Vol.2022-DPS-190 No.19
Vol.2022-CSEC-96 No.19
2022/3/10

分散 MEC 環境における コンテナオーケストレーションシステム用の ネットワークプラグイン

山田 光樹[1,a)] 渡邊 大記[2,b)] 安森 涼[2,c)] 近藤 賢郎[3,d)] 熊倉 顕[4,e)] 前迫 敬介[4,f)] 張 亮[4,g)] 寺岡 文男[1,h)]

概要：筆者らは UE (User Equipment) がコンテナクラスタとして構成されるアプリケーションの一部を MEC (Multi-Access Edge Computing) サーバにオフロードすることを可能にする ContMEC アーキテクチャを提案している．ContMEC での使用が想定されている AR (Augmented Reality) や VR (Virtual Reality) などのアプリケーションは，大容量のデータ通信をする可能性がある．ContMEC はコンテナオーケストレーションシステムとして Kubernetes を採用しているが，Kubernetes 内のネットワークを構築する CNI plugin は，現状では帯域保証されたネットワークを構築できない．本稿では，Kubernetes を採用した ContMEC において，帯域保証されたネットワーク構築を可能とする Concorde CNI plugin を提案する．Concorde の導入により，アプリケーション開発者がアプリケーションのデプロイファイルに必要帯域を指定するだけで，帯域保証されたネットワークを構築できる．評価では，Pod (コンテナの集合) 間，Docker コンテナ間のスループットを測定し，Concorde CNI plugin が正常に動作することを確認した．

1. はじめに

近年，クラウドアプリケーションはコンテナクラスタを用いて構成することが増えている [1]．コンテナクラスタは複数のサーバで構成されるコンピューティングクラスタで構築することが多く，コンテナを効率良く運用・管理するために，コンテナオーケストレーションシステムが必要である．また，近年，MEC (Multi-access Edge Computing) [2] という技術が台頭している．MEC は少数のクラウドサーバと地理的に分散した Edge Station で構成され，各 Edge Station には複数の MEC サーバが配置される [3]．計算処理を MEC サーバへ委譲することをオフロードと呼び，UE (User Equipment) やクラウドサーバから MEC サーバに計算処理をオフロードすることで，UE やクラウドサーバにかかる計算負荷の削減や低遅延な通信を実現できる．

Edge Station は比較的広い範囲をカバーするため，UE がアプリケーションの一部を Edge Station にオフロードする場合，UE が移動しても同じ Edge Station を利用し続ける可能性が高い．このような MEC 環境において，コンピューティングクラスタを構成するためには，(i) UE や Edge Station 数に対するスケーラビリティと (ii) コンピューティングクラスタ間での効率的な計算資源共有の 2 つの要件を満たす必要がある．

そこで，筆者らは上記の 2 つの要件を考慮し，UE がアプリケーションの一部を MEC 環境にオフロードできる ContMEC アーキテクチャ [4] を提案している．ContMEC は，(i) Edge Station 毎にコンピューティングクラスタを構築し，UE 数に応じたスケーラビリティを実現，(ii) 階層的なリソース管理により，コンピューティングクラスタ間のスケーラビリティと効率の良いリソース共有を実現，(iii) コンピューティングクラスタの重ね合わせにより，効率の良いリソース共有を実現，の 3 つの特徴があ

1 慶應義塾大学理工学部
Faculty of Science and Technology, Keio University
2 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University
3 慶應義塾情報セキュリティインシデント対応チーム
Computer Security Incident Response Team, Keio University
4 ソフトバンク株式会社
SoftBank Corp.
a) ham@inl.ics.keio.ac.jp
b) nelio@inl.ics.keio.ac.jp
c) moririn@inl.ics.keio.ac.jp
d) latte@itc.keio.ac.jp
e) ken.kumakura@g.softbank.co.jp
f) keisuke.maesako@g.softbank.co.jp
g) cho.ryo@g.softbank.co.jp
h) tera@keio.jp

ⓒ 2022 Information Processing Society of Japan

る．ContMEC では 1 つの Edge Station はコンテナを配置する 1 つ以上の Cluster Worker Node と，コンテナが配置されたコンピューティングクラスタを管理する 1 つの Cluster Master で構成する．また，1 つの Cluster Worker Node が別の Edge Station の Cluster Master にも属する構成を overlapped worker と呼ぶ．

ContMEC の要求事項の 1 つとして，アプリケーションを構成するマイクロサービス間の帯域保証がある．ContMEC での使用が想定されている AR (Augmented Reality) や VR (Virtual Reality) などのアプリケーションでは，大容量のデータを送受信する可能性がある．そのため，帯域保証されていないネットワークを利用する場合，アプリケーションの応答時間が長くなることが想定される．

現在，ContMEC はコンテナオーケストレーションシステムとして Kubernetes [5] を実装に採用している．Kubernetes では 1 つ以上のコンテナの集合を Pod と呼び，Pod を配置する Worker Node と Worker Node や Pod を管理する Master Node で構成する．また，Pod 間ネットワークを構築するためには CNI (Container Network Interface) が使われる．CNI とはコンテナを既存のネットワークに接続する技術であり，コンテナの I/F (interface) の提供・削除，コンテナネットワークの設定と CNI のバージョンの取得の 4 つの仕様を定義している．前述のように CNI が満たすべき仕様がとてもシンプルであるため，CNI の仕様に沿った実装である CNI plugin の実装も容易である．Kubernetes を採用した ContMEC では，Flannel [6] によって Pod 間ネットワークを構築することはできるが，帯域保証されたネットワークを構築することはできない．

