マルチテナント向けコンテナ環境における軽量かつ柔軟なARPスプーフィング対策の実現

2020/2/28

情報処理学会
第148回 システムソフトウェアとオペレーティング・システム研究会

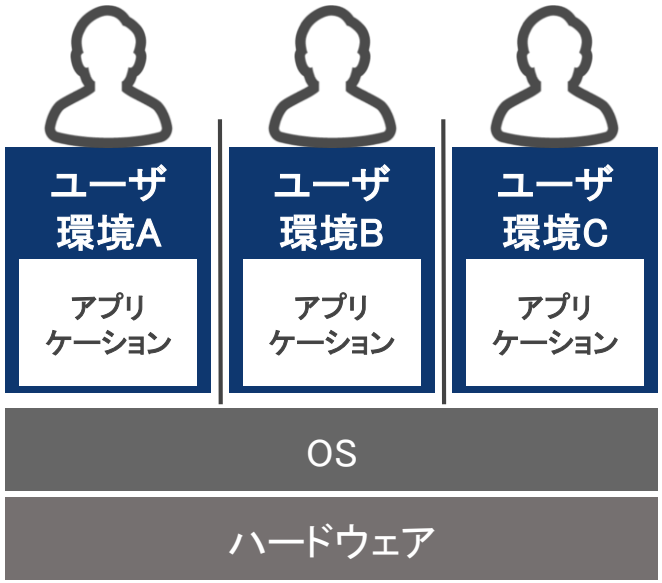
中田裕貴[*1], 松原克弥[*1], 松本亮介[*2]

*1 公立はこだて未来大学
*2 さくらインターネット株式会社 さくらインターネット研究所

背景：クラウドコンピューティング基盤の環境隔離

マルチテナント：ユーザ間でCPUやメモリ等のリソースを共有

- 複数ユーザのアプリケーションを共有マシンで運用するには，
  リソースの分配と環境の隔離が必要
  - **仮想化技術が広く用いられる**
- PaaS*1やFaaS*2と呼ばれるサービス形態では，ユーザ環境の隔離に
  コンテナ型仮想化技術の使用が増加
  - リクエスト毎など，頻繁にユーザ環境の
    起動と終了が行われるため，
    起動時間の軽量さが求められる
  - 使用されるコンテナランタイムの例：
    Docker[1]，LXD[2]，Haconiwa[3]



*1PaaS : Platform as a Service, *2FaaS : Function as a Service

[1]Docker Inc.: Docker. https://www.docker.com/,(参照2019-01-21).
[2]Canonical Ltd.: Linux Containers. https://linuxcontainers.org/lxd,(参照2019-01-21)
[3]Uchio kondo.: Haconiwa : MRuby on Container,https://github.com/haconiwa/haconiwa,(参照2019-01-21)

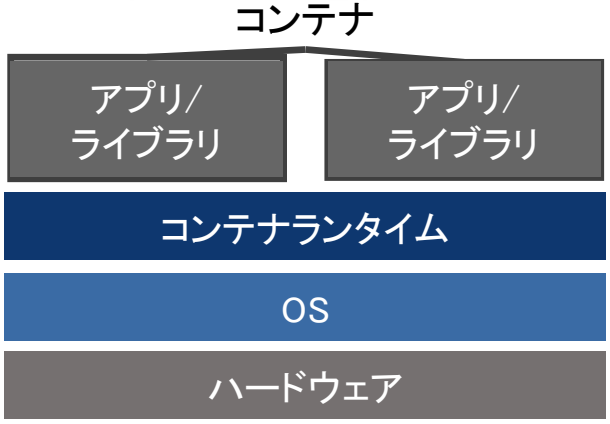
コンテナ型仮想化技術

独立したOS環境を実現する仮想化技術

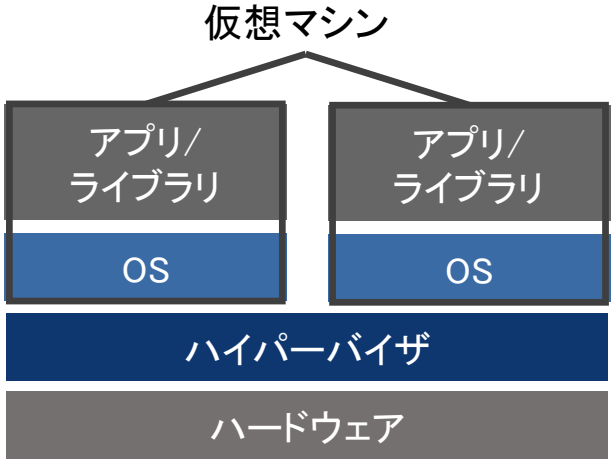
- コンテナランタイムはLinux NamespacesやLinux cgroups，chrootなどLinuxのOS機能を用いてOSリソースを多重化

  e.g. ファイルシステムやプロセスID等

- OSカーネルを共有したままユーザが使用する環境の隔離を実現
- ハードウェア仮想化技術と比べて複数のOS起動を行わないため軽量に環境隔離を実現



コンテナ型仮想化技術



ハードウェア仮想化技術

課題：コンテナに付与するネットワーク権限の粒度

隔離から抜けることが可能なネットワークリソースは

コンテナ内での実行が制限

e.g. 低レベルネットワークインターフェース，

ネットワークに関連したカーネルモジュール等

- コンテナ内でネットワークスタック等の機能を扱う

  ソフトウェアはこれらの制限されたネットワークリソースが必要

  e.g. pingコマンド，dnsmasq(DNS)

