社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

クラウド基盤における Workload の Identity を用いたネットワーク制御とパケットフィルタリングへの適用

大西憲太郎† 小谷 大祐† 市原 裕史†† 金丸 洋平†† 岡部 寿男†

† 京都大学 〒606-8501 京都市左京区吉田本町
†† LINE 株式会社 〒160-0022 東京都新宿区新宿 4-1-6 JR 新宿ミライナタワー 23 階
E-mail: †ohnishi@net.ist.i.kyoto-u.ac.jp, †{kotani,okabe}@media.kyoto-u.ac.jp,
††{hirofumi.ichihara,yohei.kanemaru}@linecorp.com

あらまし 従来のネットワーク制御では、IP アドレスとポート番号（Locator）を通信するプロセスやコンテナ（Workload）の識別子として用いてきた。しかし、昨今のマイクロサービスアーキテクチャやコンテナ技術を採用するクラウド基盤では Locator が大量に使用かつ頻繁に更新される環境となっており、Locator に頼ったネットワーク制御を行うことが困難になってきている。そこで、本研究では Workload の性質（Identity）のうちネットワーク制御に必要な情報を Service としてパケットに付加することにより、Locator に依存せずに Ientity に基づくパケットの処理を行えるシステムを提案する。そして、提案システムを利用したクラウド内のパケットフィルタリング機構を設計・構築し、パケットフィルタリングが実現できることを示す。

キーワード ネットワーク制御、パケットフィルタリング、クラウド基盤、Identity、 Workload

Network Control in a Cloud Platform Using Identities of Workloads and It's Application to Packet Filtering

Kentaro OHNISHI†, Daisuke KOTANI†, Hirofumi ICHIHARA††, Yohei KANEMARU††, and Yasuo OKABE†

† Kyoto University Yoshida-honmachi, Sakyo-ku, Kyoto, 606-8501 Japan
†† LINE Corporation JR Shinjuku Miraina Tower 23rd FL, 4-1-6 Shinjuku, Shinjuku-ku, Tokyo, 160-0022 Japan
E-mail: †ohnishi@net.ist.i.kyoto-u.ac.jp, †{kotani,okabe}@media.kyoto-u.ac.jp,
††{hirofumi.ichihara,yohei.kanemaru}@linecorp.com

Abstract Conventional network controls use IP addresses and port numbers (Locators) as identifiers of Workloads like processes and containers. However, in cloud platforms adopting microservices architecture and container technology, Locators are massibly used and frequently updated. This makes it difficult to control network with relying on Locators. In this paper, we propose a new system to process packets based on the Identity of a Workload without depending on Locators, by marking packets with the necessary information extracted from the Identity. We then design and construct a packet filtering mechanism in a cloud platform using the proposed system and show that mechanism can be implemented with the proposed system.

Key words Network Control, Packet Filtering, Cloud Platform, Identity, Workload

1. はじめに

クラウド基盤では利用者の VM・コンテナや管理者の基盤管理アプリケーションなど様々なソフトウェア (Workload) が動作しており、クラウド基盤におけるパケットフィルタリングや帯域制御、優先制御といったネットワーク制御は利用者に安全なサービスを提供するために重要である。

従来、ネットワーク制御では様々な Workload を識別するために IP アドレスやポート番号といった Locator を識別子として用いてきた。しかし、クラウド基盤においては、次の理由から Locator に基づくネットワーク制御が困難になってきている。

a） クラウド基盤における Workload の分散

従来のネットワーク管理では、同様の制御を行うべき Workload をネットワーク上で集約して配置した上で少数のプレ

フィックスを割り当てて管理することが一般的であり、それによってネットワーク制御に必要なエントリ（パケットの送信元と宛先が特定の処理を行う対象かどうかを決定するための情報）の数は抑えられていた。一方、クラウド基盤では、アプリケーションや仮想化された実行環境をアプリケーションの種類や利用者に関わらず配置することで、利用者の需要に応じた柔軟な資源の提供を行っている。そのため、同じネットワーク制御を行うべき Workload でもネットワーク上では分散して配置される [1]。その結果、Workload の IP アドレスはネットワーク制御において集約して扱うことができず、個別に扱う必要があり、ネットワーク制御に必要なエントリ数が増加してしまう。

b） マイクロサービス化による Workload の増加

近年マイクロサービスアーキテクチャという、1 つの大きなアプリケーションを相互に通信する疎結合な複数のアプリケーションに分割して開発・運用を行う手法が多く採用されている。これはアプリケーションの複雑さを減少させ開発を行いやすくするが、元々 1 つであった Workload は相互に通信する複数の Workload に分割されることになる。その結果、Workload の数が単純に増加する上、分割された Workload の間で行われる通信を新たに制御する必要がある [2] ため、制御に用いるエントリ数が増加する。

c） コンテナ化による Workload の増減頻度の増加

コンテナは Linux における仮想化技術の 1 つで、OS 内でネットワーク、プロセスなどの名前空間や権限の分離を利用して仮想的な Linux システム内でプロセスを動作させるものである。ホスト OS からは権限分離がなされた 1 プロセスとして動作するため、既存の VM と比べて軽量で起動・終了が非常に高速という特徴を持っている。その結果、負荷状況に応じたスケールイン・アウトやアプリケーションの更新をより頻繁に行うことができる [3] ため、アプリケーションの運用手法として一般的なものになっている。しかし、ネットワーク制御という面では、コンテナ化によって Workload が頻繁に増減するようになった結果、制御に用いるエントリの更新頻度が高くなるという問題がある。

