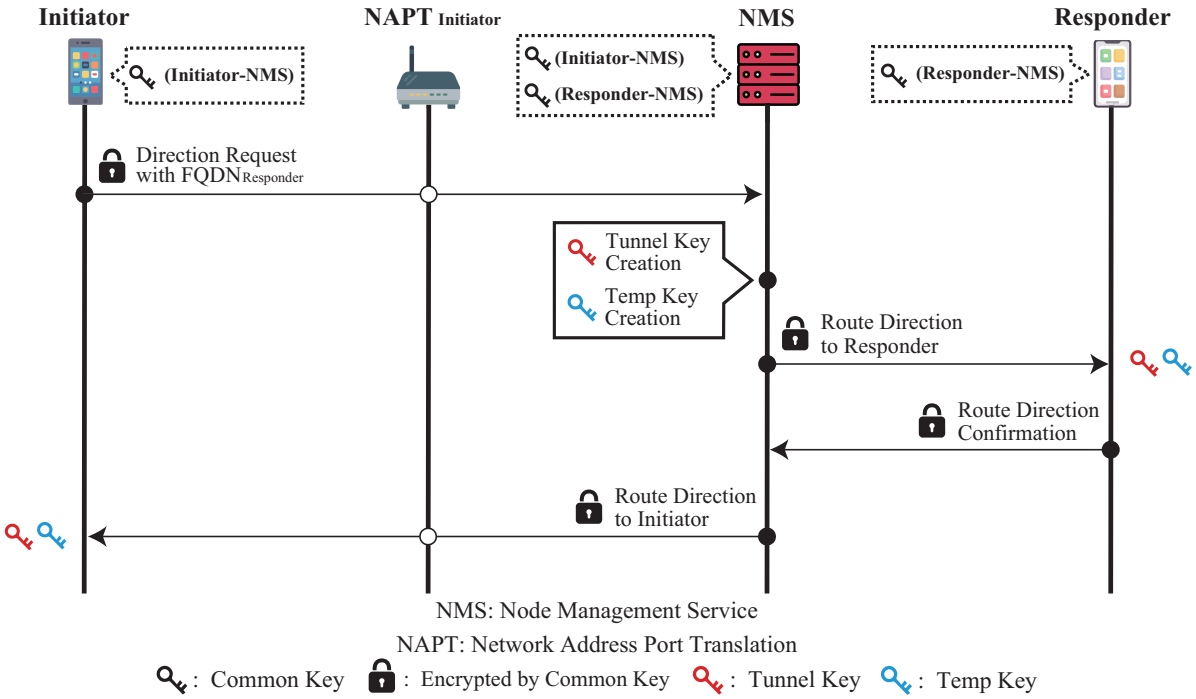


図 3.4: Route selection process

図 3.3 に位置情報登録処理の通信シーケンスを示す. CYPHONIC ノードは, ネットワークの位置情報を登録するために Registration Request メッセージを NMS に送信する. Registration Request メッセージには実 IP アドレスや, 使用しているポート番号, IP バージョン等のネットワーク情報が含まれる. NMS は Registration Request メッセージを受信した際, CYPHONIC ノードのネットワーク情報を登録及び更新する. NMS は CYPHONIC ノードのネットワーク情報の登録を完了すると, Registration Response メッセージを CYPHONIC ノードに送信する. Registration Response メッセージには, CYPHONIC ノードが NAPT 配下に存在しているか否かの情報が含まれる. また, CYPHONIC ノードは, コネクションを維持するために, NMS 対して Keep Alive メッセージを定期的に送信する. これにより, NAPT が存在する場合は, NAPT のマッピングテーブルを維持することが可能なため, NMS からの通信を受信可能となる.

3.3.3　経路選択処理

経路選択処理は CYPHONIC ノード間で通信を行う際に, 通信経路の選択を行う処理である. 経路選択処理は CYPHONIC ノード間において, 直接通信が可能な場合と不可能な場合の 2 種類に分けられる. 前者の場合は, NMS のみの経路指示によって直接通信を確立する. 後者の場合は, NMS からの経路指示を受けた後, トンネル確立に伴う処理を TRS を介して行う. また, 経路選択処理, 及び以後のトンネル確立処理, データ通信処理で用いる 3 種類の暗号鍵を以下に説明する.

- End Key
  CYPHONIC ノード間でオーバーレイネットワーク通信を行う際に, 送受信データを暗号化するための共通鍵である. End Key は Initiator が生成を行い, Responder に付与する. そのため, 通信を行う Initiator, 及び Responder のみが知り得る暗号鍵である.

- Tunnel Key

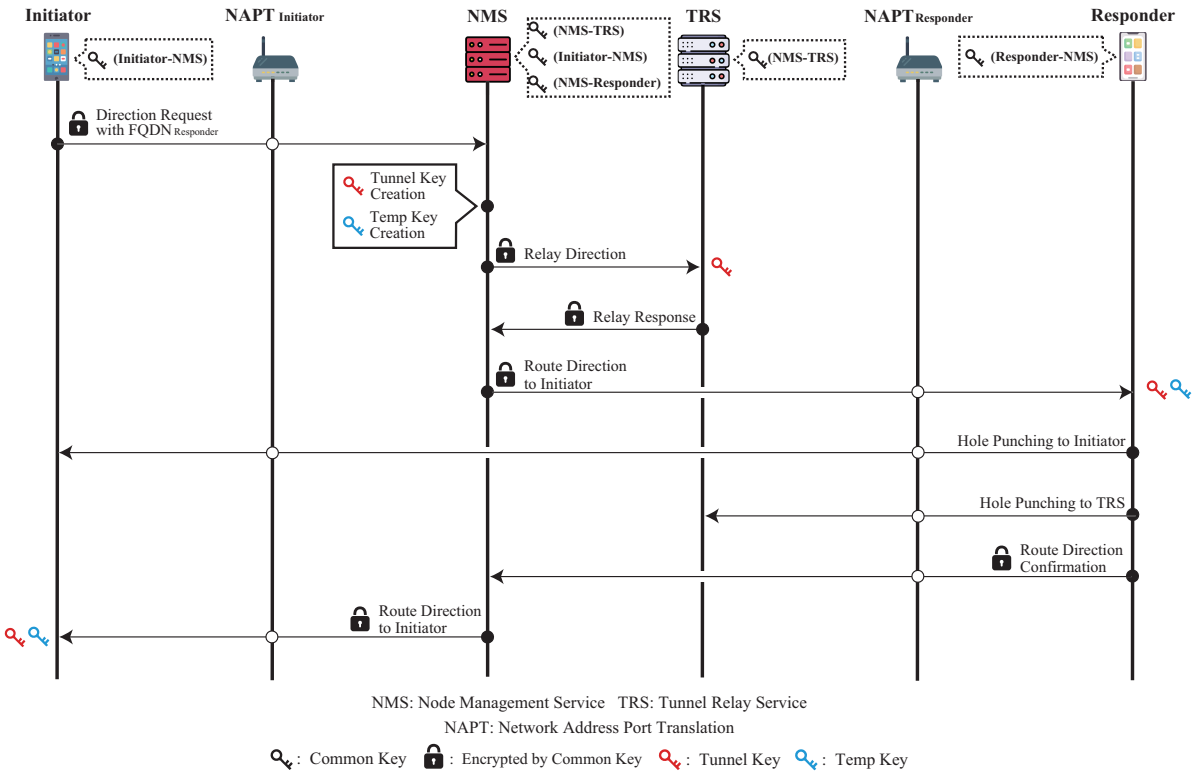


図 3.5: Route selection process (via TRS)

CYPHONIC ノード間で End Key を交換する際のシグナリングを暗号化するために用いる共通鍵である．Tunnel Key は NMS が生成を行い, 経路指示を行うシグナリングの際に Initiator, 及び Responder に付与される．また, 直接通信が不可能な状況において中継処理が必要な場合, TRS にも付与する．

- Temp Key
  CYPHONIC ノード間で End Key を交換する際のシグナリングが TRS を介して行われる場合に, End Key を暗号化するための共通鍵である．Temp Key は NMS が生成を行い, 経路指示を行うシグナリングの際に Tunnel Key と合わせて Initiator, 及び Responder に付与される．これにより, Temp Key で暗号化された End Key を TRS は復号不可能なため, 中継処理が必要な場合においても, 第三者の盗聴や改ざんを防止可能である．

図 3.4 に直接通信が可能な場合の通信シーケンス, 図 3.5 に直接通信が不可能な場合の通信シーケンスを示す．Initiator が Responder と通信を開始する際, Responder の FQDN を含んだ Direction Request メッセージを生成して NMS に送信する．NMS は, Initiator から Direction Request メッセージを受信すると, 以後のトンネル確立処理で用いる Tunnel Key と Temp Key を生成する．その後, 生成した 2 種類の暗号鍵を含めて Route Direction to Responder メッセージを Responder に送信する．また, Route Direction パケットには, 両ノードのネットワーク環境情報が含まれる．Responder は, Route Direction to Responder メッセージを受信すると, 自身の通信相手となる Initiator を認識し, 2 種類の暗号鍵を取得する．次に, Responder は NMS に通信可能であることを示すため, Route Direction Confirmation メッセージを送信する．NMS は, Responder から Route Direction Confirmation メッセージを受信すると, Route Direction to Initiator メッセージを Initiator に返送する．Route Direction to Initiator メッセージには, Responder の仮想 IP アドレスが含まれている．そのため, Initiator のアプリケーションは, FQDN

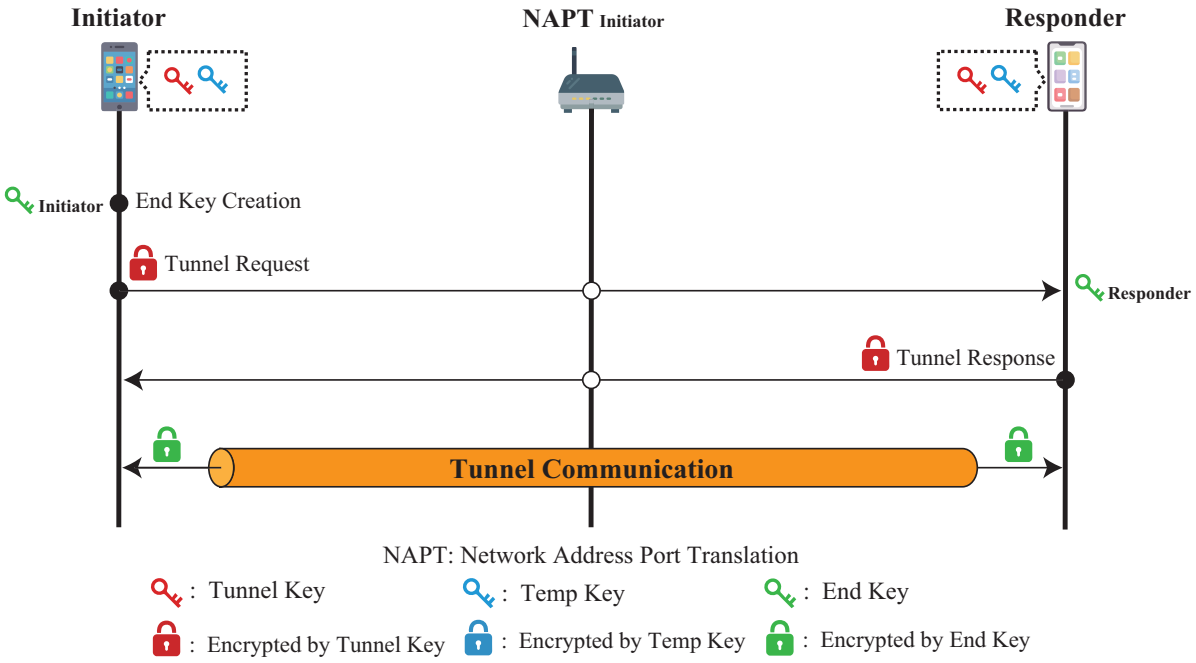


図 3.6: Tunnel establishment process

から Responder の仮想 IP アドレスを解決可能である.

一方, 両 CYPHONIC ノードが NAPT 配下に存在しており, かつその NAPT が特定のフィルタリングルールやマッピングの特性を持つ場合, 端末間での直接通信は不可能となる. また, 同様に IPv4-IPv6 間の通信も直接通信は極めて困難なことから, TRS による中継処理が必要となる. TRS による中継処理を行う場合, NMS は Initiator から Direction Request メッセージを受信した後, TRS へ Relay Request メッセージを送信する. この時, Relay Request メッセージに含める暗号鍵は, トンネル確立処理に用いる Tunnel Key のみであり, End Key を暗号化する Temp Key は含まれない. その後, TRS から Relay Response メッセージを受信すると, TRS への中継依頼処理が完了する. また, TRS を用いる場合, NMS は Responder に対して, TRS による中継処理を行うことを明示する Route Direction to Responder メッセージを送信する. Responder は Route Direction to Responder メッセージを受信すると, NMS に対して Route Direction Confirmation メッセージを返送する. この時, 同時に Initiator, 及び TRS に対して Responder 側から UDP Hole Punching を行う. 前者の UDP Hole Punching は, 後述する経路最適化処理を実行するために送信される. 後者の UDP Hole Punching は, TRS から中継されたメッセージを受信するために, Responder 側の NAPT のマッピングテーブルを維持する目的で送信される.

3.3.4　トンネル確立処理

トンネル確立処理はオーバーレイネットワークの構築を行う処理であり, Initiator と Responder 間の通信を暗号化するための End Key の交換を目的としている. Initiator と Responder は, 経路選択処理時に受け取った Route Direction メッセージの指示に従い, オーバーレイネットワークを構築する. オーバーレイネットワークは, 実際のネットワークでは UDP トンネルで構築される. また, トンネル間で送受信されるデータは End Key により暗号化されるため, セキュアな通信が実現可能である. トンネル確立処理も, CYPHONIC ノード間において, 直接通信が可能な場合と不可能な場合の 2 種類に分けられる.

図 3.6 に直接通信が可能な場合の通信シーケンス, 図 3.7 に直接通信が不可能な場合の通信シーケンスを示す. まず, トンネル確立の際, Initiator が End Key を生成する. 次に, End Key を Tunnel Request

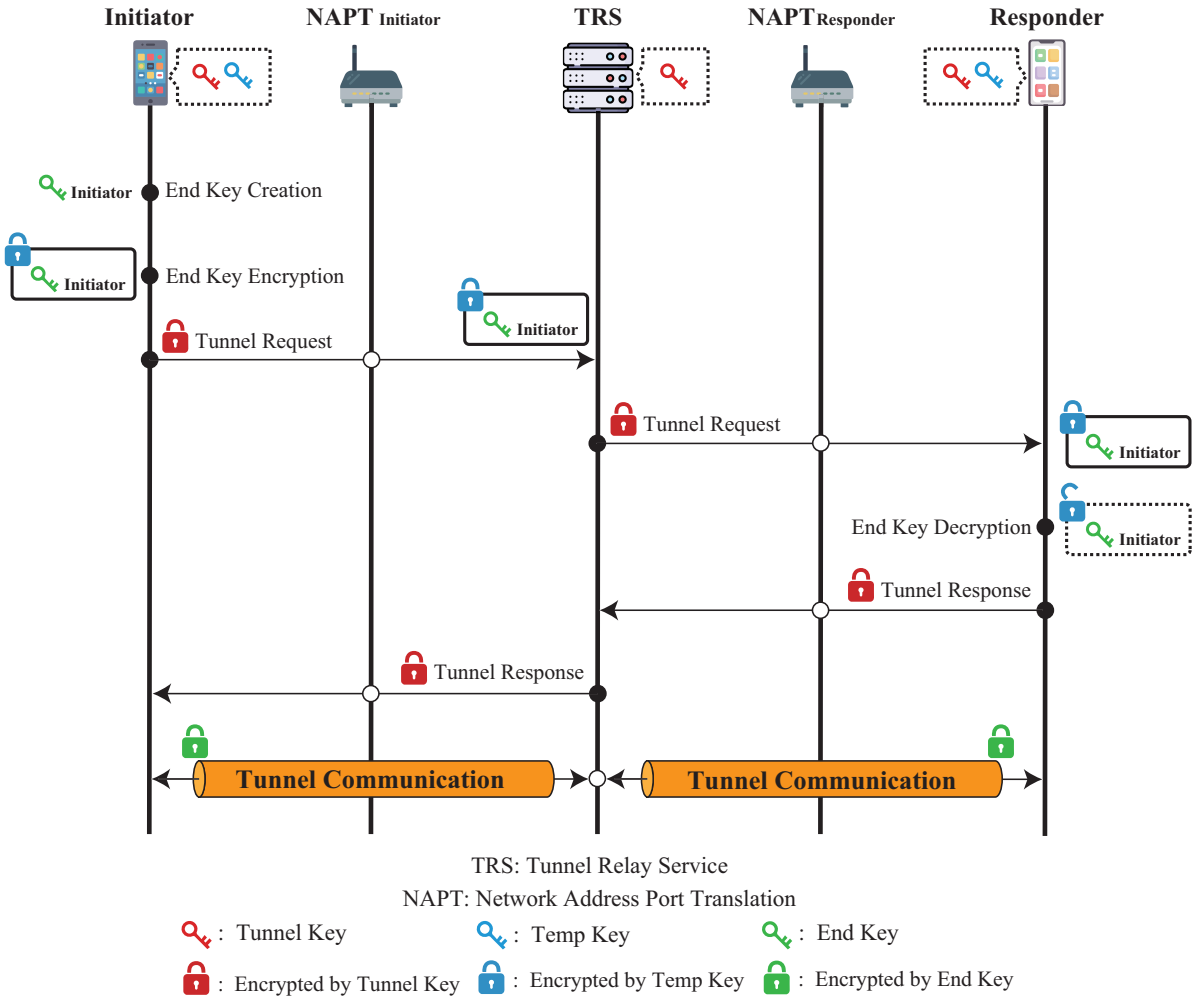


図 3.7: Tunnel establishment process (via TRS)

表 3.1: 両端末 : NAPT 1 台

Initiator \ Responder	FC	ARC	PRC	S
Full Cone (FC)	○	○	○	○
Address-Restricted Cone (ARC)	○	○	○	△
Port-Restricted Cone (PRC)	○	○	○	×
Symmetric (S)	○	○	×	×

メッセージに含め, Tunnel Request メッセージを Tunnel Key で暗号化して Responder へ送信する. Responder は受信した Tunnel Request メッセージを, Tunnel Key によって復号し, End Key を取り出す. Responder が Initiator の End Key を取得すると, Tunnel Key を用いて Tunnel Response メッセージを暗号化し, Initiator へ返送する. Initiator は受け取った Tunnel Response メッセージの復号及び検証を行い, End Key が正常に共有されたかを確認する. 以上より, 直接通信が可能な場合のトンネル確立処理が完了する.

一方, 直接通信が不可能な場合, End Key は TRS を経由して交換する必要がある. Initiator は, Temp Key で暗号化された End Key を Tunnel Request メッセージに含め, Tunnel Request メッセージを Tunnel Key で暗号化して TRS へ送信する. TRS は, Initiator から Tunnel Request メッセージを受信すると, Responder へ中継転送する. Responder は受信した Tunnel Request メッセージを, Tunnel Key によって復号し, メッセージ内に格納された End Key を, Temp Key で復号して取り出す. Responder が Initiator の End Key を取得すると, Tunnel Key を用いて Tunnel Response メッセージを暗号化し, TRS へ送信する. TRS は, Responder から Tunnel Response メッセージを受信すると, Initiator へ中継転送する. Initiator は受け取った Tunnel Response メッセージの復号及び検証を行い, End Key が正常に共有されたかを確認する. 以上より, 直接通信が不可能な場合のトンネル確立処理が完了する.

3.3.5　データ通信処理

データ通信処理はアプリケーションがオーバーレイネットワークを利用した通信を行う際の処理である. アプリケーションは仮想 IP アドレスを用いてオーバーレイネットワーク上で通信を行う. しかし, 実際には物理ネットワークを通じて相手ノードと通信を行う必要がある. そのため, 仮想 IP アドレスから作成した仮想 IP パケットを, 物理ネットワーク上で送信するために, 実 IP アドレスを用いてカプセル化する必要がある. その際, 生成した End Key を用いて, 仮想 IP パケットを暗号化する. 実 IP パケットは UDP トンネルを通じて相手ノードに送信される. 相手ノードが実 IP パケットを受け取ると, デカプセル化するとともに, 同じく End Key を用いて復号を行い, 仮想 IP パケットを取り出す. 取り出した仮想 IP パケットをアプリケーションに転送すると, データ通信処理が完了となる. 以上より, CYPHONIC ノードで動作するアプリケーションは仮想 IP アドレスを利用した通信を実現する.

3.3.6　経路最適化処理

経路最適化処理は, 両 CYPHONIC ノードが NAPT 配下に存在する場合に, 直接通信を実現するための処理である. Initiator, Responder がともに異なる NAPT 配下に存在しており, かつ NAPT が特定のフィルタリングルールやマッピングの特性を持つ場合, TRS による中継処理が必要となる. そのため, データ通信処理において常に TRS を経由する必要があるため, 経路が冗長になる問題がある. CYPHONIC では, この問題に対処するため, Initiator と Responder は TRS によるトンネル確立処理によって End

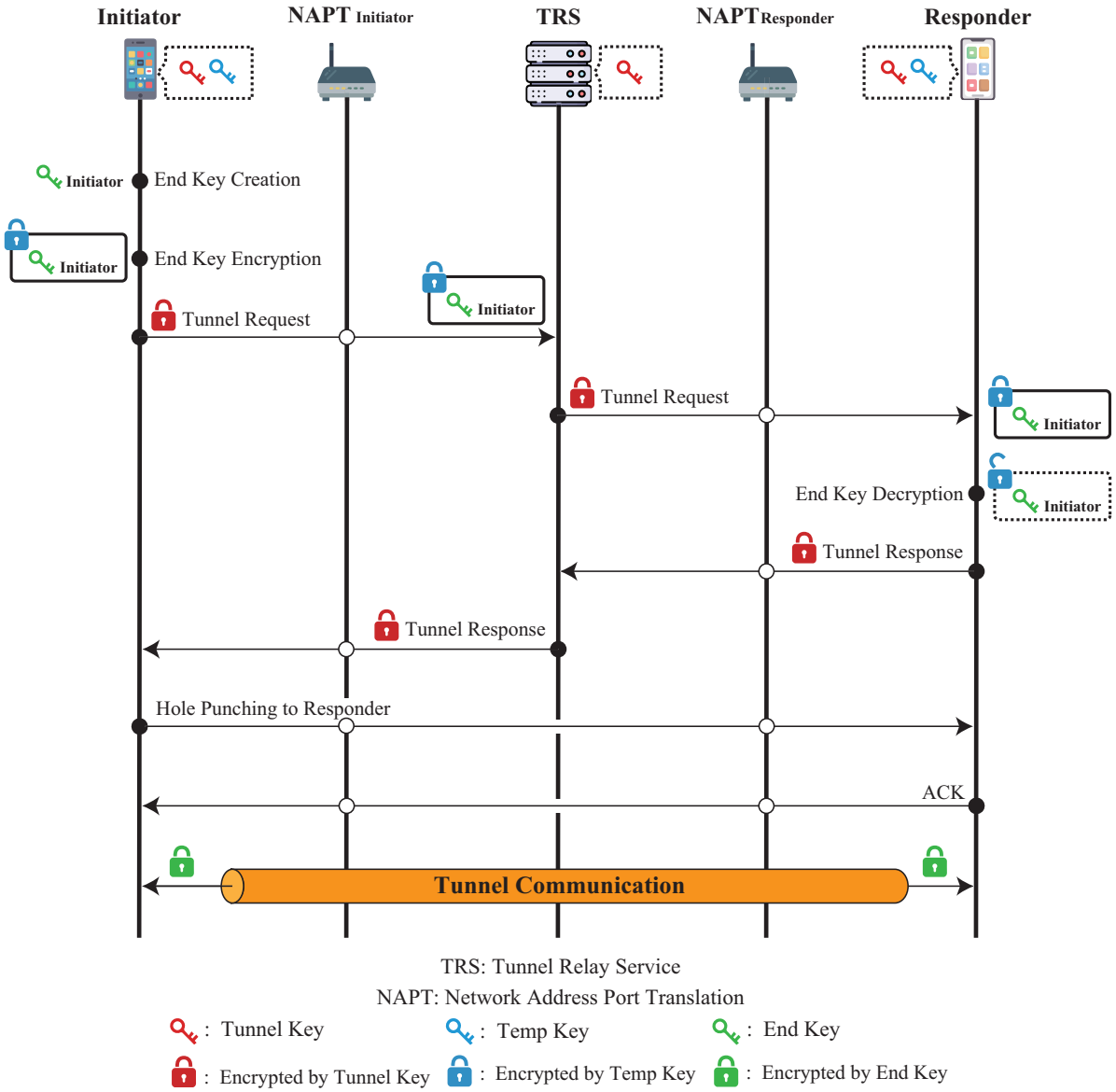


図 3.8: Route optimization process

Key を交換した後, 直接通信を実現するための経路最適化処理を試みる. なお, 両 CYPHONIC ノードがともに Symmetric 型 NAPT 配下に存在する場合, 経路最適化の実現が困難である. これは, Symmetric 型 NAPT は, 一般に用いられる Cone 型 NAPT と異なり, NAPT のマッピング情報を宛先毎に生成するためである. 故に, 双方の NAPT にそれぞれ相手ノードのマッピング情報を登録することが極めて困難である. また, 両端末が異なる 1 台の NAPT 配下に存在する場合と, 多段 NAPT と呼ばれる複数台の NAPT 配下に存在する場合, 両端末が同一の NAPT 配下に存在する場合がある.

