ホスト OS のルーティングテーブルを用いたコンフィグファイルによる DHCP サーバコンテナの自動構築

森井 佑誠[1]　大野 有樹[2]　串田 高幸[1]

概要：コンテナを用いた仮想化は, 移植性の高いアプリケーションの構築を可能にする. しかし, 既存の DHCP サーバは, ユーザが手動で用意する必要があるコンフィグファイルによりコンテナの移植性の利点を得ることができない. 本研究ではこの問題に対し, ホスト OS のルーティングテーブルを用いてコンフィグファイルを作成し, DHCP サーバコンテナを自動構築することで移植性を確保する手法を提案する. 利用するルーティングテーブルは, ホスト OS に用意されているルーティングテーブルである /proc/net/route である. 基礎実験では, ISC DHCP を使いホストとコンテナでコンフィグファイルを変えて起動実験を行った. その結果, ホストで利用したコンフィグファイルはコンテナでは適用できないという結果を得ることができ, DHCP サーバコンテナの移植性の低さを示した.

1. はじめに

背景

仮想化技術の一つに, コンテナを用いた仮想化があげられる [1]. コンテナはコンテナイメージをもとに, コンテナランタイムが読み込むことで作成される. コンテナイメージやコンテナランタイムの仕様は Open Container Initiative (OCI) によって定められている. コンテナを用いることで, 従来のホスト型やハイパーバイザ型の仮想化に比べ, 容易にアプリケーションの実行環境を作成することができる. また, コンテナの廃棄, 再作成もこれらの仮想化環境と比べ容易である. 加えて, コンテナは軽量であるとともに, 仮想化によるオーバヘッドが少ない [2–4]. そのため, コンテナはクラウドだけでなく, IoT システムにも取り入れられている [5]. 図 1 に httpd コンテナイメージを使用したアプリケーション構築の例を示す. なお, httpd コンテナで実行されるアプリケーションは Apache HTTP Server のことである.

図 1 では, サーバの管理者が Ubuntu 環境上でコマンドを入力, 実行をすることで httpd コンテナを作成、実行するまでの流れを表している. Docker[*1] はコンテナイメージからコンテナを作成, 実行できる環境をもつソフトウェアである [6–9]. このようなソフトウェアを本研究では Runtime と表記する. Runtime は主に Linux カーネルの機能を利用しているため, Ubuntu のように, Runtime を実行するホスト OS は Linux カーネルを使用した OS を使用しなければならない. Docker は, イメージリポジトリの Docker Hub から Apache HTTP Server のコンテナイメージをダウンロードし, そのコンテナイメージをもとにコンテナを作成、実行を行う. Docker Hub[*2] はインターネット上に存在するコンテナイメージが保存されているリポジトリである. このようなリポジトリはイメージリポジトリと呼ばれる. このように, コンテナ作成と実行はイメージリポジトリと Runtime のみで可能であるため, 管理者はホスト OS を気にしないでコンテナの実行が可能である. 同時に, Runtime は Linux カーネルを使用したホスト OS

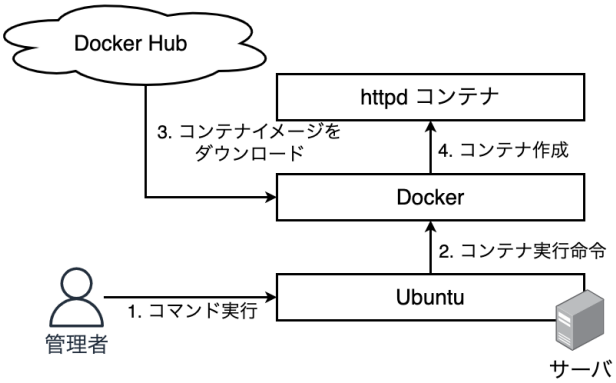

図 1　コンテナによるアプリケーション構築の例

[1] 東京工科大学コンピュータサイエンス学部
〒 192–0982 東京都八王子市片倉町 1404-1
[2] 東京工科大学院バイオ・情報メディア研究科コンピュータサイエンス専攻
〒 192–0982 東京都八王子市片倉町 1404-1

[*1] https://www.docker.com

[*2] https://hub.docker.com

であれば実行できるため, Ubuntu と Red Hat Enterprise Linux (RHEL) を例とした違うホスト OS であっても, イメージリポジトリと Runtime が存在すれば同じコンテナを何度も作成し実行することが可能である. このため, アプリケーションをコンテナとして用意することで, アプリケーションの移植性を高める利点を得ることができる. また, 比較のため, 図 2 にコンテナを用いない従来の Apache HTTP Server アプリケーション構築の例を示す.