このように、マイクロサービスアーキテクチャやコンテナ技術が導入されるようになった近年のクラウド基盤で Locator に基づいた既存のネットワーク制御を行うと、大量のエントリを頻繁に更新する必要があり、エントリの管理コストが高くなってしまう。

この問題を解決するために、ネットワーク制御を Locator ではなく通信している Workload が何であるか (Identity) に基づいて行うことを考える。

通信時に Identity を識別できるプロトコルには HIP [4], Mobile IP [5], LISP [6] といったものがある。これらを用いるとネットワーク中で個別の Worklaod を識別することができるが、Workload に一対一対応した識別子に基づいたネットワーク制御を行ってもエントリ数やエントリの更新頻度の問題は緩和されない。

これに対して、本研究では、個別の Workload を識別してネットワーク制御を行うのではなく、Workload の Identity の情報のうち実際にネットワーク制御に必要な情報を識別してネットワーク制御を行う手法を提案する。具体的には、Workload の Identity のうちネットワーク制御で用いる情報を Service で示すようにして、各 Workload を Service に紐付け、その識別子をパケットに付加することにより、Service に基づいたネットワーク制御を行う。同じ Service に紐付く Workload が増加・減少してもエントリに対して変更を行う必要がなくなり、エントリの管理が容易になる。

上記のアイデアに基づき Service に基づいたネットワーク制御を行うシステム・Acila を設計・実装するとともに、実際に Service を利用してパケットフィルタリングを行う機構を設計・実装し、その実現性を示す。

2. 関 連 研 究

第 1 章で述べたとおり、HIP [4], Mobile IP [5], LISP [6] のようなプロトコルでは通信時に Identity を識別することができるが、これらは Locator に依存せずに通信相手を識別する Mobility を主眼に置いており、ホストやデバイスに一対一対応する識別子を与えることで達成している。しかし、本研究が目的としているパケットフィルタリングや帯域制御のようなネットワーク制御においては個別の Workload を識別する必要はない。提案する手法は、Identity の情報のうちネットワーク制御に必要な情報をパケットから識別できるようにすることでエントリ数や更新頻度の削減を行うものである。

VXLAN [7] といったオーバーレイネットワークではネットワークの分離を提供するため、従来のプレフィックスによる IP アドレスの集約ができるようにオーバーレイネットワークを構成することが可能である。一方で、プレフィックスによる管理がクラウド基盤の特徴である柔軟な構成の変更を妨げる可能性がある。

MPLS は特定のネットワーク経路に対して LSP (Label Switched Path) を確立し、パケットにラベルを付与して高速なルーティングや帯域制御、優先制御 [8] を実現している。全てのパケットに送信元と宛先に関する情報を付加することで、その情報に基づいた様々なネットワーク制御を行うことを目的とする本研究とは趣旨が異なる。

クラウド基盤において Workload に Identity を付与するものとしては SPIFFE [9] が挙げられる。これは、Workload の認証を行い Identity を示す証明書をその Workload に配布する。SPIFFE は証明書を用いて mTLS で通信相手と相互認証を行うことを想定しており今回の目的とは異なるものの、Workload の管理や認証手法については関連している。

Workload に対して Identity に基づいてパケットフィルタリングやトレーシングを行うものとして Cilium [10] がある。Cilium は宛先ノードで送信元と宛先の Identity に基づいたパケットフィルタリングなどの処理を行っており、VXLAN などのカプセル化を用いる場合は送信元の ID をパケットに付加して宛先に送信している。Cilium は基本的に処理を送信元か宛先で行っており、優先制御などネットワーク中で行う必要がある

