Vol.2019-DPS-179 No.28
Vol.2019-MBL-91 No.28
Vol.2019-ITS-77 No.28
2019/5/24

COTSデバイス向けアプリケーション処理連接基盤

佐藤 友範[1,a)]　渡邉 大記[1,b)]　林 和輝[1,c)]　寺岡 文男[2,d)]

概要：Internet of Things（IoT）時代におけるクラウドコンピューティングの問題点の解決のため，Mobile Edge Computing（MEC）の考え方が近年広まっている．筆者らは MEC にサービスチェイニングを組み合わせた技術である Application Function Chaining（AFC）を提案している．AFC はアプリケーションの処理を機能 (Application Function; AF）ごとに分割し，AF のチェイニング（AF Chaining; AFC）によってアプリケーションを構成する．本稿は Commercial off-the-shelf（COTS）device，すなわちパラメータ等の設定変更は可能であるがソフトウェアの変更はできないことを想定した既成品の通信デバイスに向けた AFC の適用基盤を実現することを目的とする．AFC の設定が可能となるネットワーク領域（AFC domain）の境界には AFC Ingress および AFC Egress と呼ぶノードを配置し，それぞれ AFC domain の入口，出口の役割を担う．本稿では AFC User と AFC domain 内で動作する各エンティティのプログラムを実装した．評価では AF および AFC の設置に要する時間とデータ転送におけるスループットを計測した．結果では AF の設置には 1 つにつき 0.9 ms の内部処理時間を要する一方で，AFC の設置は AF の数に関わらず 2 ms 以下の処理時間で済むことから，一度起動した AF を様々な設定の AFC で用いるのに適していることを示した．また今後の課題として，データ転送における帯域使用率の向上や同一の AF が設置された AF Node の動的な決定が挙げられる．

Application Function Chaining Infrastructure for Commercial Off-The-Shelf Devices

1. はじめに

Internet of Things（IoT）の浸透により，インターネットに接続されるデバイスの数は増加の一途を辿っている中，現在の IoT アプリケーションは，Amazon Web Service（AWS）や Google Cloud Platform（GCP），Microsoft Azure などのクラウドサービスが手がける IoT フラットフォーム上で展開されている．ユーザはクラウドにリクエストを送信することでレスポンスを得るが，すべての処理がデータセンターで実行されるため遅延の増大やデータセンタ内に存在するアプリケーションしか利用できない点などが問題となる．

IoT 時代におけるクラウドコンピューティングの問題点を解決するために，近年 Mobile Edge Computing（Multi-access Edge Computing; MEC）[1], [2] という考え方が広まってきている．MEC は，バッテリや計算能力に制限があるスマートフォンなどのモバイル端末から，動画像のレンダリングなどの計算量が比較的大きい処理をユーザ近傍（エッジ）のコンピューティング資源にオフローディングする概念である．MEC のおかげで，アプリケーションの処理をネットワーク内に配置することができる．一方，ネットワーク事業者がファイアウォールなどのネットワーク機能（Network Function; NF）を複数連接してトラフィックに適用する，サービスチェイニング（Service Chaining）という技術が提案されている．Service Function Chaining（SFC） [3] は代表的なサービスチェイニング手法である．SFC をはじめとするサービスチェイニング手法は現在のところ，ネットワーク事業者が運用を自動化したりソフトウェアとして提供される NF（Virtualized NF; VNF）と組み合わせて運用コスト削減を図る目的で研究や標準化が進められている．MEC とサービスチェイニングを組み合わせることで，IoT デバイスや IoT ゲートウェイ（Gateway;

[1] 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University
[2] 慶應義塾大学理工学部
Faculty of Science and Technology, Keio University
[a)] glue@inl.ics.keio.ac.jp
[b)] nelio@inl.ics.keio.ac.jp
[c)] gordon@inl.ics.keio.ac.jp
[d)] tera@keio.jp

GW）と通信しつつ，ネットワーク内部でアプリケーションのデータを開発者の意図に則して加工することが期待できる．

現状の MEC は特定のアプリケーションを想定した設計になっていたり，5G や LTE などのモバイルネットワーク網を前提としたアーキテクチャになっており，汎用性に乏しいといえる．また一方で，既存のサービスチェイニング手法はネットワーク事業者向けの技術であり，アプリケーションのデータを加工するなどの，アプリケーション開発者の意図を反映させるような機構になっていない．加えて，現在のサービスチェイニング手法では既存の End-to-End（E2E）の通信路しか構築することができず，同時に複数の通信相手を想定したアプリケーションを動作させることができない．