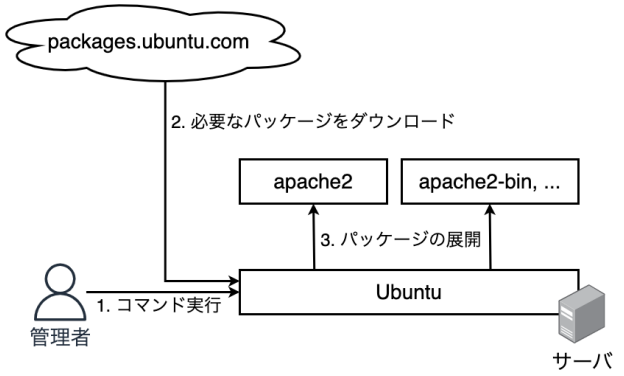

図 2 従来のアプリケーション構築の例

図 2 では, サーバの管理者が Ubuntu 環境上でコマンドを入力, 実行をする過程は図 1 における過程と同じである. 図 1 との違いは, ホスト OS である Ubuntu がアプリケーションの実行まで全てを担っていることにある. Ubuntu はパッケージリポジトリからアプリケーションの実行に必要なパッケージをダウンロードし, Ubuntu 上にパッケージを展開する. ダウンロードされるパッケージには Apache HTTP Server である apache2 の他に apache2 の実行に必要な apache2-bin を含めた複数の依存ソフトウェアも含まれている. Ubuntu のパッケージリポジトリは APT リポジトリとも呼ばれ, バージョンによって参照するリポジトリ URL やパッケージの内容は異なる. また, APT リポジトリ以外にも RPM リポジトリも存在し, こちらは RHEL や CentOS で使用される. このように, 違う環境で同じアプリケーションを構築する場合, 必ずしもアプリケーションは新しい環境と古い環境で同じ構成になるとは限らない. そのため, コンテナによるアプリケーションと比べ移植性は低くなる.

以下にコンテナネットワークについて説明をする. コンテナは通常 Runtime によって用意されたコンテナネットワークの中で実行される. 図 3 にホストマシンとコンテナネットワークの関係を示す.

図 3 において, ホストマシンは 192.168.10.0/24 のネットワークに所属しており, IP アドレスは 192.168.10.2 である. コンテナは Runtime によって作成された 172.17.0.0/16 で構成されたコンテナネットワーク内に配置されている. 図 3 中のコンテナの IP アドレスはそれぞれ, 172.17.0.3 と

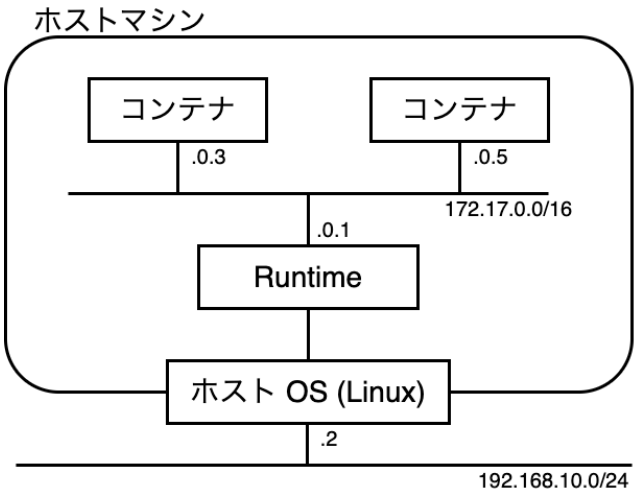

図 3 ホストマシンとコンテナネットワークの関係

172.17.0.5 であり, それらは Runtime から自動的に割り当てている. また, Runtime の IP アドレスは 172.17.0.1 でホスト OS やコンテナネットワーク内から通信可能である. コンテナ間の通信は Runtime を経由して通信を行っている. 加えて, ホスト OS からコンテナネットワーク内のコンテナへの通信も Runtime を経由して行われる. しかし, コンテナネットワーク内にいるコンテナは Runtime とホスト OS が繋がっているにも関わらず, ホスト OS への通信はできない. このように, コンテナネットワークは Runtime によってホストマシン内で独立している. これにより, コンテナはホスト OS との通信が制限され, コンテナの安全性を高めることを可能としている [10, 11].

以下に DHCP について説明をする. LAN 内の端末に IP アドレスを自動的に割り当てるプロトコルとして RFC2131 によって勧告されている Dynamic Host Configuration Protocol (DHCP) がある [12]. DHCP はクライアントサーバモデルを採用しており, 要求を行う DHCP クライアントと応答を返す DHCP サーバの二つが必要となる. DHCP クライアントと DHCP サーバは User Diagram Protocol (UDP) を使用して通信をしている. DHCP クライアントが DHCP サーバから IP アドレスを取得し, IP アドレスを開放するまでの通信を図 4 に示す.

図 4 はクライアントとサーバ間で以下の通信が行われている.

(1) クライアントは DHCPDISCOVER パケットをブロードキャストで送信する.
(2) クライアントからの DHCPDISCOVER パケットを受信したサーバはクライアントに, DHCPOFFER パケットを送信する.
(3) サーバからの DHCPOFFER パケットを受信したクライアントは DHCPREQUEST パケットをブロードキャストで送信する.
(4) クライアントからの DHCPDISCOVER パケットを受