  - pingはrawソケット(低レベルネットワークインターフェース)を使用
- 制限を解除するには，特別な実行権限を付与する必要

特別な実行権限を付与する方法

1. root権限の付与
2. Linux Capabilitiesを用いた実行権限の付与

root権限の付与とその課題

root権限：OSの全リソースへアクセスできる権限

- 付与することで全ネットワークリソースへのアクセスだけではなく，
  OS機能の全権限が利用可能

- 共存するコンテナに影響をあたえるアクセスや制御が可能
  e.g. 他コンテナが使用しているファイルの書き換え

Linux Capabilitiesを用いた権限付与とその課題

Linux Capabilities：root権限を37種類に分割

- 分割された権限を組み合わせて権限付与

  e.g. CAP_NET_RAW：低レベルネットワークインターフェースの使用を許可

- 37種類の分割では粒度が粗い

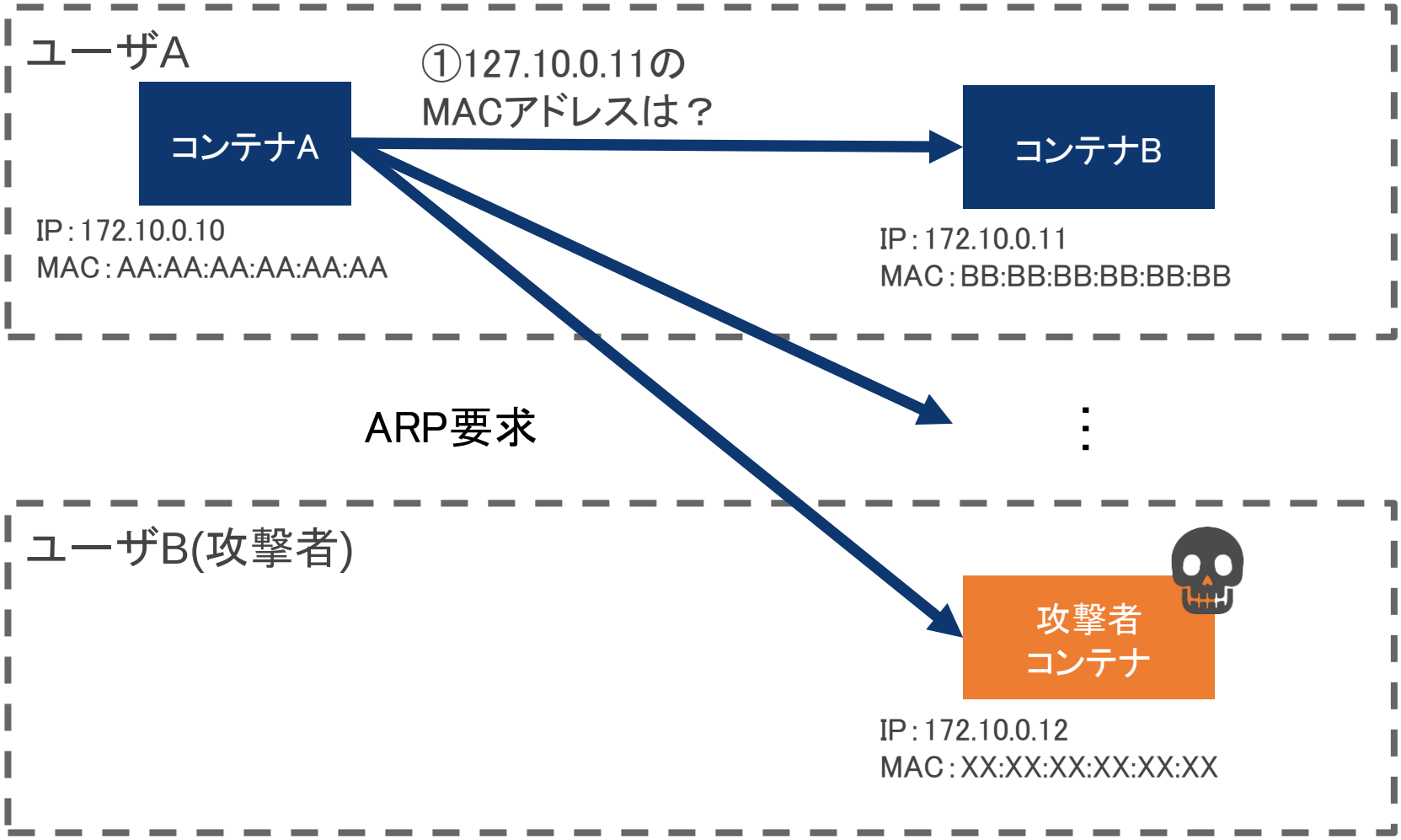
  e.g. コンテナ内でpingを使用するためにCAP_NET_RAWを付与

  - OSのTCP/IPプロトコルスタックを迂回して直接データリンク層とのデータ通信が可能
  - マルチテナント内の別ユーザが使用するコンテナへARP スプーフィング攻撃が可能[4][5]

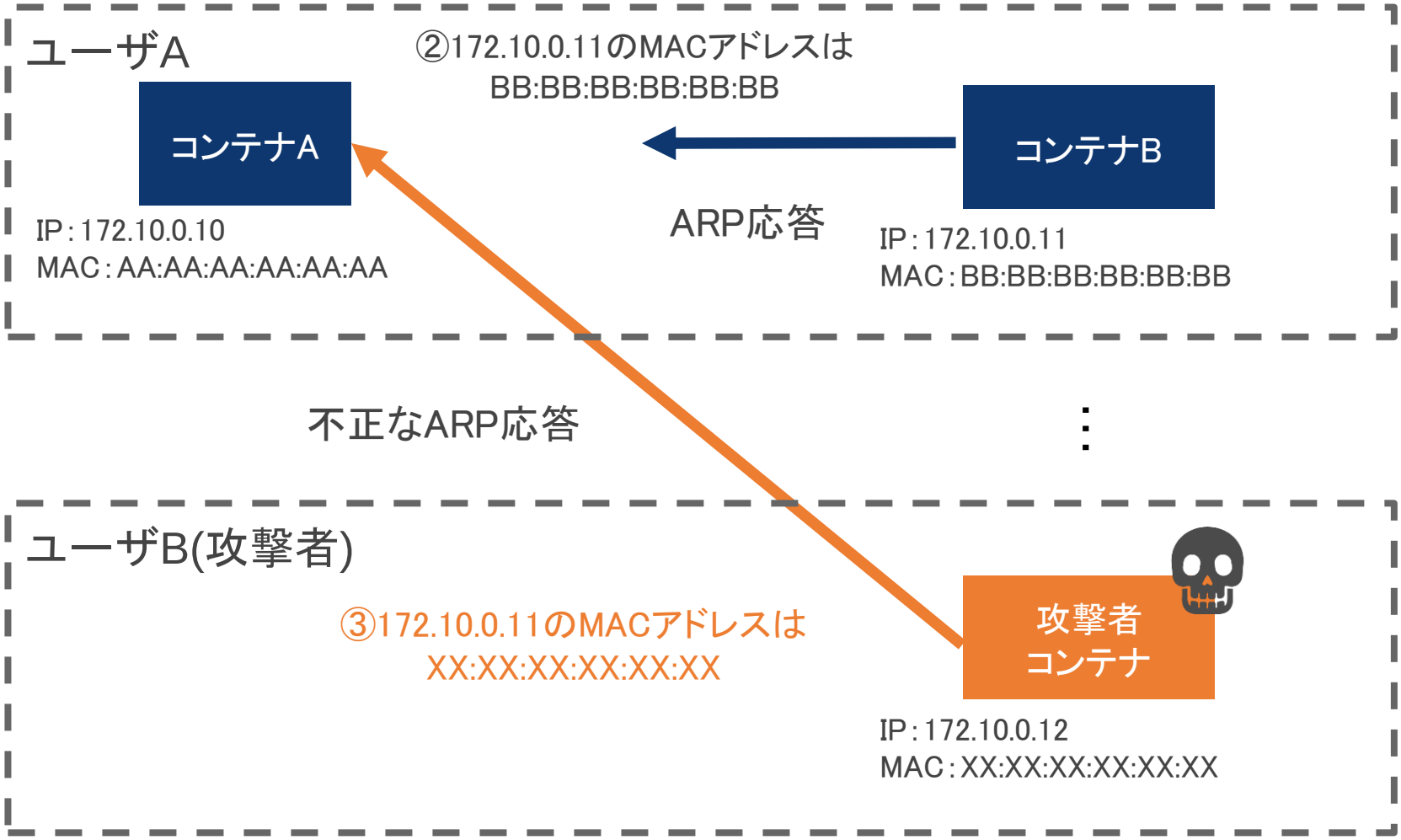
[4]Hertz, J.: Abusing privileged and unprivileged linux containers, Whitepaper, NCC Group, Vol. 48 (2016).

[5]Liz, R.: KubeCon + CloudNativeCon North America 2019:CAP NET RAW & ARP Spoofing in Your Cluster.

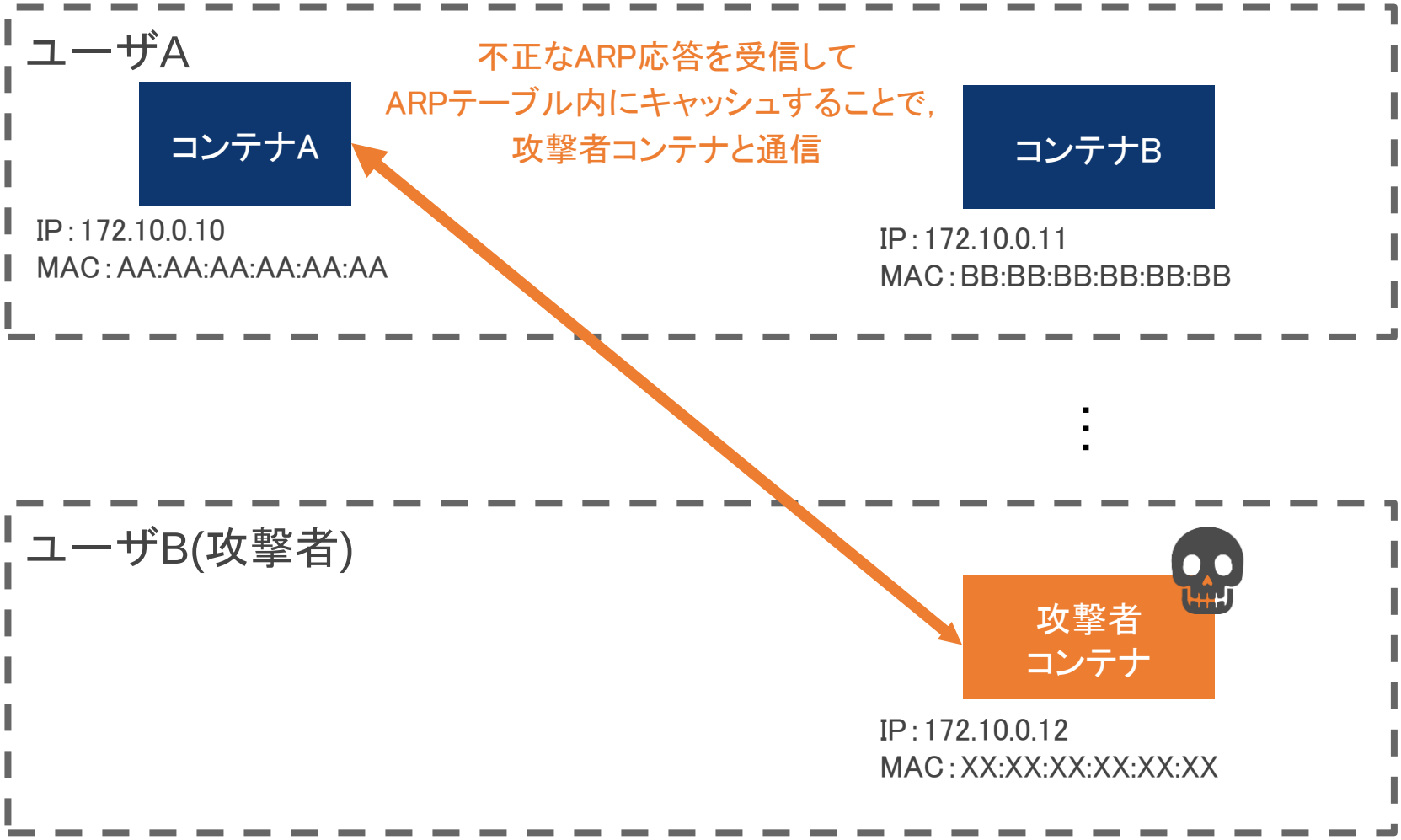
マルチテナント環境における ARPスプーフィング攻撃の例(1/3)