本稿では，ContMEC における Pod 間ネットワークに帯域保証をするための仕組みを検討し，独自の Concorde CNI plugin を PoC (Proof of Concept) で設計・実装した．CNI plugin は組み合わせることができるため，ネットワークの接続性には Flannel を用い，IP アドレスの割り当てや帯域保証は Concorde を用いた．また，CNI plugin は ContMEC 環境を想定していないため，Concorde を ContMEC 環境に対応させ，評価では，Kubernetes 環境と ContMEC 環境の両環境で正常に動作するか検証した．

2. 関連技術・研究

コンピューティングクラスタの構築時点では，各ノード間の疎通性がない．したがって，ノードを跨いだ通信を可能とする Pod 間ネットワークを構築する必要がある．

図 1 は Flannel [6] が適用された Worker Node 上のネットワーク構成の概要を示している．Flannel は複数のノード間で仮想的な L2 (Layer 2) ネットワークを構築することで，ノードを跨いだ Pod 間通信を可能とする．仮想的な L2 ネットワークの構築には VXLAN (Virtual eXtensible

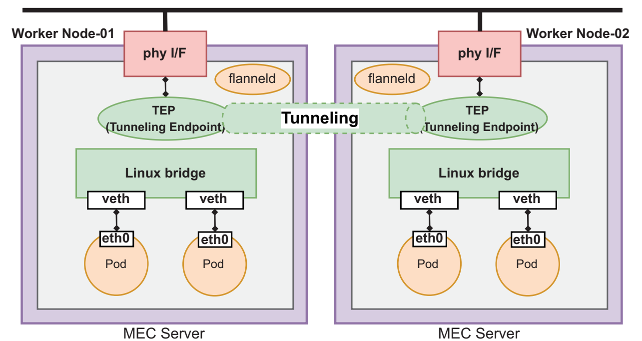

図 1: Flannel のネットワーク構成の概要．

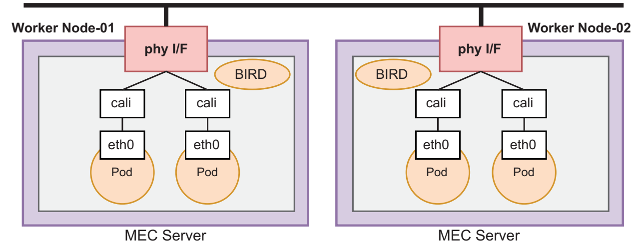

図 2: Calico のネットワーク構成の概要．

Local Area Network) などのオーバーレイ技術を用い，オーバーレイネットワークのエンドポイントが TEP (Tunneling Endpoint) である．TEP は flanneld という Flannel デーモンが作成し，flanneld は Flannel を Kubernetes 上に適用した際に Pod として全てのノード上にデプロイされる．Pod 作成時には，VTEP と Pod を接続する Linux bridge と bridge と Pod を接続する veth pair [7] を作成する．

図 2 は Calico [8] が適用された Worker Node 上のネットワーク構成の概要を示している．Calico は BGP (Border Gateway Protocol) を利用した L3 (Layer 3) ネットワークを構築することで，ノードを跨いだ Pod 間通信を可能とする．オーバーレイ技術は柔軟性がある一方，L2 フレームのカプセル化が必要であるため，オーバーヘッドやフレームサイズ制限によってパフォーマンスが劣化する可能性がある．Calico では，BIRD [9] というルーティングデーモンが Pod の経路情報を BGP を利用して外部ネットワークに配布するため，オーバーヘッドがない L3 ネットワークを構築できる．また，L3 ネットワークを扱っているため，Flannel のような bridge は使用せず，cali から始まる TAP (Terminal Access Point) インタフェースを利用する．

Cilium [10] は eBPF (Extended Berkeley Packet Filter) という Linux カーネル技術を基盤にして，ノードを跨いだ Pod 間通信を可能とする．eBPF によってネットワークを制御することで，高速な通信を実現できる．

表 1 に主要な CNI plugin が提供する機能を示す．全ての CNI plugin は Pod 間のネットワーク接続性を提供し，Calico はネットワークポリシ，Cilium は負荷分散や帯域制限の機能も提供する．一方，CNI は 2017 年に提案された

表 1: 主要な CNI plugin が提供する機能.

	Flannel	Calico	Cilium
ネットワーク接続性	○	○	○
ネットワークセキュリティポリシ		○	
負荷分散・帯域制限			○
帯域保証			

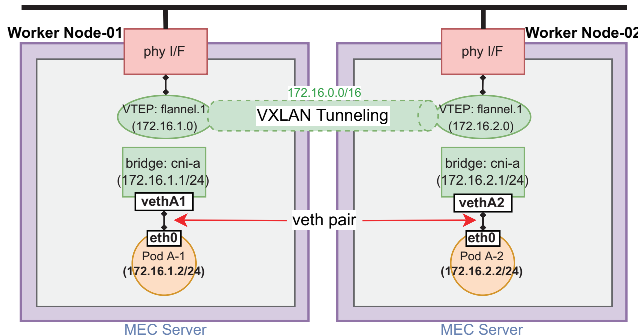

図 3: Worker Node のネットワーク構成の概要.