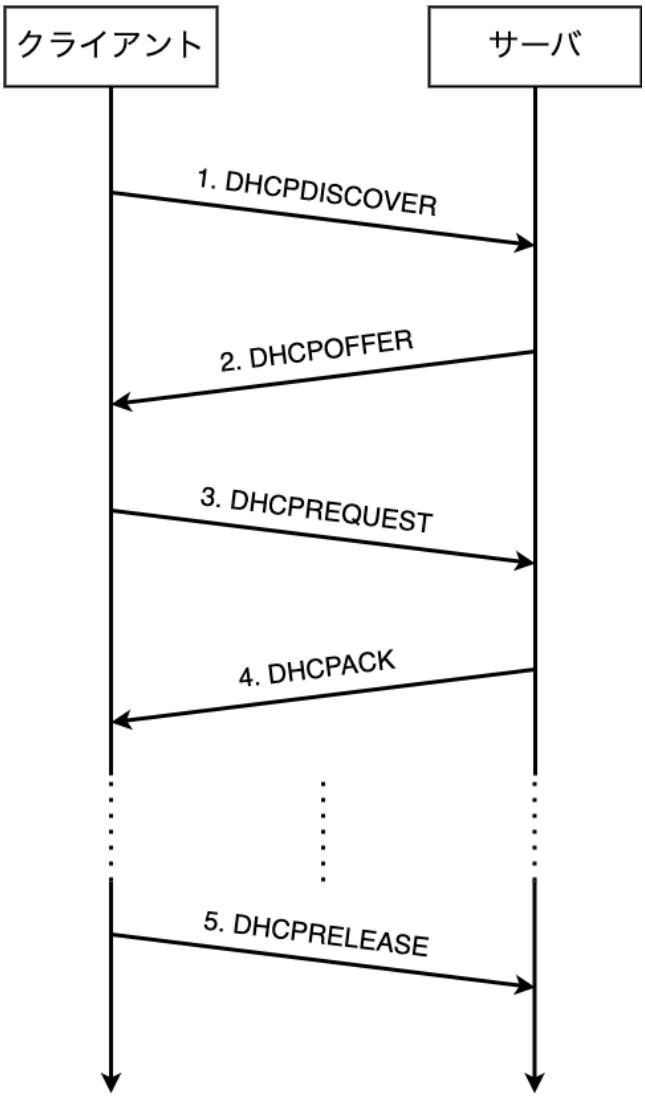

図 4　クライアントとサーバ間との通信

信したサーバはクライアントに, IP アドレスが含まれている設定パラメータつきの DHCPACK パケットを送信する．DHCPACK パケットを受信したクライアントは設定パラメータをもとに自身の IP アドレスを設定する．

(5) クライアントが IP アドレスを返却する際, そのことをサーバに伝えるために, DHCPRELEASE パケットをサーバに送信する．

LAN 内に IP を振り分けるための DHCP サーバをコンテナを用いて構築する例について説明する．DHCP サーバコンテナを動かしているホストマシンには, DHCP サーバコンテナの他に別のコンテナも動いており, DHCP サーバコンテナはそれらのコンテナとも通信可能である．図 5 にホストマシンの環境, 図 6 にホストマシンの LAN 環境を示す．

図 5 では, ホストマシン上にホスト OS (Linux), Runtime が存在し, Runtime 上に DHCP サーバコンテナと通常のコンテナが存在する．ホストマシン上におけるホスト OS (Linux) は Linux カーネルを使用した OS であり, Runtime と通信ができる．Runtime 上における DHCP サーバコンテナともう一つのコンテナはお互いに通信可能である．

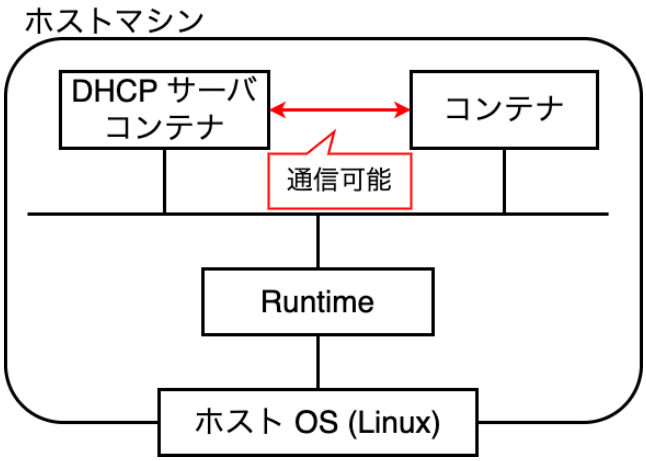

図 5　ホストマシンの環境

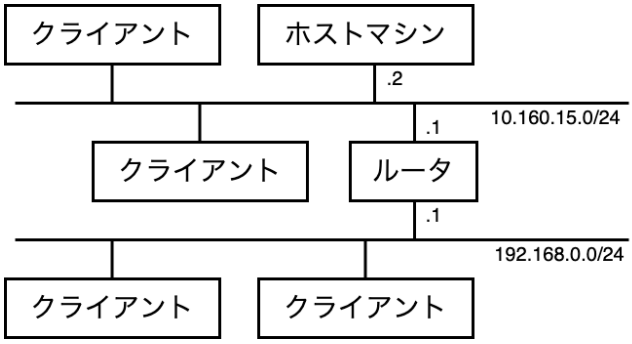

図 6　ホストマシンの LAN 環境

図 6 では, ホストマシンに繋がっているネットワーク 10.160.15.0/24 と繋がっていないネットワーク 192.168.0.0/24 の二つのネットワークが存在する．ネットワーク 10.160.15.0/24 上にあるホストマシンの IP アドレスは 10.160.15.2 である．これらの二つのネットワークはルータを通して通信可能である．ルータの IP アドレスはそれぞれのネットワークで, 10.160.15.1 と 192.168.0.1 である．図中におけるクライアントは IP アドレスが設定されておらず, DHCP サーバであるホストマシンに対してリクエストを送信する．また, ネットワーク 192.168.0.0/24 上のクライアントは DHCP リレーエージェントであるルータを中継することで, ホストマシンにリクエストを送信できる．DHCP リレーエージェントは別ネットワークで DHCP クライアントと DHCP サーバが DHCP で用いるパケットを送受信できるための中継用のエージェントである．