両端末が異なる 1 台の NAPT 配下に存在する場合, 端末は互いに相手端末側の NAPT の変換情報を取得する必要がある. また, 少なくとも一方の NAPT が Address-Restricted Cone NAPT や Port-Restricted Cone NAPT の場合, NAPT へのマッピングを行う必要がある. さらに, 少なくとも一方の NAPT が Symmetric NAPT の場合, 宛先毎に変換情報が異なるため, 外側に存在する相手端末が変換情報を直接取得する必要がある.

少なくとも一方の端末が多段 NAPT 配下に存在する場合, 構成する NAPT の中で最も強い制約が多

段 NAPT 全体の制約として適用される．NAPT は, Full Cone NAPT, Address-Restricted Cone NAPT, Port-Restricted Cone NAPT, Symmetric NAPT の順に制約が強くなる．即ち, 多段 NAPT において外側の NAPT が Full Cone であった場合でも, 内側に Symmetric NAPT である場合, 多段 NAPT 全体の制約は Symmetric NAPT として適用される．

また, 両端末が同一 NAPT 配下に存在する場合, 端末は NAPT を経由することなく, プライベートネットワーク内で通信が完結することが望ましい．そのため, 同一 NAPT であるかを確認し, 必要に応じてプライベート IP アドレスでカプセル化を行い, LAN 内でオーバーレイネットワーク通信を行う．

NAPT の組み合わせによる経路最適化パターンを表 3.1 に示す．また, 両端末が同一 NAPT に存在する場合や, 多段 NAPT が存在するネットワーク環境における経路最適化パターンについて, 付録 B.1, B.2, B.3, B.4, B.5, B.6 に示す．なお, 市場に出回る NAPT の約 82 %は Cone 型 NAPT であることが先行研究により明らかにされている．従って, NAPT 配下に存在する端末同士の通信の多くが経路最適化可能である [77, 78].

図 3.8 に経路最適化処理を行う場合の通信シーケンスを示す．この時, 双方の NAPT の何れかは Cone 型 NAPT とする．前述の通り, TRS を経由してトンネル確立処理を行う際, Responder は Initiator に対して UDP Hole Punching を実行済みである．これは, Responder 側の NAPT に Initiator のマッピング情報を登録するために実行される．その後, データ通信処理の際に, Initiator は Responder に対して直接通信を試みる．この時, Responder 側の NAPT には Initiator のマッピング情報が登録されているため, Responder は UDP Hole Punching を受信可能である．Responder は Initiator から UDP Hole Punching を受信すると, Initiator に応答メッセージとして ACK を送信する．Initiator は Responder から ACK を受信すると経路最適化に成功したと判断し, 以後の送信パケットを直接 Responder に送信する．以上より, CYPHONIC ノード間で経路最適化処理を実行することで, 最終的に端末間での直接通信が実現する．

3.4　通信処理で用いられるメッセージフォーマット

CYPHONIC における P2P トンネル確立のためのシグナリングメッセージの交換処理, 及びデータ通信処理では, 独自のヘッダを付与した CYPHONIC メッセージによる通信を行う．CYPHONIC では, TLS による通信を行う AS を除く, NMS や TRS, CYPHONIC ノードとの通信の際は, CYPHONIC ノード間で共通鍵を共有することを前提としている．CYPHONIC の通信に用いられる CYPHONIC のメッセージは, Base Header, Message Body, Hash-based Message Authentication Code (HMAC) で構成される．Message Body は, 共通鍵で暗号化した後, HMAC を付与して相手ノードに送信する．HMAC は, メッセージの改ざんの検出に用いられるメッセージ認証符号の一つで, 暗号鍵とハッシュ関数によって生成される．以下に, CYPHONIC の通信で用いるメッセージフォーマット, 及びシグナリングメッセージについて詳述する．

- Base Header
  付録の図 A.1 で示される Base Header は, 全ての CYPHONIC メッセージに付与される．CYPHONIC メッセージを受信した AS, NMS, TRS, CYPHONIC ノード, CYPHONIC アダプタは Base Header の各フィールドをもとに Message Body を解読する．以下に, Base Header のフィールドの詳細を説明する．

  - Transaction ID (32 bit)
    シグナリングで用いられる識別子を格納する．一連の処理開始時に, Universally Unique Identifier (UUID) やハッシュ値等の一意な値を挿入する．
  - Version (8 bit)
    CYPHONIC のバージョンを示す．

- Flag (8 bit)
  処理結果の成否や, 送信者が位置情報登録の初回通信などの状態を示す.
- Type (8 bit)
  メッセージの種類を示す.
- Count (8 bit)
  メッセージの同時送信数を示す. 一つの Base Header に対し, 2 つ以上のメッセージを付与して送信する場合, 追加分のメッセージ数を挿入する.
- Sequence Number (32 bit)
  一連の処理によるシグナリング中に, メッセージカウンタとして用いる. Sequence Number は, メッセージの送信時に値を増加して用いる. 当フィールドと HMAC により, リプレイ攻撃への対策を行なっている.
- Message Length (16 bit)
  Base Header, Message Body, HMAC を含めたシグナリングメッセージの長さを格納する.
- Next Opt (8 bit)
  Base Header, Message Body 部の間に挿入して用いる拡張ヘッダを使用する場合に用いる. Next Opt は, 拡張ヘッダの種類を示す. IPv6 で用いられる拡張ヘッダの形式と同様のものであり, 複数個の拡張ヘッダを用いる場合は先頭にチェーンして用いる.
- Reserved (8 bit)
  パディング及び将来的に利用される, 予約フィールドである.
- ID (128 bit)
  シグナリングメッセージ及びカプセルメッセージの送受信時に, CYPHONIC ノードを特定する NodeID または, 通信経路を特定する PathID を挿入する. NodeID とは, シグナリングプロセスを実行する際に, 各ノードを一意に特定するための識別子である. なお, 本文脈における ”Node” は, CYPHONIC ノード, 及び一般ノードの ”Node” とは区別される. また, NMS, TRS もそれぞれ 一意の NodeID を保持しており, Relay Request, 及び Relay Response 時に使用される. CYPHONIC ノードは, Login Response を受信することで NodeID を取得する. NodeID は, 認証時にユーザが任意に設定できる DeviceID とは区別される.

  また, PathID とは, CYPHONIC ノード間でトンネルを確立する際に, 通信経路を一意に特定するための識別子である. 通信経路とは, CYPHONIC における Route Direction パケットに含まれる各フィールド要素の集合体のことであり, ネットワーク情報を示す. シグナリング, 及びデータ送受信時, 必要に際して, PathID をキーとして通信経路情報を参照する. PathID は両 CYPHONIC ノードの FQDN を元にして, Initiator ノードが生成し, Direction Request に含めて NMS に送信する. 双方の CYPHONIC ノードは PathID を参照することで, 相手 CYPHONIC ノードのネットワーク情報を参照して, 最適なオーバーレイネットワーク通信を行う.

- Hash-based Message Authentication Code (HMAC)
  付録の図 A.2 で示される HMAC は, シグナリングメッセージ及びカプセル化メッセージの末尾に付与され, メッセージの改ざん検出に用いられる.
- ACK
  付録の図 A.3 で示される ACK は, シグナリングメッセージを受信した際に, 情報を送り返す必要がない場合に送信する確認応答用のメッセージである. 必要に応じて, Base Header の Flag フィールドに処理結果の成否を付与する.

- Login Request
  付録の図 A.4 で示される Login Request は, CYPHONIC ノードが AS に認証処理を依頼するメッセージである. Login Request には, CYPHONIC によるオーバーレイネットワークを実現するために, アプリケーションの識別子と認証情報を格納する. 認証情報は, ベーシック認証を用いる場合, DeviceID・パスワードを格納し, 証明書認証の場合は, CYPHONIC におけるクライアント証明書を格納する. Login Request は, TLS によって通信内容が暗号化されるため, Message Body の暗号化, HMAC の付与は行わない.

- Login Response
  付録の図 A.5 で示される Login Response は, AS が認証を行った CYPHONIC ノードに対して認証結果を返すメッセージである. Login Response には, 認証結果, Login Request によって認証された CYPHONIC ノードを管理する NMS の IP アドレス, NMS との通信の際に用いる共通鍵, CYPHONIC ノードに割り当てる仮想 IP アドレス, 及び FQDN を格納する. Login Response は, TLS によって通信内容が暗号化されるため, Message Body の暗号化, HMAC の付与は行わない.

- Registration Request
  付録の図 A.6 で示される Registration Request は, CYPHONIC ノードが自身のネットワーク環境に関する情報を NMS に登録する際に送信するメッセージである. Registration Request には, NMS の通信を受信するために生成された NAPT に変換される前のポート番号, NMS からのシグナリングを受信する際に用いる通知方法, CYPHONIC ノードの実 IP アドレスを格納する. CYPHONIC は, 将来的にスマートフォン上で動作させることを想定している. スマートフォンでは, エネルギー消費の観点からバックグラウンドでアプリケーションを動作させ続けることが困難である. そのため, 通常の Keep Alive を用いた NMS からのメッセージの受信に代わる別の通知方法を用いる場合に, その通知方法を指定する必要がある. 通知方法を指定するフィールドは, 端末に適した通知方法を格納する際に用いられる.

- Registration Response
  付録の図 A.7 で示される Registration Response は, NMS が CYPHONIC ノードに対して Registration Request の結果を応答する際に送信するメッセージである. Registration Response には, NAPT 配下に存在するかを示すフラグを格納する.

- Keep Alive
  付録の図 A.8 で示される Keep Alive は, 各サービス及び相手ノードとの通信経路を維持する際に送信するメッセージである. Keep Alive を定期的に送信することで, UDP Hole Punching の役割を担い, NAPT のマッピングテーブルを維持する.

- Direction Request
  付録の図 A.9 で示される Direction Request は, CYPHONIC ノードが NMS に対して経路選択を依頼する際に送信するメッセージである. Direction Request には, アプリケーションの識別子と相手ノードの FQDN を格納する.

- Route Direction
  付録の図 A.10 で示される Route Direction は, CYPHONIC ノードがオーバーレイネットワークを構築する場合に必要となる通信経路を NMS が指示する際に送信するメッセージである. Route Direction には, 通信経路を一意に示す PathID, 相手ノードのネットワーク環境に関する情報と FQDN, 通信経路情報を格納する. また, Route Direction にはオーバーレイネットワークを構築する際に使用する Tunnel Key と Temp Key の 2 種類の共通鍵を格納する.

- Relay Request
  付録の図 A.11 で示される Relay Request は, NMS が TRS に通信の中継を依頼する際に送信するメッセージである. NMS は, 経路選択を要求された際に, 両 CYPHONIC ノードのネットワーク情報から通信経路を判断し, 必要に応じて TRS に対して通信の中継を依頼する. Relay Request には, 経路を一意に示す PathID と, Tunnel Request の復号, 及び Tunnel Response の暗号化に使用する Tunnel Key を格納する.

- Relay Response
  付録の図 A.12 で示される Relay Response は, Relay Request の受信に成功したことを通知するメッセージである. NMS は, TRS から Relay Response を受信すると, TRS を経由する通信経路を CYPHONIC ノードへ指示する.

- Hole Punching
  付録の図 A.13 で示される Hole Punching は, CYPHONIC ノードが TRS や相手ノードに対して, 通信の際に指定すべき宛先情報を通知するメッセージである. Hole Punching は, 自身の CYPHONIC ノードが NAPT 配下に存在する場合に使用する. Hole Punching の受信者は, 送信者に対して通信を行う際に使用する宛先情報として, メッセージの送信元 IP アドレスを記録する. メッセージの送信元 IP アドレスは, NAPT を経由する際に変換されるため, 変換後の IP アドレスに対して通信を行うことで, NAPT 越えを実現する.

- Tunnel Request
  付録の図 A.14 で示される Tunnel Request は, NMS の経路指示を受け取った CYPHONIC ノードが, 相手ノードまたは TRS に送信するメッセージである. Tunnel Request には, 通信経路を特定する PathID と, 後述する Capsule Message の暗号化に使用する End Key を格納する.

- Tunnel Response
  付録の図 A.15 で示される Tunnel Response は, Tunnel Request の受信に成功したことを通知するメッセージである. CYPHONIC ノードは, 相手ノードから Tunnel Response を受信することで, オーバーレイネットワークの構築を完了する.

- Capsule Message
  付録の図 A.16 で示される Capsule Message は, 構築されたオーバーレイネットワーク上で仮想 IP アドレスを用いて送受信されるパケットをカプセル化し, 物理ネットワーク上で送受信を行う際に用いられるメッセージである. Capsule Message では, Base Header の ID フィールドに PathID が格納される. また, Payload フィールドには, 仮想 IP アドレスを使用するパケットの IP ヘッダ, トランスポートヘッダ, データが格納される.

3.5　CYPHONIC ノードの概要

CYPHONIC ノードは, CYPHONIC Daemon と呼ばれる端末プログラムを導入することで CYPHONIC を用いた通信を実現している. そのため, 既存の端末は新たに CYPHONIC Daemon を動作させるプログラムを導入する必要がある. アプリケーションは CYPHONIC Daemon を介して通信を行うことで, 仮想 IP アドレスを用いた通信が可能となる. 以下に, CYPHONIC Daemon が提供する機能の概要と, CYPHONIC ノードのオーバーレイネットワークを使用した通信について詳述する.

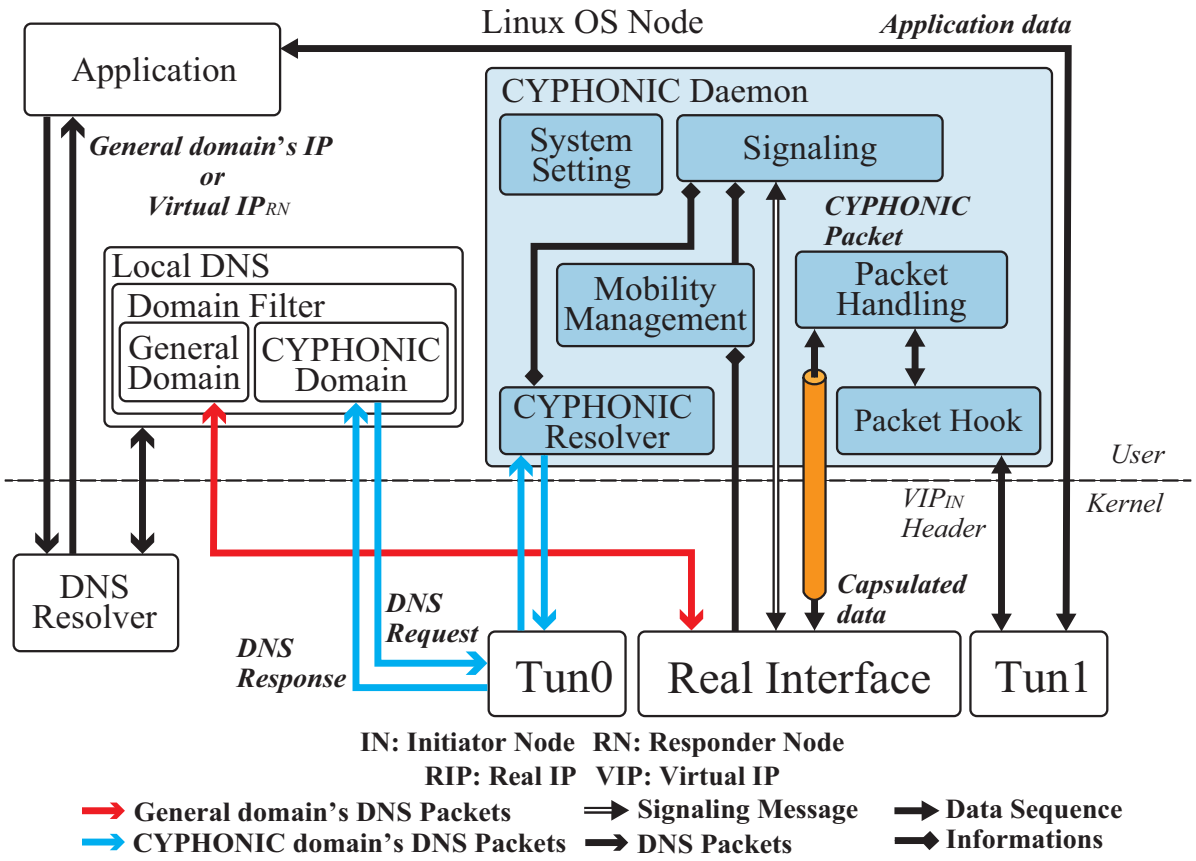


図 3.9: System model of CYPHONIC Node

3.5.1　CYPHONIC Daemon

図 3.9 に CYPHONIC ノードのシステムモデルを示す．CYPHONIC ノードは，オーバーレイネットワーク通信を行うために，CYPHONIC Daemon と仮想インターフェースが必要である．CYPHONIC Daemon とは，CYPHONIC ノードのユーザ空間にバックグラウンドプロセスとして常駐し，CYPHONIC 上での通信を実現するために必要な機能を提供する端末プログラムである．CYPHONIC Daemon は以下の機能を提供する．

- 各クラウドサービスとシグナリングを行う機能．
- FQDN から仮想 IP アドレスを解決するための名前解決機能．
- アプリケーションから仮想 IP パケットを取得する機能．
- 仮想 IP アドレスを用いたパケットに対する，CYPHONIC パケットへのカプセル化及びデカプセル化処理を行う機能．
- 物理ネットワークを介して相手ノードとカプセルメッセージを送受信する機能．

また，仮想インターフェースとは，アプリケーションが CYPHONIC Daemon と仮想 IP パケットの交換を行うための Tunnel (Tun) インターフェースであり，仮想 IP アドレスを割り当てる．仮想インターフェースは，アプリケーションから受信したデータを，仮想 IP アドレスを用いてカプセル化し，CYPHONIC Daemon へ受け渡す．そして，CYPHONIC Daemon は，通信プロセスに応じて前述した CYPHONIC

メッセージを生成し, CYPHONIC クラウドや相手ノードと通信を行う. CYPHONIC Daemon は, 5 つのモジュールで構成され, それぞれ以下の機能を提供する.

- Signaling Module
  CYPHONIC Daemon を起動した際に, 最初に呼び出されるモジュールであり, CYPHONIC クラウドとの一連のシグナリング処理を管理する. また, 仮想インターフェースを作成して, 仮想 IP アドレスを割り当て, 仮想 IP パケットを Tun インターフェースへルーティングされるように, ルーティングテーブルの設定を変更する.

- Packet Hook Module
  各クラウドサービスや CYPHONIC ノードと通信を行う際に, 仮想インターフェースを介して仮想 IP アドレス宛てのパケット取得処理を行う. その後, 取得した仮想 IP パケットを, 後述する Packet Handling Module へ受け渡す. また, Packet Handling Module からデカプセル化されたパケットを受け取った場合には, 仮想インターフェースを介してアプリケーションへ転送する.

- Packet Handling Module
  定義された CYPHONIC のパケット形式に従って, パケットの生成, 暗号化及び復号, カプセル化及びデカプセル化処理を行う. また, 送信する CYPHONIC パケットの末端に改ざん検出を行うための HMAC を付与する. 受信側は, 受信パケットの HMAC を計算することで, データの完全性を保証した通信を実現する.

- CYPHONIC Resolver Module
  FQDN のクエリに対する仮想 IP アドレスをアプリケーションに伝達するため, Domain Name System (DNS) パケットの処理を行う. アプリケーションから受信した FQDN をもとに, DNS Request を生成して, Signaling Module へ受け渡す. また, Signaling Module から相手ノードの仮想 IP アドレスを受信すると, DNS Response を生成してアプリケーションへ通知する.

- System Setting Module
  設定ファイルから CYPHONIC の通信に必要な情報の取得処理を行う. 設定ファイルには, 仮想インターフェースの設定や, クラウドサービスとの通信に用いるポート番号の情報が記述されており, System Setting Module によって他のモジュールに伝達される. また, AS によって端末の認証を行う際の認証方式を決定する.

- Mobility Management Module
  実 IP アドレスの変更を検出して, Signaling Module へ通知することで移動透過処理を行う. 移動透過処理は, NMS に対して再度, 端末のネットワーク位置情報を登録し直すことで, トンネル経路の再構築を行う.

3.5.2　オーバーレイネットワーク通信

図 3.10 に, オーバーレイネットワークを介した通信の DNS パケットフローを示す. アプリケーションは, 相手ノードを FQDN によって識別する. そのため, DNS リゾルバは, CYPHONIC で用いる FQDN の DNS クエリを検出する. Local DNS は CYPHONIC で用いる FQDN の DNS クエリのみをフィルタリングし, CYPHONIC Resolver Module へ受け渡す. CYPHONIC Resolver Module は, DNS クエリを受信すると経路選択処理を開始するため, Signaling Module へ DNS Request を生成して転送する. Signaling Module は, NMS に対して相手ノードのネットワーク情報を取得するため, Direction Request メッセージを生成して送信する. その後, Signaling Module を通じて, NMS から Route Direction to Initiator メッ

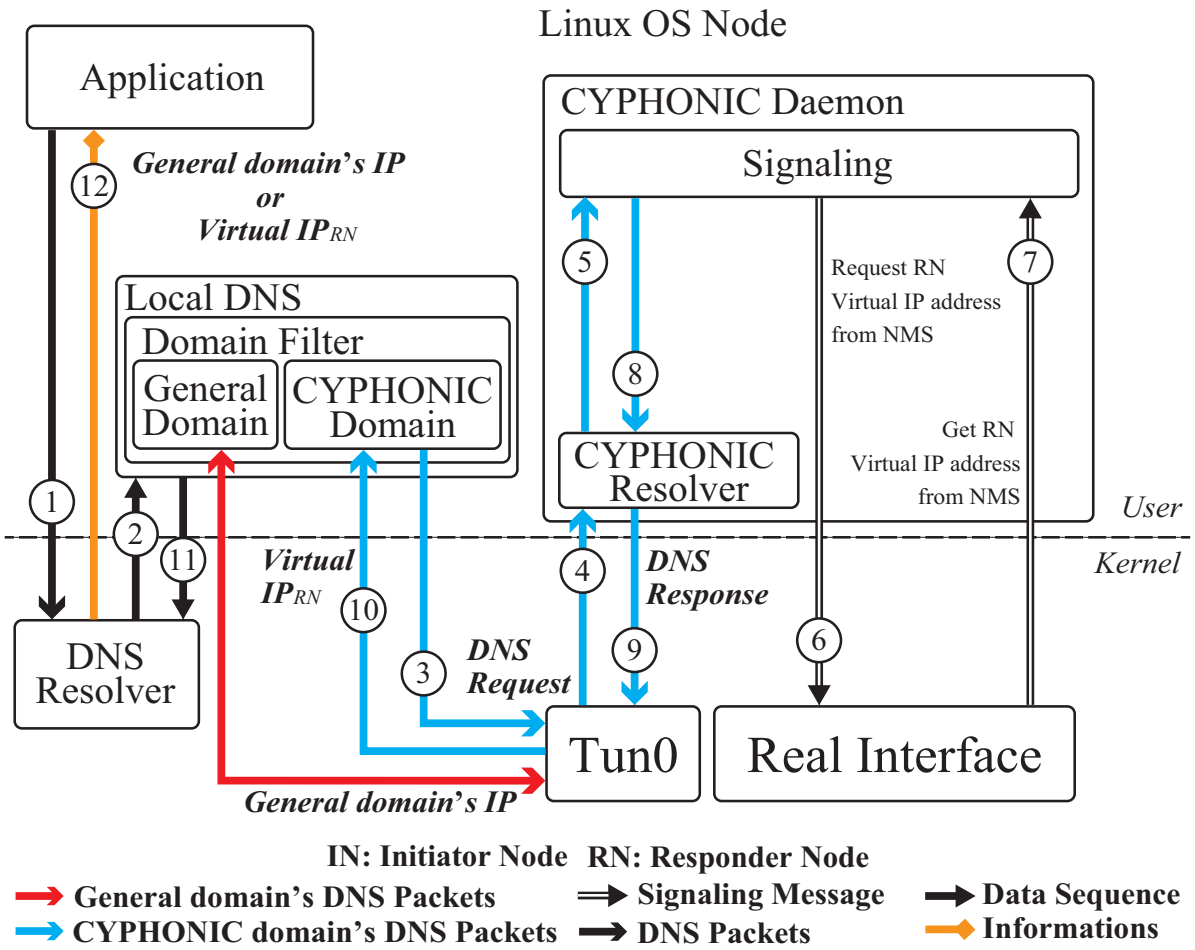


図 3.10: DNS packet flow

セージによって相手ノードの仮想 IP アドレスを取得すると, CYPHONIC Resolver Module へ受け渡す. 最後に, CYPHONIC Resolver Module は, 相手ノードの仮想 IP アドレスを含む DNS Response を生成してアプリケーションへ転送する.