新しい技術であり，研究・開発が追いついていない．そのため，表 1 の 3 つを含めた既存の CNI plugin では帯域保証された Pod 間ネットワークを構築できない．また，ネットワーク資源は共有されており，帯域保証するためにユーザ毎にネットワーク資源の使用量を調整する必要があるため，帯域保証の仕組みを構築することは一般的に難しい．

3. Concorde CNI plugin

2 章で述べたように，既存の CNI plugin では帯域保証された Pod 間ネットワークを構築できないという課題がある．本章では，この課題を解決するために独自に設計・実装した Concorde CNI plugin の設計を説明する．また，ContMEC 環境への対応に関する設計と CNI plugin が適用された ContMEC 環境での Pod 間の通信手順を述べる．

3.1 ネットワーク構成

本稿では，Flannel と Concorde を組み合わせてコンピューティングクラスタ内のネットワークを構築している．Flannel はノードを跨いだ通信を可能とする Pod 間ネットワークを構築し，Concorde は Pod の IP アドレス割り当てと帯域保証の設定をする．図 3 は Flannel と Concorde が適用されたコンピューティングクラスタの Worker Node におけるネットワーク構成の概要を示している．ノードを跨いだ Pod 間通信を可能にするためには，VXLAN などのオーバーレイネットワークによってノード間を接続する必要がある．Flannel では，VTEP (VXLAN Tunneling Endpoint) をエンドポイントとする VXLAN のオーバーレイネットワークを構築することで，ノード間を接続している．また，Concorde は Flannel によって構築されたネットワーク上に適用し，Linux bridge や veth pair を作成する．

図 3 に示す cni-a は Linux bridge であり，Pod と VTEP 間を接続する仮想 I/F で，同一コンピューティングクラスタの全ての Pod が共有する．また，vethA1 や eth0 などの veth pair は Odin 社の Virtuozzo や，そのオープン版である OpenVZ に実装されている仮想ネットワーク I/F であり，Pod と Linux bridge 間の接続性を提供する [7].

3.2 帯域保証

本稿では PoC 実装として，帯域保証機能の実装に Linux tc コマンド [11] を採用し，始点 Pod 毎に送信方向への帯域保証を実現する．tc コマンドは I/F を通過するトラフィックを TBF (Token Bucket Filter) などのキューイングによって制御することができ，I/F に対して設定する．Kubernetes や Kubernetes を採用した ContMEC では，tc コマンドの設定先候補として物理 I/F，VTEP，Linux bridge，bridge と接続した veth，Pod と接続した veth (eth0) の 5 つの I/F がある．これらの I/F に帯域保証の設定をした際の利点や欠点を述べる．

3.2.1 物理 I/F への設定

物理 I/F はノード外に転送するトラフィックが全て通過する I/F であるため，全てのトラフィックを制御することができ，複数トラフィック間で帯域を融通することも容易である．一方，始点 IP アドレスによるパケットの制御ができないという tc コマンドの実装の都合上，始点 Pod 毎の帯域保証ができない．

3.2.2 VTEP への設定

VTEP は物理 I/F と同様の動作を示し，ContMEC の overlapped worker 環境ではコンピューティングクラスタ毎に分離されている．したがって，コンピューティングクラスタ毎に帯域保証の設定や保守・運用ができる．一方，VTEP は Flannel によって作成されるため，Flannel を必ず使用しなければならない．また，物理 I/F と同様に始点 Pod 毎の帯域保証ができない．

3.2.3 Linux bridge への設定

bridge は Flannel でも導入しているが，本稿では Concorde が作成している．したがって，Flannel 以外の CNI plugin と組み合わせることができる．また，VTEP と同様に ContMEC ではコンピューティングクラスタ毎に分離されているため，コンピューティングクラスタ毎に帯域保証の設定や保守・運用ができる．一方，トラフィックが Pod に流れる方向へ帯域保証されてしまう bridge と tc コマンドの実装の都合上，送信方向への帯域保証ができない．

3.2.4 bridge と接続した veth への設定

bridge と接続した veth は Pod 毎に作成されるため，始点 Pod 毎の帯域保証ができ，tc の設定がシンプルになる．一方，bridge と同様に，トラフィックが Pod に流れる方向へ帯域保証されてしまうため，送信方向への帯域保証が

ⓒ 2022 Information Processing Society of Japan

できない．また，tc が設定された Pod 以外に流れるトラフィックを考慮していないため，帯域保証値を設定しても，別のトラフィックが流れていると帯域が保証されない可能性がある．したがって，帯域保証値と帯域制限値を同値にする必要がある．

3.2.5 Pod と接続した veth (eth0) への設定

eth0 は bridge と接続した veth と同様に Pod 毎に作成されるため，始点 Pod 毎の帯域保証ができ，tc の設定がシンプルになる．また，eth0 は物理 I/F と同様の動作をするため，送信方向への帯域保証ができる．一方，bridge と接続した I/F と同様に，tc が設定された Pod 以外に流れるトラフィックを考慮していないため，帯域保証値と帯域制限値を同値にする必要がある．

3.2.6 設定先 I/F の決定

上記で述べたように，物理 I/F や VTEP への設定では始点 Pod 毎の帯域保証ができず，bridge や bridge と接続した veth への設定では送信方向への帯域保証ができない．したがって，本稿では Pod と接続した veth (eth0) に対して tc の設定をする．また，上記のように帯域保証されない可能性があるため，帯域保証値と帯域制限値を同値にして tc の設定をする．