マルチテナント環境における ARPスプーフィング攻撃の例(2/3)



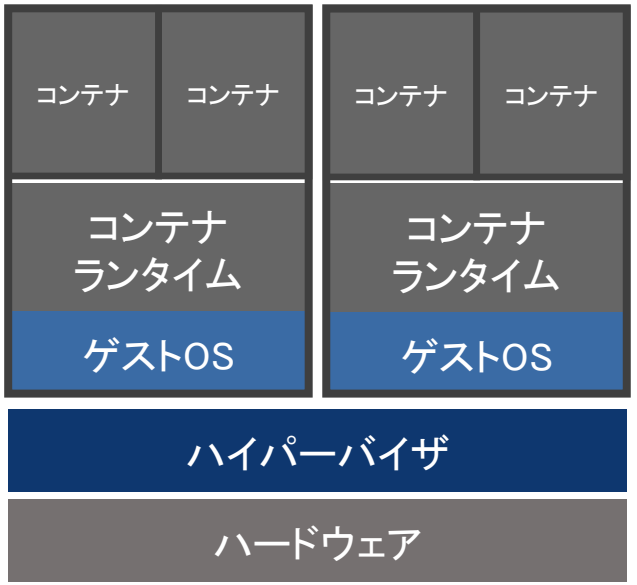
マルチテナント環境における ARPスプーフィング攻撃の例(3/3)



関連研究1: 仮想マシンによるネットワークリソース多重化

概要

- Kata Containers[6]はPod(コンテナ群)毎にハードウェア仮想化を用いて仮想マシンを生成
  - Pod毎にネットワークリソースの多重化
  - ネットワークリソースの多重化によって隔離から抜け，他コンテナに対する攻撃を防止



課題

- 起動時間にかかるオーバヘッド[7]
  - コンテナ生成毎に仮想マシン作成とOS起動

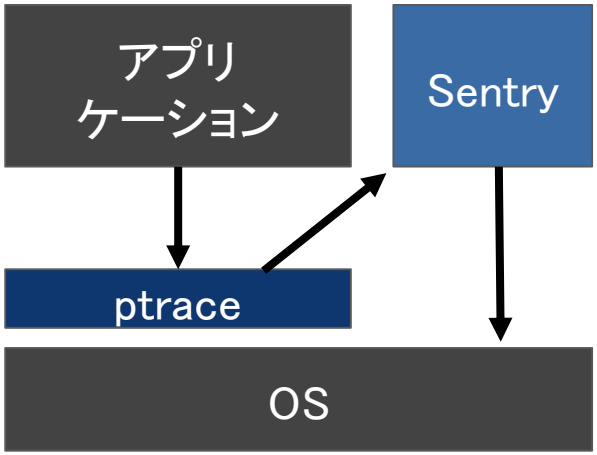
[6]The OpenStack Foundation: Kata Containers: The speed of containers, the security of VMs, https://katacontainers.io (参照 2019-01-21).

[7]Xu,W.: KubeCon+CloudNativeCon North America 2018: Kata and gVisor :A Quantitative Comparison

関連研究2: ユーザランドにおけるネットワークリソース再構築

概要

- gVisor[8]はSentryと呼ばれるユーザ空間カーネルでコンテナを動作
  - コンテナが発行するシステムコールをptraceでフック
- Sentryで再実装された安全なシステムコールを使用して他コンテナへの攻撃面を削減
- プロトコルスタックは再構築することで隔離から抜け，他コンテナに対する攻撃を防止



課題

- ネットワークにかかるオーバヘッド[9]
  - プロトコルスタックのユーザ空間上での再実装

[8]google LLC: gvisor: container runtime sandbox, https://github.com/google/gvisor (参照 2019-01-21).

[9]Young, E. G., Zhu, P., Caraza-Harter, T., Arpaci-Dusseau, A. C. and Arpaci-Dusseau, R. H.

: The true cost of containing: A gVisor case study, 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19) (2019).

関連研究3: OSカーネルのセキュリティ機能を用いた手法

概要

- AppArmor[10]はプログラム単位でセキュリティポリシに基づいて強制アクセス制御を実現
- RAWソケットを拒否し，ICMPを許可といった，Linux Capabilitiesより細かい粒度の制御が可能

課題

- セキュリティポリシの柔軟性
  - 宛先ごとのパケット許可・不許可のような柔軟な設定はできない

[10]Ubuntu Documentation Team: AppArmor,https://wiki.ubuntu.com/AppArmor(参照2019-01-21)

関連研究4: コンテナのセキュリティ機能を用いた手法

概要

- Calico[11]はコンテナのネットワーキング機能とセキュリティ機構を提供
- コンテナ群ごとのセキュリティポリシに，
  IPアドレスやプロトコル単位でのアクセス制御を記述することが可能

課題

- 補足できないパケットの存在
  - コンテナ型仮想化技術と共に，Linuxカーネル内で動作するeBPFや
    カーネル空間をバイパスするDPDKの利用が注目
  - リファレンスモニタという考え方では，
    全アクセスに対して必ず介在可能であることが必須
  - コンテナでeBPFやDPDKが用いられた場合，パケットに必ず介在できない

[11]Tigera, Inc.: Project Calico – Secure networking for thecloud native era,https://www.projectcalico.org/(参照2019-01-21)

マルチテナントにおけるARPスプーフィング攻撃を防止する軽量かつ柔軟なネットワーク隔離実現の要件

1. コンテナからのパケット送受信に介在が可能であること

   理由：eBPFやDPDKのようなアーキテクチャを用いられても

   適切なアクセス制御を実現するため

2. 軽量にネットワーク隔離が実現可能であること

   理由：ネットワークリソースの隔離性向上によって，

   ネットワーク性能が大きく低下するのを防ぐため

3. セキュリティポリシを柔軟に記述可能であること

   理由：複数の制御条件の考慮や，多数のコンテナが増減を繰り返すことで

   制御内容が頻繁に変化する環境においても，ネットワーク隔離を実現するため

4. コンテナが迅速に起動可能であること

   理由：本発表の対象であるマルチテナント環境では，

   コンテナが頻繁に起動・終了を繰り返すため

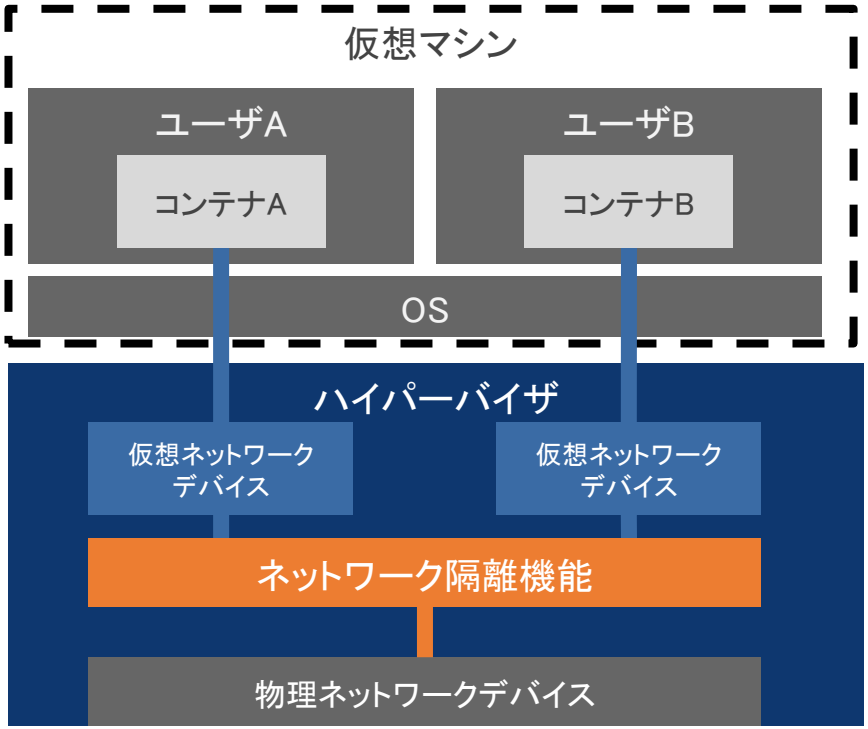
本研究の目的と提案

目的

- 不必要な権限付与によるネットワークリソースを用いた他コンテナへの攻撃を防止する軽量かつ柔軟なネットワーク隔離の実現
  - 本発表は，ARPスプーフィング攻撃を対象に説明