図 3.11 に, オーバーレイネットワークを介した通信の Protocol Data Unit (PDU) カプセル化フローを示す. アプリケーションのデフォルトルートは, 仮想インターフェイスに設定される. そのため, アプリケーションは, 仮想 IP アドレスを使用して通信を行う. 前述の DNS に関する処理が完了した後, アプリケーションは取得した相手ノードの仮想 IP アドレス宛てにソケット通信を開始する. アプリケーションが, 仮想インターフェースへアプリケーションデータを送信すると, Packet Hook Module によって取得し, Packet Handling Module に受け渡して処理を行う. CYPHONIC では, 送受信する全てのパケットを UDP パケットでカプセル化する. そのため, 取得した仮想 IP パケットに対して UDP ヘッダを付与し, CYPHONIC ノードの実 IP アドレスでカプセル化を行う. その後, 物理インターフェースを介して, 構築した UDP トンネルによって相手ノードへ送信する. また, 相手ノードへパケットが着信し, デカプセル化されると, 実 IP ヘッダと UDP ヘッダが取り除かれ, 仮想 IP パケットを取得する. 最後に, 仮想 IP パケットからアプリケーションデータを取り出し, アプリケーションへ転送する. 以上より, アプリケーションは仮想 IP アドレスを用いて CYPHONIC を用いた通信を行うことが可能となる.

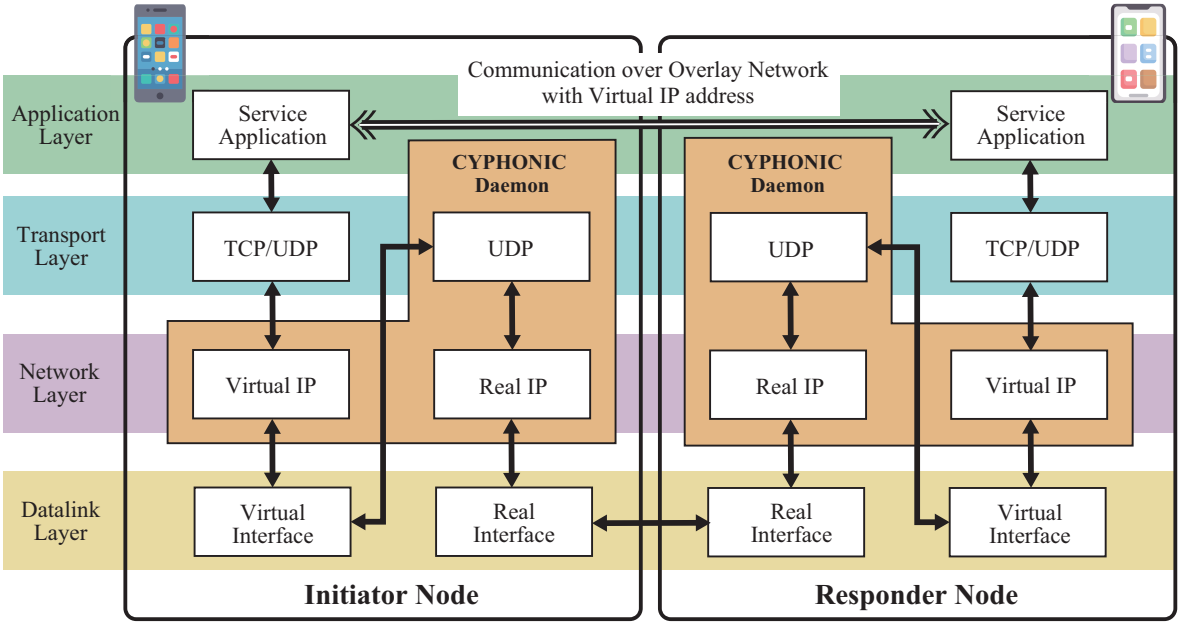


図 3.11: PDU encapsulation flow

3.6　CYPHONIC アダプタの概要

CYPHONIC アダプタは, Adapter Daemon と呼ばれるプログラムを搭載した端末であり, CYPHONIC Daemon を搭載することが極めて困難な一般ノードにオーバーレイネットワーク通信機能を提供する. CYPHONIC アダプタは, オーバーレイネットワーク構築処理のためのシグナリングや, パケット処理機能において, CYPHONIC Daemon と同様の機能を搭載する. また, TCP/IP における標準的なプロトコルを用いて一般ノードに仮想 IP アドレスを割り当てる. 以下に, Adapter Daemon が提供する機能の概要と, 一般ノードにおける CYPHONIC アダプタを介したオーバーレイネットワーク通信について詳述する.

3.6.1　Adapter Daemon

図 3.12 に CYPHONIC アダプタのシステムモデルを示す. CYPHONIC アダプタは物理 NIC を 2 枚搭載しており, 片方を一般ノードとブリッジ接続, もう片方をインターネットに接続して利用する. Adapter Daemon は, CYPHONIC アダプタとして構成された端末のユーザ空間にバックグラウンドプロセスとして常駐し, CYPHONIC 上での通信を実現するために必要な機能や, 一般ノードをサポートするための機能を提供する. Adapter Daemon は CYPHONIC Daemon を拡張して設計されている. そのため, CYPHONIC Daemon の既存機能を用いた上で, アダプタとして動作するための諸機能が追加されている. 以下に, Adapter Daemon を構成する機能について詳述する.

- CYPHONIC Daemon が提供する機能
  既存の CYPHONIC Daemon には, 以下のモジュールが存在し, Adapter Daemon において, CYPHONIC 上の通信に必要な処理を実行する.
  - Signaling Module
    端末が起動し, CYPHONIC Daemon が動作した際に, 最初に呼び出されるモジュールである.

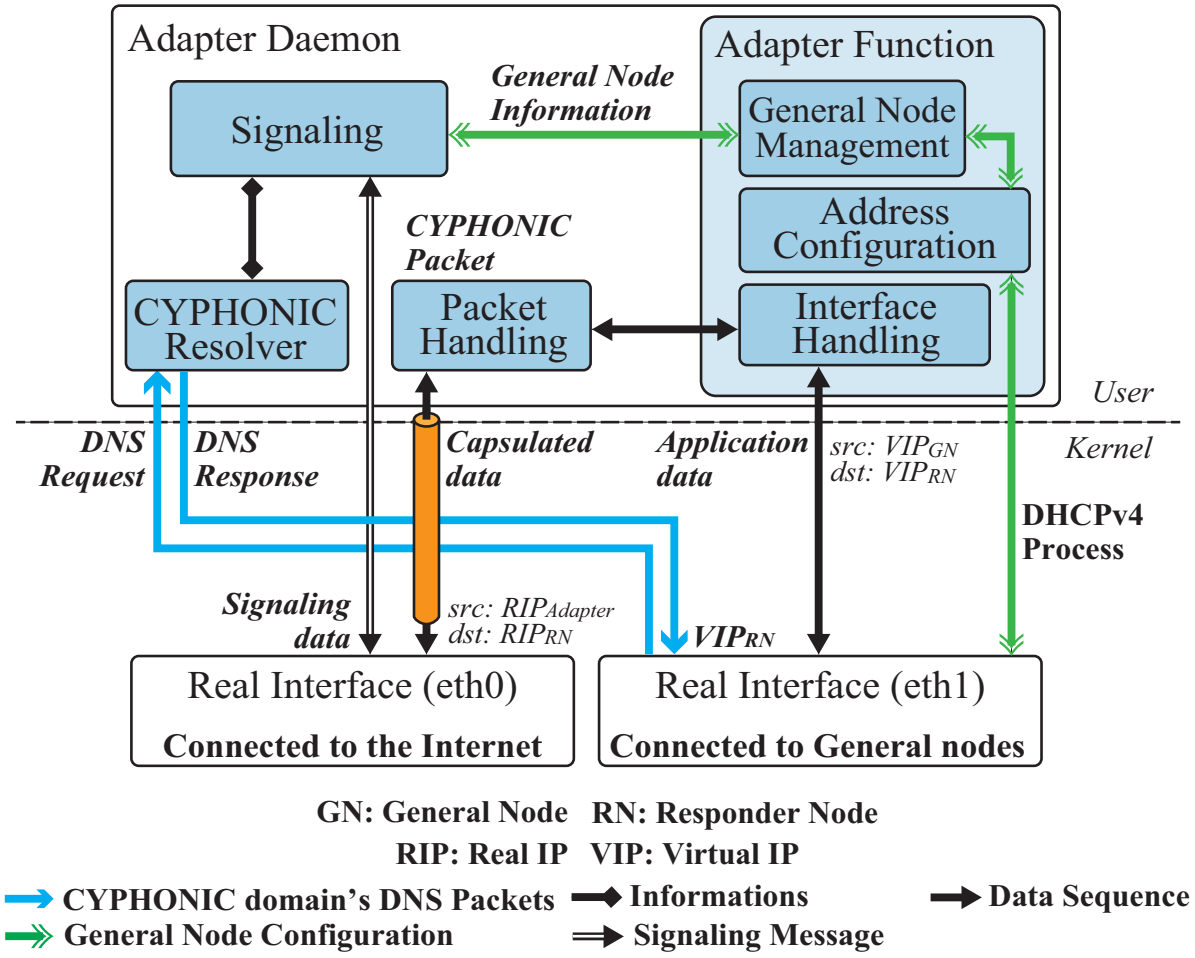


図 3.12: System model of CYPHONIC Adapter

前述した CYPHONIC のクラウドサービスである AS, NMS, TRS との一連のシグナリング処理を管理する.

- Packet Handling Module
  定義された CYPHONIC のパケット形式に従って, パケットの生成, 暗号化処理及び復号処理, カプセル化処理及びデカプセル化処理を行う. また, 送信する CYPHONIC パケットの末尾に改ざん検出を行うための HMAC を付与する. 受信側は, 受信パケットの HMAC を計算することで, データの完全性を保証した通信を実現する.
- CYPHONIC Resolver Module
  相手ノードのネットワーク情報をアプリケーションに伝達するために用いられる. 既存のアプリケーションが DNS を用いて通信を開始するのと同様に, CYPHONIC を用いたオーバーレイネットワーク上での通信においても FQDN を用いる. CYPHONIC Resolver Module は, 取得した相手ノードの仮想 IP アドレスから DNS パケットを生成する.

- Adapter Function の各モジュールが提供する機能
  Adapter Daemon において, Adapter Function は, 一般ノードをオーバーレイネットワークへ参加させるための諸機能を提供する.
  - General Node Management Module

一般ノードの情報を管理するモジュールである．General Node Management Module は, 接続される一般ノードの仮想 IP アドレス, 及び FQDN と, 一般ノードを認識するための Media Access Control (MAC) アドレスを保持する．また, 一般ノードの通信プロセス毎に暗号鍵を生成し, 相手ノードと直接交換することで, アダプタ配下に存在する一般ノードを識別したセキュアな通信を行うことが可能である．

– Address Configuration Module
General Node Management Module から取得した情報に基づいて, 一般ノードへ仮想 IP アドレスを付与するモジュールである．CYPHONIC アダプタが付与するアドレスは, 一般ノードでは実 IP アドレスとして認識されるが, CYPHONIC 上では一意な仮想 IP アドレスとして機能する．

一般ノードが IPv4 で動作する場合は, Dynamic Host Configuration Protocol for IPv4 (DHCPv4) プロセスを用いて, 仮想 IPv4 アドレスを付与する．また, ゲートウェイアドレス及び DNS サーバアドレスとして CYPHONIC アダプタの仮想 IP アドレスを付与する．ここで, CYPHONIC アダプタが使用する仮想 IPv4 アドレスは, 一般ノードとの通信のみに使用されるオーバーレイネットワーク用のゲートウェイアドレスであり, CYPHONIC におけるデータ通信処理の際には使用されない．

一般ノードが IPv6 で動作する場合は, Neighbor Discovery Protocol (NDP) とステートフル Dynamic Host Configuration Protocol for IPv6 (DHCPv6) を組み合わせて, 仮想 IPv6 アドレスを付与する．また, NDP における Router Advertisement (RA) メッセージを一定間隔で送信することにより, 一般ノードに対して CYPHONIC アダプタを近隣ルータとして確立する．IPv4 と同様に, CYPHONIC アダプタが使用する仮想 IPv6 アドレスは, 一般ノードとの通信のみに使用されるオーバーレイネットワーク用のゲートウェイアドレスであり, CYPHONIC におけるデータ通信処理の際には使用されない．

– Interface Handling Module
各クラウドサービスや相手ノードと通信をする際に, 一般ノードが接続される物理インターフェースを介して仮想 IP アドレス宛てのパケットを取得する．取得したパケットは, Packet Handling Module へ受け渡すことにより処理する．また, Packet Handling Module からデカプセル化されたパケットを受け取った場合には, 物理インターフェースを介して一般ノードへ送信する．

3.6.2　CYPHONIC アダプタを介したオーバーレイネットワーク通信

図 3.13 に, 一般ノードにおける, オーバーレイネットワークを介した通信の DNS パケットフローを示す．一般ノードは相手ノードを FQDN によって識別する．そのため, 一般ノードは相手ノードの FQDN に対する DNS Request を生成し, 物理インターフェースを通して CYPHONIC アダプタへ送信する．CYPHONIC Resolver Module は, DNS パケットを受信すると経路選択処理を開始するため, 受信した DNS クエリをもとに DNS Request を生成して Signaling Module へ転送する．Signaling Module は, NMS に対して相手ノードのネットワーク情報を取得するため, Direction Request メッセージを生成して送信する．その後, Signaling Module を通じて, NMS から Route Direction to Initiator メッセージによって相手ノードの仮想 IP アドレスを取得すると, CYPHONIC Resolver Module へ受け渡す．CYPHONIC Resolver Module は, 相手ノードの仮想 IP アドレスを含む DNS Response を生成し, 物理インターフェースを介して一般ノードへ送信する．

図 3.14 に, オーバーレイネットワークを介して一般ノードと CYPHONIC ノードが通信を行う際の PDU カプセル化フローを示す．CYPHONIC アダプタは, DNS Response を返送する際に相手ノードの

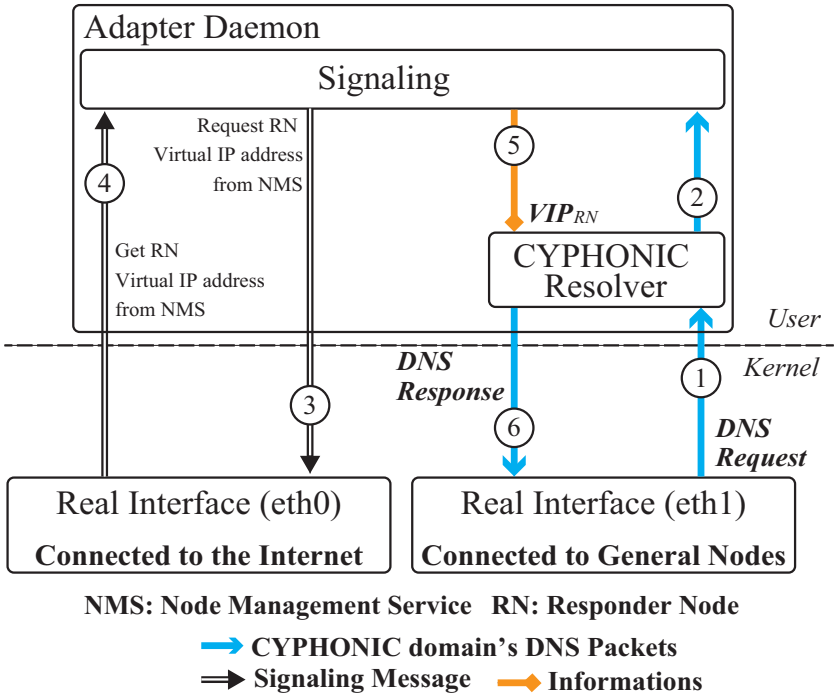


図 3.13: DNS packet flow of General Node using CYPHONIC Adapter

仮想 IP アドレスを一般ノードに通知すると, 一般ノードから送信される Address Resolution Protocol (ARP) Request を受信する. CYPHONIC アダプタは, ARP Request に対して 相手ノードの仮想 IP アドレスに CYPHONIC アダプタ自身の MAC アドレスをマッピングする Proxy ARP を実行する. その結果, 以後のデータ通信通信の際, リンクレイヤ通信によって相手ノードに送信する仮想 IP パケットを一般ノードから取得可能である. 前述の DNS に関する処理が完了した後, 一般ノードは取得した相手ノードの仮想 IP アドレス宛てに通信を開始する. 一般ノードが仮想 IP アドレス宛てに通信を開始すると, Interface Handling Module を通じてパケットを取得する. その後, Packet Handling Module に受け渡して仮想 IP パケットを CYPHONIC パケットへカプセル化し, 管理している一般ノードの End Key を用いて暗号化する. そして, CYPHONIC アダプタの実 IP アドレスを用いて, 実 IP パケットへカプセル化し, 構築した UDP トンネルによって相手ノードへ送信する. また, 相手ノードに着信したパケットがデカプセル化されると, 一般ノードの仮想 IP アドレスが取り出されるため, アダプタ配下の一般ノードを一意に識別した通信を行うことが可能である. 以上より, 一般ノードは CYPHONIC アダプタに接続することで, CYPHONIC による通信サービスを利用することが可能となる.

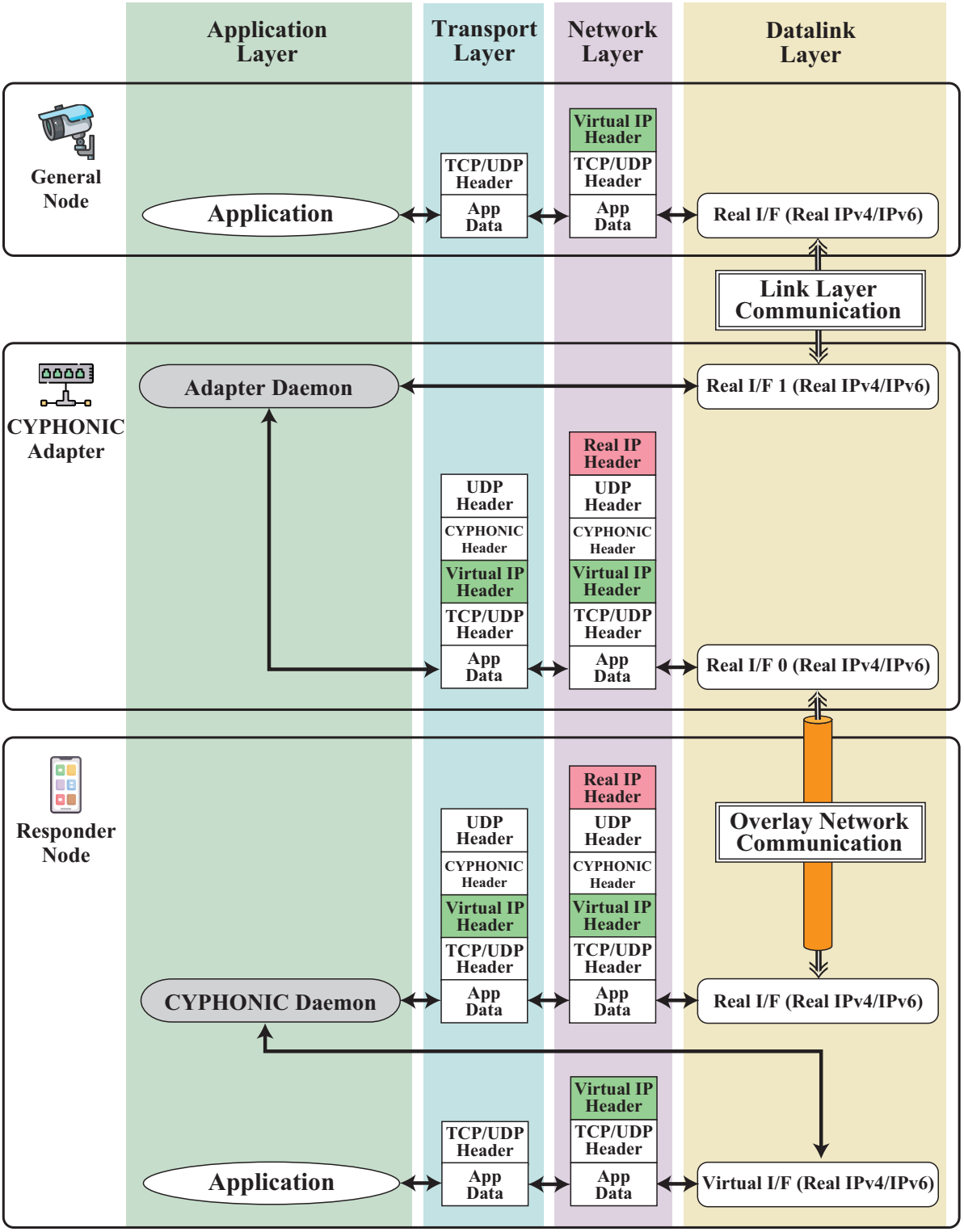


図 3.14: PDU encapsulation flow of General Node using CYPHONIC Adapter

第4章　提案システム

4.1　概要

本章では, CYPHONIC におけるいくつかの課題に対応するべく, エンドノード, 及びクラウドサービスのそれぞれに着目し, CYPHONIC サービスの拡張性を実現する設計手法について提案を行う. エンドノードでは, 複数の一般ノードをサポートするべく, CYPHONIC アダプタの拡張設計を提案する. また, クラウドサービスは, 複数のインスタンスを用いてサービスを多重化し, 効率的かつ高可用性を実現するためのインフラストラクチャ設計を提案する. 以下に, それぞれの提案概要について詳述する.

4.1.1　CYPHONIC アダプタ

既存の CYPHONIC アダプタは, 一般ノードと 1 対 1 対応を想定して設計されており, 一つの CYPHONIC アダプタは, 1 台の一般ノードのみにしか対応していない. 一方で, 近年では複数の IoT 端末を連携したソリューションやサービスも多数登場していることから, CYPHONIC アダプタにおいても複数の一般ノードを同時にサポートすることが求められる. また, CYPHONIC アダプタのプログラムである Adapter Daemon は, オーバーレイネットワーク通信に伴い, 一般ノードから取得したパケットをシングルスレッド方式によって処理する. そのため, ビデオストリーミングをはじめ, トラフィック量が比較的多くなるサービスでは, 受信パケットがスタックすることによる処理遅延や CYPHONIC アダプタの機能全体に負荷が掛かる可能性がある. 結果として, 一般ノードのネットワーク品質が低下する懸念が生ずる.

そこで, 本提案では, CYPHONIC アダプタの処理機能をマルチスレッド化するとともに, 複数の一般ノードを同時にサポート可能にするため, Adapter Daemon のマルチノード対応設計及びマルチスレッド処理方式を提案する. 提案方式では, 複数の一般ノードを管理するため, Adapter Daemon 内部にキャッシュストアを追加する. キャッシュストアは, 比較的読み書きが軽量に動作するインメモリで導入する. また, 各モジュールは内部キャッシュを Adapter Daemon のデータベースとして, 相互に情報を共有することで, 同時実行的に処理機能を動作させることが可能となる. CYPHONIC アダプタの機能の一つである, Packet Handling Module は, 複数の一般ノードから受信したパケットを高速に処理する必要がある. オーバーレイネットワーク通信に伴う, カプセル化や暗号化処理は Adapter Deamon 内の他のモジュールと比較して, 特に負荷が高いタスクである. 従って, 提案処理方式では Packet Handling Module に専用のワーカースレッドを生成して割り当てる. ワーカースレッドは, 生成と破棄に要するオーバーヘッドを抑制することが可能であるため, 処理の軽量化が見込める. また, ワーカースレッドは非同期で実行されるため, アダプタから送信されるパケット順序が, 着信時とずれる可能性がある. そのため, Packet Handling Module における処理の際に, パケット順序を維持するための順序付けスキームを導入する. 検証評価として, 提案する CYPHONIC アダプタに複数の一般ノードを接続し, コネクションが増加した場合の通信スループット, 及び通信遅延のパフォーマンス傾向を確認する. また, 提案方式ではワーカースレッドを生成して処理を移譲することから, スレッド数を変化させて, パフォーマンスがどのように向上あるいは劣化するかを検証する. 最後に, 得られた検証データをもとに, 実運用を想定した考察を述べる.

4.1.2　CYPHONIC クラウド

CYPHONIC アダプタのマルチノード対応が完了すると, CYPHONIC サービスを利用可能となる端末の種類は格段に増えるため, 利用者やさらなるノードの増加が想定される. 一方で, 各ノードの管理機能やトンネル確立をサポートする CYPHONIC クラウドは, 単一インスタンスのみで稼働している. そのため, 過負荷や障害により, クラウドサービスが停止した場合, CYPHONIC ノードや一般ノードは P2P 通信を確立することが不可能となる. また, CYPHONIC クラウドの可用性を向上させる既存方式として, フェールオーバーを用いたマスター・スレーブ方式によるサーバ多重化が提案されている. しかし, マスター・スレーブ方式に対応したクラウドサービスは NMS のみであり, 認証機能を担う AS や, トンネル通信を中継する TRS は可用性を担保する仕組みが備わっていない. また, NMS において採用されたフェールオーバ方式も, 障害を前提とした設計であるため, 少なからず, マスター・スレーブの切り替えに時間を要することから, ダウンタイムが発生する. さらに, 複数のスレーブサーバを通知する術として, 認証応答時に返送される, Login Response に NMS の宛先情報を複数含める必要があるため, 冗長度に応じてパケットが肥大化するという課題があった. CYPHONIC ノードは NMS に対してネットワーク環境情報を登録する際や, 端末の移動に伴って位置情報を更新する際に, 多重化した全ての NMS に対してシグナリングを実行する必要がある. そのため, CYPHONIC ノードを含む末端ノードの負荷が高くなる上, 処理機能が冗長になる課題も存在する. 従って, CYPHONIC クラウドの障害を未然に防ぐ仕組みや, クラウドサービス内部で発生した障害をノードに対して隠蔽する仕組みが必要である. また, CYPHONIC クラウドは, 機能毎に負荷の傾向が異なるため, 各サービスのリクエスト数に応じた適切なスケールアウト機能が必要である.