3.3 ContMEC 環境への対応

ContMEC では overlapped worker 環境を想定している．そのため，コンピューティングクラスタ毎に独立したネットワークを構成していないと，Worker Node 上の Pod は，その Pod が属しているコンピューティングクラスタ以外のネットワークに属する可能性がある．別のコンピューティングクラスタのネットワークに属してしまうと，コンピューティングクラスタ毎の保守・運用が困難になる．したがって，Worker Node 上のネットワークをコンピューティングクラスタ毎に分離する必要がある．しかし，CNI plugin は Kubernetes 環境を想定しているため，ContMEC に適用することはできない．そこで，文献 [4] で使用している Flannel は，使用する CNI plugin やネットワーク情報が記載されている CNI 構成ファイルや，CNI に関連する様々なファイルをコンピューティングクラスタ毎に分離して ContMEC 環境に対応している．これに加え，本稿では文献 [4] で構築された Flannel によるネットワークに合わせて，ContMEC 環境に Concorde を対応させている．具体的には，Concorde の割り当て済み IP アドレス管理ファイルと bridge を分離し，Pod に割り当てる IP アドレスとトラフィックの管理を分離することで，コンピューティングクラスタ毎に独立したネットワークを構築する．

3.3.1 割り当て済み IP アドレス管理ファイルの分離

割り当て済み IP アドレス管理ファイルは Concorde が導入し，自ノード上の Pod に割り当て済みの Pod IP アドレスを管理するために使用する．図 4 は，Worker Node 上に Cluster-A と Cluster-B の 2 つのコンピューティングクラスタが存在する ContMEC 環境で，コンピューティングクラスタ毎に分離した割り当て済み IP アドレス管理ファイルの内容を示している．Cluster-A の Pod サブネットは 172.16.1.0/24，Cluster-B の Pod サブネットは 172.17.1.0/24 であり，図 4 のように，コンピューティングクラスタ毎に IP アドレスを管理している．コンピューティングクラスタ毎に分離されていない場合，コンピューティングクラスタ数が増加すると管理する IP アドレスも増加し，IP アドレスの管理が困難になる．また，コンピューティングクラスタ毎に分離された CNI plugin が同時に同じファイルにアクセスする可能性があり，ロック機能などを新たに実装する必要がある．したがって，割り当て済み IP アドレス管理ファイルをコンピューティングクラスタ毎に分離する．

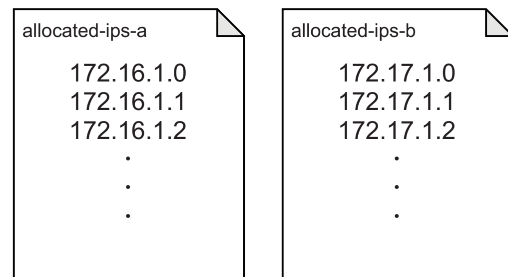

図 4: 割り当て済み IP アドレス管理ファイル．

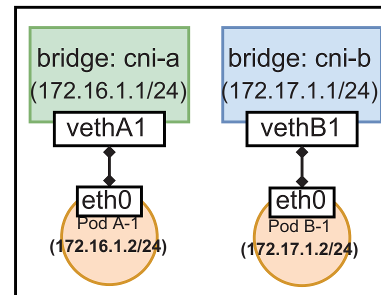

図 5: Linux bridge と Pod.

3.3.2 Linux bridge の分離

bridge は Pod が送信するトラフィックが最初に転送される仮想 I/F で，Pod と VTEP を接続するために使用する．bridge は Flannel でも導入しているが，本稿では Concorde が作成する．図 5 は，3.3.1 項で述べた ContMEC 環境で，Pod，veth pair およびコンピューティングクラスタ毎に分離した bridge を示している．Pod サブネットは 3.3.1 項で述べた通りで，コンピューティングクラスタ毎に Pod の送受信トラフィックを分離している．コンピューティングクラスタ毎に分離されていない場合，コンピューティングクラスタ数が増加すると bridge を通過するトラフィックも増加し，bridge の負荷が大きくなる．また，bridge に障

ⓒ 2022 Information Processing Society of Japan

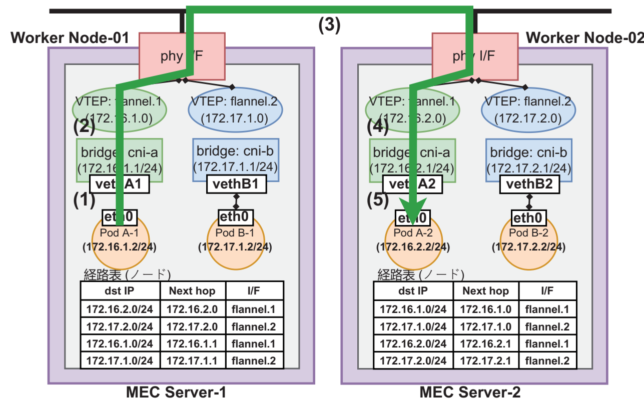

図 6: Worker Node における Pod 間通信.

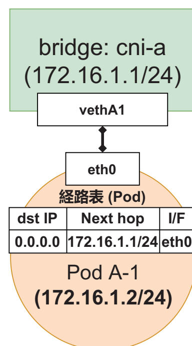

図 7: Pod と bridge の詳細.