課題

既存の DHCP サーバではコンフィグファイルを読み込み, そのファイルに設定されている内容から DHCP サー

バの動作を決定する．このコンフィグファイルはホストマシンのユーザである管理者が設定を行う．設定を記述するために管理者は，ホストマシンの環境や LAN 環境を事前に知る必要がある．また，このコンフィグファイルは設定した環境でしか基本的に適用できず，ホストマシンの環境や LAN 環境が変わった新しい環境に移行する場合，同じ設定で適用できるとは限らない．これは，DHCP サーバをコンテナ化したときも同様である．具体例として，既存の DHCP サーバを新しいネットワークでコンテナとして動作させる場合を図 7 に示す．

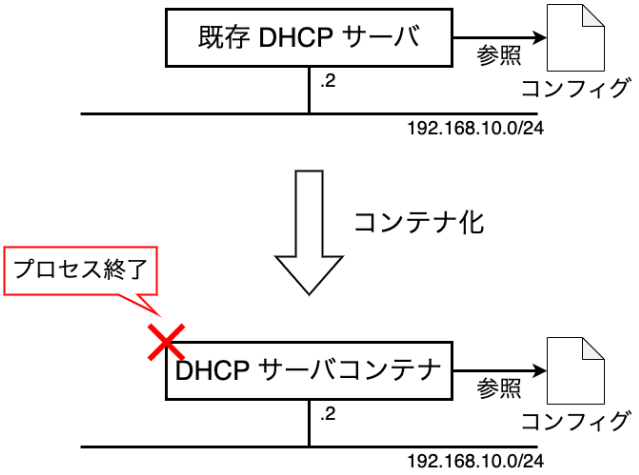

図 7　課題の具体例

図 7 では，ネットワーク 192.168.10.0/24 上に既存の DHCP サーバが存在する．この DHCP サーバをコンテナ化し，DHCP サーバコンテナとして動作させる．このとき，コンフィグは既存の DHCP サーバで使用したコンフィグを使用する．しかし，このコンフィグは新しく作成した DHCP サーバコンテナでは適用できず，コンテナはプロセス終了してしまう．このため，DHCP サーバをコンテナとして動作させる場合，コンテナの移植性という利点を得ることができないという課題がある．

各章の概要

本研究は，次のように構成される．第 1 章では，本研究の背景・課題について述べる．第 2 章では，本研究の関連研究について述べる．第 3 章では，第 1 章で述べた課題を解決するための提案手法，ユースケース・シナリオについて述べる．第 4 章では，提案手法を実現するための実装と実験方法について述べる．第 5 章では，基礎実験とその分析について述べる．第 6 章では，本研究における議論について述べる．第 7 章では，本研究のまとめについて述べる．

2.　関連研究

Network Function Virtualization (NFV) と呼ばれる物理ネットワークから分離してソフトウェアから操作する方法がある [13, 14]．NFV は 2012 年に提唱され，European Telecommunications Standards Institute (ETSI) によって，策定が進められている．NFV を用いることで，従来，ユーザ側で必要となる DHCP サーバや Firewall をサービスプロバイダ側に委譲することができ，ユーザ側は管理をする必要がなくなる．サービスプロバイダ側に委譲された機能は仮想化され，物理的に操作することなく設定を変更することが可能である．加えて，この例の他に，Long Term Evolution (LTE) の中心ネットワークである Evolved Packet Core (EPC) に適用できる例がある．NFV を適用することで，EPC で使われる機能を仮想化し，データセンタ側に委譲できるため，容易にスケーリングできる利点が存在する．しかし，コンテナ環境での NFV については追加の研究が必要とされている．そのため，コンテナを前提とした本研究では NFV による仮想化は適用できない．

NFV に関連して，サーバレス環境で NFV を用いた DHCP 環境を構築し，性能測定をした研究がある [15]．この研究では，サーバレス環境を構築できるソフトウェア OpenWhisk を用いて，コンテナ化した DHCP サーバをコンテナとしてサーバレス環境で動作させている．この環境とサーバレスではない従来の環境での DHCP サーバと NAT の性能比較をこの研究は行っている．結論として，DHCP サーバのサーバレス環境での有用性を示せたが，必ずしもサーバレス環境での実行が優れているということではないと述べられている．本研究では，DHCP サーバコンテナの移植性に焦点をおいているため，性能評価となるこの研究は適用できない．

3.　提案

提案方式

提案方式ではホスト OS のルーティングテーブルを利用し，コンフィグファイルを作成することで DHCP サーバコンテナを自動構築を行う提案をする．ここでのルーティングテーブルとは，ホストマシン内で用意されているルーティングテーブルである /proc/net/route ファイルのことである．/proc/net/route ファイルで，ホストマシンが所属するネットワークを判別することができる．図 6 を例に，ホストマシンを基準としたネットワークを図 8 に示す．