そこで, CYPHONIC クラウドの可用性を担保するため, 障害許容性と高負荷耐性を実現する仕組みを導入する. 本研究では, CYPHONIC クラウドにコンテナオーケストレータをベースとしたインフラストラクチャを導入し, 各サービスサーバの台数を多重化したクラスタを用いる. また, クラスタを構成する各サーバの CPU やメモリの使用率といったメトリクスを一定間隔で収集することにより, 処理負荷に応じたオートスケーリング機能を追加する. 加えて, クラスタの前段にロードバランサを設置することで, 個々のサーバに掛かる負荷を平準化する. 一般的な Software Load Balancer (SLB) では, クライアント端末がロードバランサの IP アドレスに通信を開始し, ロードバランサが各サーバにリクエストを振り分けることで負荷分散を実現する. しかし, サーバに到達するリクエストの送信元 IP アドレスは SLB の IP アドレスとなるため, クライアント端末の所属ネットワークを特定することが困難となる. CYPHONIC では, クラウドサービスがエンド端末から取得したネットワーク環境情報を元に, 通信経路を指示する. そのため, 送信元 IP アドレスが正しく取得されない場合, 通信経路を適切に指示することが極めて困難となる. そこで, クラスタの外部に Border Gateway Protocol (BGP) ルータを設け, Equal Cost Multi Path (ECMP) による等コストパスを用いた負荷分散方式を導入する. BGP ルータとクラスタを構成する各ワーカーノードは internal BGP (iBGP) としてピアリングされるため, SNAT を回避した上で負荷分散機構を実現可能である. 検証評価として, 提案したクラウドサービスに対して複数のクライアントからリクエストを送信し, 処理負荷に応じたコンテナのオートスケーリングを確認する. また, サーバ終了時において, 既存の処理を終えた後にコンテナを安全に停止する Graceful Shutdown を確認し, エンドノードに対してスケールインの挙動を隠蔽可能であるかを検証する.

4.2　CYPHONIC アダプタの拡張設計

4.2.1　イベント駆動処理モデル

複数の一般ノードに対応するため, Adapter Daemon のマルチスレッド化を行う. 既存の Packet Handling Module は, 受信パケットを直列に処理するため, 複数の一般ノードが同時にオーバーレイネットワーク通

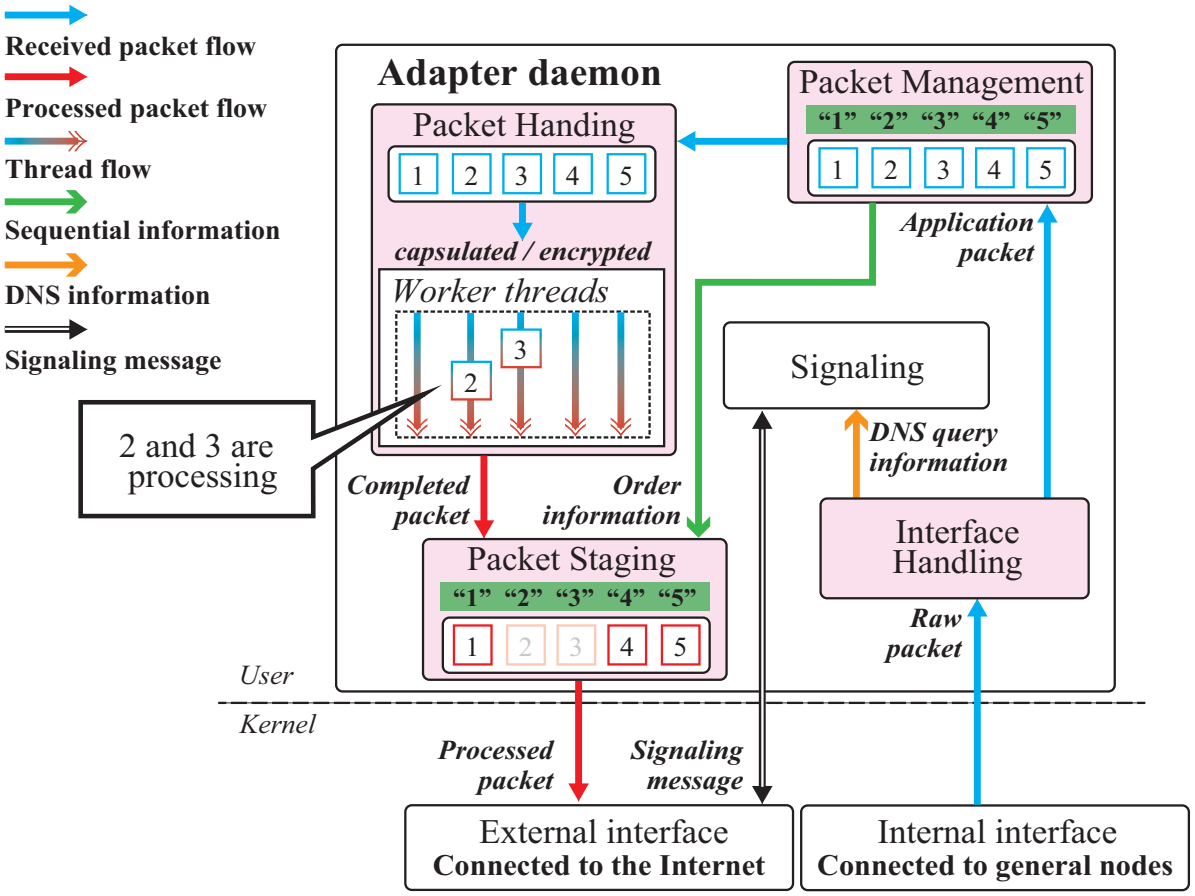


図 4.1: Overview of MultiThread-based CYPHONIC Adapter

信を行う場合, ネットワーク性能が劣化する懸念がある. そこで, Packet Handling Module に専用のワーカースレッドを準備してパケットを並行処理する. 一方で, 各ワーカースレッドの処理は非同期で実行され, OS や言語ランタイムの負荷状況に依存する. そのため, 一般ノードから受信したパケット順序と, 相手ノードに送信する処理済みのパケット順序が異なる可能性がある. その結果, 既存の CYPHONIC アダプタは TCP で極端な性能劣化が確認され, UDP においても Jitter 値が高くなる傾向にあった.

そこで, 本提案では処理方式の見直しを行い, Packet Handling Module のマルチスレッド化に伴い, パケット処理方式を 3 つのステージに分離する. 提案方式を採用した CYPHONIC アダプタの処理スキームを図 4.1 に示す. Adapter Daemon は, Interface Handling Module を介して一般ノードからパケットを受信すると, Packet Management Module に受け渡す. Packet Management Module は, 受信パケットの順序を保存した後, Packet Handling Module に受け渡して処理を行う. Pakcet Handling Module では, Packet Management Module から受信したパケットを複数のワーカースレッドを用いて非同期で並行処理する. その後, 処理されたパケットから順番に Packet Staging Module に受け渡す. Packet Staging Module は, 処理済みパケットを受信した後, Packet Management Module で管理している受信順序に従って, 逐次的に送信する. その結果, 送受信時点におけるパケット順序を維持することが可能である.

また, CYPHONIC Daemon は受信したフラグメントパケットを個別に処理することを前提としている. 一方で, Linux カーネル version 2.6 以降では, フラグメント化されたパケットを Network Interface Card (NIC) で再結合する Generic Receive Offload (GRO) を備えている [79]. そのため, Maximum Transmission Unit (MTU) 値に基づいて受信したパケットは, ユーザ空間で動作する Adapter Daemon に受け渡される際に, デフラグメントされる可能性がある. そこで, 本提案では NIC オフロードの機能を

使用せずに, 受信パケットを前述の処理スキームを用いてユーザ空間上で個別に処理する.

既存の CYPHONIC アダプタは単一の一般ノードのみを想定しているため, Adapter Daemon が起動した際に, 自動的に一般ノード情報をロードする. しかし, CYPHONIC で通信を行う全ての端末情報はクラウドサービスで管理されていることから, 一般ノードの情報も予めクラウドサービスに登録しておくことが望ましい. CYPHONIC アダプタが起動すると, クラウドサービスに一般ノードを取得するためのリクエストを送信し, 任意のアダプタに接続される複数の一般ノード情報を受け取る. また, これらの情報は内部キャッシュで管理され, 一般ノードにおける CYPHONIC のシグナリング処理を代行する際, また, 一般ノードが CYPHONIC アダプタを介してオーバーレイネットワーク通信を行う際に, 参照される. 一方で, インメモリ上のキャッシュデータは, CYPHONIC アダプタの停止に伴いって消去される. そこで, CYPHONIC アダプタは取得したデータを不揮発性メモリに保存しておき, 必要に応じてインメモリキャッシュに再ロードすることにより, 効率的に一般ノード情報を管理する. また, 一般ノードの情報がクラウドサービスで更新された場合, NMS を通じて, CYPHONIC アダプタに更新リクエストを送信する. NMS は, CYPHONIC アダプタと常時 Keep Alive を実行しているため, クラウドサービスからプライベートネットワーク内に設置される CYPHONIC アダプタに通信開始することが可能である. CYPHONIC アダプタは, NMS から一般ノード情報が更新された旨の通知を受け取ると, 再度クラウドサービスに一般ノード情報を問い合わせる.

4.2.2　CYPHONIC アダプタの拡張シグナリング

以下に, CYPHONIC アダプタのシグナリングパターンについて詳述する.

4.2.2.1　CYPHONIC アダプタの認証処理

Adapter daemon が起動すると, CYPHONIC ノードと同様に認証処理が実行される. CYPHONIC アダプタは, AS に Login Request を送信し, Login Response によって CYPHONIC アダプタが NMS との通信で使用する共通鍵, CYPHONIC アダプタの FQDN, CYPHONIC アダプタの 仮想 IP アドレス, CYPHONIC アダプタの NodeID を取得する. CYPHONIC アダプタはゲートウェイアドレスと呼ばれる特殊な仮想 IP アドレスを内部インターフェースに割り当てて使用する. 内部インターフェースとは, 一般ノードに接続されるネットワークインターフェースのことである. なお, 内部インターフェースに割り当てられる IP アドレスは, 一般ノードに仮想 IP アドレスを付与する際や, CYPHONIC アダプタと一般ノード間で行われる通信を実行するために割り当てられる. また, 図 3.14 で示す通り, オーバーレイネットワーク通信において, CYPHONIC アダプタは, 一般ノードの仮想 IP アドレスを CYPHONIC アダプタの実 IP アドレスでカプセル化する. そのため, 内部インターフェスに割り当てられたゲートウェイアドレスは, オーバーレイネットワーク通信には用いない.

CYPHONIC アダプタ, 及び一般ノードは, 認証方式を選択することが可能である. 認証の種類として, DeviceID・パスワードによるベーシック認証, またはクライアント証明書認証を選択する. また, ユーザインターフェースや, 証明書機能がサポートされていない端末を考慮し, MAC アドレスを用いた単純照合認証が使用可能である.

4.2.2.2　CYPHONIC アダプタの位置情報登録処理

AS との認証処理が完了すると, 次に NMS に対して所属ネットワークの情報を登録するために, 位置情報登録処理が実行される. Registration Request には, CYPHONIC アダプタの外部インターフェースに接続されているネットワークの実 IP アドレスが含まれる. NMS は, Registration Request に含まれているネットワークアドレスと, 実際に NMS が CYPHONIC アダプタから受け取ったパケットの送信

元 IP アドレスを比較する．その後, NAPT の有無を判定するフラグを Registration Response に含め, CYPHONIC アダプタに返送する．以後, オーバーレイネットワーク通信を行う際に, NAPT フラグ及び相手ノードのネットワーク情報を元に, 実行するシグナリングパターンを変更する．また, CYPHONIC アダプタと NMS との通信は UDP ベース で行われるため, 介在する NAPT のマッピング情報を保持する必要がある．CYPHONIC アダプタは, NMS に対して UDP Hole Punching を行うことで, Keep Alive を実行する．なお, Keep Alive パケット は Registration Response を受信後, 他のシグナリングや処理の結果に影響されず, 非同期で NMS に送信され続ける．

4.2.2.3　一般ノード情報取得処理

位置情報登録処理が完了した後, CYPHONIC アダプタは自身に接続される一般ノードの情報を取得するために, 一般ノード情報取得処理を実行する．このシグナリングは, CYPHONIC アダプタのみが実行するシグナリングである．一般ノード情報の取得リクエストは, CYPHONIC クラウドのソフトウェアコントローラに対して送信される．CYPHONIC アダプタは自身の DeviceID を含めることで, 管理する一般ノードの情報を取得する．なお, ソフトウェアコントローラは予め, AS と同様にルート認証局によって認証済みである．CYPHONIC アダプタとソフトウェアコントローラ間は HTTP/TLS 通信を行うことで, 信頼性及び完全性を確保した状態で通信を行う．ソフトウェアコントローラは, CYPHONIC アダプタの DeviceID に紐付く一般ノード情報を取得して API レスポンスとして応答する．

4.2.2.4　一般ノードの認証処理及び DHCP リクエスト

CYPHONIC アダプタは, 一般ノードの情報を CYPHONIC クラウドから取得することで, 一般ノードを受け入れ可能となる． CYPHONIC アダプタは IEEE 802.1X 認証をサポートする． また, 現在では多くの端末が IEEE 802.1X をサポートしており, Supplicant として機能することが可能である．CYPHONIC アダプタの文脈では, 一般ノードが Supplicant, CYPHONIC アダプタが Authenticator, AS が 1X Authentication Server に該当する．Supplicant は, Extensible Authentication Protocol (EAP) によって認証手続きを開始する．また, この時, CYPHONIC アダプタは同時に, 一般ノードがブロードキャストした DHCP リクエストを何度か受信する可能性がある．CYPHONIC アダプタは, EAP Request を送信したノードについて, 構成情報を事前に持ち合わせている場合は, 信頼されたノードとして, 一般ノードをアクティベートする．

一般ノードの認証処理として, CYPHONIC アダプタが代理で Login Request を送信する．この時, ベーシック認証であれば, 一般ノードの DeviceID 及び パスワード を含め, 証明書認証の場合は, EAP で取得した証明書を Login Request に含める．AS で一般ノードの認証が完了すると, Login Response によって, 一般ノードが NMS との通信で使用する共通鍵, 一般ノードの FQDN, 一般ノードの 仮想 IP アドレス, 一般ノードの NodeID を含めて返送する．前述した手続きを取ることで, AS は統一された処理を実行することが可能である．つまり, CYPHONIC ノード, CYPHONIC アダプタ, 一般ノードでシグナリングやパケットフォーマットを使い分ける必要がない．なお, Login Response によって取得した情報は全て CYPHONIC アダプタが管理をする．CYPHONIC アダプタは, Read Only Memory (ROM) 等の不揮発性メモリに一般ノードの情報を持つが, 一般ノードが通信をする際には, ファイルから Random Access Memory (RAM) にロードして内部キャッシュに保持する．この時 CYPHONIC アダプタに接続できる一般ノードは事前に CYPHONIC クラウドに登録済の端末のみである．稼働中はインメモリとして管理することで, CYPHONIC アダプタは都度ファイルをリードする必要がないため, 高速に動作することが可能である．また, 一般ノードの通信中に CYPHONIC アダプタが再起動した場合, 保存された一般ノードの情報を再度読み込んで処理を継続する．

4.2.2.5　DHCP レスポンス

EAP による IEEE 802.1X 認証が完了すると, AS から受信した仮想 IP アドレス, 及び事前に管理している一般ノードの MAC アドレスと DHCP DISCOVER パケットの内容を参照して, DHCP プロセスに則って仮想 IP アドレスを割り当てる. また, DHCP の処理は, 後述する一般ノードの位置情報登録処理と並行して非同期で実行される. DHCP レスポンスによって仮想 IP アドレスの他, デフォルトゲートウェイアドレス及び DNS サーバアドレスも同様に通知する. なお, 一般ノードが IPv6 モードで動作している場合は, NDP (ICMPv6) と DHCPv6 を連携して仮想 IP アドレスを割り当てる. DHCPv6 はステートフル方式で動作するため, 仮想 IP アドレスをはじめ, ネットワーク接続に必要情報の管理が容易である. また, 一般ノードに対して, NDP パケットとして Router Advertisement (RA) メッセージを定期的に送信することで, CYPHONIC アダプタを近隣ルータとして確立する.

4.2.2.6　一般ノードの位置情報登録処理

AS から Login Response を受け取ると, DHCP 処理と並行して, NMS に Registration Request を送信する. この時, Registration Request には CYPHONIC アダプタの外部インターフェースの実 IP アドレスを含める. つまり, 一つの CYPHONIC アダプタに接続される一般ノードの実 IP アドレスは何れも CYPHONIC アダプタと同一となる. また, Registration Request を暗号化する際には, 特定の一般ノードと NMS 間で使用する共通鍵を使用する. NMS は, Registration Request を受信すると, NAPT フラグを Registration Response に含めて返送する.

4.2.2.7　一般ノードのプロビジョニング

一般ノードに対する Registration Response を受信すると, CYPHONIC アダプタは受信した一般ノードに対する起動ステートをアクティベートする. これにより, 以後の処理で必要となるスレッドを起動して待機し, 一般ノードから送信されたパケットを取得, 及び処理する準備が完了する. CYPHONIC アダプタが一般ノードの送受信に伴って生成するスレッドは, 親スレッドからフォークされたワーカースレッドであり, イベント駆動で処理を継続する. パケットの送受信に必要となるメモリ領域を予め準備しておくことで, スレッドの生成と破棄に要するシステムコールを避けることで, 処理の効率化が可能である. なお, CYPHONIC アダプタ側で一般ノードの起動ステートがアクティベートされるまで, 一般ノードはオーバーレイネットワーク通信を開始することは不可能である. また, 一般ノードが一定期間通信を行わない場合, CYPHONIC アダプタは, 生成した内部スレッド及びメモリバッファを破棄する. これにより, 不要なメモリ領域がスタックするのを回避するとともに, メモリフラグメントを最小限に抑制する.

4.2.2.8　経路確立処理

一般ノードが Initiator である場合の経路確立処理を取り上げて次に説明する. CYPHONIC はピアノードに対する FQDN クエリ及び DNS リクエストパケットをトリガとしてオーバーレイネットワークの構築処理を開始する. 一般ノードは通信を開始する際に, ピアノードの FQDN を DNS クエリに内包して CYPHONIC アダプタへ送信する. この時, DNS パケット及び L3 パケットは, 一般ノードの仮想 IP アドレスを送信元 IP アドレス, ゲートウェイアドレスを宛先 IP アドレスとして送信される. CYPHONIC アダプタは, 一般ノードから DNS パケットをフックすると, Direction Request パケットを生成して NMS へ送信する. Direction Request には, 一般ノードのネットワーク環境情報, ノード情報, 通信経路を一意に定める PathID 等が含まれる.

NMS は CYPHONIC アダプタから Direction Request を受信すると, NodeID を元に, データベースから一般ノードの情報を取得して, Route Direction To Responder パケットを Responder ノードに送信する. Responder ノードは, 通信可能な状態にある場合, Route Direction Confirmation パケットを NMS に返送する. なお, Responder ノードが iOS, Android 等のモバイル端末の場合, Apple Push Notification Service (APNS) や Firebase Cloud Messaging / Firebase Notifications と NMS を連携させ, プッシュ通知を用いた Route Direction を行う. MNS は, Route Direction Confirmation パケットを Responder ノードから受信すると, Base Header 内に通信経路を一意に定める PathID を含めて, Route Direction To Initiator パケットを CYPHONIC アダプタに送信する. CYPHONIC アダプタは Route Direction To Initiator パケットを NMS から受信すると, PathID をキーとして, インメモリキャッシュで管理している一般ノードの情報及び NMS との共通鍵を取得する. その後, NMS との共通鍵を用いて Route Direction To Initiator パケットを復号し, Body 部を取得する. 最後に, Body 部に含まれる, 仮想 IP アドレスを元に, DNS レスポンスパケットを生成して, 一般ノードに応答する.

4.2.2.9　ゲートウェイプロキシ処理

一般ノードが DNS レスポンスを受信すると, 取得した仮想 IP アドレスに対する ARP リクエストが送信される. これは, 一般ノードと同じ帯域のネットワークアドレスが通知されたため, 対象ノードが LAN 内に存在すると, 一般ノードが判断するためである. しかし, オーバーレイネットワーク上は同一ネットワークであっても, 物理ネットワークでは ピアノードは異なるネットワークに存在する. そのため, 一般ノードのパケットは通常, ピアノードに直接送信することは不可能なため, ARP レスポンスとして CYPHONIC アダプタの MAC アドレスをピアノードの仮想 IP アドレスにマッピングして応答する. これにより, L2 ネットワークの特徴を用いて, 一般ノードと CYPHONIC アダプタ間はリンクレイヤ通信を行う. 従って, ゲートウェイアドレス及び L3 アドレスがこの間の通信で使用されることはない. 以上の手続きは, 一般に Proxy ARP として知られる.

また, IPv6 では, Proxy ARP に変わる L3 プロトコルとして ND Proxy を使用する. ND Proxy は, RA メッセージを定期的に送信をすることで, 一般ノードに対して近隣確立処理を行うとともに, ピアノードに対する Neighbor Solicitation (NS) メッセージに対して, CYPHONIC アダプタの MAC アドレスをピアノードの仮想 IP アドレスにマッピングした Neighbor Advertisement (NA) を代理で応答する. これにより, CYPHONIC アダプタを通じて, 一般ノードとピアノードは近隣確立を行う.

4.2.2.10　トンネル確立処理

一般ノードが Initiator である場合のトンネル確立処理を取り上げて次に説明する. 内部インターフェース側で Proxy ARP もしくは ND Proxy を実行をする際, 同時に外部インターフェースでは並行してトンネル確立処理を実行する. CYPHONIC アダプタは, DNS Response を生成した後, 一般ノードの End Key を生成して, Tunnel Request に内包してピアノードへ送信する. この時, 既に, Route Direction To Initiator パケットによって, ピアノードへの通信経路は確定している. また, 通信経路は PathID によって取得される. ピアノードは, Tunnel Request パケットを受信すると, 同様に End Key を生成して Tunnel Response に内包し CYPHONIC アダプタへ返送する. CYPHONIC アダプタは, Tunnel Response を受信すると, 一般ノード情報を管理するインメモリキャッシュに End Key を格納して管理する.

4.2.2.11　データ通信処理

トンネル確立処理が完了し, オーバーレイネットワークの構築が完了すると, 一般ノードと CYPHONIC アダプタは, リンクレイヤ通信を行い, CYPHONIC アダプタとピアノードは L3 以上のオーバーレイネッ

トワーク通信を行う．CYPHONIC アダプタは, データ通信処理の際, 常に MAC アドレスをキーとして一般ノード情報を内部キャッシュから参照する．また, オーバーレイネットワーク通信の際は, Tunnel Response によって取得された End Key を用いて Body 部を暗号化し, セキュアなエンド間通信を実現する．オーバーレイネットワーク構築後は, 介在する NAPT のマッピング情報を保持するため, UDP Hole Punching を ピアノードに送信し続けることで Keep Alive を行う．なお, ピアノードに対する Keep Alive パケットも, 他のシグナリングや処理の結果に影響されず, 非同期で送信され続ける．

4.2.2.12　後続処理

一般ノードがピアノードと通信を一定期間行わない場合, CYPHONIC アダプタは, 内部キャッシュを削除する．また, 一般ノードの起動ステートを非アクティベート化して, 生成されたスレッド及び, メモリバッファを, 起動時に確保したプールに返却する．一般ノードがが再び通信を開始する場合, DNS パケットをトリガとして内部キャッシュを生成し, 再度経路確立処理を開始する．

4.3　CYPHONIC クラウドの規模拡張性設計

4.3.1　オートスケーリング及びオートヒーリング

CYPHONIC クラウドは, AS, NMS, TRS の 3 種類のマイクロサービスで構成される．これらのサービスは各機能毎に負荷の傾向が異なる．特に, TRS は端末間での直接通信が困難なネットワーク環境では, 通信トラフィックを常に中継し続ける必要がある．本提案では, トラフィックに応じた柔軟なスケールアウト機能を搭載するため, コンテナオーケストレーションを用いる．既存の CYPHONIC クラウドはシングルバイナリで実行可能であるため, コンテナとして扱うことが容易である．コンテナは仮想マシン, 及びクラスタのワーカーノード上で柔軟に生成・破棄できるため, トラフィックに応じたスケールアウトの実現が可能である．また, 各コンテナのメトリクスを一定間隔で収集することにより, コンテナの台数やトラフィックのルーティング先を切り替える．本提案では, コンテナのメトリクスとして CPU 使用率を用いる．AS は TCP コネクションを確立していることから, 通信の際にサービスが停止すると既存処理の継続が困難となる．そのため, Graceful Shutdown を導入することにより, サービスアウトしたコンテナに対してトラフィックを送信しないようにする．また, スケールインに伴い, 既存のコンテナを安全に停止するとともに, 他のコンテナが処理を継続することでダウンタイムの発生を防ぐ．さらに, コンテナオーケストレーションツールは宣言的な構成に基づいて, サーバの台数を一定に維持可能である．そのため, スケールアウトされたコンテナはトラフィックが減少すると, スケールインを実行する．また, 何らかの理由によりコンテナが停止した場合, セルフヒーリングによって自動的にサービスを復旧する．その結果, 管理者は CYPHONIC クラウドの運用に伴い, 手動でサーバの台数を増加させたり, 障害からの復旧作業を行ったりする必要がなく, 継続してクラウドサービスを運用することが可能である．

4.3.2　ECMP を活用した負荷分散機構

CYPHONIC では, クラウドサービスがエンド端末から取得したネットワーク環境情報を元に, 通信経路を指示する．そのため, 送信元 IP アドレスが正しく取得されない場合, 通信経路を適切に指示することが極めて困難となる．一般的な Software Load Balancer (SLB) では, クライアント端末がロードバランサの IP アドレスに通信を開始し, ロードバランサが各サーバにリクエストを振り分けることで負荷分散を実現するこの時, サーバに到達するリクエストの送信元 IP アドレスは SLB の IP アドレスとなるため, クライアント端末本来の IP アドレスを特定することが困難となる Source Network Address Translation (SNAT) が生ずる．そこで, 本提案では, クラスタの外部に Border Gateway Protocol (BGP) ルータを設

け, Equal Cost Multi Path (ECMP) による等コストパスを用いた負荷分散方式を導入する. BGP ルータとクラスタを構成する各ワーカーノードは internal BGP (iBGP) としてピアリングされるため, SNAT を回避した上で負荷分散機構を実現可能である.