害が発生した場合，障害が全てのコンピューティングクラスタに影響してしまう．したがって，bridge をコンピューティングクラスタ毎に分離する．

3.4 Pod 間通信の手順

図 6 は，Cluster-A と Cluster-B の 2 つのコンピューティングクラスタが存在する環境で，Cluster-A において，Worker Node-01 上の Pod A-1 (172.16.1.2) から Worker Node-02 上の Pod A-2 (172.16.2.2) にパケットを送信する手順を示している．本節では，Concorde と Flannel によって ContMEC の Pod 間ネットワークが構築された後，Pod 間でどのようにパケットが送信されるのかを説明する．

(i) Pod (Worker Node-01) ⇒ Linux bridge (Worker Node-01); まず，Pod A-1 が図 7 に示すような，Pod に設定されている経路表を参照して転送先を判断し，cni-a までパケットを転送する (図 6-(1))．(ii) Linux bridge (Worker Node-01) ⇒ VTEP (Worker Node-01); cni-a に到達したパケットは，今度はノードの経路表を参照し，終点 IP アドレスから使用する I/F と転送先を決定する (図 6-(2))．終点 IP アドレスは 172.16.2.2 なので，図 6 に示す経路表では，1 番目のエントリにマッチし，I/F が flannel.1，転送先が 172.16.2.0 の IP アドレスを持つ Worker Node-02 の VTEP に決定される．(iii) VTEP (Worker Node-01) ⇒ VTEP (Worker Node-02); Worker Node-01 の VTEP と Worker Node-02 の VTEP 間は Flannel が作成したトンネルによって通信する (図 6-(3))．このトンネルは VXLAN によるオーバーレイネットワークで構成されており，ノード間で L2 ネットワークを構成することができる．Worker Node-01 の VTEP に到達したパケットは，この L2 ネットワークを通って Worker Node-02 の VTEP に転送される．(iv) VTEP (Worker Node-02) ⇒ Linux bridge (Worker Node-02); Flannel のトンネルを通って Worker Node-02 の VTEP に到達したパケットは，Worker Node-02 のホストの経路表を参照し，終点 IP アドレスから次の転送先が決定される (図 6-(4))．終点 IP アドレスは 172.16.2.2 なので，図 6 に示す経路表では，3 番目のエントリにマッチし，次の転送先が 172.16.2.1 の IP アドレスを持つ cni-a に決定される．(v) Linux bridge (Worker Node-02) ⇒ Pod (Worker Node-02); cni-a にパケットが到達すると，cni-a が ARP (Address Resolution Protocol) によって IP アドレスから MAC アドレスを解決し，その MAC アドレスを持つ Pod にパケットを転送する (図 6-(5))．

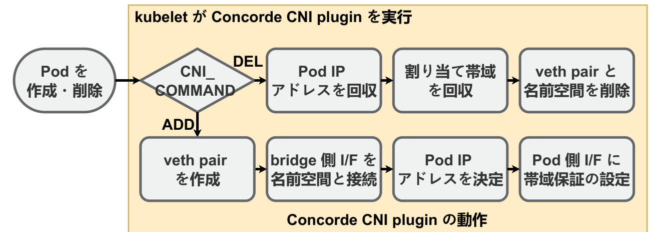

図 8: Concorde 実行時の動作手順.

4. Concorde CNI plugin の実装

本稿では Concorde CNI plugin を bash を用いて実装した．図 8 に Concorde 実行時の動作手順を示す．Concorde の実装詳細を説明する前に，CNI plugin の動作概要を説明する．まず，Pod が作成・削除されると，kubelet という各ノードで実行されるエージェントによって CNI plugin の動作を指示する環境変数 `$CNI_COMMAND` の値が `ADD`・`DEL` に変化する．この `$CNI_COMMAND` の値が変化すると，kubelet が CNI plugin を実行する．CNI plugin は `$CNI_COMMAND` の値の違いによって挙動が異なり，`ADD` の場合は Pod 作成時，`DEL` の場合は Pod 削除時の動作をする．本章では，Pod の作成時と削除時に分け，図 8 に沿って Concorde CNI plugin の実装を説明する．

4.1 Pod 作成時

Pod 作成時には，まず bridge が存在するかどうかを判断する．bridge が存在しない場合は bridge を作成し，bridge が存在する場合は何もせずに次の動作に移る．bridge と Pod を接続するために veth pair を作成し，片方の veth を bridge に接続する．その後，反対側の veth (eth0) を Pod

ⓒ 2022 Information Processing Society of Japan

```
1  allocate_ip(){
2      for ip in "${all_ips[@]}"
3      do
4          reserved=false
5          for my_pod_ip in "${my_pod_ips[@]}"
6          do
7              if [ "$ip" = "$my_pod_ip" ]; then
8                  reserved=true
9                  break
10             fi
11         done
12         if [ "$reserved" = false ]; then
13             echo "$ip" >> $MY_POD_IP_STORE
14             echo "$ip"
15             return
16         fi
17     done
18 }
```

図 9: Concorde の実装 (割り当て IP アドレス決定関数).

```
1  bandwidth_control(){
2      bandwidth_value=$(bandwidth_allocate $2)
3      ip netns exec $1 tc qdisc add dev eth0 root
           handle 1: htb default 1
4      ip netns exec $1 tc class add dev eth0
          parent 1: classid 1:1 htb rate
          $bandwidth_value ceil $bandwidth_value
          burst 10Mb cburst 10Mb
5  }
```

図 10: Concorde の実装 (帯域保証設定関数).