図 8 において，10.160.15.0/24 はホストマシンが所属するネットワークであり，192.168.0.0/24 はホストマシンが所属しないネットワークである．このうち，ホストマシンが所属するネットワーク 10.160.15.0/24 は /proc/net/route ファイルを用いることでホストマシン側で認識することができる．

以下に /proc/net/route ファイルについて説明を行う．/proc/net/route ファイルの内容の例をファイル 1 に示す．

/proc/net/route は procfs と呼ばれるファイルシステムに存在するファイルである．procfs はコンピュータを構成

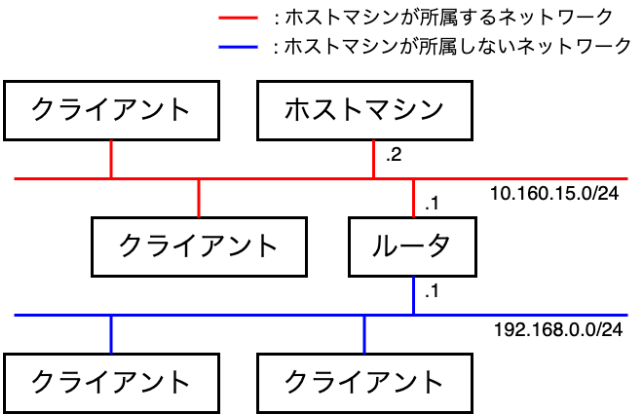

図 8　ホストマシンを基準としたネットワーク

```
1 Iface   Destination     Gateway         Flags   RefCnt
  Use     Metric  Mask            MTU     Window  IRTT
2 ens34   00000000        0164A8C0        0003    0
  0       100     00000000        0       0       0
3 ens34   2346C80A        0164A8C0        0007    0
  0       100     FFFFFFFF        0       0       0
4 docker0 000011AC        00000000        0001    0
  0       0       0000FFFF        0       0       0
5 ens34   0064A8C0        00000000        0001    0
  0       100     00FFFFFF        0       0       0
6 ens34   0164A8C0        00000000        0005    0
  0       100     FFFFFFFF        0       0       0
7 ens34   0664A8C0        00000000        0005    0
  0       100     FFFFFFFF        0       0       0
```

ファイル 1: /proc/net/route

するハードウェアの情報をファイルとして読み込むことができるファイルシステムである．ハードウェアの例として CPU やメモリがあげられ，その情報の例には CPU コア数，使用メモリ量があげられる．UNIX や UNIX 系の OS ではこの procfs によって，ハードウェア情報をファイルの一つとして扱うことが可能である [16]．/proc/net/route は 11 個の要素で構成されている．このうち DHCP サーバのコンフィグで必要となる要素は，Iface, Destination, Gateway, Flags, Mask である．これらの要素の各説明を表 1 に示す．

表 1　/proc/net/route における必要な要素の説明表

要素	説明
Iface	ネットワークインターフェース
Destination	宛先 IP アドレス
Gateway	ゲートウェイ
Flags	<linux/route.h>に定義されたフラグ
Mask	サブネットマスク

また，<linux/route.h>に定義されているフラグを Linux ソースコード上の include/uapi/linux/route.h[*3] から抜粋しファイル 2 に示す．

[*3] https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/include/uapi/linux/route.h

```
51 #define RTF_UP          0x0001          /* route
   usable                  */
52 #define RTF_GATEWAY     0x0002          /* destination
   is a gateway     */
53 #define RTF_HOST        0x0004          /* host entry
   (net otherwise)   */
54 #define RTF_REINSTATE   0x0008          /* reinstate
   route after tmout  */
55 #define RTF_DYNAMIC     0x0010          /* created
   dyn. (by redirect)   */
56 #define RTF_MODIFIED    0x0020          /* modified
   dyn. (by redirect)  */
57 #define RTF_MTU         0x0040          /* specific
   MTU for this route  */
58 #define RTF_MSS         RTF_MTU         /*
   Compatibility :-(            */
59 #define RTF_WINDOW      0x0080          /* per route
   window clamping    */
60 #define RTF_IRTT        0x0100          /* Initial
   round trip time      */
61 #define RTF_REJECT      0x0200          /* Reject
   route                  */
```

ファイル 2: include/uapi/linux/route.h

表 1 とファイル 2 を使用して，ファイル 1 の内容について説明する．ファイル 1 の 3 行目の内容を例として説明する．内容の要素と値を表 2 に示す．

表 2　内容の要素と値

要素	値
Iface	ens34
Destination	2346C80A
Gateway	0164A8C0
Flags	0007
Mask	FFFFFFFF

表 2 において，Destination, Gateway, Mask は 16 進数表記のリトルエンディアンで表されている．これらを変換した結果を表 3 に示す．

表 3　変換結果

要素	値
Destination	10.200.70.35
Gateway	192.168.100.1
Mask	255.255.255.255

また，Flags は<linux/route.h>に示されているフラグのビット演算のため，0007 を変換すると，RTF_UP, RTF_GATEWAY, RTF_HOST の 3 つのフラグが存在することがわかる．よって，これらの結果から 3 行目の内容を説明することができる．/proc/net/route ファイルではこの内容が一行ごとに記載されている．提案手法では，ホストマシンが所属するネットワークを判別するために，

Gateway が存在し, かつ自身のネットワークがわかるネットワークを /proc/net/route ファイルから複数選択する. ファイル 1 では, 2 行目と 5 行目が該当する. また, ネットワークインターフェースが複数存在する場合, ネットワークインターフェースごとに同じ処理を行う. このアルゴリズムをアルゴリズム 1 に示す.