第5章　検証及び評価

5.1　概要

前章では, CYPHONIC におけるいくつかの課題に対応するべく, エンドノード, 及びクラウドサービスのそれぞれに着目し, CYPHONIC サービスの拡張性を実現する設計手法について提案を行った. エンドノードでは, 複数の一般ノードをサポートするべく, CYPHONIC アダプタの拡張設計を提案した. また, クラウドサービスは, 複数のインスタンスを用いてサービスを多重化し, 効率的かつ高可用性を実現するためのインフラストラクチャ設計を提案した. 本章では, 提案手法を採用した CYPHONIC アダプタの実装詳細と, 規模拡張性を考慮した CYPHONIC クラウドのインフラストラクチャ設計の実装について説明する. また, 実装を行なった CYPHONIC アダプタ, 及び CYPHONIC クラウドについて検証・評価を実施し, その結果について考察する.

5.2　提案システムの実装

CYPHONIC アダプタ, 及び CYPHONIC クラウドは従来から利用されてきた Go 言語を用いて拡張実装を行う. Go 言語は, ホスト OS に依存することなく, コンテナ型仮想化上のアプリケーションでも動作することが可能である. また, コンパイル言語でありシングルバイナリで動作するため, 移植性や冪等性の観点でも優れている. 本研究における提案したシステムにおいて, 実装に用いた言語及びパッケージのバージョン情報を表 5.1 に示す. また, 検証に用いる機器の詳細を, 表 5.2 に示す. CYPHONIC のシグナリングでは, TCP, UDP, TLS を用いる. そのため, Go の標準ライブラリである net ライブラリを用いて実装する. 各サービスは, イベント駆動型アーキテクチャを使用してスレッドプールでイベント処理を実行するように実装されている. また, CYPHONIC アダプタは, 処理の途中で状態を保持するプロセスがある. そのため, 独自でインメモリキャッシュに状態を格納する機能を Go で実装した. 暗号化・復号処理をするスレッドは, Goroutine を用いてマルチスレッド対応する.

CYPHONIC クラウドは分散コンテナ実行基盤として Kubernetes を採用し, シングルバイナリでコンパイルしたソフトウェアをコンテナとして展開する.

5.3　CYPHONIC アダプタの実装詳細

既存の CYPHONIC Daemon は, CYPHONIC クラウドとシグナリングを行う Signaling Module, 仮想 IP アドレスを用いたパケットの処理や, 暗号化処理, データの送受信を行う Packet Handling Module を実装している. また, CYPHONIC では FQDN を用いて各ノードを識別することから, DNS パケットを処理するための CYPHONIC Resolver Module を実装している. CYPHONIC アダプタは, 一般ノードに対して通信処理を代行するために, これらのモジュールを活用する. 以下に, 各モジュールの実装について詳述する.

- Signaling Module
  Signaling Module は, CYPHONIC クラウドとの一連のシグナリング処理を管理する. そのため,

表 5.1: Tool versions associated with implementation

Language / Package	CYPHONIC Adapter	CYPHONIC Cloud
go	v1.20.1	v1.20.1
x/net	v0.18.04	v0.15.0
google/gopacket	v1.1.19	v1.1.19
google/uuid	v1.4.0	v1.3.1
miekg/dns	v1.1.51	-
go-ping/ping	v1.1.0	-
insomniacslk/dhcp	v0.0.0-20230307103557-e252950ab961	-
mdlayher/dhcp6	v0.0.0-20190311162359-2a67805d7d0b	-
spf13/cobra	v1.7.0	-
zap	v1.24.0	v1.26.0
natefinch/lumberjack.v2	v2.2.1	-
yaml.v2	v2.4.0	-
github.com/aws/aws-sdk-go	-	v1.45.14
driver/postgres	-	v1.5.2
gorm	-	v1.25.4
golang/mock	-	v1.6.0
sideshow/apns2	-	v0.23.0

Signaling Module はソケット通信を管理するために, OS 標準のソケット API を用いて実装する. AS とのシグナリングは SSL/TLS による通信を行い, NMS や TRS とのシグナリング時は, UDP による通信を行う. CYPHONIC クラウドとの間で送受信されるパケットは, 暗号化を行う必要があるため, OpenSSL を使用する. さらに, UDP 通信では CYPHONIC クラウドから受信するメッセージに応じて, 通信プロセスを管理している. また, Signaling Module は, UDP 通信で受信したデータのヘッダ情報から, パケットの種類を特定し, その種類に応じた処理を各モジュールに指示する.

- Packet Handling Module
  Packet Handling Module は, 定義された CYPHONIC パケットの形式に従って, カプセルメッセージを生成する. また, CYPHONIC パケットの暗号化と, HMAC のハッシュ計算には OpenSSL を用いる. パケットの共通鍵暗号アルゴリズムには AES-256-CBC, 認証アルゴリズムには HMAC-MD5 をそれぞれ使用する. カプセル化の際は, パケットの末端に HMAC を付与する. デカプセル化の際は, 受信した CYPHONIC パケットの HMAC を計算することで, 改ざん検出を行う.

- CYPHONIC Resolver Module
  CYPHONIC Resolver Module は一般ノードから受信した FQDN クエリに基づいて DNS パケットの処理を行う. CYPHONIC アダプタは一般ノードから様々な DNS パケットを受信することが想定されるため, CYPHONIC で用いられるドメインのみをフィルタリングする. パケットのフィルタリング機能は, Go 言語ベースで作成された CoreDNS を用いて実装し, DNS パケットの分析は, miekg/dns, google/gopacket を使用して実装する. CYPHONIC Resolver Module は, 一般ノードから受信した FQDN クエリをもとに, DNS Request を生成して Signaling Module へ渡す. また, 相手ノードの仮想 IP アドレスを受信した際は, DNS Response を生成して, 一般ノードへ送信する.

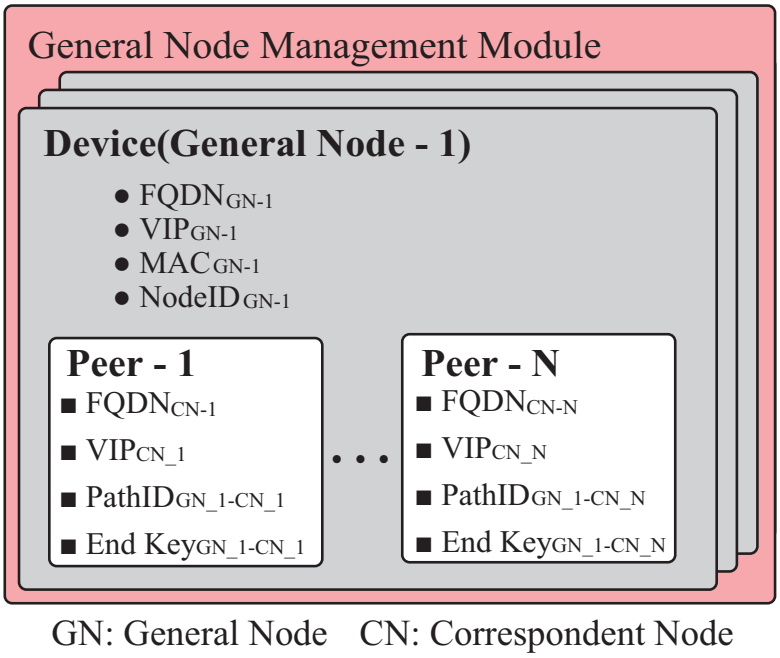


図 5.1: Management of General Node information

CYPHONIC アダプタは, 接続される一般ノードの情報として仮想 IP アドレスや FQDN を管理する必要がある. また, 一般ノードが用いる仮想 IP アドレスを MAC アドレスに基づいて付与するため, 一般ノードが使用するインターフェースの MAC アドレスを管理する必要がある. さらに, 一般ノードの通信に伴い, 一般ノードが CYPHONIC 上の相手ノードに送信するパケットを取得する機能が必要である. 接続される一般ノードの諸情報を管理する機能は, General Node Management Module として実装する. 一般ノードに対して DHCP プロセスを用いた仮想 IP アドレスの割り当て処理は, Address Configuration Module として実装する. 一般ノードが接続されるインターフェースを通して, 一般ノードとパケットを送受信するための機能は, Interface Handling Module として実装する. これらのモジュール群を Adapter Function として CYPHONIC Daemon 内に追加する. Adpater Function と CYPHONIC Daemon が協調処理を行うことで, 一般ノードに対して CYPHONIC の機能を提供する. 以下に, Adapter Function の各モジュールの実装について詳述する.

- General Node Management Module
  General Node Management Module は, 接続される一般ノードの情報をアダプタ内部のインメモリキャッシュを用いて保管する. CYPHONIC アダプタが起動すると, 一般ノード情報取得処理により, CYPHONIC クラウドから一般ノードの仮想 IP アドレス, FQDN, MAC アドレスを取得する. そして, 一般ノードが通信を行う際に, キャッシュ情報を参照することにより, CYPHONIC Daemon へ必要な情報を引き渡す. 図 5.1 に, CYPHONIC アダプタに実装するキャッシュテーブルの構成を示す. 通常, CYPHONIC ノードは Device と呼ばれるキャッシュテーブルに自身の FQDN や仮想 IP アドレス, CYPHONIC の通信に伴って送信者を識別する NodeID が格納される. また, 相手ノードとの通信プロセスごとに, Peer と呼ばれるキャッシュテーブルを生成して, 通信相手の FQDN や仮想 IP アドレス, 通信経路を示す PathID を格納する. さらに, トンネル確立処理に用いるために CYPHONIC ノード自身が生成した End Key を格納する. CYPHONIC アダプタは, 自身の Device テーブルの中に, さらに一般ノードの情報を格納する Device テーブルを, 接続される一般ノードの数分保持する. また, 一般ノードの通信プロセスごとに処理を行う必要があるため, Peer テーブルは一般ノードの Device テーブルごとに, それぞれ生成を行う. 通信の際は, 一般ノードの

仮想 IP アドレスを用いて Device テーブルを参照し, その中に格納されている Peer テーブルから End Key を取り出す. その後, 一般ノードから受信したパケットとともに CYPHONIC Daemon へ受け渡す. また, 相手ノードからアダプタ配下の一般ノードに対する通信開始要求を受信した場合, CYPHONIC アダプタは Device テーブルを参照して, FQDN に紐付けられている仮想 IP アドレスを応答する. CYPHONIC アダプタが管理する Peer テーブルは, 一定期間通信が行われない場合, 自動的に削除する. これにより, CYPHONIC アダプタのメモリ浪費を防止する.

- Address Configuration Module
  Address Configuration Module は, 接続される一般ノードに対して仮想 IP アドレスを付与するため, DHCP の機能を実装する. アドレス付与の際は, General Node Management Module を参照して一般ノードの MAC アドレスをもとに, 管理している仮想 IP アドレスを取得する. そして, DHCP の静的アドレス割り当て方式によって, 一般ノードへ仮想 IP アドレスを付与する. 仮想 IP アドレスは MAC アドレスに紐付けられるため, リース期間が終了した場合にも, Address Configuration Module は再度同じアドレスを割り当てる.

- Interface Handling Module
  Interface Handling Module は, 接続される一般ノードと, データの送受信を行うために用いられるモジュールである. Interface Handling Module は, 一般ノードから全てのパケットを受信して処理する必要があることから, Raw ソケットとプロミスキャスモードを実装する. Interface Handling Module は, 一般ノードから受信したパケットに応じて処理を実行する. まず, DHCP パケットを受信した場合, 前述した Address Configuration Module によって, DHCP プロセスを実行する. また, 一般ノードが通信を行う際に, CYPHONIC アダプタが経路選択処理を実行すると, 相手ノードの仮想 IP アドレスが一般ノードへ通知されるため, CYPHONIC アダプタは一般ノードから ARP Request を受信する. この ARP Request は, 相手ノードの MAC アドレスを要求するために送信されたものである. CYPHONIC アダプタは, 相手ノードに対する仮想 IP パケットを取得するため, 自身の MAC アドレスを代理で応答する Proxy ARP の機能を実装する. これにより, 一般ノードの ARP テーブルには, 相手ノードの仮想 IP アドレスと, CYPHONIC アダプタの MAC アドレスがマッピングされるため, 一般ノードから受信したパケットを CYPHONIC Daemon で処理することが可能となる. データ通信処理の際は, 一般ノードより受信したイーサフレームからイーサヘッダを取り除き, CYPHONIC Daemon へ仮想 IP パケットを転送する. また, CYPHONIC Daemon からデカプセル化されたパケットを受け取った場合には, CYPHONIC アダプタの MAC アドレスを送信元として, イーサフレームを一般ノードへ送信する.

5.4　検証及び評価

図 5.2 の評価モデルを用いて, 一般ノードの台数を増加させた時の, 1 台あたりの平均通信性能を測定する. CYPHONIC アダプタ, 及び通信相手である CYPHONIC ノードはそれぞれ NAPT 配下に設置される. また, CYPHONIC アダプタは L2 スイッチを介して, 複数の一般ノードを接続する. 両者の通信は, 両 NAPT 環境であるためトンネル確立に伴って TRS を使用するが, その後, 経路最適化を実行することでエンド間通信に切り替える. 本検証において, 通信遅延の測定には ping を使用し, TCP 及び UDP のスループットの測定には iperf3 を使用する.

図 5.3, 図 5.4 に, CYPHONIC アダプタのスレッド数を変化させて場合の評価結果を示す. 評価より, CYPHONIC アダプタは 5 台の一般ノードを接続した場合, 1 台あたり 30Mbps 程度の処理性能を発揮していることが分かる. また, 一般ノード台数の増加に伴う通信遅延が一定であることが確認された. 評価結果より, 例えば, コンビニエンスストア 1 店舗に防犯カメラを 5 台設置する場合を想定すると, 1 台

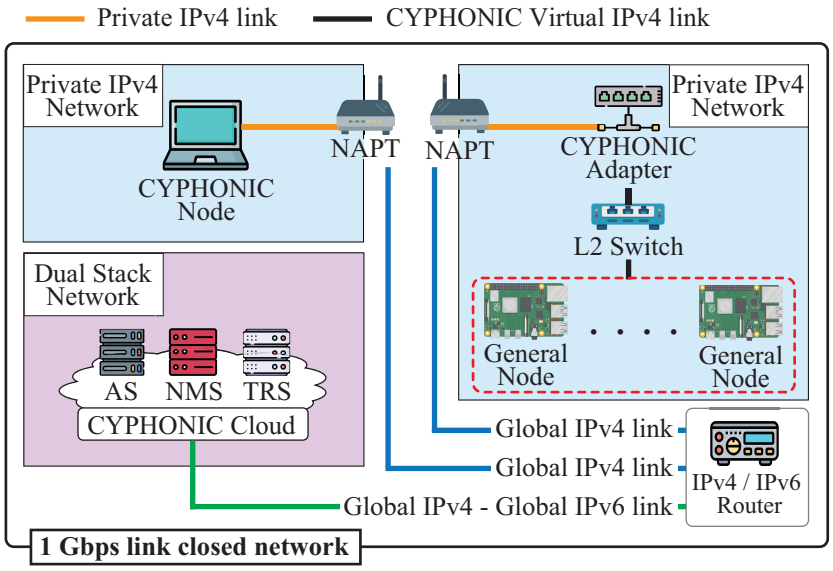


図 5.2: Evaluation model for CYPHONIC Adapter

あたり解像度 4K, 30 fps 程度のストリームデータを処理可能である. 一般的な防犯カメラは 3 – 5 fps 程度で十分に機能するため, さらに多くの IoT 端末をサポートすることが可能であると考察する. さらに, スレッド数を 8 に増加させた場合, 処理性能が大幅に向上し, 一般ノードの通信品質が向上している. そのため, CYPHONIC アダプタのスペックを上げることで, さらに高品質な通信環境を提供することが可能であると考えられる.

クラウドサービスにおけるかクラウドサービスへの負荷分散, オートスケーリングの検証には k6 を用いる. 検証に用いる Kubernetes のバージョン, 及びサーバ機器の使用をそれぞれ, 表 5.3, 表 5.4 に示す. k6 を用いら試験では, 100 RPS で発生させたトラフィックを 30 秒毎に増加させ, 最終的に 500,000 RPS まで増加させる. また, Kubernetes 及び k6 を用いて CYPHONIC クラウドのスケーリングを検証する. 提案手法ではクラスタの前段に BGP ルータを設置し, 各ワーカーノードとピアリングすることで ECMP を用いた負荷分散メカニズムを導入した. 検証の結果, NMS はエンドノードのネットワーク環境情報を正常に取得可能であった. また, CPU 使用率をメトリクスとした水平オートスケーリングの挙動を確認した. 評価結果を, 図 5.5 に示す.

スケールインに伴う検証では特定のリクエストを受信する Pod 及びコンテナを強制的に停止した. コンテナ終了までの待機時間を変化させた場合のリクエストエラーの傾向を表 5.5, 表 5.6, 表 5.7 に示す. 提案した CYPHONIC クラウドは停止に伴う時間を 5 秒程度設けることにより, クラウドサービス内部で発生した障害をエンドノードに対して隠蔽可能であることが確認された.

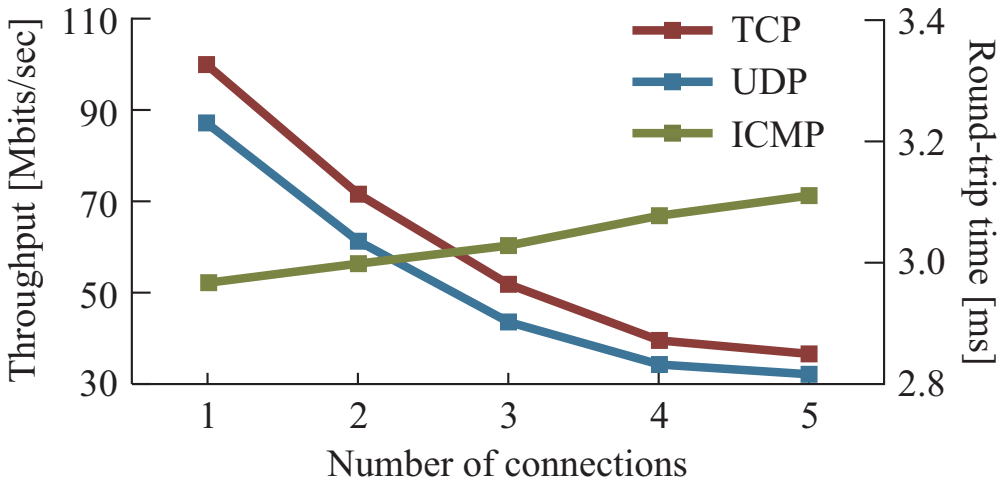


図 5.3: CYPHONIC Adapter: Number of worker threads 4

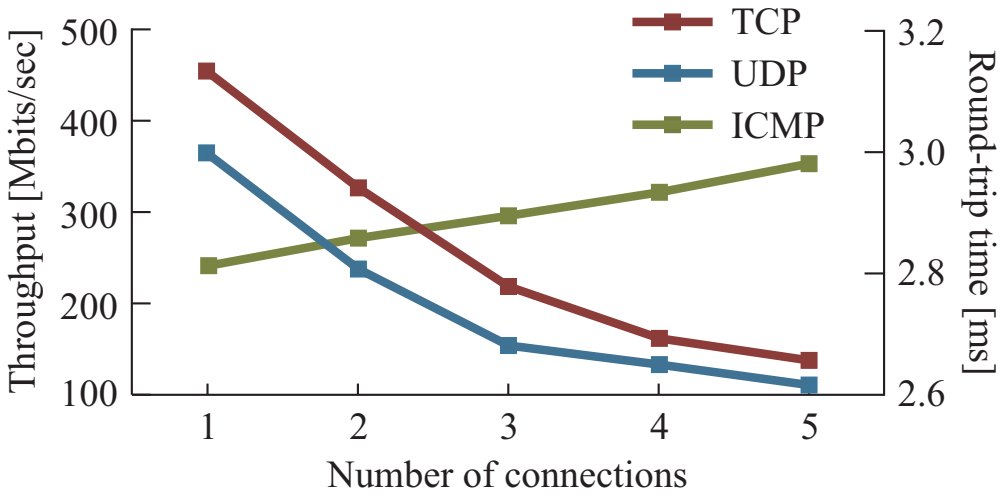


図 5.4: CYPHONIC Adapter: Number of worker threads 8

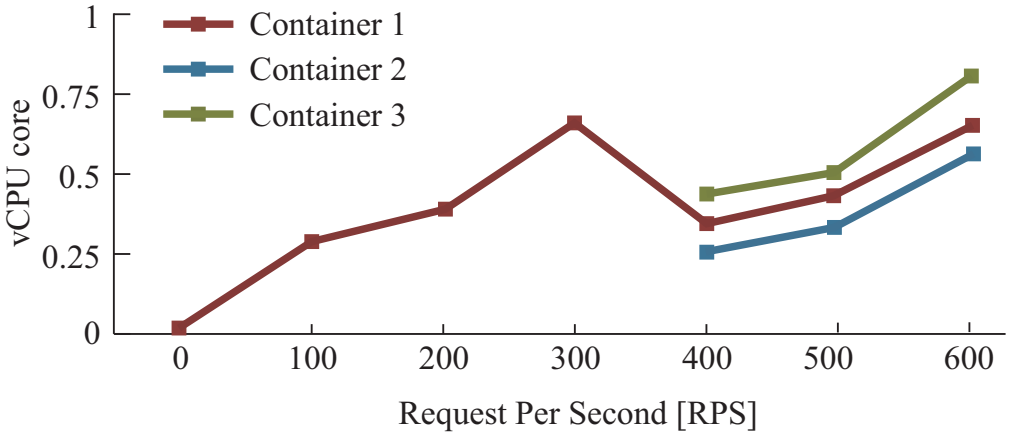


図 5.5: Trends of pod auto scaling

表 5.2: Specifications of the measuring devices for Evaluation of CYPHONIC Adapter

AS, NMS, Controller, CockroachDB	
Virtual Machine	Vagrant v2.2.19 / libvirt v8.0.0 / qemu v6.2.0
OS	Ubuntu 22.04.3 (Jammy Jellyfish) 64 bit
CPU	AMD Ryzen 3 5300U @2.60GHz 4cores 8threads
Architecture	x86_64
RAM	1 GiB DDR4-3200

TRS	
Machine	Raspberry Pi 4 Model B
OS	Debian GNU/Linux 11 (bullseye) 64 bit
CPU	Cortex-A72(ARMv8) BCM2711 @1.5GHz 4cores 4threads
Architecture	aarch64
RAM	4 GiB LPDDR4-2400 SDRAM

CYPHONIC Node	
Machine	Lenovo ThinkPad E14 Gen3
OS	Ubuntu 22.04.3 (Jammy Jellyfish) 64 bit
CPU	AMD Ryzen 3 5300U @2.60GHz 4cores 8threads
Architecture	x86_64
RAM	16 GiB DDR4-3200

CYPHONIC Adapter (Number of worker threads 2)	
Machine	OpenBlocks IoT VX2
OS	Debian GNU/Linux 11 (bullseye) 64 bit
CPU	Intel(R) Atom(TM) CPU E3805 @1.33GHz 2cores 2threads
Architecture	x86_64
RAM	2 GiB DDR3L

CYPHONIC Adapter (Number of worker threads 4)	
Machine	Raspberry Pi 4 Model B
OS	Debian GNU/Linux 11 (bullseye) 64 bit
CPU	Cortex-A72(ARMv8) BCM2711 @1.5GHz 4cores 4threads
Architecture	aarch64
RAM	4 GiB LPDDR4-2400 SDRAM

CYPHONIC Adapter (Number of worker threads 8)	
Machine	Lenovo Yoga Slim 7-14ITL05
OS	Ubuntu 22.04.3 (Jammy Jellyfish) 64 bit
CPU	Intel(R) i5-1135G7 @2.40GHz 4cores 8threads
Architecture	x86_64
RAM	8 GiB DDR4-3200

General Node	
Machine	Raspberry Pi 4 Model B
OS	Debian GNU/Linux 11 (bullseye) 64 bit
CPU	Cortex-A72(ARMv8) BCM2711 @1.5GHz 4cores 4threads
Architecture	aarch64
RAM	4 GiB LPDDR4-2400 SDRAM

表 5.3: Kubernetes component versions

Kubernetes Component	Software	Version
Kubernetes	kubeadm	v1.28.2
Open Container Initiative (OCI)	Docker	v24.0.6
Key-Value Store (KVS)	etcd	v3.2.26
Container Runtime Interface (CRI)	containerd	v1.6.24
Container Network Interface (CNI)	flannel	v0.20.1
Software Load Balancer (SLB)	MetalLB	v0.13.12
Container Storage Interface (CSI)	Ceph	v17.2.6
Kubernetes operator for distributed storage	Rook	v1.11.0
Object Storage	MinIO	v5.0.5
Distributed Database system	CockroachDB	v23.1.8
Distributed In-memory database system	Redis	v6.2.5
Internal Monitoring service	kubernetes-dashboard	v2.7.0
External Monitoring service	kube-prometheus-stack	v42.1.0
kube-dns	CoreDNS	v1.11.1
Add-on	Metrics Server	v0.6.2
Continuous Integration (CI)	GitHub Actions	actions/checkout@v4
Continuous Delivery & Deployment (CD)	ArgoCD	v2.4.11

表 5.4: Specifications of the server machines

Control Plane (Master Node x1)	
Virtual Machine	Vagrant v2.2.19 / libvirt v8.0.0 / qemu v6.2.0
OS	Ubuntu 22.04.3 (Jammy Jellyfish) 64 bit
CPU	Intel(R) i9-11900K @3.50GHz 8cores 16threads
Architecture	amd64
RAM	128 GiB DDR4-3200

Data Plane (Worker Node x3)	
OS	Ubuntu 22.04.3 (Jammy Jellyfish) 64 bit
CPU	Intel(R) i9-13900 @5.60GHz 24cores 32threads
Architecture	amd64
RAM	128 GiB DDR4-3200

BGP Router x1	
Virtual Machine	Vagrant v2.2.19 / libvirt v8.0.0 / qemu v6.2.0
OS	VyOS 1.5 (Circinus) 64 bit
CPU	Intel(R) i9-11900K @3.50GHz 8cores 8threads
Architecture	amd64
RAM	32 GiB DDR4-3200

表 5.5: Scale-in wait time: 0s

Overview	Value
Request count	54230
2xx response	51859
5xx response	2371
Error rate	4.37 %

表 5.6: Scale-in wait time: 3s

Overview	Value
Request count	54230
2xx response	52961
5xx response	1269
Error rate	2.34 %

表 5.7: Scale-in wait time: 5s

Overview	Value
Request count	54230
2xx response	54230
5xx response	0
Error rate	0 %

第6章　総括

6.1　本研究のまとめ

本研究では, エンドノード及びクラウドサービスのそれぞれに着目し, CYPHONIC の拡張性設計を提案した. 評価結果より, CYPHONIC アダプタは複数の一般ノードを同時にサポート可能であることを確認した. また, サーバクラスタリングと ECMP による負荷分散機能を導入した CYPHONIC クラウドの規模拡張性設計が実用可能であることを確認した.