の名前空間に接続する．ここまでの流れが，CNI の仕様の 1 つであるコンテナの I/F の提供である．

Pod の I/F が作成されると，割り当てる IP アドレスを決定する．まず Concorde 実行時に，CNI 構成ファイルに記載されているネットワーク設定の内容が Concorde に渡され，Pod のサブネットを取得する．次に Pod のサブネットから，割り当て可能な IP アドレスを全通り，割り当て済み IP アドレス管理ファイルから，割り当て済みの IP アドレスをそれぞれ探索し，配列として保存する．このとき，ホスト部が全て 0 である IP アドレスはネットワークアドレスと定義し，ホスト部の末尾が 1，それ以外が全て 0 である IP アドレスは Pod のゲートウェイの IP アドレスと定義したため，Pod への割り当てを回避するために割り当て済み IP アドレス管理ファイルに強制的に追記する．

図 9 に，割り当て IP アドレスを決定する関数を示す．2 行目の `all_ips[@]` は割り当て可能な全ての IP アドレスであり，5 行目の `my_pod_ips[@]` は既に割り当てられている全ての IP アドレスである．この関数で，`all_ips[@]` と `my_pod_ips[@]` を比較することで，まだ割り当てられていない IP アドレスを決定する．関数の実行が終了すると，Pod に接続した I/F (eth0) に IP アドレスを付与する．

図 10 に帯域保証設定関数を示す．3〜4 行目の `$1` は `bandwidth_control` 関数の第一引数で，Pod の名前空間の ID である．帯域保証値は MANO 機構が割り当てることを想定しているが，MANO 機構には帯域管理機能がまだ実装されていない．したがって，現段階では割り当て帯域値を決定・管理する `bandwidth_allocate` 関数を別途実装し，割り当て帯域を決定している．`bandwidth_control` 関数の第二引数である 2 行目の $2 は Pod に割り当てる IP アドレスである．MANO 機構は IP アドレス毎の帯域管理を想定しているため，`bandwidth_allocate` 関数で IP アドレス毎に帯域を管理している．割り当て帯域が決定すると，tc コマンドによって帯域保証の設定をする．tc コマンドは qdisc (キューイング規則) というスケジューラで I/F を通るパケットを管理しており，キューイングによって，設定した I/F を通るトラフィックの帯域保証をすることができる．qdisc は classful qdisc という階層化構造をとることができ，階層化によってトラフィック毎にパケットを制御することができる．本稿では，3.2 節で述べたように，設定先 I/F や tc コマンドの実装の都合上，それぞれの Pod の I/F (eth0) に対して帯域保証の設定をしているため，1 階層構造としている．実際には，3 行目で qdisc の設定，4 行目で階層化の設定をしており，階層化設定の際に帯域値の設定もされる．設定する帯域値は 4 行目にある `rate` が帯域保証値，`ceil` が帯域制限値であり，3.2 節で述べたように帯域保証値と帯域制限値は同値にしている．

4.2 Pod 削除時

Pod 削除時には，まず削除される Pod の IP アドレスを取得し，割り当て済み IP アドレス管理ファイルから削除する．次に，Pod 作成時と同様に MANO 機構に代わって `bandwidth_collect` 関数を別途実装し，`bandwidth_allocate` 関数が割り当てた帯域値を回収する．その後，bridge と接続した veth を削除し，最後に Pod の名前空間を削除する．ここで，名前空間の削除とともに eth0 が削除され，eth0 の削除とともに tc の設定も削除されるため，eth0 と tc の設定の明示的な削除はしていない．

5. 評価

本稿では，アプリケーション開発者がアプリケーションのデプロイファイルに必要帯域を指定するだけで，帯域が保証されるネットワークを構築できる Concorde CNI plugin を設計・実装した．デプロイファイルは，アプリケーションを構成するために必要な CPU や RAM などの様々な情報をまとめたファイルであり，このデプロイファイルに必要帯域も指定する．本章では，Concorde CNI plugin が ContMEC 環境と Kubernetes 環境の両環境で tc コマンドが正常に動作し，帯域を指定するだけで帯域保証されたネットワークを構築できるかを検証する．

ⓒ 2022 Information Processing Society of Japan

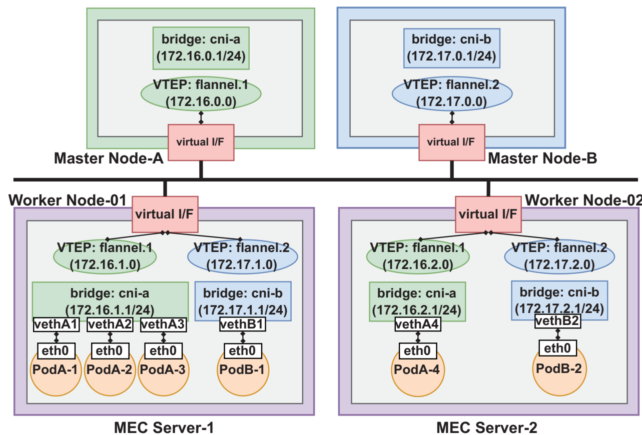

図 11: ContMEC 評価環境.