筆者らは，このような MEC とサービスチェイニングの欠点を踏まえ，Application function Chaining（AFC）と呼ばれるアプリケーションレベルのサービスチェイニングを実現する通信機構を提案している [4], [5]．従来の AFC 手法では Commercial off-the-shelf（COTS）デバイスのような，パラメータ等の設定変更は可能であるがソフトウェアの変更はできないデバイスは AFC に参加することが出来なかった．COTS デバイスの例としては，センサデバイスや監視カメラなどが挙げられる．本稿の目的は COTS デバイス向けの AFC の実装および性能評価であり，ソフトウェア上の制約がある COTS デバイスに対しても AFC を適用できることを示す．

2. 関連研究・関連技術

2.1 エッジコンピューティング

ACACIA Mobile Edge Network [6] は LTE コア網に Mobile Edge Computing（MEC）を実現するシグナリングプロトコルを Software Defined Networking（SDN）や Network Functions Virtualization（NFV）で実現する研究である．ACACIA が対象とするのは，Augmented Reality（AR）のような，低遅延性を要求し，かつ動画像処理のような比較的計算量の大きいアプリケーションである．具体的には，AR をアパレルショップなどの商品展示に応用するアプリケーションや，地図と AR を組合せたような地理的情報を用いるロケーションアウェアなアプリケーションなどである．また，ACACIA の特徴として，通信先を SDN で決定することが挙げられる．そのため ACACIA アーキテクチャ上ではアプリケーションの特性やユーザの位置によって通信先や演算場所を自在に変更できることは大きな利点である．一方で，ACACIA は，LTE コア網において MEC を実現することができるが，LTE 以外のネットワーク，たとえば広域インターネットに適用できないことが欠点となる．また，ACACIA では演算のオフローディング先がすべてエッジという前提になっており，クラウドとの連携が考慮されていない．

2.2 サービスチェイニング

Service Function Chaining（SFC）[3] は代表的なサービスチェイニング手法である．SFC では，事業者の展開するネットワーク内の classifier というエンティティが SFC ドメインの出入り口となる．パケットが classifier に到達すると，classifier はパケットをクラス化し，Service Function Forwarder が解釈できるフォーマットでチェインするファンクションの情報をパケットに付与する．クラス化や forwarder の実現の手法は実装依存で，例えば IETF で標準化が進められている Network Service Header [7] や，IPv6 Segment Routing（SRv6） [8]，MPLS Segment Routing（SR-MPLS） [9] などで研究および標準化が進められている．

NSH [7] では，classifier がオリジナルのパケットを SFC ドメイン内で有効なカプセル化を施し，forwarder が NSH を解釈してファンクション（Service Function; SF）をチェインする．SF には，NSH に対応していない従来のネットワークファンクションも使用できる．NSH では新たにプロトコルが定義されているため，そのための実装を classifier や forwarder および SF にする必要があり，場合によっては新たにネットワーク機器を用意する必要がある．

Segment Routing [8], [9] を用いた SFC の実現方法では，既存のネットワーク機器をそのまま流用できるという利点がある．特に SRv6 では広域網にそのまま展開でき，SF に固有のアドレスを割り当てるようなアドレッシングが考案されている．SRv6 は NSH と同様に新たなヘッダが定義されるため，SRv6 の拡張ヘッダを解釈するための実装が必要となる．一方で，SR-MPLS ではデータプレーンをそのまま使用でき，運用ポリシをどのようなラベリングに置き換えればよいかを考えればよい．しかし，MPLS は事業者ごとに運用ポリシが異なる場合が多いため，共通のポリシをどのように実現するかが課題となる．

SFC のアーキテクチャをアプリケーションレベルのサービスチェイニングに適用しようとすると，いくつかの問題点があげられる．まず，ファンクションの適用はユーザから透過的であるということがアーキテクチャから読み取れる．つまり，通信の両端は従来の通信を想定しているため，複数の相手と同時に通信する通信路を構築することができない．また，現在までのクラス化手法では，SF を実行する場所を動的に決定することができない．そのため，あるアプリケーションの利用率が高まると，特定の SF が動作するノードの負荷が上昇する可能性がある．