6.2　今後の課題

CYPHONIC アダプタに一般ノードを何台まで接続可能であるかを確認する必要がある. また, CYPHONIC アダプタはオーバーレイネットワーク通信とインターネット通信が同時に実現できない課題が存在する. これは, CYPHONIC アダプタが一般ノードのオーバーレイネットワーク通信サポートする際に実行する Proxy ARP によって, クラスレスネットワークにおける現在のインターネット接続環境では Redirect 処理が発生するためである.

クラウドサービスの検証では, 今回ローカルネットワーク環境に BGP ルータを配置した. しかしながら, 物理ネットワーク環境での検証が未実施であるため, 検証を行う必要がある.

付録

A　パケットフォーマット定義

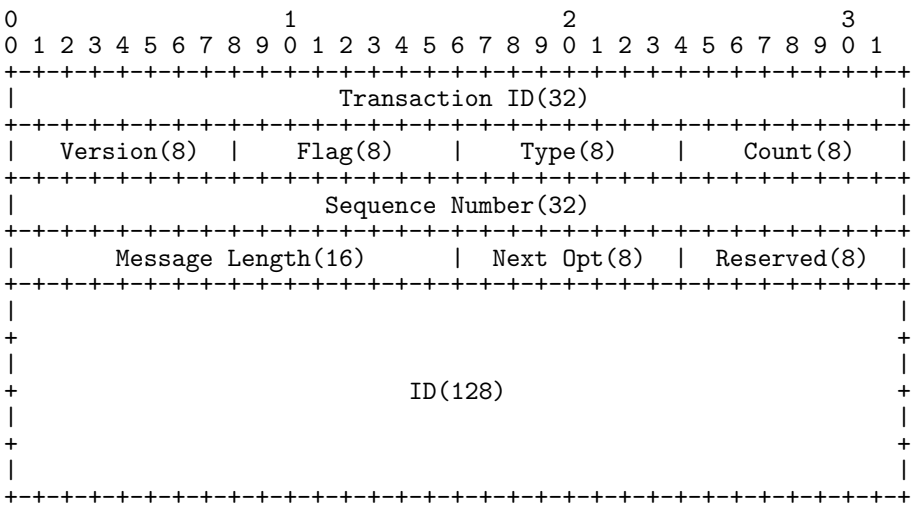


図 A.1: Structure of Base Header

図 A.2: Structure of Hash-based Message Authentication Code

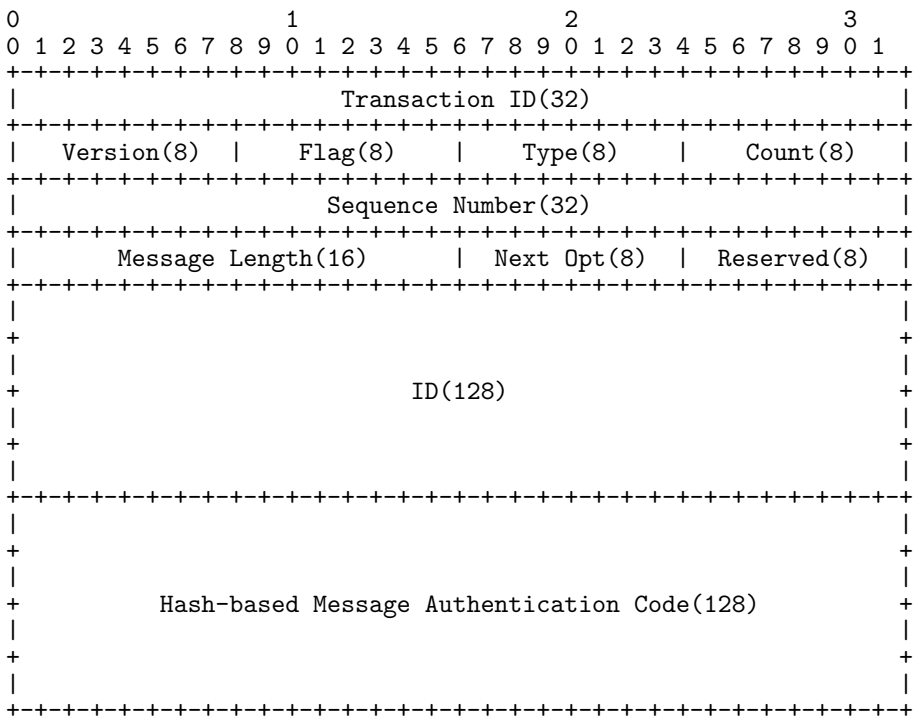


図 A.3: Structure of ACK

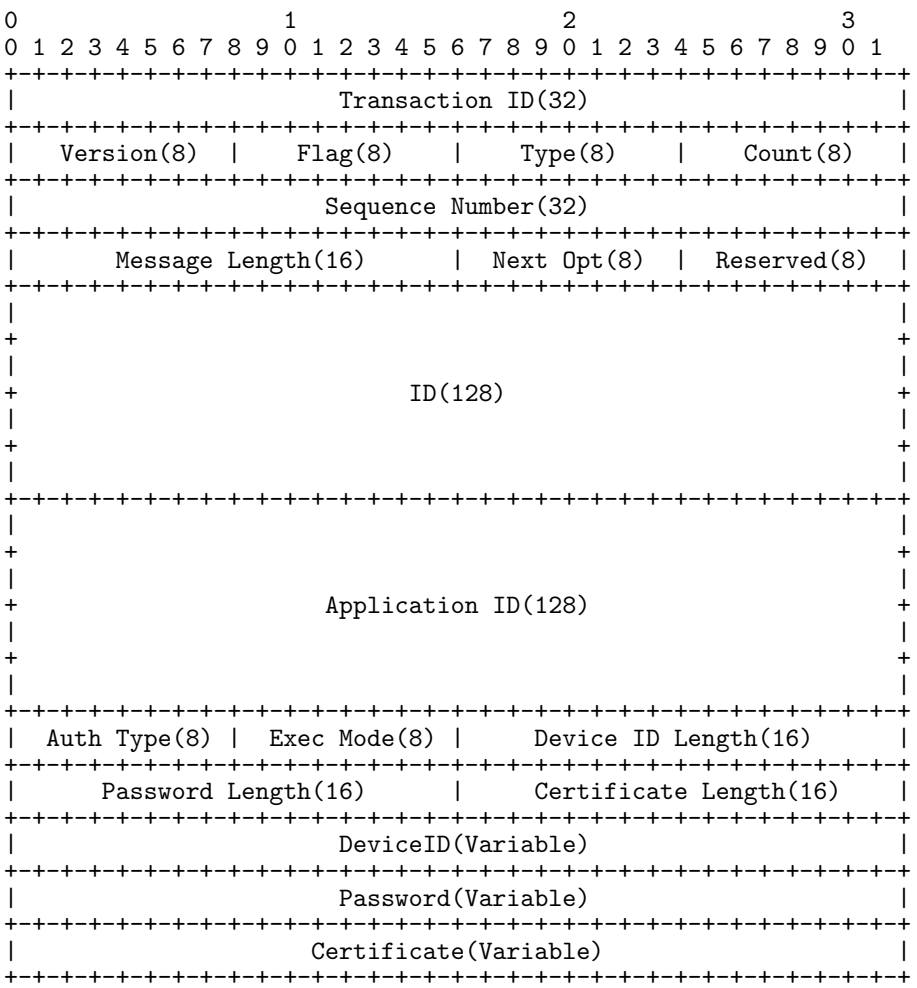


図 A.4: Structure of Login Request

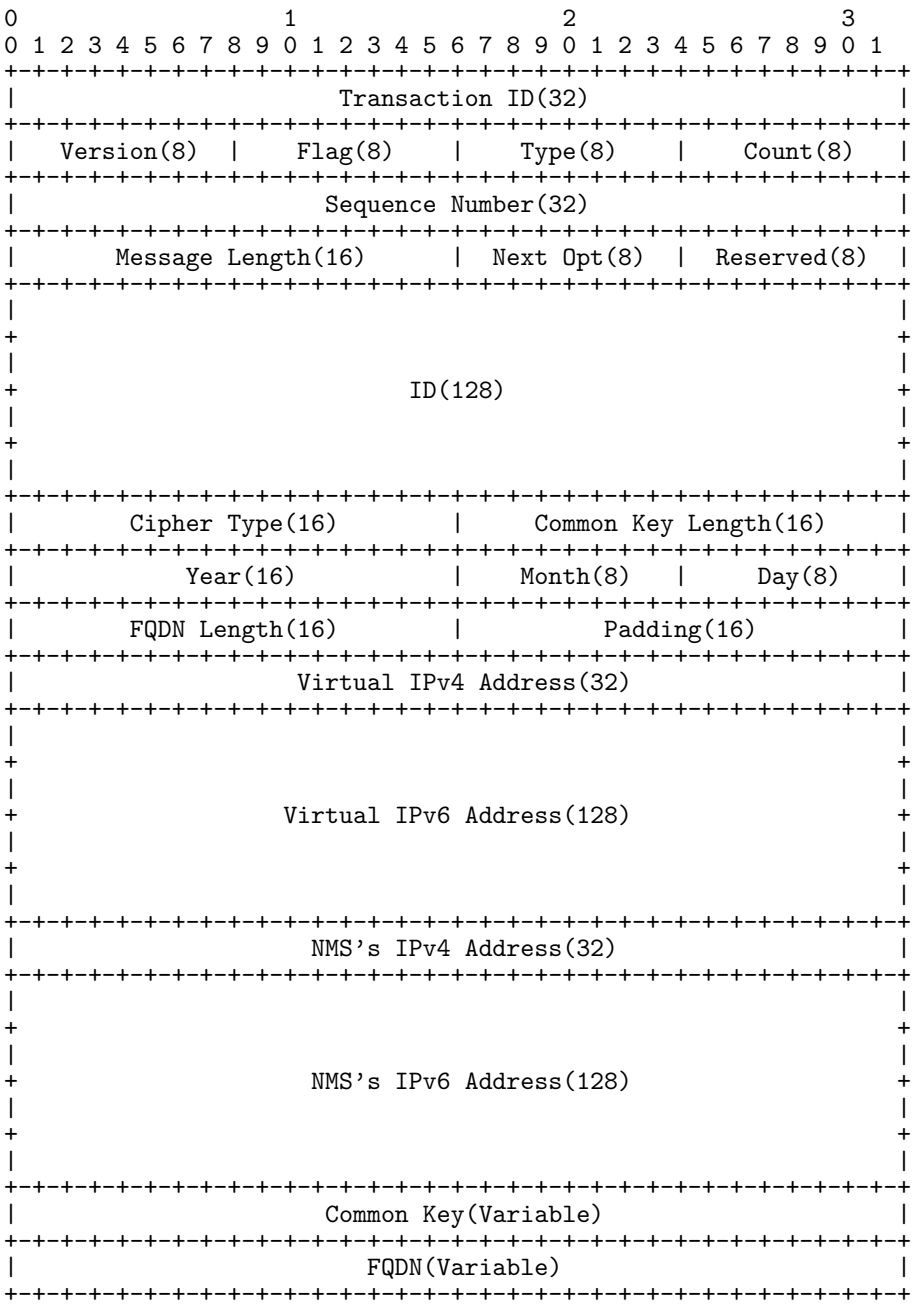


図 A.5: Structure of Login Response

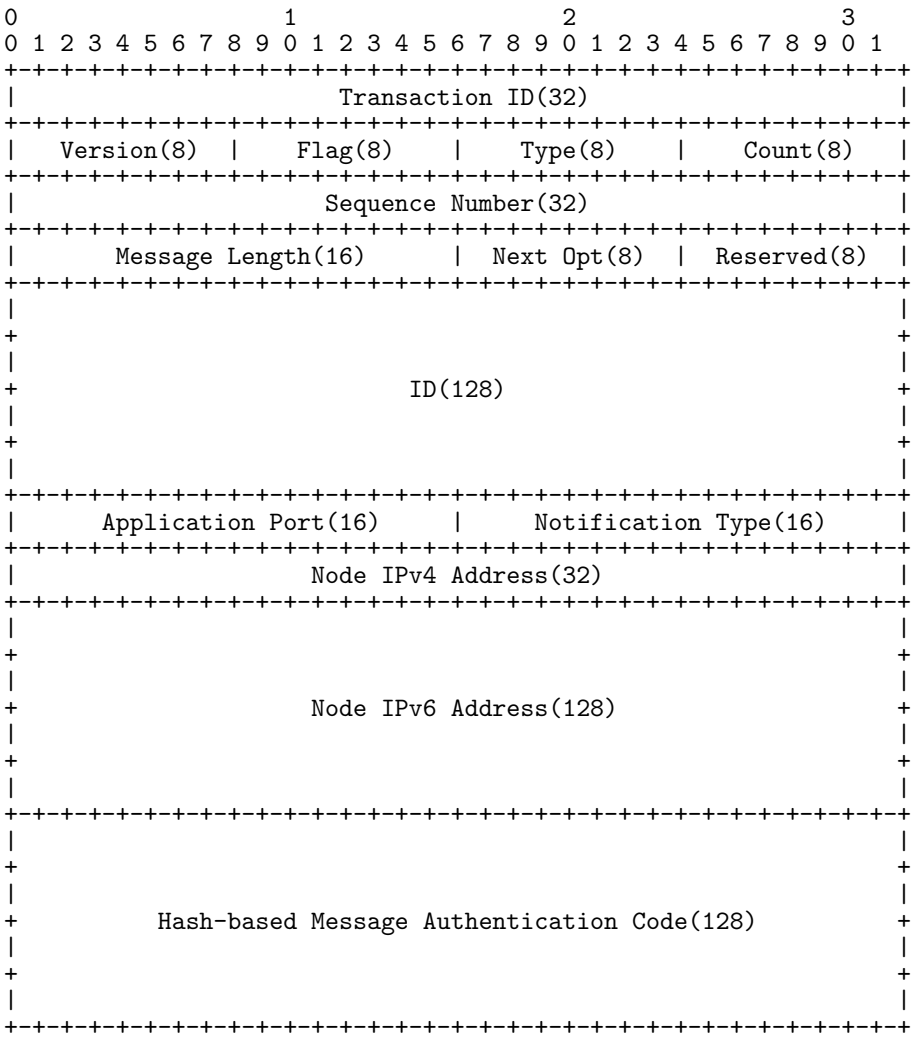


図 A.6: Structure of Registration Request

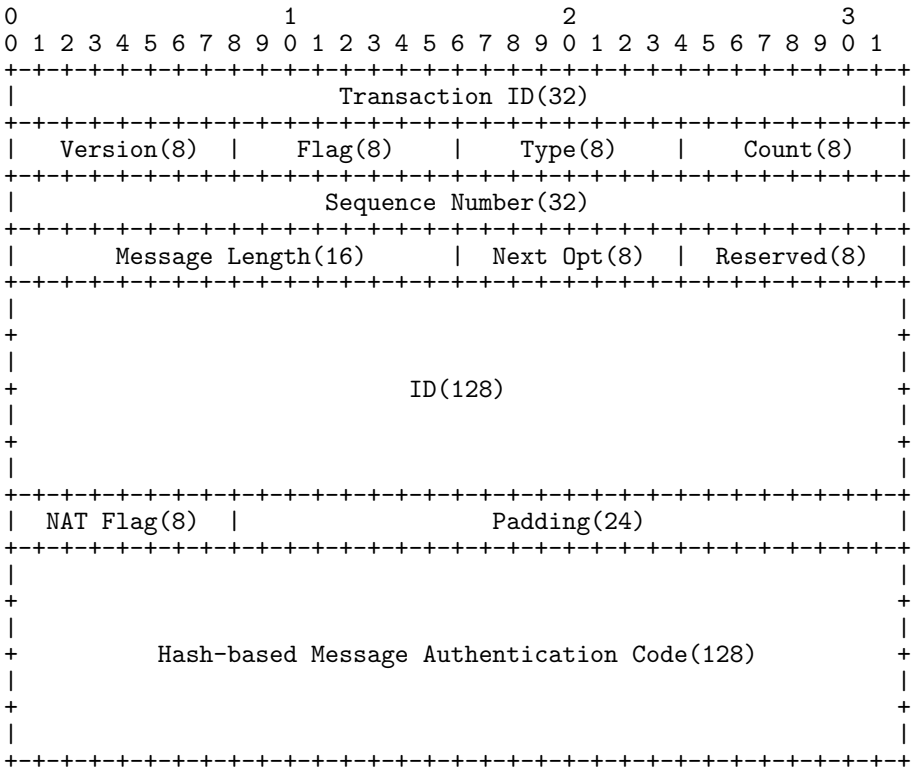


図 A.7: Structure of Registration Response

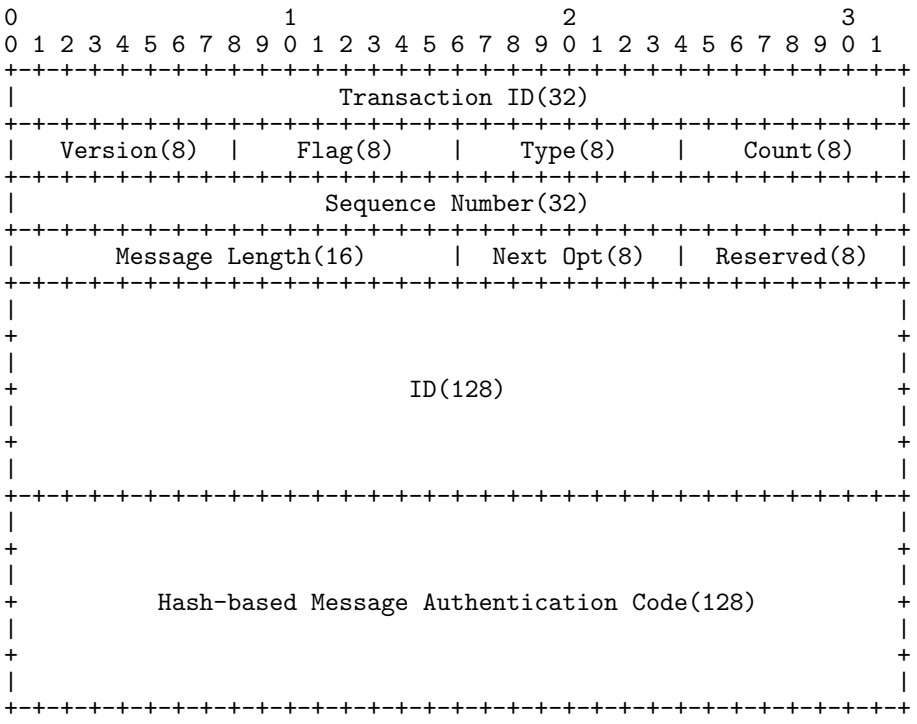


図 A.8: Structure of Keep Alive

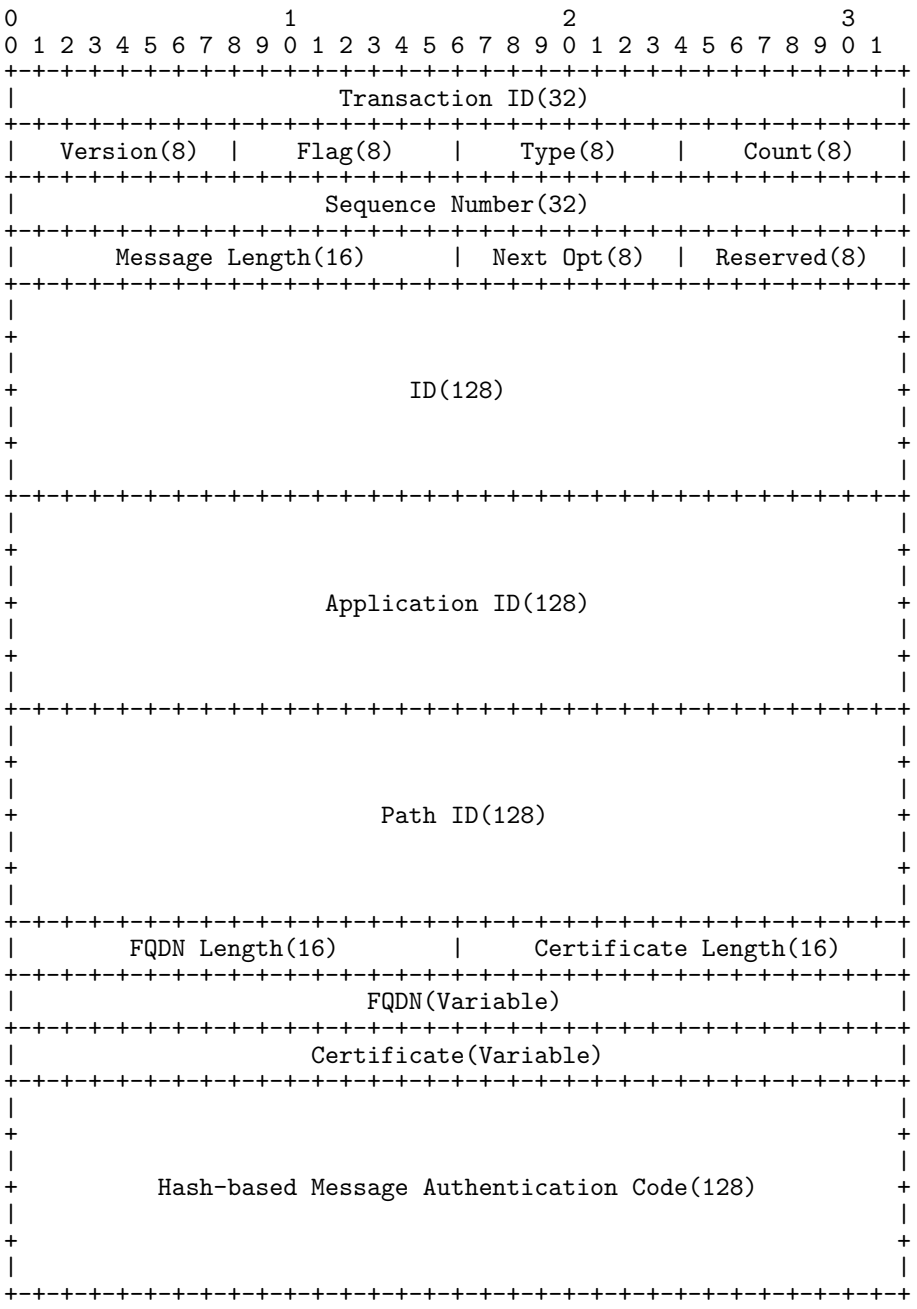


図 A.9: Structure of Direction Request

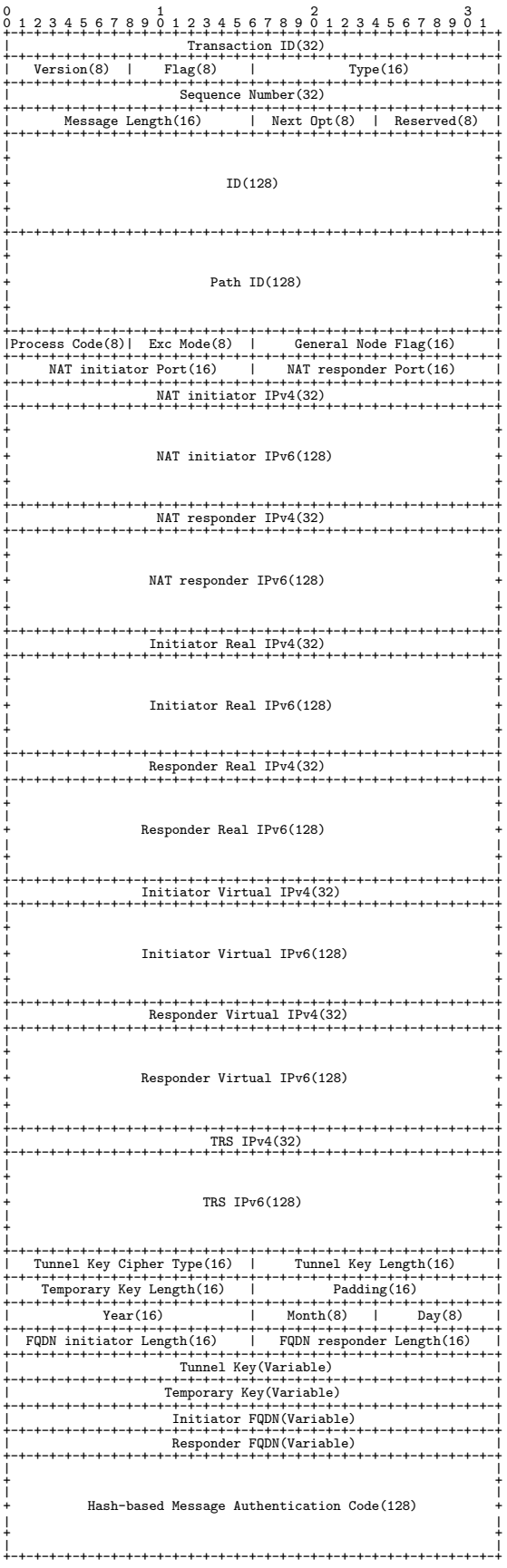


図 A.10: Structure of Route Direction

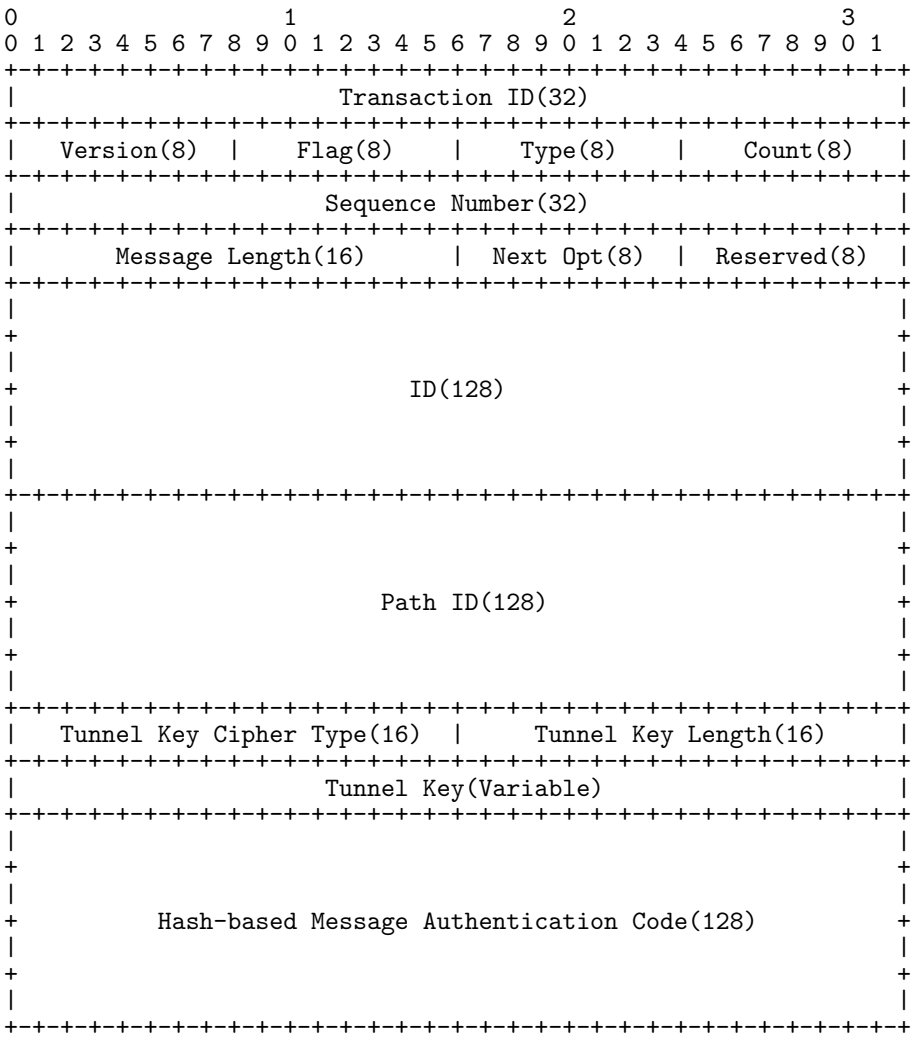


図 A.11: Structure of Relay Request

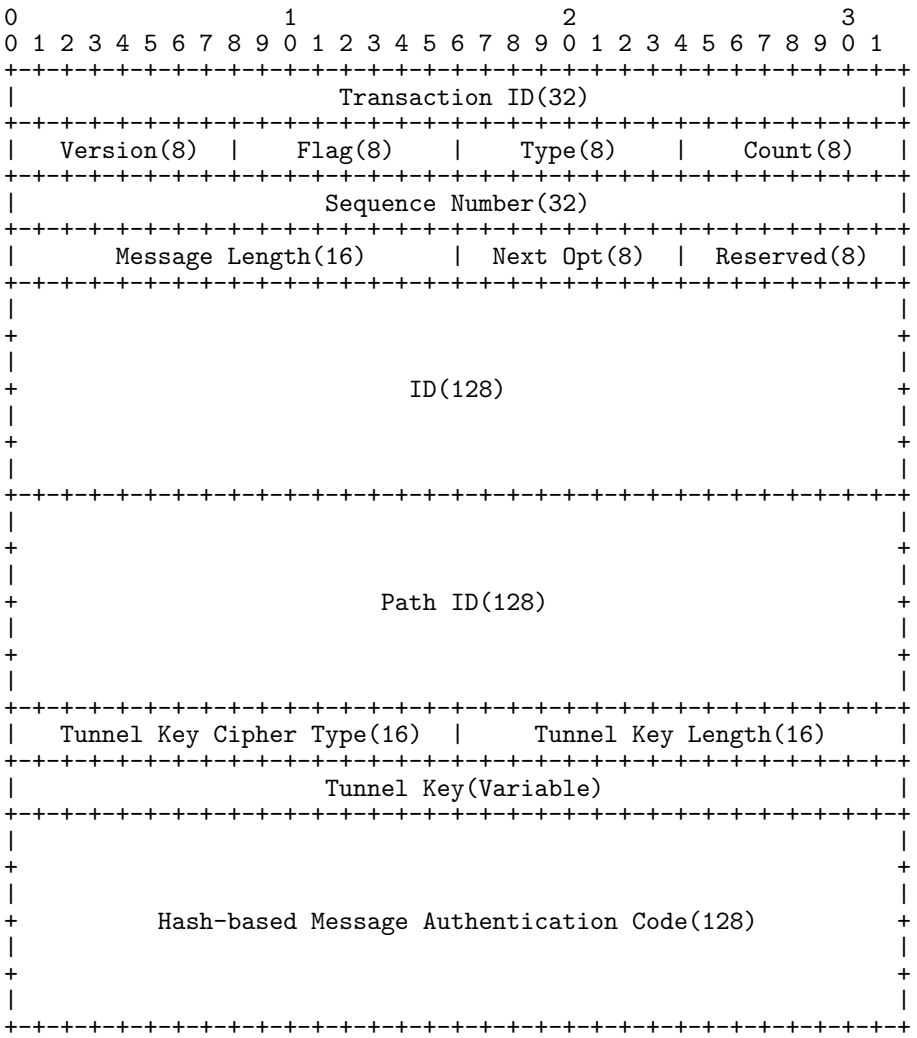


図 A.12: Structure of Relay Response

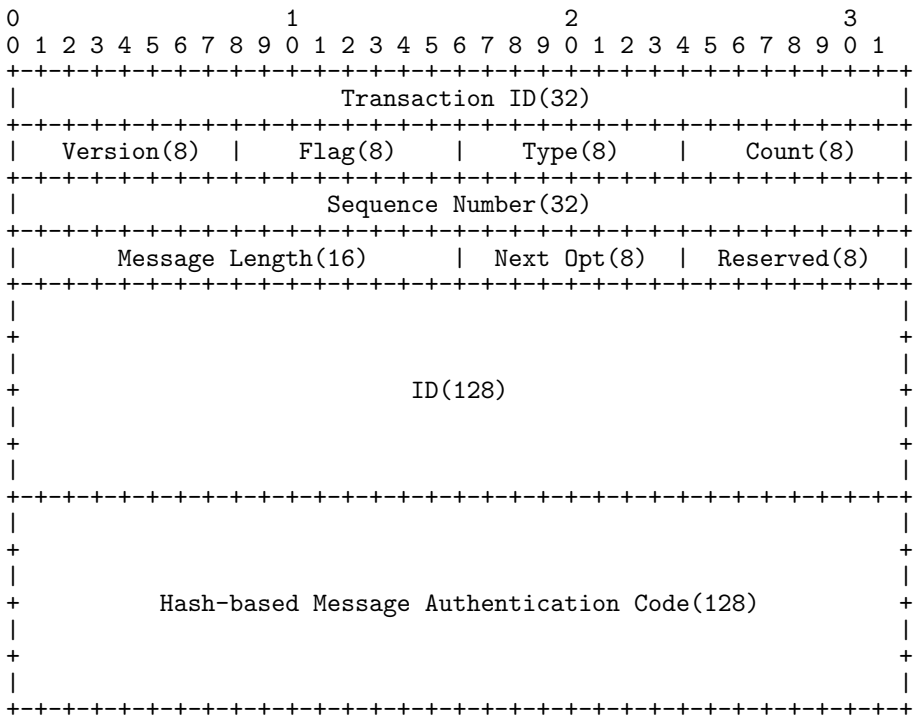


図 A.13: Structure of Hole Punching

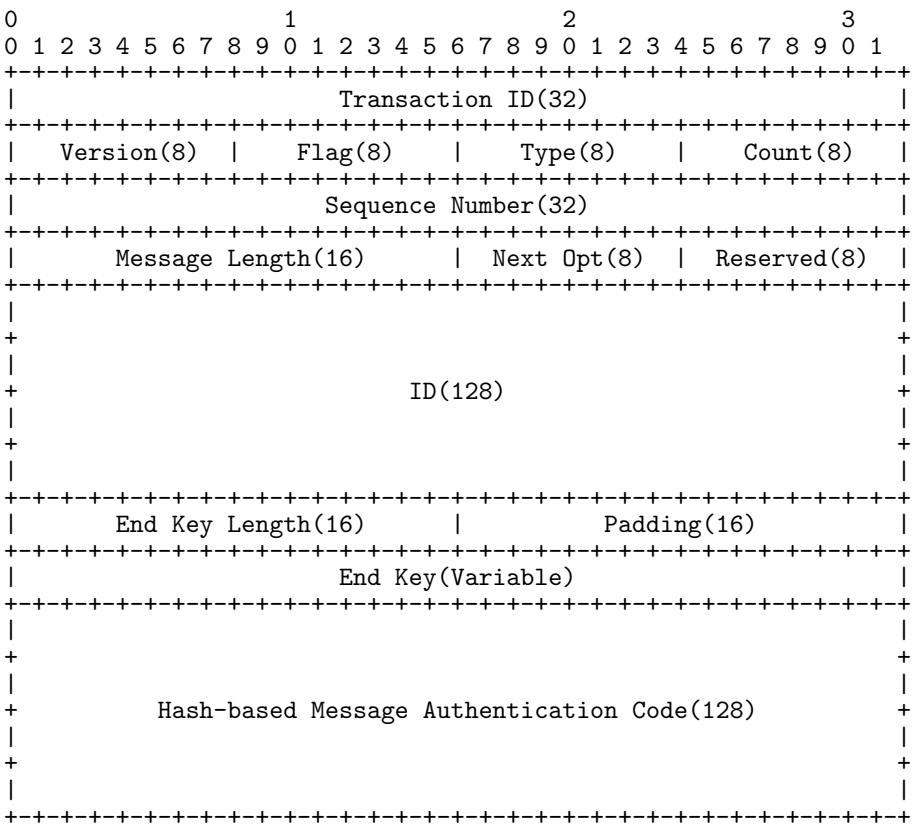


図 A.14: Structure of Tunnel Request

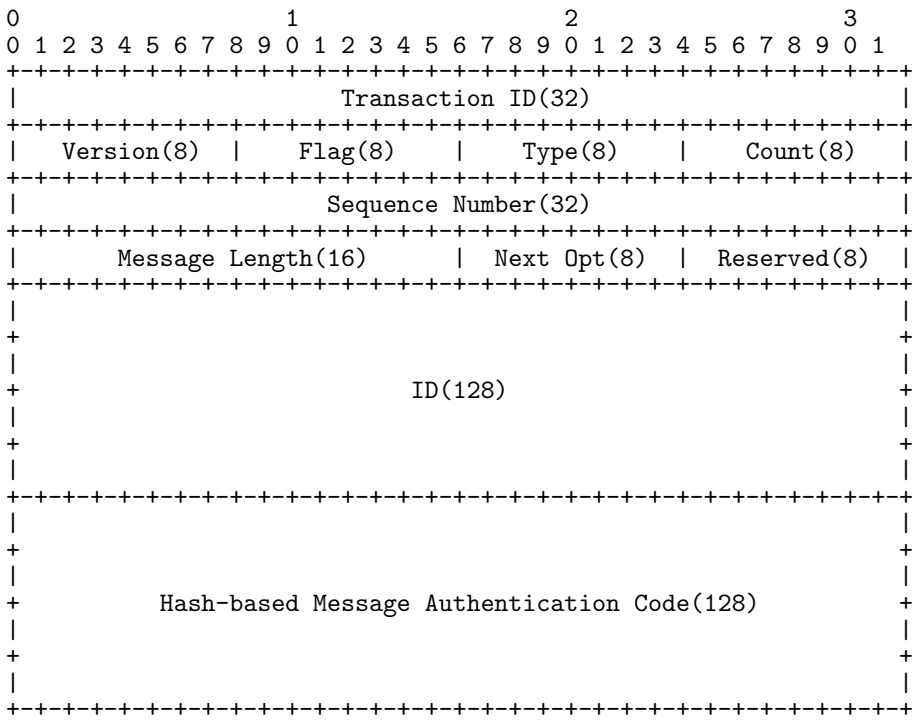


図 A.15: Structure of Tunnel Response

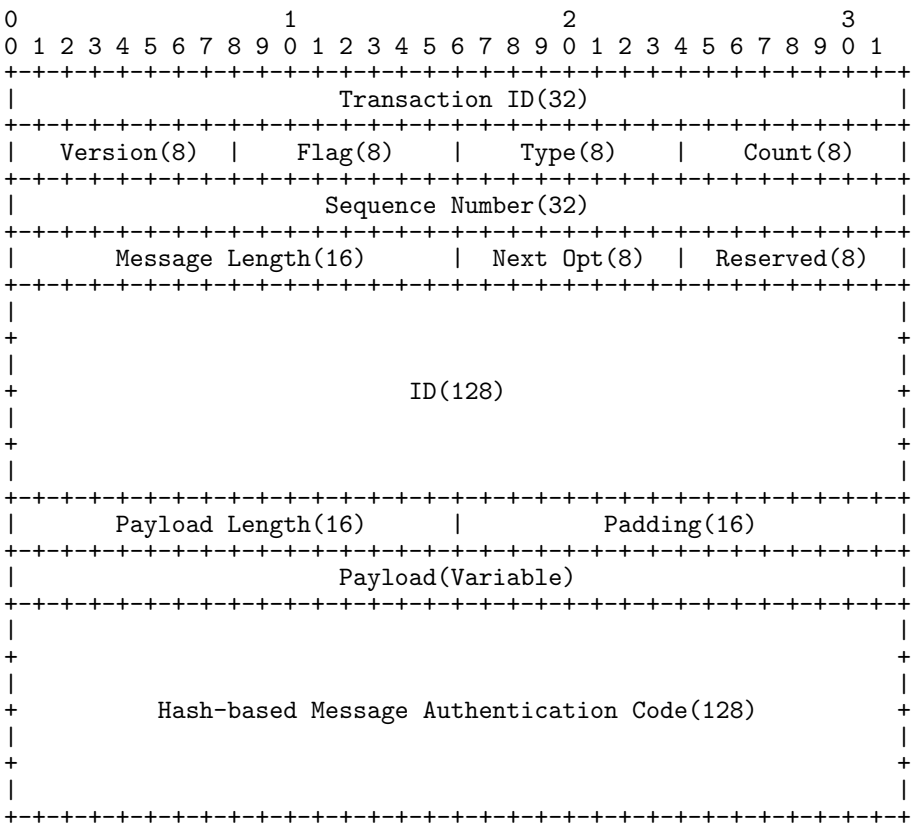


図 A.16: Capsule Message

B　NAPT 環境における経路最適化パターン

表 B.1: Initiator : NAPT 1 台, Responder : NAPT 2 台

Responder \ Initiator		Full Cone	Address-Restricted Cone	Port-Restricted Cone	Symmetric
NAPT (Outer)	NAPT (Inner)				
Full Cone (FC)	FC	○	○	○	○
	ARC	○	○	○	○
	PRC	○	○	○	×
	S	○	△	×	×
Address-Restricted Cone (ARC)	FC	○	○	○	○
	ARC	○	○	○	○
	PRC	○	○	○	×
	S	○	△	×	×
Port-Restricted Cone (PRC)	FC	○	○	○	×
	ARC	○	○	○	×
	PRC	○	○	○	×
	S	○	△	×	×
Symmetric (S)	FC	○	△	×	×
	ARC	○	△	×	×
	PRC	○	△	×	×
	S	○	△	×	×

表 B.2: Initiator : NAPT 2 台, Responder : NAPT 1 台

Initiator \ Responder		Full Cone	Address-Restricted Cone	Port-Restricted Cone	Symmetric
NAPT (Outer)	NAPT (Inner)				
Full Cone (FC)	FC	○	○	○	○
	ARC	○	○	○	△
	PRC	○	○	○	×
	S	○	○	×	×
Address-Restricted Cone (ARC)	FC	○	○	○	△
	ARC	○	○	○	△
	PRC	○	○	○	×
	S	○	○	×	×
Port-Restricted Cone (PRC)	FC	○	○	○	×
	ARC	○	○	○	×
	PRC	○	○	○	×
	S	○	○	×	×
Symmetric (S)	FC	○	○	×	×
	ARC	○	○	×	×
	PRC	○	○	×	×
	S	○	○	×	×

表 B.3: 両端末 : NAPT 2 台 (Responder の外側 NAPT : Full Cone, Address-Restricted Cone)

Responder / Initiator		NAPT (Outer)	Full Cone				Address-Restricted Cone			
		NAPT (Inner)	FC	ARC	PRC	S	FC	ARC	PRC	S
NAPT (Outer)	NAPT (Inner)									
Full Cone (FC)	FC		○	○	○	○	○	○	○	○
	ARC		○	○	○	△	○	○	○	△
	PRC		○	○	○	×	○	○	○	×