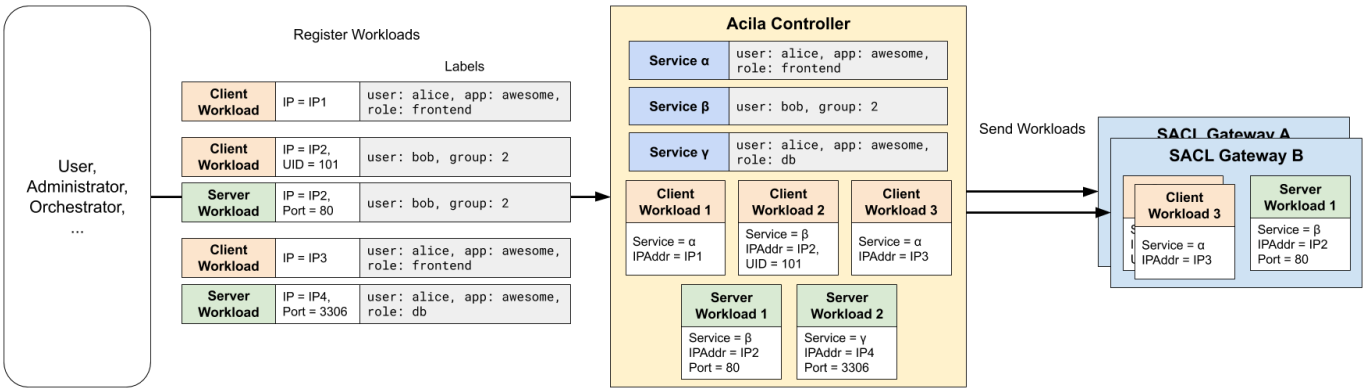

図 2　Workloload への Service の付与と SACL Gateway への配布

ネットワーク制御については想定していない。本研究では、送信元と宛先両方の Identity に関する情報をパケットに付加することによって、ネットワーク中での制御に応用できるようにするものである。

3. 前提とするクラウド基盤

3.1 Workload

			Labels
(1)	Client Workload	IP = IP1	user: alice, app: awesome, role: frontend
(2)	Client Workload	IP = IP2, UID = 101	user: bob, group: 2
(3)	Server Workload	IP = IP2, Port = 80	user: bob, group: 2
(4)	Client Workload	IP = IP3	user: alice, app: awesome, role: frontend
(5)	Server Workload	IP = IP4, Port = 3306	user: alice, app: awesome, role: db

図 1　Workload

クラウド基盤内で単一の機能を提供する処理を行うソフトウェアを Workload と呼ぶ [9]。Workload が示す対象は提供される機能や利用者によって様々である。例えば IaaS (Infrastructure as a Service) では利用者に貸し出す VM の実行環境やクライアント・サーバーとなる VM 内のプロセスが、CaaS (Container as a Service) ではコンテナや Kubernetes [11] における Pod が 1 つの Workload となり得る。

本研究ではこの Workload 間の通信を Workload の Identity に基づいて制御する。Workload を通信を開始するクライアント側 (Client Workload) と通信を受け付けるサーバー側 (Server Workload) に分けて扱う。Client Workload は IP アドレスと UID などの追加情報、Server Workload は IP アドレスとポート番号[1] のような Workload を特定するための情報を持つ。

3.2 Workload のラベル

今回想定するクラウド基盤では、図 1 のように Workload は Key Value 形式のラベルの集合によって統一的に管理されており、 Workload が作成・更新された場合にはそのラベルとともに Acila Controller (Acila 内で用いる様々なデータを管理するアプリケーション) に伝えられる。Workload にラベルを付与するものは様々である。例えば PaaS などの管理者が管理する Workload は管理者が指定する。IaaS の VM のような利用者が管理している Workload であれば利用者にラベルでの管理機能といった形で提供される。また、CaaS の 1 つである Kubernetes のように Key, Value 形式のラベルによる管理を前提としたサービスではそのオーケストレーターが保持している情報をそのままラベルの集合として用いる。

また、利用者の管理する実行基盤は完全には信頼できないものとして扱い、管理者の管理するシステム・ネットワークについては信頼できるものとして扱う。

(注 1)：IP アドレスのみで識別する場合はポート番号は指定されない

4. Workload への Service の付与

4.1 Service

Workload の Identity には Locator の他にアプリケーションの種類・特性、バージョン、動作している機器、稼働時間などの様々な情報がある。これらの情報は Workload によって異なるが、ネットワーク制御においては全ての Workload に別々の処理を行うことはほぼなく、行う処理が同じである Workload はまとめて扱うことが望ましい。処理の決定に必要な情報はアプリケーションの種類など一部の情報であり、これを Service として表す。そして、各 Workload を Identity に基づいて何かしら 1 つの Service に紐付ける。

例えば、図 1 において、ラベルが `{user: alice, app: awesome, role: frontend}` である Client Workload には同じ処理を行うものとする。その場合、ラベルがその内容であることを表す Service を作成し、その Service に Workload (1) と Workload (4) を紐付ける。

各 Service に SACL ID と呼ぶ識別子を割り振る。この SACL ID を Workload 間のパケットに付加することで、ネットワーク制御において送信元・宛先 Workload の各 Service を識別して、SACL ID に基づいた様々な処理を行う。