アルゴリズム 1 ネットワークの判別アルゴリズム

```
Require: lines は /proc/net/route の 2 行目以降の行
results ← {}
RTF_UP ← 0x0001
RTF_GATEWAY ← 0x0002
for all line ← lines do
    element ← SPLIT(line)
    Iface ← element.Iface
    Destination ← element.Destination
    Gateway ← element.Gateway
    Flags ← element.Flags
    Mask ← element.Mask
    segments ← COUNT(Mask, 1)
    if Flags & RTF_UP then
        if Gateway and Flags & RTF_GATEWAY then
            if not results[Iface] then
                results[Iface] ← Iface
            end if
            results[Iface].Gateway ← Gateway
        else if notSUBSTR(Destination, segments, 32) then
            if not results[Iface] then
                results[Iface] ← Iface
            end if
            results[Iface].Destination ← Destination
            results[Iface].Mask ← Mask
        end if
    end if
end for
```

アルゴリズム 1 では, /proc/net/route の 2 行目以降の行で, ホストが所属しているネットワークを Iface をキーとした results オブジェクトとして取得している. results には Destination, Gateway, Mask の値が格納されているため, Iface ごとのネットワークの概要を取得することが可能である.

ユースケース・シナリオ

ユースケースとして, 複数の部署に分かれた企業を想定する. ユースケースの例を図 9 に示す.

図 9 では, 部署が複数存在し, それぞれにルータと DHCP サーバコンテナが存在している環境である. このような環境において, 既存手法では, 部署ごとに DHCP サーバコンテナを手動で設定する必要がある. また, 部署が増えたり, 分割したりした場合, 新たに同様の作業を繰り返さなければならない. しかし, 提案手法を用いることで, DHCP サーバコンテナを導入するだけで完結するため, 既存手法で存在した同じ作業の繰り返しを減らすことが可能である.

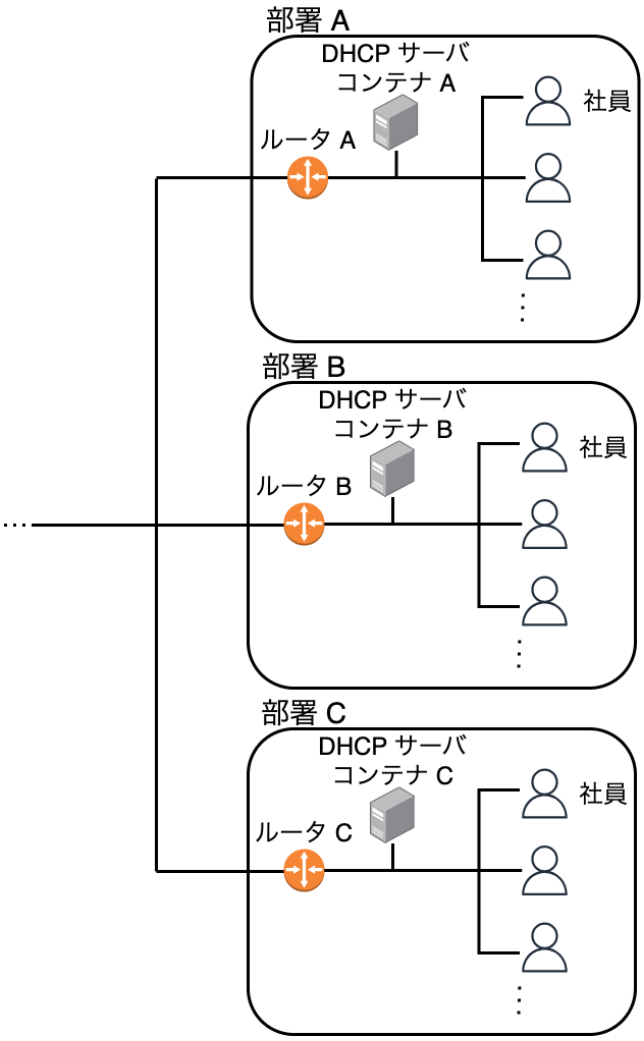

図 9 ユースケースの例

4. 実装

実装の全体図を図 10 に示す.

実装では, 図 10 中の 1. 及び 2. の部分を実装する. 1. では, 提案方式で提案した手法をホストマシンが所属しているネットワークを判別する. ネットワーク判別のためにホストから read-only でコンテナ内にボリュームマウントされた /proc/net/route ファイルを参照する. 参照したファイルからコンフィグに必要な内容を 2. のコンフィグ作成で利用する. ファイルから取得した内容はコンフィグファイルである dhcpd.conf の記述形式として作成する. 最後に, 3. の ISC DHCP が 2. で作成した dhcpd.conf を参照し, ループすることで DHCP サーバとして動作する. また, 1., 2., 3. はそれぞれ独立したプログラムなため, エラーやコンテナの強制終了が発生した場合, プロセスが終了する.