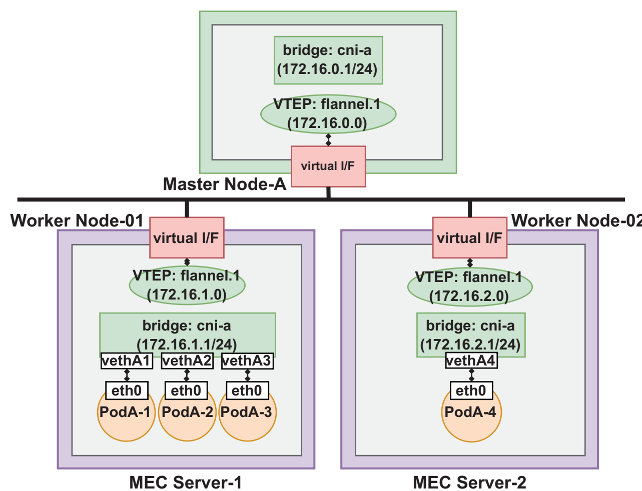

図 12: Kubernetes 評価環境.

5.1 評価環境

図 11 に Kubernetes を採用した ContMEC の評価環境を示し，図 12 に Kubernetes の評価環境を示す．ContMEC では Master Node と Worker Node を VM で 2 台ずつ用意し，それぞれの Master Node が同じ 2 台の Worker Node を管理している構成とした．Kubernetes では Master Node を 1 台，Worker Node を 2 台 VM で用意し，Master Node が 2 台の Worker Node を管理している構成にした．全ての VM は表 2 に示す環境を用いた．また，ContMEC では Worker Node-01 に 4 個，Worker Node-02 に 2 個，Kubernetes では Worker Node-01 に 3 個，Worker Node-02 に 1 個の Pod を配置し，両環境で Docker コンテナを 2 つの Worker Node それぞれに 1 つずつ配置した．

5.2 評価方法

評価は，iperf3 コマンドを使用して TCP トラフィックを発生させ，Pod 間や Docker コンテナ間のスループットを 100 ミリ秒間隔で測定した．トラフィックは ContMEC では図 11 における (1) PodA-1 → PodA-4，(2) PodA-2 → PodA-3，(3) PodB-1 → PodB-2 の方向に流した．Kubernetes では図 12 における (1) PodA-1 → PodA-4，(2) PodA-2 → PodA-3 に流した．また，両環境で Docker コンテナ間にも流した．表 3 は，両環境において，トラフィックを流す各 Pod 間に設定された帯域保証値を示しており，Docker コンテナ間では帯域設定をしていない．

表 2: 評価環境の詳細 (全 VM 共通).

項目	内容，バージョン
OS	Ubuntu 18.04 LTS
CPU	vCPU × 2
RAM	16 GB
Storage	16 GB
K8s, kubeadm	v1.22.2

表 3: 各 Pod 間の帯域保証値.

	始点 Pod	終点 Pod	帯域保証値
(1)	PodA-1	PodA-4	1 Gbps
(2)	PodA-2	PodA-3	3 Gbps
(3)	PodB-1	PodB-2	4 Gbps

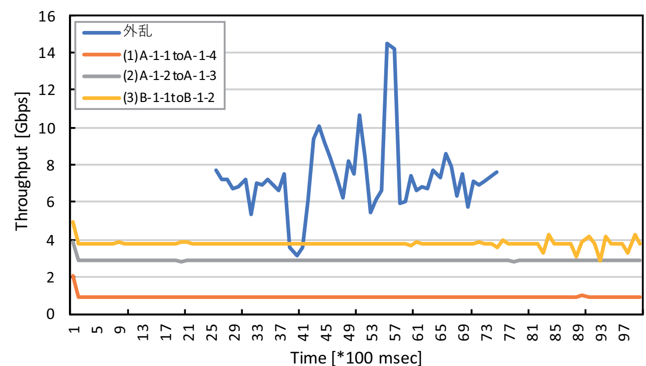

図 13: Pod 間スループット (ContMEC).

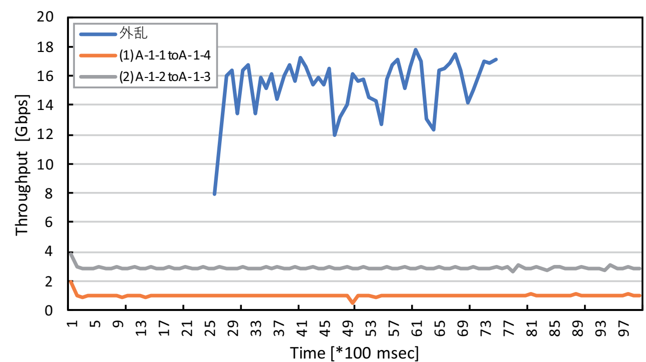

図 14: Pod 間スループット (Kubernetes).