4.2 Workload への Service の付与

本研究では、ネットワーク制御で行う処理の決定には Workload に紐付くラベルの集合を用いる。そのため、ラベルに基づいて Workload をどの Service に紐付けるかを決定する。

全てのラベルがネットワーク制御での処理の決定に必要とは

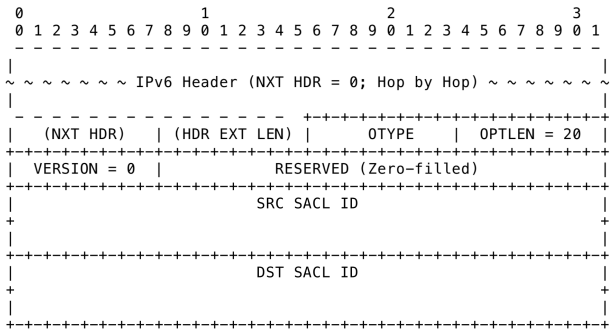

図 3　SACL ID を付加するためのパケットフォーマット

限らないが、今回は簡単のためにラベルの集合と Service は一対一対応とした。Workload の作成・更新時に Acila Controller は同じラベルの集合を持つ Service が既に存在すればその Service を、そうでなければラベルの集合に対応した Service を新しく作成して Workload に割り当てる。

これまでに述べた Workload とラベルの集合 (Labels)、Service の関係を 図 2 に示す。

5.　パケットへの SACL ID の付加

5.1　パケットフォーマット

SACL ID をパケットに付加するためのパケットフォーマットとして、大規模なクラウド基盤では IPv4 アドレスが枯渇することから IPv6 の採用が進んでいること、また次の 2 つの理由から IPv6 の拡張ヘッダである Hop-by-Hop Options 内の TLV [12] を用いる。SACL ID が 64 bit である場合のパケットフォーマットを図 3 に示す。

a）柔　軟　性

IPv6 のフローラベルなど、長さが予め定められたフォーマットを用いると SACL ID のビット長がそのフォーマットによって制限されてしまう。Hop-by-Hop Options 内の TLV は可変長であり、SACL ID に十分な長さをもたせた自由なフォーマットの構築ができ、将来の拡張も容易である。

b）既存のネットワークとの親和性

VXLAN や Geneve、MPLS などの既存のプロトコルのフォーマットを流用して SACL ID をパケットに付加することも考えられる。しかし、目的の異なるプロトコルを流用することでネットワーク機器が望まない解釈・処理を行う可能性がある。また、SACL ID を付加するために新たなカプセル化を行うことで NIC のオフローディング機能が利用できなくなりパフォーマンスが悪化する可能性がある。

IPv6 の Hop-by-Hop Options は IPv6 の拡張ヘッダであるため新たにカプセル化がされることはなく、ネットワーク機器は通常の IPv6 パケットとして扱うことができる。さらに、Hop-by-Hop Option (TLV) では Type の先頭 2bit でその Type に機器が対応していない場合の機器の対応を指定することができ、`00` にすると「オプションをスキップして処理を続行する」[12] となる。よって、SACL ID の拡張を実装していない既存のネットワーク機器でも未知のプロトコルとしてパケットが破棄されることはなく、SACL ID を無視した通常のパケット処理を行うことができる。

5.2　パケットへの SACL ID の付加

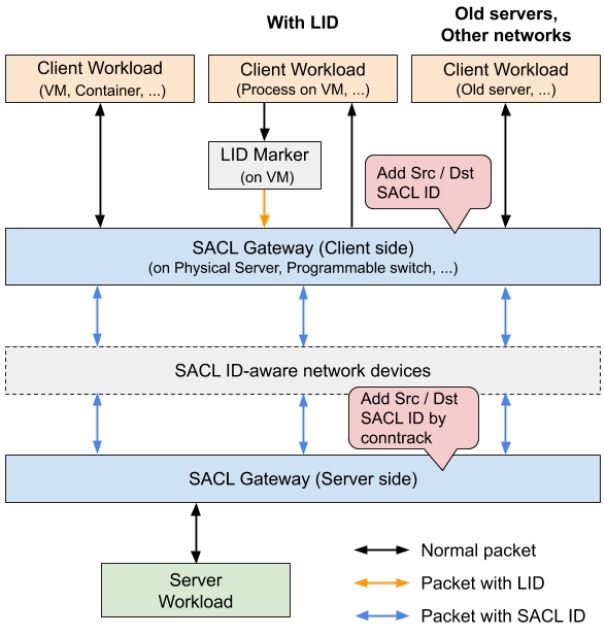

図 4　パケットへの SACL ID の付加

本節では、Workload から発せられるパケットに SACL ID を付加する手法を説明する。概要を図 4 に示す。

パケットに対して SACL ID の付加と削除を行うものを SACL Gateway と呼ぶ。SACL Gateway は配下に Client Workload と Server Workload を持ち、それらの情報と別途 SACL ID の付加に必要な Server Workload の情報を持つ。例えば、パケットフィルタリングの場合は配下の Client Workload から通信が許可された Server Workload の情報を保持する。