	S		○	○	×	×	○	○	×	×
Address-Restricted Cone (ARC)	FC		○	○	○	△	○	○	○	△
	ARC		○	○	○	△	○	○	○	△
	PRC		○	○	○	×	○	○	○	×
	S		○	○	×	×	○	○	×	×
Port-Restricted Cone (PRC)	FC		○	○	○	×	○	○	○	×
	ARC		○	○	○	×	○	○	○	×
	PRC		○	○	○	×	○	○	○	×
	S		○	○	×	×	○	○	×	×
Symmetric (S)	FC		○	○	×	×	○	○	×	×
	ARC		○	○	×	×	○	○	×	×
	PRC		○	○	×	×	○	○	×	×
	S		○	○	×	×	○	○	×	×

表 B.4: 両端末 : NAPT 2 台 (Responder の外側 NAPT : Port-Restricted Cone, Symmetric)

Responder / Initiator		NAPT (Outer)	Port-Restricted Cone				Symmetric			
		NAPT (Inner)	FC	ARC	PRC	S	FC	ARC	PRC	S
NAPT (Outer)	NAPT (Inner)									
Full Cone (FC)	FC		○	○	○	○	○	○	○	○
	ARC		○	○	○	△	△	△	△	△
	PRC		○	○	○	×	×	×	×	×
	S		×	×	×	×	×	×	×	×
Address-Restricted Cone (ARC)	FC		○	○	○	△	△	△	△	△
	ARC		○	○	○	△	△	△	△	△
	PRC		○	○	○	×	×	×	×	×
	S		×	×	×	×	×	×	×	×
Port-Restricted Cone (PRC)	FC		○	○	○	×	×	×	×	×
	ARC		○	○	○	×	×	×	×	×
	PRC		○	○	○	×	×	×	×	×
	S		×	×	×	×	×	×	×	×
Symmetric (S)	FC		×	×	×	×	×	×	×	×
	ARC		×	×	×	×	×	×	×	×
	PRC		×	×	×	×	×	×	×	×
	S		×	×	×	×	×	×	×	×

表 B.5: 両端末：同一 NAPT 1 台

Full Cone	Address-Restricted Cone	Port-Restricted Cone	Symmetric
○	○	○	○

表 B.6: 両端末：同一 NAPT 2 台

NAPT (Inner) / NAPT (Outer)	Full Cone	Address-Restricted Cone	Port-Restricted Cone	Symmetric
Full Cone	○	○	○	○
Address-Restricted Cone	○	○	○	○
Port-Restricted Cone	○	○	○	○
Symmetric	○	○	○	○

謝辞

本論文は, 筆者が愛知工業大学大学院 経営情報科学研究科 経営情報科学専攻 博士前期課程に在籍中の研究成果をまとめたものである. 本研究を進めるにあたり, 愛知工業大学 情報科学部 情報科学科 内藤克浩教授には多大な時間をかけて終始ご指導をいただいた. ここに, 感謝の意を表する. また, 菱田隆彰教授, 並びに, 伊藤暢浩教授には副査としてご助言をいただくとともに, 本論文の細部にわたりご指導をいただいた. ここに感謝の意を表する. 最後に, 日常の議論を通じ, 多くの知識や示唆を頂いた内藤研究室の皆様に感謝の意を表する.

参考文献

[1] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman, "Identifying diverse usage behaviors of smartphone apps," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. [Online]. Available: https://doi.org/10.1145/2068816.2068847 New York, NY, USA, Association for Computing Machinery, p. 329–344, November 2011.

[2] "Statista, Global Digital Population as of January 2020." [Online]. Available: https://www.statista.com/statistics/512650/worldwide-connected-devices-amount/ June 2023.

[3] T. Nishio, R. Shinkuma, T. Takahashi, and N. B. Mandayam, "Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud," in *Proceedings of the First International Workshop on Mobile Cloud Computing and Networking*, ser. MobileCloud '13. [Online]. Available: https://doi.org/10.1145/2492348.2492354 New York, NY, USA, Association for Computing Machinery, p. 19–26, July 2013.

[4] R. Kolcun, D. Boyle, and J. A. McCann, "Efficient In-Network Processing for a Hardware-Heterogeneous IoT," in *Proceedings of the 6th International Conference on the Internet of Things*, ser. IoT'16. [Online]. Available: https://doi.org/10.1145/2991561.2991568 New York, NY, USA, Association for Computing Machinery, p. 93–101, November 2016.