5.3 評価結果

各 Pod 間のトラフィックを 0 秒から 10 秒の 10 秒間，コンテナ間の外乱トラフィックを 2.5 秒から 7.5 秒の 5 秒間流した際のスループットを 100 ミリ秒単位で測定した．ContMEC での測定結果を図 13 に，Kubernetes での測定結果を図 14 に示す．図 13 および図 14 から，Pod 間のみ

にトラフィックを流した際，どの Pod 間でも表 3 の値で設定したスループットを示している．また，外乱トラフィックを発生させた場合でも，Pod 間トラフィックは設定された値のスループットを保ったままである．したがって，Concorde CNI plugin が ContMEC 環境と Kubernetes 環境の両環境で正常に動作し，tc コマンドによる設定が正常に動作していることが分かった．また，必要帯域を指定するだけで，ネットワークが輻輳している状況でも帯域保証されたネットワークを構築できたことが分かった．

6. 議論

本稿では，始点 Pod 毎に送信方向への通信に対して帯域保証の設定をしたが，始点 Pod 毎に加えて終点 Pod 毎でも帯域保証するためには，他のノード上で起動している Pod の IP アドレスを取得する必要がある．しかし，CNI plugin は CNI plugin が存在するノード上のネットワークのみをサポート範囲としているため，他のノード上の Pod の IP アドレスは把握していない．そのため，CNI plugin を実行する際に，各ノード上の CNI plugin が割り当てる IP アドレス情報を共有する必要がある．この IP アドレス情報の共有は，MANO 機構に実装することを想定しているため，CNI plugin は MANO 機構に指示された IP アドレスと割り当て帯域を元に帯域保証の設定をする．したがって，各 Pod 間毎に帯域保証する場合，IP アドレス情報共有システムを MANO 機構に別途実装する必要がある．

前述のように，CNI plugin は CNI plugin が存在するノード上のネットワークのみをサポート範囲としているため，トラフィックがノード外に転送された後は CNI plugin のサポート範囲外である．したがって，物理 I/F を通過した後のトラフィックはベストエフォートで扱われる．ノード外に転送されたトラフィックを確実に帯域保証するためには，ノード内だけでなくノード外のネットワーク上にも帯域保証機構が必要となる．しかし，Kubernetes はノード間のネットワーク上のルータやスイッチは対象外であるため，ノード間のネットワーク上に MANO 機構等を独自に設計する必要がある．また，Kubernetes は元々 overlapped worker 環境を想定していない．そこで ContMEC では Kubernetes を overlapped worker 環境に対応させている．CNI plugin も overlapped worker 環境を想定していないため，ContMEC では Flannel を同じように overlapped worker 環境に対応させている．加えて，Kubernetes を採用して ContMEC を実現するためには，Pod の適切な配置先ノードを決定する MANO 機構や，MANO 機構と通信をする機構など，独自に設計・実装すべきものが数多くあり，都合が悪い点が多い．したがって，Kubernetes 以外のコンテナオーケストレーションシステムを用いて ContMEC を実現する必要がある．

7. おわりに

本稿では帯域保証機能を有する Concorde CNI plugin を提案した．Concorde により，アプリケーション開発者がアプリケーションのデプロイファイルに必要帯域を指定するだけで帯域保証されるネットワークを構築でき，ContMEC 環境での適用も実現した．評価では，ContMEC と Kubernetes における Pod 間やコンテナ間のスループットを測定した．測定結果より，Concorde が正常に動作し，帯域保証されたネットワークを構築できたことを確認した．一方，CNI plugin のサポート範囲は CNI plugin が存在するノード内であるため，ノードを跨ぐ Pod 間通信はベストエフォートで扱われるという課題がある．今後は，Kubernetes 以外のコンテナオーケストレーションシステムを用いた ContMEC を想定し，CNI plugin 以外の方法による帯域保証されたネットワークの構築を目指す．

参考文献

[1] Nguyen, C., Mehta, A., Klein, C. and Elmroth, E.: Why Cloud Applications Are Not Ready for the Edge (Yet), *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, Association for Computing Machinery, pp. 250–263 (2019).
[2] Porambage, P., Okwuibe, J., Liyanage, M., Ylianttila, M. and Taleb, T.: Survey on Multi-Access Edge Computing for Internet of Things Realization, *IEEE Communications Surveys Tutorials*, Vol. 20, No. 4, pp. 2961–2991 (2018).
[3] Hu, Y. C., Patel, M., Sabella, D., Sprecher, N. and Young, V.: Mobile edge computing—A key technology towards 5G, *ETSI white paper*, Vol. 11, No. 11, pp. 1–16 (2015).
[4] Watanabe, H., Yasumori, R., Kondo, T., Kumakura, K., Maesako, K., Zhang, L., Inagaki, Y. and Teraoka, F.: ContMEC: An Architecture of Multi-access Edge Computing for Offloading Container-Based Mobile Applications, *Proceedings of 2022 IEEE International Conference on Communications (ICC)*, pp. 1–7 (2022). (to appear).
[5] Cloud Native Computing Foundation: Kubernetes. https://kubernetes.io.
[6] CoreOS, Inc: Flannel. `https://github.com/flannel-io/flannel`.
[7] Claassen, J., Koning, R. and Grosso, P.: Linux containers networking: Performance and scalability of kernel modules, *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 713–717 (2016).
[8] Tigera, Inc: Calico. `https://projectcalico.docs.tigera.io/about/about-calico`.
[9] CA.NIC, Inc: BIRD internet routing daemon. `https://bird.network.cz`.
[10] Isovalent, Inc: Cilium. `https://cilium.io/learn`.
[11] The Linux Documentation Project: Linux Advanced Routing Traffic Control HOWTO. `https://tldp.org/HOWTO/Adv-Routing-HOWTO/index.html`.

ⓒ 2022 Information Processing Society of Japan