SACL Gateway の処理は Workload の通信が入出力される管理者が管理するインターフェースで行われる。例えば VM が Workload の場合、ハイパーバイザーホストが VM と通信を行う TAP インターフェースやハイパーバイザーホストの物理インターフェースが考えられる。

管理者が管理するインターフェースで行う理由は、もし VM やコンテナなど利用者が管理している Workload 内で SACL Gateway の処理を行うと、利用者によって付加された SACL ID を管理者は信頼することができず、SACL ID を検証する機構が別途必要になってしまうためである。また、利用者に SACL Gateway のための対応・管理コストを強いてしまう。

加えて、Workload に近いインターフェースであることが望ましい。これは、SACL ID の付加時にパケットの宛先から Server Workload を識別する際、Kubernetes における Service IP のような、Client Workload の付近でないと分からない情報を識別に用いることで SACL Gateway のエントリ数の削減やより細かな Wokrload の識別を行うためである。また、同じ SACL Gateway 配下の Workload を少なくすることで SACL ID で制御できない通信を減らすこともできる。

また、SACL Gateway の実装ができない機器や外部のネッ

トワークに対しては、境界にプログラマブルスイッチなどのSACL ID の拡張に対応したネットワーク機器を設置して SACL Gateway とする。

5.2.1 SACL ID の付加

Client Workload から Server Workload へのパケットに対しては、送信元の IP アドレスなどから Client Workload を特定し Source SACL ID を、Server Workload の IP アドレスやポート番号から Server Workload を特定し Destination SACL ID を付加する。

逆向きのパケットに関しては、クライアントが用いるポート番号は一定でないため、パケットから Client Workload の特定を行うのが困難である。そのため、Connection Tracking (Conntrack) [13] をサーバー側 SACL Gateway で行い、Workload 間のコネクションがある間 Locator と SACL ID の対応を記憶することで SACL ID を付加できるようにする。

a） LID

Client Workload に関して、利用者の管理する VM 内のプロセスなど、管理者が持つ情報のみでは Workload が特定できないものがある。クライアントは送信元のポート番号をランダムに用いるため、 Locator で特定することができない。また、通信を開始する際には Conntrack にコネクションが記憶されていない。

そのような Workload を識別するためには、利用者が管理している実行環境内の情報を用いざるを得ない。例えば利用者の VM 内のプロセスが Workload である場合、VM は通信がどのプロセスのものかを特定することが可能である。しかし、利用者が管理する実行環境を直接 SACL Gateway とするのは、5.2 節で述べた通り SACL ID を容易に偽装できてしまうため信頼できなくなる問題がある。

そこで、その Workload を管理する利用者にしか影響しない形で SACL ID の信頼を部分的にその利用者に委ね、Workload を識別するための情報を利用者の実行環境から SACL Gateway に伝える仕組みを追加する。管理者はその情報に従って SACL Gateway で SACL ID を付加する。

具体的には、先程の例における VM のような、 Workload と SACL Gateway の間にある Workload の識別ができる実行環境でパケットに一時的な Workload の識別子を付加して SACL Gateway に伝える[2]。その識別子を LID (Local ID)、LID を付加するものを LID Marker と呼ぶ。LID は LID Marker の中で一意である。SACL Gateway はパケットから識別した LID Marker と LID の情報から Client Workload を識別し SACL ID をパケットに付加する。LID Marker と LID の組で　Client Workload と対応させることで、悪意のある LID Marker が自身の配下にない Workload に偽装することを防ぐ。

本手法では、管理者は LID Marker が自身の配下の別の Client Workload に偽装することを防ぐことはできない。利用者は LID Marker を用いると管理者だけでは行えないより細かなネットワーク制御を行えるが、その管理と信頼性の担保は利用者に委ねられる。

5.2.2 SACL ID の削除

宛先側の SACL Gateway では到達したパケットから SACL ID を取り除いて Server Workload に転送する。

6. SACL ID を用いたパケットフィルタリング

本章では、これまでに述べた提案手法を用いたパケットフィルタリング手法を示す。

Client Workload から Server Workload へのパケットをフィルタリングする場所はサーバー側の SACL Gateway とする（図 4 における "SACL Gateway (Server side)"）。この理由は、もしクライアント側のみで行うと、ある Server Workload への望まないアクセスを防ぐには全ての SACL Gateway がパケットフィルタリングを正しく行う必要があり、ポリシーの更新時に反映の遅れが懸念される問題がある一方、 サーバー側で行えば Server Workload を配下に持つ 1 つの SACL Gateway のみが正しいパケットフィルタリングを行えばよく、反映の遅れの問題が少ないためである。