5. 実験

課題にて述べた, DHCP サーバコンテナの移植性の低さを示すために実験を行う.

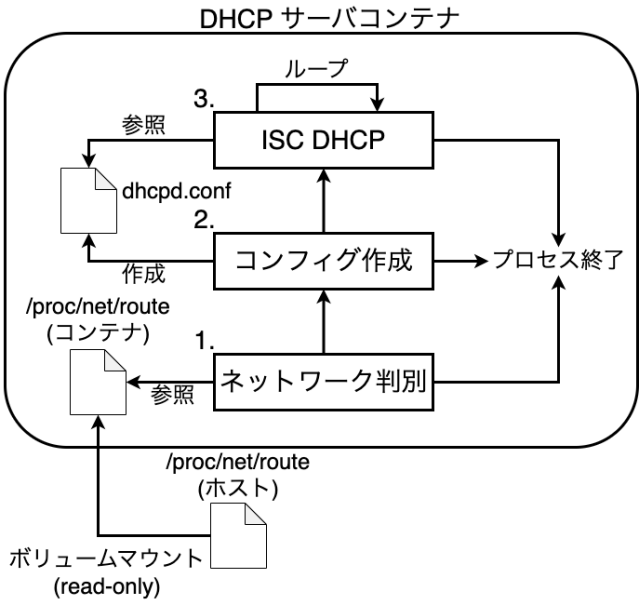

図 10　実装の全体図

実験環境

実験環境は ESXi 上に構築した VM 環境で行う．ホスト OS は Ubuntu 22.04, Runtime は Docker を利用する．なお，ホストマシンが所属するネットワークは 192.168.100.0/24, コンテナネットワークは 172.17.0.0/16 である．Debian 上で ISC DHCP を実行するコンテナを作成できる Dockerfile をファイル 3 に示す．

この Dockerfile を用いて，実験用のコンテナイメージを作成した．また，実行時に参照する dhcpd.conf ファイルをファイル 4, 5 に示す二種類用意した．

ファイル 4, 5 は，ISC DHCP の起動時に参照されるコンフィグファイルである．これらの違いとして，ファイル 4 では，ホストマシンが所属するネットワークのみの設定を記述しているが，ファイル 5 では，コンテナネットワークの設定も記述している．コンテナネットワークの設定はファイル 5 の 11 行目に示されており，172.17.0.0/16 のネットワークについて記述している．ファイル 3, 4, 5 は GitHub 上で閲覧可能である[*4]．これらのファイルを使用し，ホスト上とコンテナ上で ISC DHCP が起動するかの実験を行った．

実験結果と分析

表 4 に起動実験の結果を示す．なお，行は dhcpd.conf ファイルの内容を示し，ファイル 4 とファイル 5 をあげている．列は実行環境を示し，ホストとコンテナをあげている．

実験の結果，ホストで起動したコンフィグファイルをコンテナで実行した場合，起動できないことを確認した．このことから，DHCP サーバコンテナではホスト環境で利用したコンフィグファイルを利用できないため，移植性が低いことが示せる．

[*4] https://github.com/cdsl-research/C0119316-exp

```dockerfile
FROM debian:bookworm-slim AS BUILDING

WORKDIR /root

RUN apt update && apt install -y wget make gcc && \
    wget https://downloads.isc.org/isc/dhcp/4.4.3-P1/dhcp-4.4.3-P1.tar.gz && \
    tar xzf dhcp-4.4.3-P1.tar.gz && cd dhcp-4.4.3-P1 && \
    ./configure && make

FROM debian:bookworm-slim AS RUNNING

WORKDIR /root

COPY --from=BUILDING /root/dhcp-4.4.3-P1/server/dhcpd /root/dhcpd

RUN mkdir -p /var/lib/dhcp && touch /var/lib/dhcp/dhcpd.leases && \
    mkdir -p /etc/dhcp && touch /etc/dhcp/dhcpd.conf && \
    mkdir -p /run/dhcp-server && touch /run/dhcp-server/dhcp.pid && \
    chmod 775 /var/lib/dhcp && chmod 664 /var/lib/dhcp/dhcpd.leases

CMD ["./dhcpd", "-f", "-4", "-pf", "/run/dhcp-server/dhcpd.pid", "-cf", "/etc/dhcp/dhcpd.conf", "-lf", "/var/lib/dhcp/dhcpd.leases"]
```

ファイル 3: Dockerfile

```
default-lease-time 600;
max-lease-time 7200;

subnet 192.168.100.0 netmask 255.255.255.0 {
    range 192.168.100.30 192.168.100.200;
    option routers 192.168.100.1;
    option subnet-mask 255.255.255.0;
}
```

ファイル 4: dhcpd.conf

表 4　ISC DHCP の起動実験

	ファイル 4	ファイル 5
ホスト	起動	起動
コンテナ	起動せず	起動

6. 議論

本研究では，単一ホストマシンにおけるコンテナ環境を

```
default-lease-time 600;
max-lease-time 7200;

shared-network container {
    subnet 192.168.100.0 netmask 255.255.255.0 {
        range 192.168.100.30 192.168.100.200;
        option routers 192.168.100.1;
        option subnet-mask 255.255.255.0;
    }

    subnet 172.17.0.0 netmask 255.255.0.0 {
    }
}
```