[5] M. J. Beevi, "A fair survey on Internet of Things (IoT)," in *2016 International Conference on Emerging Trends in Engineering, Technology and Science (ICETETS)*, pp. 1–6, February 2016.

[6] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 4, pp. 2347–2376, June 2015.

[7] A. Recse, R. Szabo, and B. Nemeth, "Elastic resource management and network slicing for IoT over edge clouds," in *Proceedings of the 10th International Conference on the Internet of Things*, ser. IoT '20. [Online]. Available: https://doi.org/10.1145/3410992.3411015 New York, NY, USA, Association for Computing Machinery, October 2020.

[8] "Edge computing and deployment strategies for communication service providers." [Online]. Available: https://cloud.report/whitepapers/edge-computing-and-deployment-strategies-for-communication-service-providers February 2020.

[9] M. Vögler, J. M. Schleicher, C. Inzinger, and S. Dustdar, "A Scalable Framework for Provisioning Large-Scale IoT Deployments," *ACM Trans. Internet Technol.*, vol. 16, no. 2. [Online]. Available: https://doi.org/10.1145/2850416 March 2016.

[10] H. Hejazi, H. Rajab, T. Cinkler, and L. Lengyel, "Survey of platforms for massive IoT," in *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*, pp. 1–8, January 2018.

[11] M. Picone, M. Mamei, and F. Zambonelli, “A Flexible and Modular Architecture for Edge Digital Twin: Implementation and Evaluation,” *ACM Trans. Internet Things*, vol. 4, no. 1. [Online]. Available: https://doi.org/10.1145/3573206 February 2023.

[12] L. F. Rivera, H. A. Müller, N. M. Villegas, G. Tamura, and M. Jiménez, “On the Engineering of IoT-Intensive Digital Twin Software Systems,” in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ser. ICSEW’20. [Online]. Available: https://doi.org/10.1145/3387940.3392195 New York, NY, USA, Association for Computing Machinery, p. 631–638, June 2020.

[13] S. Mittal, A. Tolk, A. Pyles, N. V. Balen, and K. Bergollo, “Digital Twin Modeling, Co-Simulation and Cyber Use-Case Inclusion Methodology for IOT Systems,” in *2019 Winter Simulation Conference (WSC)*, pp. 2653–2664, December 2019.

[14] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC ’12. [Online]. Available: https://doi.org/10.1145/2342509.2342513 New York, NY, USA, Association for Computing Machinery, p. 13–16, August 2012.

[15] E. A. Lee, “Cyber Physical Systems: Design Challenges,” in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pp. 363–369, May 2008.

[16] S. K. Khaitan and J. D. McCalley, “Design Techniques and Applications of Cyberphysical Systems: A Survey,” *IEEE Systems Journal*, vol. 9, no. 2, pp. 350–365, June 2015.

[17] E. Bertino, K.-K. R. Choo, D. Georgakopolous, and S. Nepal, “Internet of Things (IoT): Smart and Secure Service Delivery,” *ACM Trans. Internet Technol.*, vol. 16, no. 4. [Online]. Available: https://doi.org/10.1145/3013520 December 2016.

[18] S. D. Gupta and S. Ghanavati, “Towards a heterogeneous IoT privacy architecture,” in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ser. SAC ’20. [Online]. Available: https://doi.org/10.1145/3341105.3374108 New York, NY, USA, Association for Computing Machinery, p. 770–772, March 2020.

[19] S. Duangsuwan, A. Chusongsang, C. Teekanakvisit, and S. Promwong, “BER Detection of A2G Wireless Communication in Rician K-Factor Fading Channel for Massive IoT Connectivity Network,” in *2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, pp. 1–2, December 2019.

[20] Z. Guo, K. Zhang, H. Xin, M. Bi, H. He, and W. Hu, “An optical access network framework for smart factory in the industry 4.0 era supporting massive machine connections,” in *2017 16th International Conference on Optical Communications and Networks (ICOCN)*, pp. 1–3, August 2017.

[21] H. Habibzadeh, C. Kaptan, T. Soyata, B. Kantarci, and A. Boukerche, “Smart City System Design: A Comprehensive Study of the Application and Data Planes,” *ACM Comput. Surv.*, vol. 52, no. 2. [Online]. Available: https://doi.org/10.1145/3309545 May 2019.

[22] Y. H. Hwang, “IoT Security & Privacy: Threats and Challenges,” in *Proceedings of the 1st ACM Workshop on IoT Privacy, Trust, and Security*, ser. IoTPTS ’15. [Online]. Available: https://doi.org/10.1145/2732209.2732216 New York, NY, USA, Association for Computing Machinery, p. 1, April 2015.

[23] M. Papoutsakis, K. Kritikos, K. Magoutis, and S. Ioannidis, “Towards Model-Driven Application Security across Clouds,” April 2018.

[24] C. H. Chua and S. C. Ng, “Open-Source VPN Software: Performance Comparison for Remote Access,” in *Proceedings of the 5th International Conference on Information Science and Systems*, ser. ICISS ’22. [Online]. Available: https://doi.org/10.1145/3561877.3561882 New York, NY, USA, Association for Computing Machinery, p. 29–34, August 2022.

[25] “Ericsson Mobility Report.” [Online]. Available: https://www.ericsson.com/4ae12c/assets/local/reports-papers/mobility-report/documents/2023/ericsson-mobility-report-november-2023.pdf November 2023.

[26] “Cisco Annual Internet Report (2018–2023) White Paper.” [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf September 2020.

[27] A. Aktypi, K. Kalkan, and K. B. Rasmussen, “SeCaS: Secure Capability Sharing Framework for IoT Devices in a Structured P2P Network,” in *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY ’20. [Online]. Available: https://doi.org/10.1145/3374664.3375739 New York, NY, USA, Association for Computing Machinery, p. 271–282, March 2020.

[28] D. R. Vasconcelos, R. M. C. Andrade, V. Severino, and J. N. D. Souza, “Cloud, Fog, or Mist in IoT? That Is the Question,” *ACM Trans. Internet Technol.*, vol. 19, no. 2. [Online]. Available: https://doi.org/10.1145/3309709 March 2019.

[29] L. Q. Tho and H. Q. Trung, “P2P shared-caching model: using P2P to improve client-server application performance,” in *Proceedings of the 4th Symposium on Information and Communication Technology*, ser. SoICT ’13. [Online]. Available: https://doi.org/10.1145/2542050.2542090 New York, NY, USA, Association for Computing Machinery, p. 222–226, December 2013.

[30] A. Rabay’a, E. Schleicher, and K. Graffi, “Fog Computing with P2P: Enhancing Fog Computing Bandwidth for IoT Scenarios,” in *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 82–89, July 2019.

[31] H.-J. Hong, “From Cloud Computing to Fog Computing: Unleash the Power of Edge and End Devices,” in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 331–334, December 2017.

[32] J. E. Bailes and G. F. Templeton, “Managing P2P security,” *Commun. ACM*, vol. 47, no. 9, p. 95–98. [Online]. Available: https://doi.org/10.1145/1015864.1015894 September 2004.

[33] I. Ahmed, T. Nahar, S. S. Urmi, and K. A. Taher, “Protection of Sensitive Data in Zero Trust Model,” in *Proceedings of the International Conference on Computing Advancements*, ser.

ICCA 2020. [Online]. Available: https://doi.org/10.1145/3377049.3377114 New York, NY, USA, Association for Computing Machinery, January 2020.

[34] L. Nishad, A. Gupta, J. Paliwal, R. Pandey, S. Beniwal, and S. Kumar, "Security, Privacy Issues and challenges In Cloud Computing: A Survey," pp. 1–7, March 2016.

[35] M. Alhawamdeh and R. Tahboub, "Enabling Security as a Service for IoT Emerging Technologies: A Survey," in *The 7th Annual International Conference on Arab Women in Computing in Conjunction with the 2nd Forum of Women in Research*, ser. ArabWIC 2021. [Online]. Available: https://doi.org/10.1145/3485557.3485582 New York, NY, USA, Association for Computing Machinery, August 2021.

[36] S. Li, "Editorial: Zero Trust based Internet of Things," *EAI Endorsed Transactions on Internet of Things*, vol. 5, p. 165168, June 2020.

[37] S. Hameed, F. I. Khan, B. Hameed, and D. F. H. Sadok, "Understanding Security Requirements and Challenges in Internet of Things (IoT): A Review," *J. Comput. Netw. Commun.*, vol. 2019. [Online]. Available: https://doi.org/10.1155/2019/9629381 January 2019.

[38] S. Sicari, A. Rizzardi, L. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead," *Computer Networks*, vol. 76, pp. 146–164. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128614003971 January 2015.

[39] J. Postel, "Internet Protocol," no. 791. [Online]. Available: https://www.rfc-editor.org/info/rfc791 September 1981.

[40] J.-J. Lin, K.-C. Wang, S.-M. Cheng, and Y.-C. Liu, "On exploiting SDN to facilitate IPv4/IPv6 coexistence and transition," in *2017 IEEE Conference on Dependable and Secure Computing*, vol. August, pp. 473–474, August 2017.

[41] L. Prehn, F. Lichtblau, and A. Feldmann, "When wells run dry: the 2020 IPv4 address market," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '20. [Online]. Available: https://doi.org/10.1145/3386367.3431301 New York, NY, USA, Association for Computing Machinery, p. 46–54, November 2020.

[42] T. L. Hain, "Architectural Implications of NAT," no. 2993. [Online]. Available: https://www.rfc-editor.org/info/rfc2993 November 2000.

[43] K. B. Egevang and P. Srisuresh, "Traditional IP Network Address Translator (Traditional NAT)," no. 3022. [Online]. Available: https://rfc-editor.org/rfc/rfc3022.txt January 2001.

[44] C.-L. Huang and S.-H. Hwang, "The Asymmetric NAT and Its Traversal Method," in *Proceedings of the Sixth International Conference on Wireless and Optical Communications Networks*, ser. WOCN'09, IEEE Press, pp. 177–180, April 2009.

[45] H.-C. Wang, C. Chen, and S.-H. Lu, "An SDN-based NAT Traversal Mechanism for End-to-end IoT Networking," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 1–4, September 2019.

[46] D. Seah, W. K. Leong, Q. Yang, B. Leong, and A. Razeen, "Peer NAT proxies for peer-to-peer games," in *Proceedings of the 8th Annual Workshop on Network and Systems Support for Games*, ser. NetGames '09, IEEE Press, November 2009.

[47] B. Hinden and D. S. E. Deering, “Internet Protocol, Version 6 (IPv6) Specification,” no. 2460. [Online]. Available: https://www.rfc-editor.org/info/rfc2460 December 1998.

[48] V. K. K. Terli, S. P. Chaganti, N. B. Alla, S. Sarab, and T. El Taeib, “Software implementation of IPv4 to IPv6 migration,” in *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pp. 1–6, April 2016.

[49] L. Yang and K. Lei, “Combining ICE and SIP protocol for NAT traversal in new classification standard,” in *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*, pp. 576–580, October 2016.

[50] F. Huang, L. Yu, T. Shen, and S. Hu, “The P2P Solution Research and Design Based on NAT Traversing Technology,” in *2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pp. 1347–1351, October 2019.

[51] Y. Mikamo, K. Asahi, H. Suzuki, and A. Watanabe, “Proposal for an ad-hoc routing protocol considering traffic conditions and evaluation of UDP using a redundant route,” in *2014 Seventh International Conference on Mobile Computing and Ubiquitous Networking (ICMU)*, pp. 72–73, January 2014.

[52] Q. Zhang, C. Guo, Z. Guo, and W. Zhu, “Efficient Mobility Management for Vertical Handoff between WWAN and WLAN,” *Communications Magazine, IEEE*, vol. 41, pp. 102–108, December 2003.

[53] P. Singh, “Enhancement of handover decision in heterogeneous networks,” in *2017 International Conference on Signal Processing and Communication (ICSPC)*, pp. 436–441, July 2017.

[54] R. Karmakar, S. Chattopadhyay, and S. Chakraborty, “Impact of IEEE 802.11n/ac PHY/MAC High Throughput Enhancements on Transport and Application Protocols—A Survey,” *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2050–2091, August 2017.

[55] C. E. Perkins, “IP Mobility Support for IPv4, Revised,” no. 5944. [Online]. Available: https://rfc-editor.org/rfc/rfc5944.txt November 2010.

[56] G. Montenegro, “Reverse Tunneling for Mobile IP, revised,” no. 3024. [Online]. Available: https://rfc-editor.org/rfc/rfc3024.txt January 2001.

[57] C. E. Perkins and D. B. Johnson, “Route Optimization in Mobile IP,” work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-mobileip-optim-11 September 2001.

[58] K. Naito, K. Tanaka, and K. Tanaka, “CYPHONIC: Overlay network technology for Cyber Physical communication,” in *The 10th International Multi-Conference on Complexity, Informatics and Cybernetics: IMCIC 2019*, International Institute of Informatics and Cybernetics, pp. 1–6, March 2019.

[59] S. Isomura, T. Yoshikawa, H. Komura, S. Kubota, C. Nishiwaki, and K. Naito, “Prototyping evaluation of CYPHONIC: Overlay network technology for Cyber-Physical communication,” in *2021 IEEE 18th Consumer Communications and Networking Conference (CCNC)*. [Online]. Available: https://doi.org/10.1109/CCNC49032.2021.9369500 IEEE Press, p. 1–2, January 2021.

[60] T. Yoshikawa, H. Komura, C. Nishiwaki, R. Goto, K. Matama, and K. Naito, "Evaluation of new cyphonic: Overlay network protocol based on go language," in *2022 IEEE 40th International Conference on Consumer Electronics (ICCE)*. [Online]. Available: https://doi.org/10.1109/ICCE53296.2022.9730323 IEEE Press, pp. 1–6, January 2022.

[61] S. Horisaki, K. Matama, K. Naito, and H. Suzuki, "A Proposal of QUIC-based CYPHONIC for Encrypted End-to-End Communications," in *2022 Tenth International Symposium on Computing and Networking (CANDAR)*. [Online]. Available: https://doi.org/10.1109/CANDAR57322.2022.00012 IEEE Press, pp. 27–35, November 2022.

[62] R. Goto, K. Matama, R. Aihata, S. Horisaki, H. Suzuki, and K. Naito, "Implementation and Evaluation of CYPHONIC client focusing on Sequencing mechanisms and Concurrency for packet processing," in *2023 IEEE 12th Global Conference on Consumer Electronics (GCCE)*. [Online]. Available: https://doi.org/10.1109/GCCE59613.2023.10315593 IEEE Press, pp. 997–1001, October 2023.

[63] A. Musaddiq, Y. Zikria, O. Hahm, H. Yu, A. Bashir, and S. W. Kim, "A Survey on Resource Management in IoT Operating Systems," *IEEE Access*, vol. PP, pp. 1–1, February 2018.

[64] R. Goto, T. Yoshikawa, H. Komura, K. Matama, N. Chihiro, and K. Naito, "Design and Basic Evaluation of Virtual IPv4-based CYPHONIC adapter," *Journal of Systemics, Cybernetics and Informatics (JSCI)*, vol. 20, no. 3, pp. 55–63, June 2022.

[65] R. Goto, K. Matama, C. Nishiwaki, and K. Naito, "Design of new CYPHONIC adapter focused on packet sequential processing scheme," *IEICE Communications Express (ComEX)*, vol. 12, no. 5, pp. 194–199, May 2023.

[66] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, June 2016.

[67] J. Portilla, G. Mujica, J.-S. Lee, and T. Riesgo, "The Extreme Edge at the Bottom of the Internet of Things: A Review," *IEEE Sensors Journal*, vol. 19, no. 9, pp. 3179–3190, January 2019.

[68] N. Pavlopoulou and E. Curry, "PoSSUM: An Entity-centric Publish/Subscribe System for Diverse Summarization in Internet of Things," *ACM Trans. Internet Technol.*, vol. 22, no. 3. [Online]. Available: https://doi.org/10.1145/3507911 March 2022.

[69] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "Maglev: A Fast and Reliable Software Network Load Balancer," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. [Online]. Available: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eisenbud Santa Clara, CA, USENIX Association, pp. 523–535, March 2016.

[70] F. Chahlaoui, M. R. El-Fenni, and H. Dahmouni, "Performance Analysis of Load Balancing Mechanisms in SDN Networks," in *Proceedings of the 2nd International Conference on Networking, Information Systems and Security*, ser. NISS '19. [Online]. Available: https://doi.org/10.1145/3320326.3320368 New York, NY, USA, Association for Computing Machinery, March 2019.

[71] J.-B. Lee, T.-H. Yoo, E.-H. Lee, B.-H. Hwang, S.-W. Ahn, and C.-H. Cho, “High-Performance Software Load Balancer for Cloud-Native Architecture,” *IEEE Access*, vol. 9, pp. 123 704–123 716, August 2021.

[72] R. Yang and M. Kogias, “HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme,” in *Proceedings of the 1st Workshop on EBPF and Kernel Extensions*, ser. eBPF ’23. [Online]. Available: https://doi.org/10.1145/3609021.3609307 New York, NY, USA, Association for Computing Machinery, p. 77–83, September 2023.

[73] M. Bishop, “HTTP/3,” no. 9114. [Online]. Available: https://www.rfc-editor.org/info/rfc9114 June 2022.

[74] “Usage statistics of HTTP/3 for websites.” [Online]. Available: https://w3techs.com/technologies/details/ce-http3 February 2024.

[75] “2019 Docker Usage Report.” [Online]. Available: https://sysdig.com/blog/sysdig-2019-container-usage-report/ October 2019.

[76] “Containers At Scale: At Google, the Google Cloud Platform and Beyond.” [Online]. Available: https://speakerdeck.com/jbeda/containers-at-scale May 2014.

[77] B. Ford, P. Srisuresh, and D. Kegel, “Peer-to-Peer Communication across Network Address Translators,” in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC ’05, USA, USENIX Association, p. 13, April 2005.

[78] K. Matama, H. Komura, R. Goto, H. Suzuki, and K. Naito, “Enhancing Route Optimization for NAPT Traversal in CYPHONIC: Design and Implementation,” in *2023 IEEE 12th Global Conference on Consumer Electronics (GCCE)*. [Online]. Available: https://doi.org/10.1109/GCCE59613.2023.10315375 IEEE Press, pp. 990–994, October 2023.

[79] D. Siemon, “Queueing in the Linux Network Stack,” *Linux Journal*, vol. 2013, no. 231. [Online]. Available: https://doi.org/10.5555/2509948.2509950 July 2013.

研究業績

- Ren Goto, Kazushige Matama, Ryouta Aihata, Michiyo Suda, Hidekazu Suzuki, and Katsuhiro Naito, "CYPHONIC Adapter: Enabling Secure P2P Connectivity for Diverse IoT Devices," *2024 IEEE 21st Consumer Communications & Networking Conference (CCNC)*, January 2024.
- Shota Horisaki, Kazushige Matama, Ren Goto, Katsuhiro Naito, and Hidekazu Suzuki, "Intercommunication Between Local Edge Computing Devices Using QUIC-Based CYPHONIC," *2024 IEEE 42nd International Conference on Consumer Electronics (ICCE)*, January 2024.
- Ren Goto, Kazushige Matama, Ryouta Aihata, Shota Horisaki, Hidekazu Suzuki, and Katsuhiro Naito, "Implementation and Evaluation of CYPHONIC client focusing on Sequencing mechanisms and Concurrency for packet processing," *2023 IEEE 12th Global Conference on Consumer Electronics (GCCE)*, October 2023.
- Guilherme Seidy Ujiie, Ren Goto, Kazushige Matama, Yoshiya Suzuki, Hidekazu Suzuki, and Katsuhiro Naito, "Proposal of CYPHONIC end-device functions on Windows OS," *2023 IEEE 12th Global Conference on Consumer Electronics (GCCE)*, October 2023.
- Kazushige Matama, Hijiri Komura, Ren Goto, Hidekazu Suzuki, and Katsuhiro Naito, "Enhancing Route Optimization for NAPT Traversal in CYPHONIC: Design and Implementation," *2023 IEEE 12th Global Conference on Consumer Electronics (GCCE)*, October 2023.
- Ren Goto, Kazushige Matama, Chihiro Nishiwaki, and Katsuhiro Naito, "Design of new CYPHONIC adapter focused on packet sequential processing scheme," *2023 IEICE Communications Express (ComEX)*, May 2023.
- Kazushige Matama, Ren Goto, Chihiro Nishiwaki, and Katsuhiro Naito, "Extension mechanism of overlay network protocol to support digital authenticates," *2023 IIIC Journal of Systemics, Cybernetics and Informatics (JSCI)*, February 2023.
- Ren Goto, Kazushige Matama, Chihiro Nishiwaki, and Katsuhiro Naito, "Proposal of an extended CYPHONIC adapter supporting general nodes using virtual IPv6 addresses," *2022 IEEE 11th Global Conference on Consumer Electronics (GCCE)*, October 2022.
- Kazushige Matama, Ren Goto, Chihiro Nishiwaki, and Katsuhiro Naito, "Extension mechanism of overlay network protocol to support digital authenticates," *2022 IIIS 26th World Multi-conference on Systemics, Cybernetics and Informatics (WMSCI)*, July 2022.
- Ren Goto, Taiki Yoshikawa, Hijiri Komura, Kazushige Matama, Chihiro Nishiwaki, and Katsuhiro Naito, "Design and Basic Evaluation of Virtual IPv4-based CYPHONIC adapter," *2022 IIIC Journal of Systemics, Cybernetics and Informatics (JSCI)*, June 2022.
- Ren Goto, Taiki Yoshikawa, Hijiri Komura, Kazushige Matama, Chihiro Nishiwaki, and Katsuhiro Naito, "Design and Basic Evaluation of Virtual IPv4-based CYPHONIC adapter," *2022 IIIS*

13th International Multi-conference on Complexity, Informatics and Cybernetics (IMCIC), March 2022.

- Taiki Yoshikawa, Hijiri Komura, Ren Goto, Kazushige Matama, Chihiro Nishiwaki, and Katsuhiro Naito, "Design and Basic Evaluation of Virtual IPv4-based CYPHONIC adapter," *2022 IEEE 19st Consumer Communications & Networking Conference (CCNC)*, January 2022.
- Taiki Yoshikawa, Hijiri Komura, Chihiro Nishiwaki, Ren Goto, Kazushige Matama, and Katsuhiro Naito, "Design and Basic Evaluation of Virtual IPv4-based CYPHONIC adapter," *2022 IEEE 40th International Conference on Consumer Electronics (ICCE)*, January 2022.
- 殿内 太一朗, 堀崎 翔太, 後藤 廉, 内藤 克浩, 吉田 勝好, 鈴木 秀和, "CYPHONIC を用いたローカル 5G 接続ネットワークカメラとの P2P 映像配信に関する検証," 情報処理学会 第 *86* 回全国大会, March 2024.
- 後藤 廉, 眞玉 和茂, 相畑 亮太, 鈴木 秀和, 内藤 克浩, "同時実行性及びパケット順序処理に着目した CYPHONIC クライアントの実装と評価," *2023* 年 電子情報通信学会 ソサイエティ大会 *(Society Conference)*, September 2023.
- ウジイエ ギレルメ セイジ, 後藤 廉, 眞玉 和茂, 鈴木 誉写, 内藤 克浩, 鈴木 秀和, "CYPHONIC の端末機能を WindowsOS で実現する基礎開発," *2023* 年 電子情報通信学会 センサネットワークとモバイルインテリジェンス研究専門委員会 *(SeMI)*, July 2023.
- 鈴木 誉写, 眞玉 和茂, 後藤 廉, ウジイエ ギレルメ セイジ, 内藤 克浩, 鈴木 秀和, "CYPHONIC クラウドにおける監視システムの設計と基礎評価," *2023* 年 電子情報通信学会 センサネットワークとモバイルインテリジェンス研究専門委員会 *(SeMI)*, July 2023.
- 眞玉 和茂, 小村 聖, 後藤 廉, 鈴木 秀和, 内藤 克浩, "CYPHONIC における NAPT 越え時の通信経路最適化に関する設計及び実装," 情報処理学会 第 *107* 回モバイルコンピューティングと新社会システム研究会 *(MBL)*, May 2023.
- 二村 洸輔, 堀崎 翔太, 後藤 廉, 内藤 克浩, 鈴木 秀和, "CYPHONIC サーバ群のオートスケーリングのためのメモリ管理手法に関する検討," 情報処理学会 第 *20* 回情報学ワークショップ *Workshop on Informatics (WiNF)*, December 2022.
- 後藤 廉, 吉川 大貴, 小村 聖, 眞玉 和茂, 内藤 克浩, "仮想 IPv4 アドレスを想定した CYPHONIC アダプタの設計と基礎評価," 情報処理学会 第 *33* 回 コンシューマー・デバイス&システム *(CDS)*, January 2022.
- 西脇 千紘, 吉川 大貴, 小村 聖, 後藤 廉, 眞玉 和茂, 内藤 克浩, "iOS におけるアプリケーション内独自プロトコル実装に関する一検討," 令和三年度 電気・電子・情報関係学会 東海支部連合大会, September 2021.
- 内藤 克浩, 後藤 廉, 眞玉 和茂, 鈴木 秀和, "通信アダプタ及び通信システム," 特願 2023-168182, September 2023.