逆向きのパケットについては、クライアント側の SACL Gateway で Conntrack を用いて Workload 間のコネクションを記憶して、Client Workload と Server Workload 間のコネクションがあれば通信を許可する。

本手法の他にもネットワーク中のネットワーク機器でパケットフィルタリングを行う手法も考えられる。その場合、プログラマブルスイッチなどの SACL ID に基づいたパケットフィルタリングを行う装置を新たにクライアント側とサーバー側の SACL Gateway 間に配置する。SACL Gateway で行う場合と比較して、許可する通信の情報を配布する先は少なくなるというメリットがある反面、パケットフィルタリングのためのエントリ数は増加する上、フィルタリングすることができない Workload 間の通信が増加する。

6.1 ポリシーの管理と配布

パケットフィルタリングのために利用者が登録するポリシーの記述は ABAC (Attribute Based Access Control) [14] によって行う。ポリシーは通信を許可する送信元と宛先の Service 群をセレクタの集合によりそれぞれ選択するものである。概要を図 5 に示す。

セレクタは Service に紐付くラベルに基づいて Service を選択するためのもので、検索するべきラベルの Key 、Key に対応する値として存在するべき (in) ものを指定するかそうでない (not_in) かを決定する Operator 、存在するべき/するべきでない値の集合である Values で構成されている。複数のセレクタがある場合は AND と解釈し、Acila Controller がポリシーにマッチする送信元と宛先の Service 群（Client Services, Server Services）を検索する。

ポリシーが追加されるか、Server Workload が追加・変更されたとき、ポリシーの Server Services のいずれかに属する Server Workload を配下に持つ全ての SACL Gateway へ、許可するべき (Client Service, Server Service) の組 (Rule) が Acila Controller か

(注2)：どのように付加するかは実行環境が用いるプロトコルによって異なる。例えば、IPv4 では DSCP や TTL フィールドを用いた実装が考えられる。

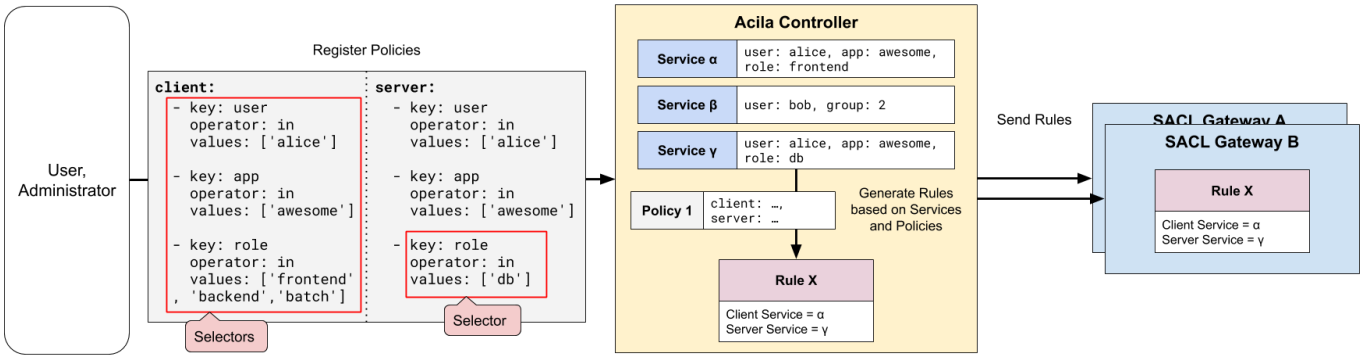

図 5　ポリシーの管理と配布

ら配布される。Client Workload が通信を始める際に、パケットに付加された SACL ID を基にサーバー側の SACL Gateway でパケットフィルタリングを行う。

また、ポリシーはクライアント側の SACL Gateway で持つネットワーク制御で必要な Server Workload を選定するためにも用いられる。Acila Controller は、各 SACL Gateway が、配下の Client Workload からの通信が許可された Server Workload の集合を持つように Server Workload の情報を配布する。

7. 動 作 検 証

これまでに述べた Workload への Service の付与とパケットへの SACL ID の付加、そして SACL ID を利用したパケットフィルタリング手法を実装し検証を行った。動作検証の概要を図 6 に示す。

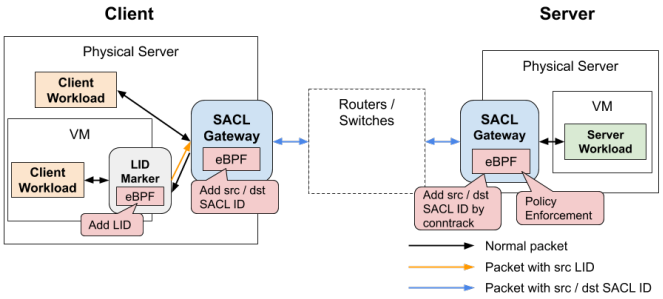

図 6　動作検証の概要

7.1 環　　境

本検証では VM 間の通信に SACL ID を付加し、パケットフィルタリングを行う。Client Workload は VM と VM 内のプロセスそれぞれのパターンを設定し、Server Workload は VM 内のデータベースサーバーとした。なお、Client Workload が VM 内のプロセスの場合、LID Marker を VM に設置した。SACL Gateway は各 VM のハイパーバイザーホストに設置した。