提案

- ハードウェア仮想化技術を用いてパケットを捕捉・制御することで，軽量かつ柔軟なネットワーク隔離を実現
- 実現には仮想ネットワークデバイスとハイパーバイザを使用



本提案手法と要件

1. コンテナからのパケット送受信に介在が可能であること

   ハードウェア仮想化技術を使用してネットワークデバイスへのパケットを
   捕捉し，OS上では捕捉できない全パケットへの介在を実現

2. 軽量にネットワーク隔離が実現可能であること

   ハードウェア仮想化技術を用いてCPUのハードウェア仮想化支援機能を
   活用し，ユーザランドでのリソース再構築より小さいオーバヘッドを実現

3. セキュリティポリシを柔軟に記述可能であること

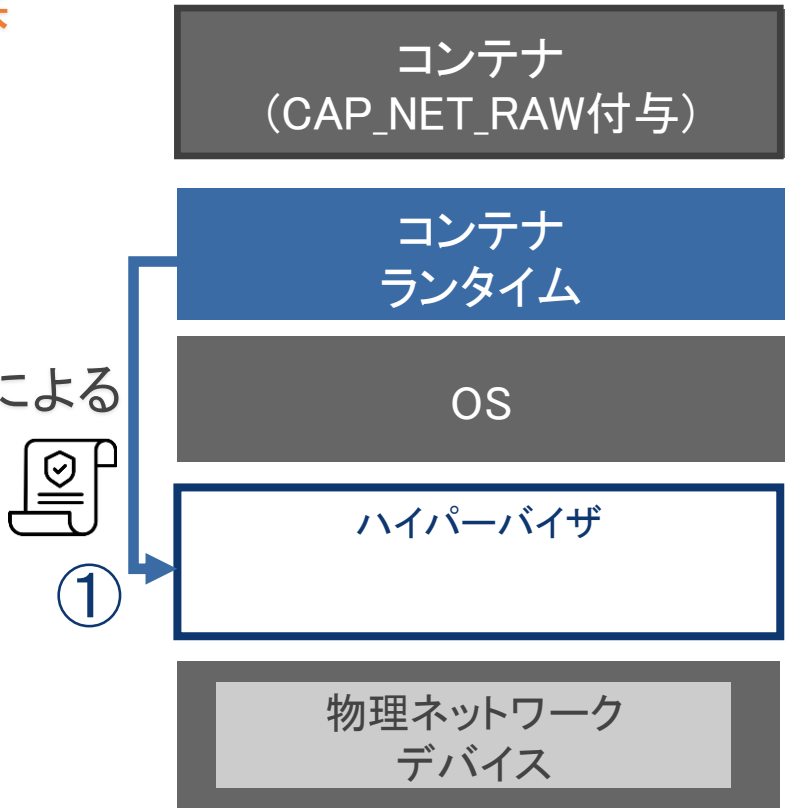
   プログラムとして記述する設定ファイルを用いることで，複数条件を考慮し
   多数のコンテナが増減する環境においても，柔軟な設定を実現

4. コンテナが迅速に起動可能であること

   OSカーネル全体の多重化は行わず，隔離が必要なネットワークリソースのみ
   仮想化することで，コンテナの迅速な起動を実現

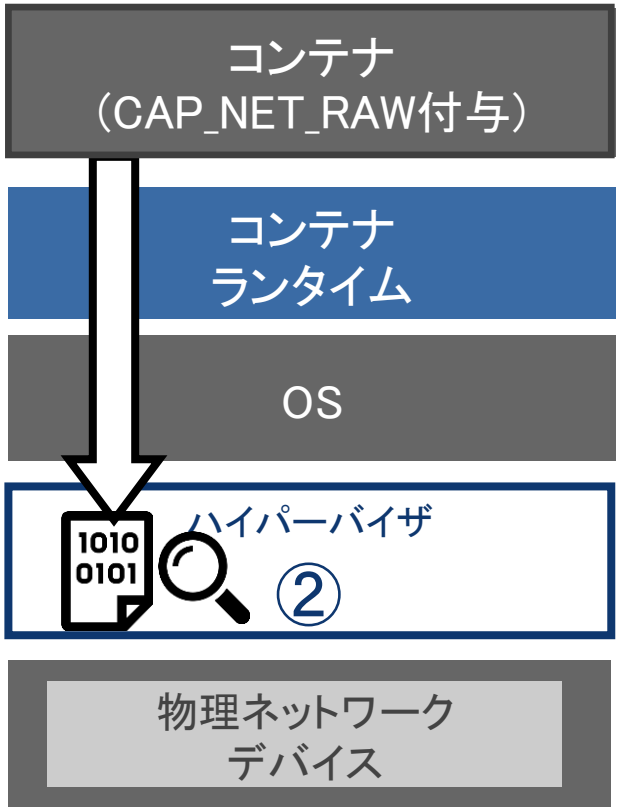
提案手法を用いたARPスプーフィング攻撃防止の流れ

1. コンテナランタイムがコンテナ作成時に
   セキュリティポリシをハイパーバイザに提供
2. ハイパーバイザで
   OSから送信されたパケットを監視
3. パケットからコンテナ特定
4. パケットの内容を解析，セキュリティポリシによる
   アクセス内容の検査
5. ポリシに基づいて送信する前に
   パケットを破棄



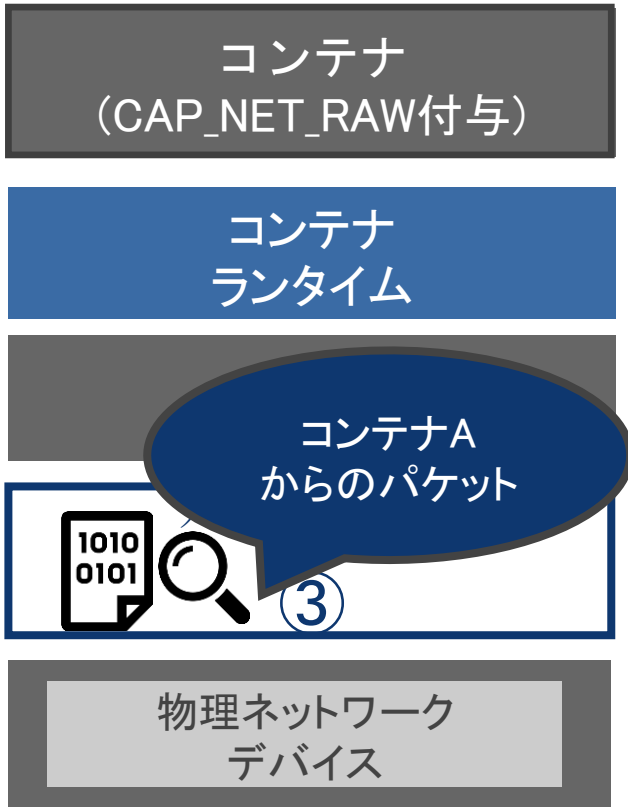
提案手法を用いたARPスプーフィング攻撃防止の流れ

1. コンテナランタイムがコンテナ作成時に
   セキュリティポリシをハイパーバイザに提供
2. ハイパーバイザで
   OSから送信されたパケットを監視
3. パケットからコンテナ特定
4. パケットの内容を解析，セキュリティポリシによる
   アクセス内容の検査
5. ポリシに基づいて送信する前に
   パケットを破棄



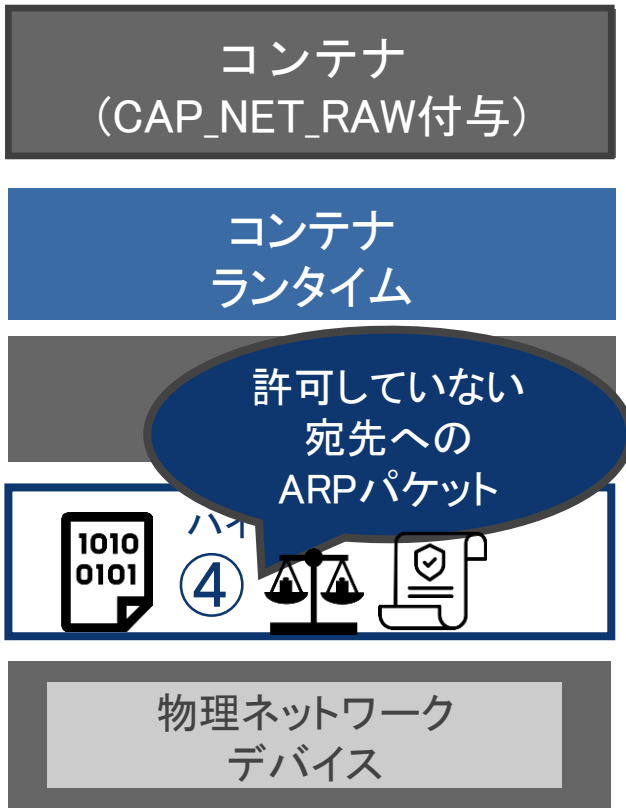
提案手法を用いたARPスプーフィング攻撃防止の流れ

1. コンテナランタイムがコンテナ作成時に
   セキュリティポリシをハイパーバイザに提供
2. ハイパーバイザで
   OSから送信されたパケットを監視
3. パケットからコンテナ特定
4. パケットの内容を解析，セキュリティポリシによる
   アクセス内容の検査
5. ポリシに基づいて送信する前に
   パケットを破棄