ファイル 5: dhcpd.conf

想定した課題と提案手法を述べた. しかし, 実際のコンテナ運用では, 複数のマシンでコンテナを管理する手法が取られている. 実際の例として, コンテナオーケストレーションツールである Kubernetes がある. Kubernetes では複数のホストマシンを用いてマスターノードとワーカーノードに分割してクラスタを構築する. Kubernetes において通常, アプリケーションコンテナはワーカーノードに配置される. このような環境の場合, DHCP サーバコンテナが配置されるワーカーノードは常に同じものとは限らない. そのため, DHCP サーバコンテナは新しいワーカーノードに配置される場合, 作成したコンフィグファイルがあるにも関わらずコンフィグ生成の処理をもう一度行わなければならない. この場合, Container Network Interface (CNI) を使用してクラスタ内で DHCP サーバの情報を共有しコンフィグ生成の処理をなくすという手法を取ることができる.

7. おわりに

本研究では, DHCP サーバコンテナの移植性が低い課題を示し, その課題を解決する提案手法として, ホスト OS のルーティングテーブルを用いたコンフィグファイルによる DHCP サーバコンテナの自動構築を提案した. 実際にホストにある DHCP サーバを同じコンフィグファイルでコンテナとして実行し, 起動するかの基礎実験を行った結果, コンテナでは起動せず, コンテナの移植性が低いことを示せた.

参考文献

[1] Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A. and Peterson, L.: Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors, *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, New York, NY, USA, Association for Computing Machinery, p. 275–287 (online), DOI: 10.1145/1272996.1273025 (2007).

[2] Xavier, M. G., Neves, M. V., Rossi, F. D., Ferreto, T. C., Lange, T. and De Rose, C. A. F.: Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments, *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 233–240 (online), DOI: 10.1109/PDP.2013.41 (2013).

[3] Li, Z., Kihl, M., Lu, Q. and Andersson, J. A.: Performance Overhead Comparison between Hypervisor and Container Based Virtualization, *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pp. 955–962 (online), DOI: 10.1109/AINA.2017.79 (2017).

[4] Kozhirbayev, Z. and Sinnott, R. O.: A performance comparison of container-based technologies for the Cloud, *Future Generation Computer Systems*, Vol. 68, pp. 175–182 (online), DOI: https://doi.org/10.1016/j.future.2016.08.025 (2017).

[5] Celesti, A., Mulfari, D., Fazio, M., Villari, M. and Puliafito, A.: Exploring Container Virtualization in IoT Clouds, *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 1–6 (online), DOI: 10.1109/SMARTCOMP.2016.7501691 (2016).

[6] Boettiger, C.: An Introduction to Docker for Reproducible Research, *SIGOPS Oper. Syst. Rev.*, Vol. 49, No. 1, p. 71–79 (online), DOI: 10.1145/2723872.2723882 (2015).

[7] Merkel, D. et al.: Docker: lightweight linux containers for consistent development and deployment, *Linux j*, Vol. 239, No. 2, p. 2 (2014).

[8] Anderson, C.: Docker [Software engineering], *IEEE Software*, Vol. 32, No. 3, pp. 102–c3 (online), DOI: 10.1109/MS.2015.62 (2015).

[9] Rad, B. B., Bhatti, H. J. and Ahmadi, M.: An introduction to docker and analysis of its performance, *International Journal of Computer Science and Network Security (IJCSNS)*, Vol. 17, No. 3, p. 228 (2017).

[10] Dua, R., Raja, A. R. and Kakadia, D.: Virtualization vs Containerization to Support PaaS, *2014 IEEE International Conference on Cloud Engineering*, pp. 610–614 (online), DOI: 10.1109/IC2E.2014.41 (2014).

[11] Felter, W., Ferreira, A., Rajamony, R. and Rubio, J.: An updated performance comparison of virtual machines and Linux containers, *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 171–172 (online), DOI: 10.1109/ISPASS.2015.7095802 (2015).

[12] Droms, R.: Dynamic Host Configuration Protocol, RFC 2131 (1997).

[13] Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F. and Boutaba, R.: Network Function Virtualization: State-of-the-Art and Research Challenges, *IEEE Communications Surveys & Tutorials*, Vol. 18, No. 1, pp. 236–262 (online), DOI: 10.1109/COMST.2015.2477041 (2016).

[14] Han, B., Gopalakrishnan, V., Ji, L. and Lee, S.: Network function virtualization: Challenges and opportunities for innovations, *IEEE Communications Magazine*, Vol. 53, No. 2, pp. 90–97 (online), DOI: 10.1109/MCOM.2015.7045396 (2015).

[15] Savi, M., Banfi, A., Tundo, A. and Ciavotta, M.: Serverless Computing for NFV: Is it Worth it? A Performance Comparison Analysis, *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 680–685 (online), DOI: 10.1109/PerComWorkshops53856.2022.9767495 (2022).

[16] Ritchie, D. M.: The UNIX System: A Stream Input-Output System, *AT&T Bell Laboratories Technical Journal*, Vol. 63, No. 8, pp. 1897–1910 (1984).