また、VM には IPv6 アドレスが設定されており、Linux Bridge 経由でハイパーバイザーホストを含めフラットに接続されている。

7.2 実　　装

SACL ID は 64bit とした。

表 1　SACL Gateway

id	IP アドレス
1	2001:db8::1
2	2001:db8::2

表 2　LID Marker

id	IP アドレス	SACL Gateway
1	2001:db8::3	1

7.2.1 SACL Gateway

SACL Gateway はハイパーバイザーホスト上で eBPF を用いて実装した。VM からのパケットに対して SACL ID を付加し、さらに VM へのパケットに対して SACL ID に基づいてパケットフィルタリングを行う eBPF プログラムを VM に接続される TAP インターフェースに設定した。なお、サーバー側の SACL Gatway における Conntrack は Locator と SACL ID の対応を記憶する簡易的な実装を行った。

7.2.2 LID Marker

LID は IPv6 パケットの Hop Limit フィールドに $LID\%128+100$ の形で設定した。なお、変更した Hop Limit 値はハイパーバイザーホストの SACL Gateway でデフォルト値に戻した。また、LID Marker 内の Workload の識別にはプロセスの UID を用いた。

これらの実装は、Client Workload が動作する VM の iptables を設定することによって行った。UID が 1000 であるプロセスのパケットの Hop Limit を 101 (LID = 1) に設定する設定例を以下に示す。

```
# ip6tables -t mangle -A OUTPUT -m owner --uid-
    owner 1000 -j HL --hl-set 101
```

7.3 検　　証

上記の環境に沿ったデータを Acila Controller に登録した。Aclia に登録したデータ例を表 1, 2, 3, 4 に示す。

この時点で登録した Workload のラベルに沿って正しく Service が作られた。作成された Service を表 5 に示す。Client Workload 1, 2 は Service 1 に、Server Workload 1 は Service 2 に紐付けられた。

また、LID Marker 1 は Client Workload 2 の情報を、SACL

表 3　Client Workload

id	IP アドレス	SACL Gateway	LID Marker	UID	Labels
1	2001:db8::4	1	null	null	`user: alice, app: awesome, role: frontend`
2	2001:db8::5	null	1	1000	`user: alice, app: awesome, role: frontend`

表 4　Server Workload

id	IP アドレス	SACL Gateway	Port	Labels
1	2001:db8::6	2	3306	`user: alice, app: awesome, role: db`

表 5　Service

id	SACL ID (Base64 encoded)	Labels
1	`cGae1gU9vSgD/iAIqp6xt/bXj04sS7ubGeeSUvV1qWc=`	`user: alice, app: awesome, role: frontend`
2	`FE5cCQgMODyIqzdMY1mbr4D9FwPDhNdN9f3c3PtvA/A=`	`user: alice, app: awesome, role: db`

Gateway 1 は Client Workload 1 と Client Workload 2 かつそれを識別するための LID の情報を取得して、LID Marker と SACL Gateway が Workload の情報を正しく取得することを確認した。この時点では Client Workload 1, 2 から Server Workload 1 への通信を行うことはできない。

その後、Client Workload 1, 2 から Server Workload 1 への通信を許可するためのポリシーを登録した。Acila Controller に登録したポリシーを YAML で表現したものを以下に示す。

```
client:
  - key: user
    operator: in
    values: ['alice']
  - key: app
    operator: in
    values: ['awesome']
  - key: role
    operator: in
    values: ['frontend', 'backend', 'batch']
server:
  - key: user
    operator: in
    values: ['alice']
  - key: app
    operator: in
    values: ['awesome']
  - key: role
    operator: in
    values: ['db']
```

ポリシーを登録すると、SACL Gateway 1 は Server Workload 1 の情報を取得し、SACL Gateway が配下の Client Workload と通信を行う Server Workload を正しく取得したことを確認した。SACL Gateway 2 は Service 1 から Service 2 への通信を許可する Rule を取得し、SACL Gateway が Rule を正しく取得したことを確認した。そして、Client Workload 1, 2 と Server Workload 1 間のパケットに SACL ID が付加されるようになり、通信が行えるようになった。

8. 考察と課題

本研究の成果に基づいて、ネットワーク中の帯域制御や優先制御といった従来の IP ベースで行っている他のネットワーク制御にも Service に基づいた処理を適用することでエントリの数や更新頻度の軽減が行えることが考えられる。