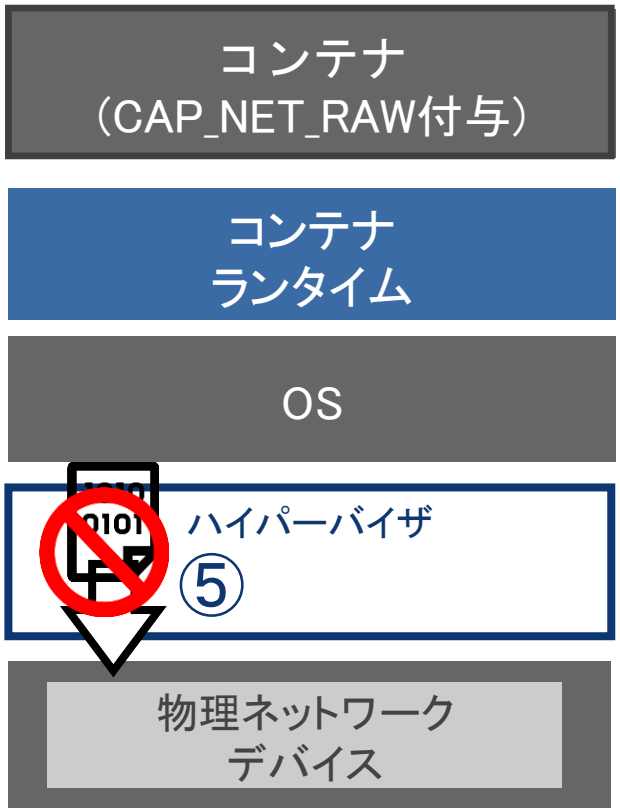
提案手法を用いたARPスプーフィング攻撃防止の流れ

1. コンテナランタイムがコンテナ作成時に
   セキュリティポリシをハイパーバイザに提供
2. ハイパーバイザで
   OSから送信されたパケットを監視
3. パケットからコンテナ特定
4. パケットの内容を解析，セキュリティポリシによる
   アクセス内容の検査
5. ポリシに基づいて送信する前に
   パケットを破棄



提案手法を用いたARPスプーフィング攻撃防止の流れ

1. コンテナランタイムがコンテナ作成時に
   セキュリティポリシをハイパーバイザに提供
2. ハイパーバイザで
   OSから送信されたパケットを監視
3. パケットからコンテナ特定
4. パケットの内容を解析，セキュリティポリシによる
   アクセス内容の検査
5. ポリシに基づいて送信する前に
   パケットを破棄



提案手法実現における技術的課題

1. 柔軟なセキュリティポリシの記述

理由：全てのパケットを拒否するのではなく，隔離が必要なパケットのみ制御

→ IPアドレス毎などの柔軟なセキュリティポリシ記述

2. コンテナ間の共有リソースへのアクセス捕捉

理由：全リソースではなく，隔離の必要があるリソースへのアクセスのみ捕捉

→ 特定のデバイスのアクセスのみを選択してフックする機構

3. アクセス元コンテナの特定

理由：コンテナはOSプロセス上で構成される概念であり，

ハイパーバイザでコンテナを識別不可能

→ プロセスの解析機構とコンテナ特定機構

4. コンテナ別のアクセス制御

理由：提供されたセキュリティポリシに沿ってアクセス制御する必要

→ パケットの内容解析

課題1:柔軟なセキュリティポリシの記述

ポリシをプログラムとして記述

- セキュリティポリシはmrubyのコードとして記述
  - IPアドレス単位や条件を用いた柔軟な設定が可能
- Haconiwaはコンテナの起動や終了時にmrubyコードを実行可能[14]
  - コンテナ起動時にハイパーバイザコールを使用して
    ハイパーバイザへセキュリティポリシを提供
    - OSがハイパーバイザを呼び出す専用命令

```
config.add_general_hook :before_start_wait do |base|
  cvisor = CVisor.new(base.pid)          #プロセスグループIDをBitVisorに送信
  cvisor.allow_ipaddr "192.168.2.100" #192.168.2.100へのパケット送信を許可
end
```

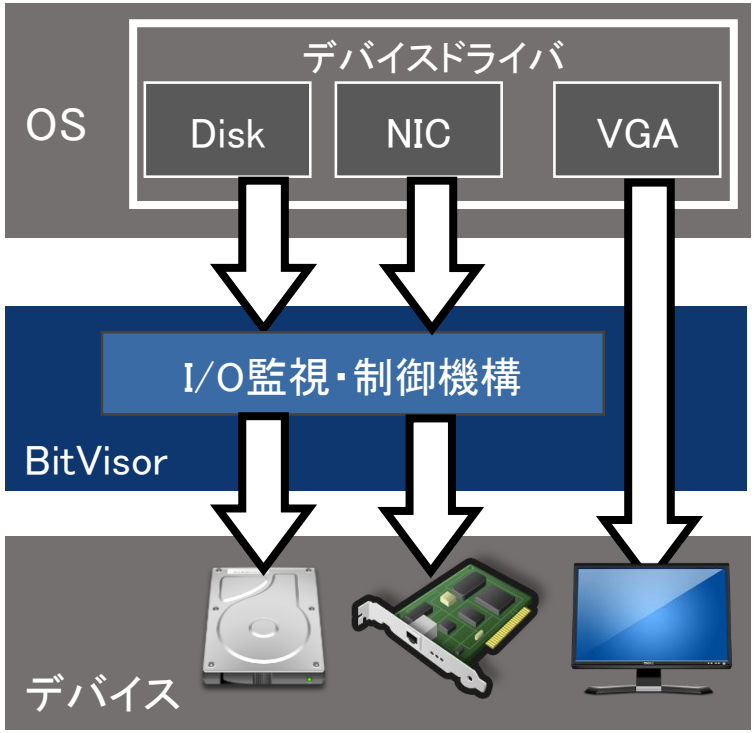
設定ファイル例(コンテナ起動時に実行)

[14]近藤宇智朗,松本亮介,栗林健太郎:
Haconiwa: プログラムによる,組み立て可能性と拡張性を持つ Linux コンテナ,第 80 回全国大会講演論文集, Vol. 2018, No. 1, pp.19-20 (2018).

課題2:ネットワークリソースへのアクセス捕捉

解決策:軽量ハイパーバイザBitVisor[15]の活用

- OSからデバイスへのI/Oを監視・制御
- 準パススルー型アーキテクチャ
  - 全てのデバイスを仮想化せず
    必要最低限のI/Oのみを捕捉
  - 複数のOSを動作させる機能は省略



[15]Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K.,Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E. and et al.: BitVisor: A Thin Hypervisor for Enforcing i/o Device Security, Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Association for Computing Machinery, pp. 121–130 (2009).

課題3:アクセス元コンテナの特定

コンテナの特定の流れ

1. パケットを発行したプロセスの解析
2. プロセスが所属しているコンテナの特定

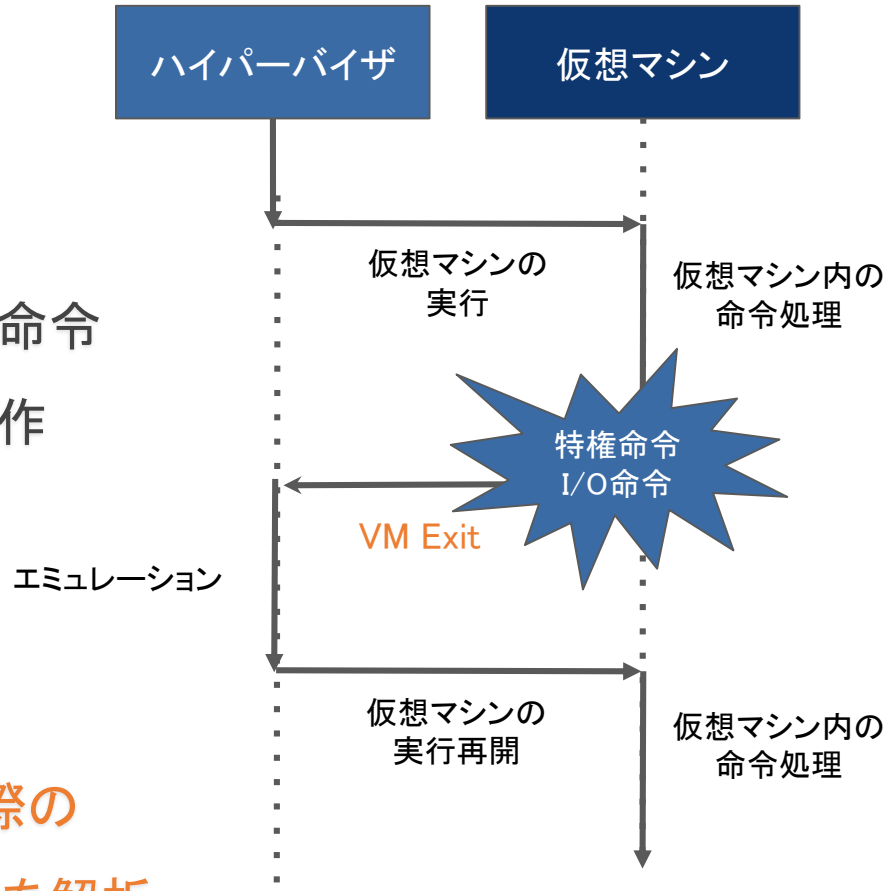
パケットを発行したプロセスの解析(1/2)

ハードウェア仮想化支援機能を活用

Intel VT-x：Intel製CPUの仮想化技術の
性能向上が目的の機能[16]

VMExit

- OSによってCPUの特権命令やI/O命令などが発行される際に発生する動作
- 発生するとOSが使用していたレジスタの内容が退避され，ハイパーバイザに制御が遷移



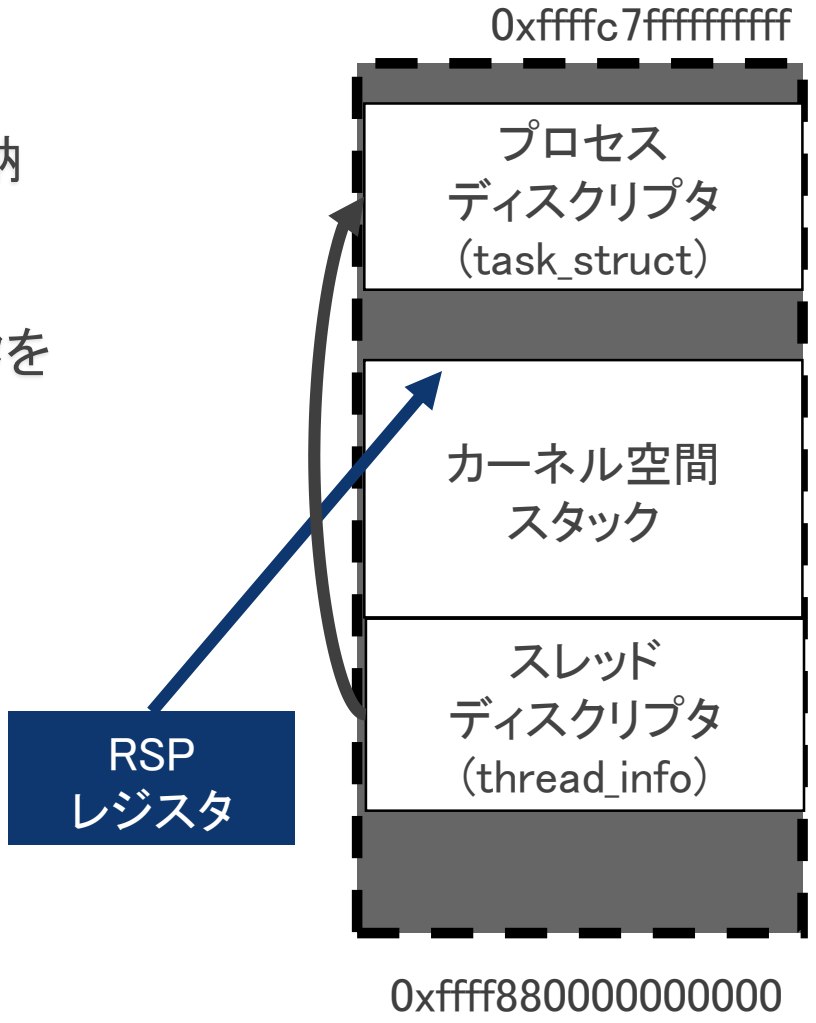
I/Oが送信され，VMExitが発生した際の
退避されたレジスタの内容からプロセスを解析

[16]Uhlig, R., Neiger, G., Rodgers, D., Santoni, A. L., Martins, F. C. M., Anderson, A. V., Bennett, S. M., Kagi, A., Leung, F. H. and Smith, L.: Intel virtualization technology, Computer, Vol. 38, No. 5, pp. 48-56 (2005).

パケットを発行したプロセスの解析(2/2)

解析対象のレジスタ：RSPレジスタ

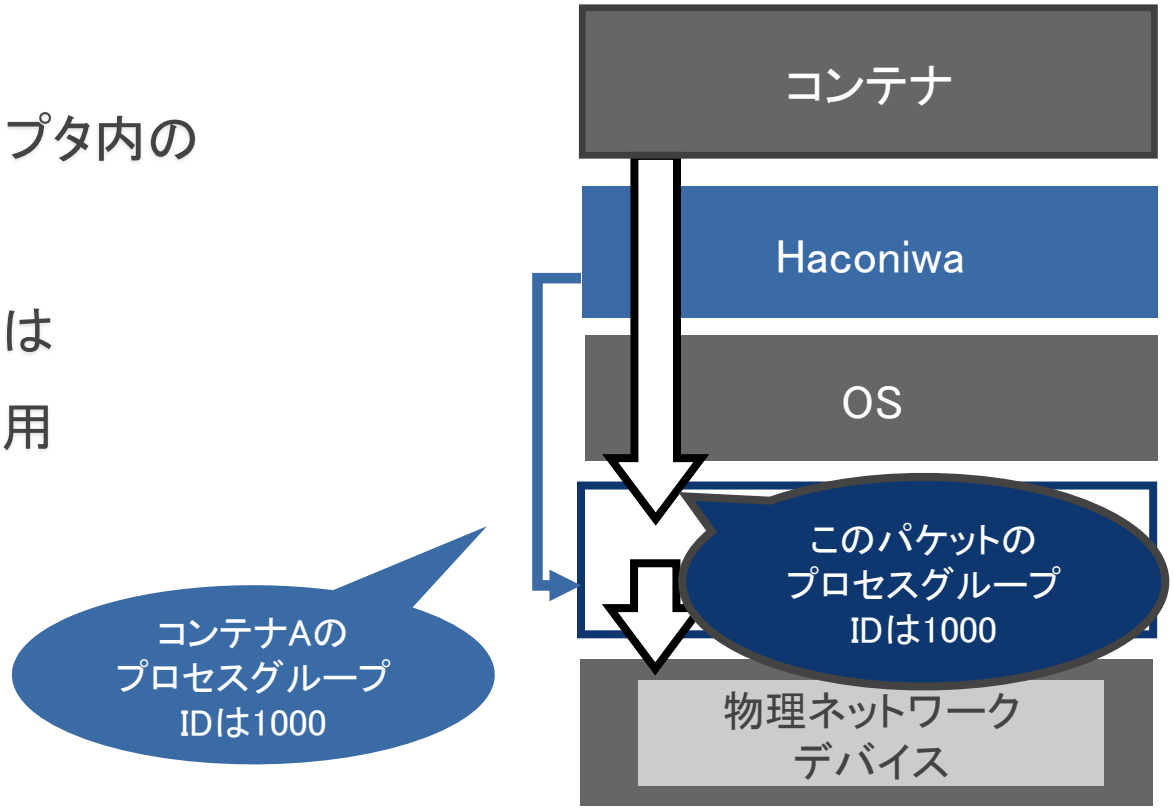
- 対象Linuxカーネル:3.10(64bit)
- カーネル空間スタックの末端アドレスを格納
- 末端アドレスからベースアドレスを算出
  - Linuxで実行中スレッドのディスクリプタを取得し，プロセスディスクリプタを特定
  - どのプロセスからのI/Oか判断を実現