また、Service について、簡単のためにラベルの集合と Service は一対一対応としているが、その結果 Service の数が多くなってしまう可能性がある。それに対して、与えられたポリシーから最適な Service の生成を行う、Service に用いるラベルの Key を限定する、といった検討が考えらえる。さらに、現在は SACL ID の生成方法については定めていないが、SACL ID の階層化を行って一部の条件を上位ビットで判別できるようにすることでエントリをより減らすことも考えられる。

SACL ID を付加するためのプロトコルには IPv6 の Hop-by-Hop Options を利用しており、現在は IPv4 パケットには対応していないため、IPv6 パケットにカプセル化する必要がある。IPv4 でもカプセル化を行わずに Service に基づいたネットワーク制御を行うために、IPv4 を用いた SACL ID を付加するプロトコルの設計について検討の余地がある。

パケットフィルタリングについて、現在は Server Workload から Client Workload へのパケットはクライアント側の SACL Gateway において Conntrack を用いることで実現しているが、これによって SACL Gateway は配下の Client Workload のコネクションを記憶する必要がある。これに対して、サーバー側の SACL Gateway と同様に SACL ID に基づいたパケットフィルタリングを行うことが考えられる。

これらの検討・検証については、今後の課題である。

9. 結　　論

近年のクラウド基盤のネットワーク制御におけるエントリの数や更新頻度が多くなりエントリの管理が困難になるという問題に対して、通信する Workload の Identity からネットワーク制御に必要な情報を Service として切り出し、複数の Workload をまとめた上で、ネットワーク中で Service を識別できるようにすることでエントリの数や更新頻度の削減を行うシステムを提案した。そして、提案システムを用いたパケットフィルタリング機構を設計し、実装と検証することで実現性を示した。

現状では、帯域制御といった他のネットワーク制御への応用やラベルの集合と Service の対応の考察などが不足している。

また、SACL ID を付加するレイテンシやエントリ数の削減数といった定量的評価については今回は評価を行っておらず、今後の課題である。

最後に、本研究を進める上で有益な助言を頂きました京都大学の當山達也氏にこの場を借りて感謝申し上げます。

文　献

[1] P.M. Mell and T. Grance, “SP 800-145. The NIST Definition of Cloud Computing,” Technical report, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, 2011.

[2] N. Dragoni, S. Giallorenzo, A.L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, “Microservices: yesterday, today, and tomorrow,” Present and ulterior software engineering, pp.195–216, Springer, 2017.

[3] A. Tosatto, P. Ruiu, and A. Attanasio, “Container-Based Orchestration in Cloud: State of the Art and Challenges,” 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems, pp.70–75, 2015.

[4] P. Nikander, A. Gurtov, and T.R. Henderson, “Host Identity Protocol (HIP): Connectivity, Mobility, Multi-Homing, Security, and Privacy over IPv4 and IPv6 Networks,” IEEE Communications Surveys Tutorials, vol.12, no.2, pp.186–204, 2010.

[5] C.E. Perkins, “Mobile IP,” IEEE Communications Magazine, vol.35, no.5, pp.84–99, 1997.

[6] Cisco, “Locator/ID Separation Protocol Architecture - Cisco,” March 2011. [Online; accessed on 2020-11-16]. https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/locator-id-separation-protocol-lisp/white_paper_c11-652502.html

[7] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, “Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks,” RFC 7348, Aug. 2014. [Online; accessed on 2020-11-16]. https://rfc-editor.org/rfc/rfc7348.txt

[8] M.R.A. Rahimi, H. Hashim, and R.A. Rahman, “Implementation of Quality of Service (QoS) in Multi Protocol Label Switching (MPLS) networks,” 2009 5th International Colloquium on Signal Processing Its Applications, pp.98–103, 2009.

[9] The SPIFFE Authors, “SPIFFE - secure production identity framework for everyone”. [Online; accessed on 2020-11-16]. https://spiffe.io/

[10] Cilium Authors, “Cilium - Linux Native, API-Aware Networking and Security for Containers”. [Online; accessed on 2020-11-16]. https://cilium.io/

[11] The Kubernetes Authors, “Production-grade container orchestration - kubernetes”. [Online; accessed on 2020-11-16]. https://kubernetes.io/

[12] D.S.E. Deering and B. Hinden, “Internet Protocol, Version 6 (IPv6) Specification,” RFC 8200, July 2017. [Online; accessed on 2020-11-16]. https://rfc-editor.org/rfc/rfc8200.txt

[13] P. Ayuso, “Netfilter’s connection tracking system,” LOGIN: The USENIX magazine，vol.31，pp.34–39，2006.

[14] E. Yuan and J. Tong, “Attributed based access control (ABAC) for web services,” IEEE International Conference on Web Services (ICWS’05), p.569, 2005.