プロセスが所属しているコンテナの特定

HaconiwaとBitVisorの連携

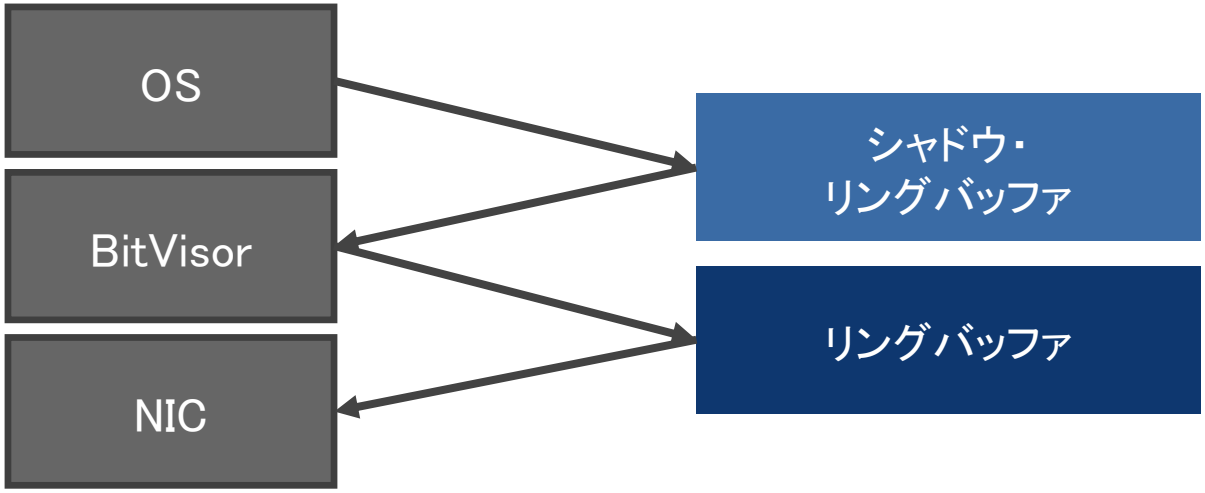
- コンテナ起動時にコンテナのプロセスグループIDをBitVisorに伝達
- 解析したプロセスディスクリプタ内のプロセスグループIDと比較
- プロセスグループIDの伝達はハイパーバイザコールを使用



課題4:コンテナ別のアクセス制御(1/2)

BitVisorでのパケット制御

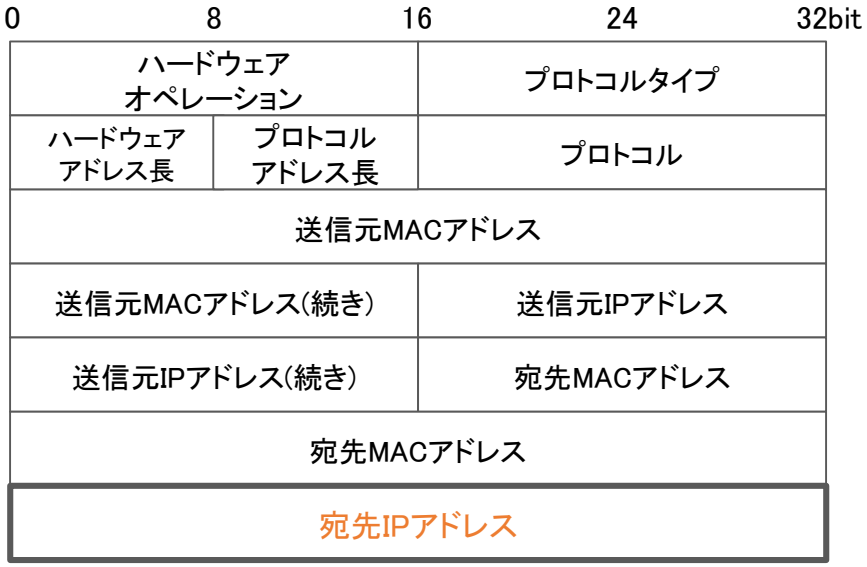
- BitVisorはパケット送信に使用するリングバッファを多重化
  - パケットの送受信への介在を実現
- BitVisorは2つのバッファを適切なタイミングで同期
  - バッファの同期する際にパケットを解析



課題4:コンテナ別のアクセス制御(2/2)

パケットの宛先特定とアクセス制御

- ARPパケットを解析
  - 宛先IPアドレス領域を確認して宛先を取得
- アクセス制御
  - セキュリティポリシで許可された宛先か確認
  - 許可されていない場合は，バッファの同期をスキップしてパケットを破棄



評価実験1：ネットワークのオーバヘッドを計測

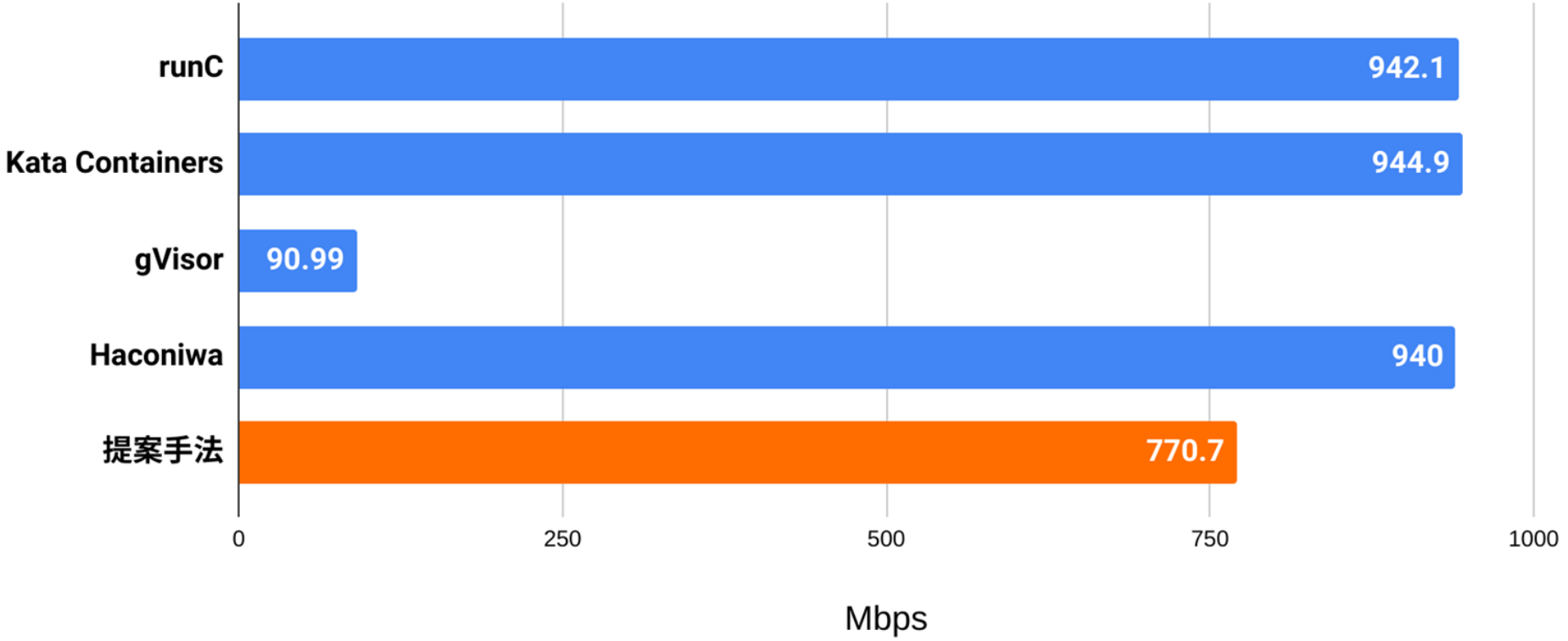
要件(2)：軽量にネットワーク隔離が実現可能であるか評価

- iperf3を用いたスループットの測定
- IPv4のUDPパケットを，1Gbpsの帯域幅で10秒(1秒×10回)送信
- 10回計測した値の平均値を評価
- 実験対象はrunC(プロセス分離コンテナ)，Kata Containers，gVisor，Haconiwa，本提案手法

実験環境(クライアント側)

OS	CentOS 7
Linux カーネル	3.10（gVisor の評価には 5.4 を使用）
CPU	Intel（R） Core（TM） i5-6260U CPU 1.80GHz
RAM	8GByte
NIC	Intel Coporation Ethernet Connection I219-V （rev 21）

評価実験1の実験結果・考察



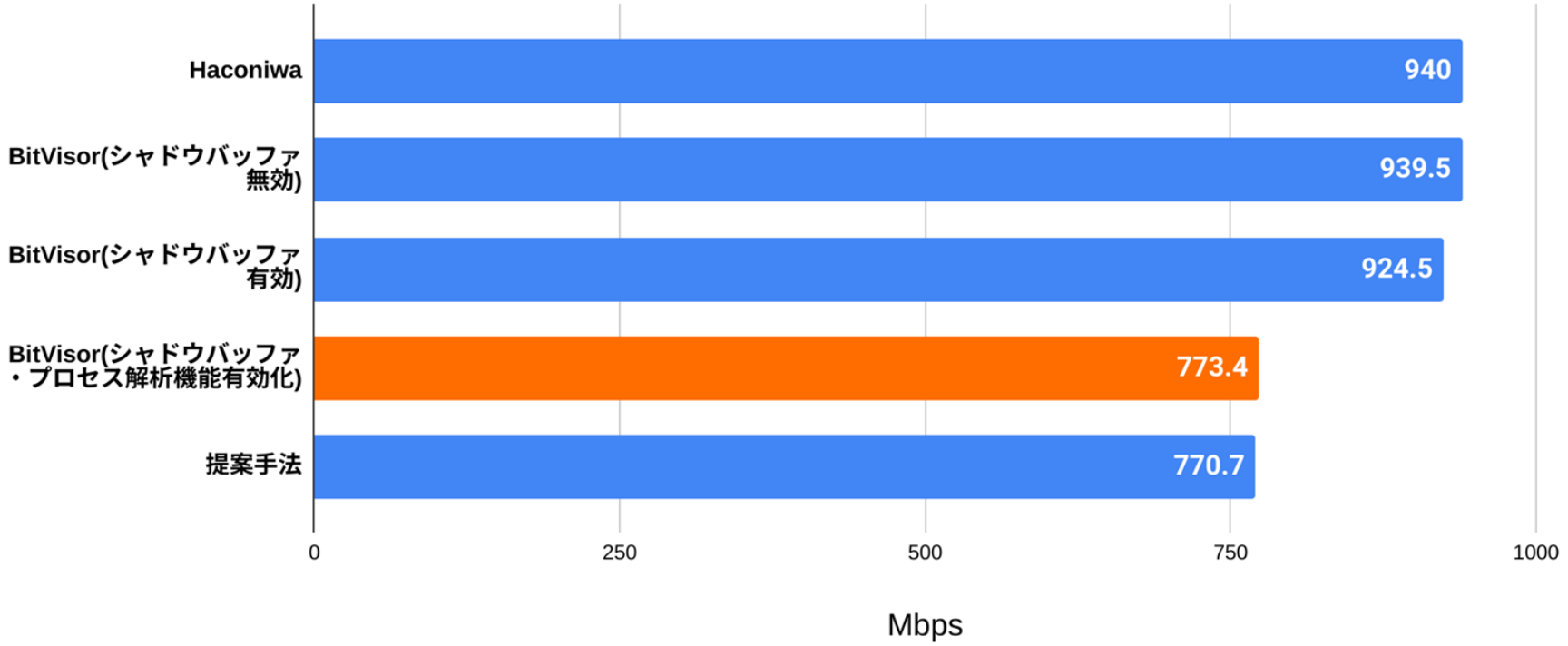
- 本提案手法はrunC(プロセス分離コンテナ)よりスループットが約18%低下
- 本提案手法はgVisorと比較すると8.4倍のスループットを達成

追加実験:オーバヘッドの特定

評価実験1で判明した18%のオーバヘッドの調査

- 実験内容は評価実験1と同一
- 評価対象は，Haconiwa，BitVisor(シャドウバッファ無効)，BitVisor(シャドウバッファ有効)，BitVisor(シャドウバッファ・プロセス解析機能有効)，本提案手法(シャドウバッファ・プロセス解析・パケット解析機能有効)

追加実験の結果・考察



- シャドウバッファのオーバヘッドは約2%
  - BitVisorによる仮想化のオーバヘッドは少ないといえる
- プロセス解析機能によるオーバヘッドは約16%
  - プロセス解析機能が大きなボトルネックであり，高速化する必要がある

評価実験2:コンテナの起動にかかる時間の計測

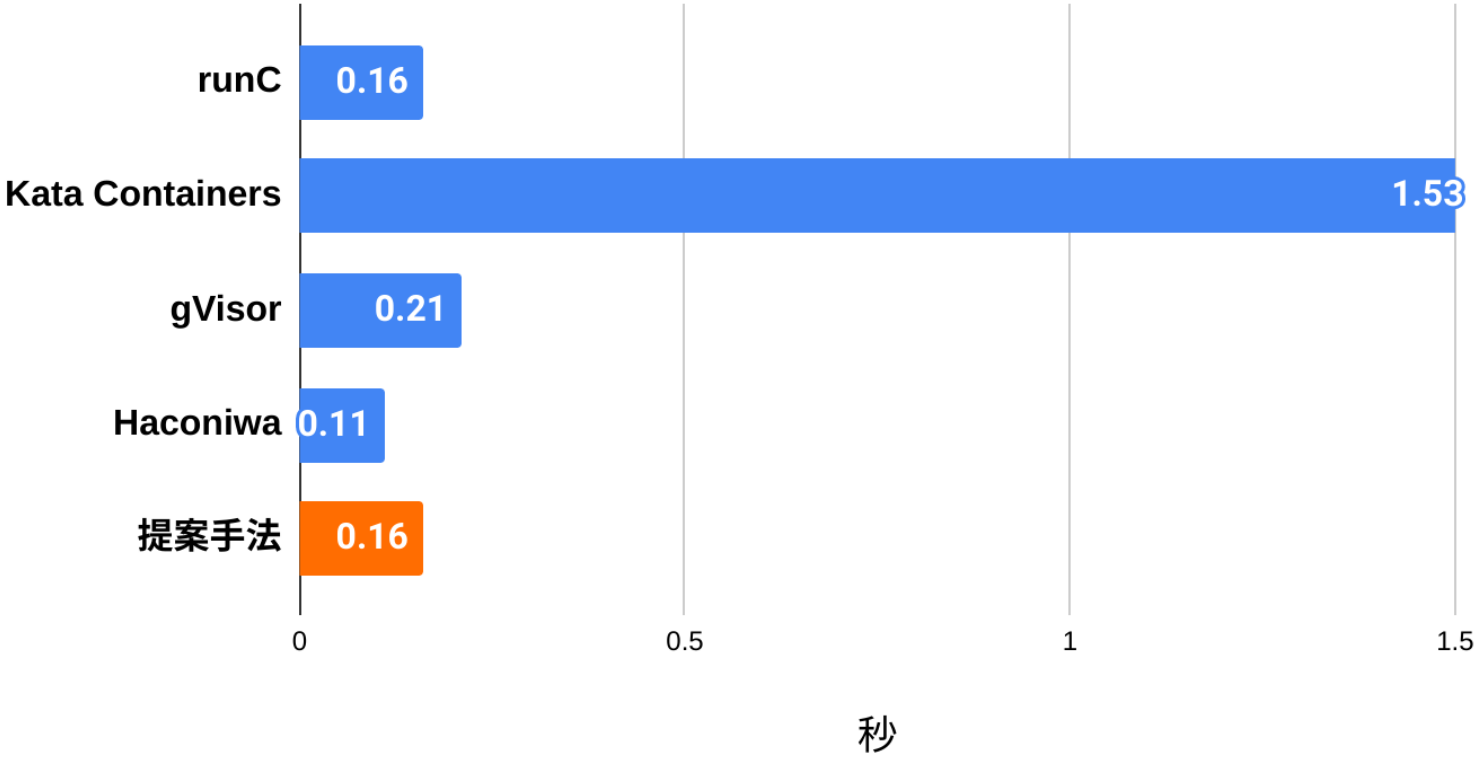
要件(4):コンテナが迅速に起動可能であるかを評価

- 標準出力にHello Worldと出力するコンテナを使用
- このコンテナの起動から終了までの時間を計測
- 評価は10回計測した値の平均を使用

実験環境

OS	CentOS 7
Linux カーネル	3.10（gVisor の評価には 5.4 を使用）
CPU	Intel（R） Core（TM） i5-6260U CPU 1.80GHz
RAM	8GByte
NIC	Intel Coporation Ethernet Connection I219-V （rev 21）

評価実験2の結果・考察



- 本提案手法はrunCと起動にかかる時間の差は小さい
  - 本提案手法がコンテナ起動にかかる時間に与える影響は小さいと判断
- 本提案手法はKata Containersと比較して9分の1の起動時間であった

まとめ

目的

不必要な権限付与によるネットワークリソースを用いた他コンテナへの攻撃を防止する

軽量かつ柔軟なネットワーク隔離の実現

提案

BitVisorを用いて，コンテナからのI/Oを捕捉し，

セキュリティポリシに基づいてI/Oの検査を行い，アクセス制御

評価実験の結果

評価実験1：プロセス解析機構が大きなボトルネックになっていることが判明

評価実験2：本提案がコンテナ起動にかかる時間に与える影響は小さいことが判明

今後の課題

- プロセス解析機構の高速化
- 単一マシン内におけるコンテナ間通信の隔離手法

補足資料: コンテナ型仮想化技術における ネットワーク隔離

Linuxのネットワーク機能を用いて実現

- Network Namespacesと仮想ネットワークインターフェース(veth), Linux Bridgeを使用
- **vethとLinux Bridgeによってコンテナ間通信を実現**