OpenFlow のような SDN プロトコルによるサービスチェイニング手法として，文献 [10], [11] は Differentiated Services Code Point（DSCP）フィールドにネットワークサービスごとのチェイニング情報を格納し，VLAN ID に

Vol.2019-DPS-179 No.28
Vol.2019-MBL-91 No.28
Vol.2019-ITS-77 No.28
2019/5/24

図 1　直線上の AF Chain

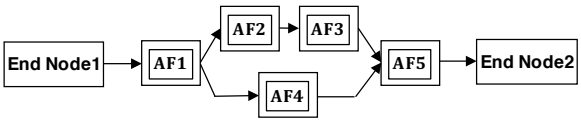

図 2　分岐，合流が存在する AF Chain

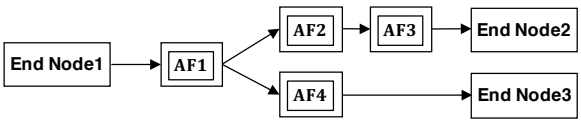

図 3　エンドノードを複数持つ AF Chain

よるルールマッチングでパケットを操作する方式を提案している．これらの研究で用いられるように，DSCP フィールドや VLAN ID は従来のネットワーク技術をそのまま流用できるため，導入コストや管理コストが低いという利点が存在する．サービスチェイニングをする際に，OpenFlow のマッチングに DSCP フィールドや VLAN ID を利用することは一般的である [10]．これらの手法をアプリケーションレベルのサービスチェイニングに適用する場合，次の問題が考えられる．まず，開発者の意図を反映したチェインを作成することが困難である．アプリケーションレベルのサービスチェイニングでは，アプリケーションユーザの位置やネットワークの状況によって，ファンクションが動作する位置が変更できる仕組みを持つ必要がある．例えば OpenFlow では，想定外のパケットが SDN スイッチに到達したときにドロップするかパケットインする場合がほとんどである．ユーザがどの位置からサービスを利用するかわからない以上，事実上全てのアクセスネットワークのルールを記述する必要がある．しかし，現在のインターネットが Open Shortest Path First（OSPF）のような Interior Gateway Protocol（IGP）と Border Gateway Protocol（BGP）のような Exterior Gateway Protocol（EGP）とで経路表の管理が分かれていることからも，SDN スイッチが全てのアクセスネットワークの情報を保持し，世界中に配置されることは極めて困難である．

3. Application Function Chaining

Application Function Chaining（AFC）は，地理的に隔たれた複数の計算資源にまたがってアプリケーションの処理（Application Function; AF）を展開し，これらをチェイニングするための技術である．AFC における AF はデータを入力して特定の演算をし，結果を出力する関数のようなものであり，AF をチェイニングすることで 1 つのアプリケーションとしての機能を提供する．AFC では AF 作成者という AF を作成し提供する事業者を想定しており，利用者は AF を組み合わせて実現したい処理を構築する．AFC の特徴の一つとして，制作者が異なる AF を組み合わせて利用できる点が挙げられる．複数の AF をチェイニングしたものを AF Chain と呼ぶ．**図 1** は 3 つの AF を直線状にチェイニングした AF Chain である．エンドノードから送信されたデータはまず AF1 で処理が施され，AF2 に送信される．データは AF2，AF3 でも同様に順次処理された後，エンドノードへと送信される．また，AF Chain は**図 2** のように分岐や合流を持つことが出来る．分岐を持つ AF Chain は**図 3** のようにそれぞれ別のエンドノードを宛先に設定出来る．各 AF はデータを処理する際に通常のデータの出力とは別に返り値を渡す．この返り値に対して関係演算を行うことで次に適用する AF を動的に決定することができる．

表 1　AFC 設計の分類

		ステートの保持	
		ヘッダ内	AF Node
終端機器	End Node	文献[5]	文献[4]
	Gate Node	AFC for COTS devices	

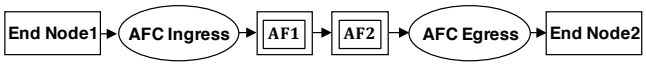

図 4　AFC Ingress，AFC Egress による AFC の適用

3.1 本稿の AFC における位置づけ

AFC の設計はこれまで複数存在し，**表 1** に示すようにどのエンティティがチェインのステートを保持するか，またどのノードが AFC の終端ノードになるかにより大きく分類することが出来る．ステートの保持はデータパケットに付け加えられヘッダ内に埋め込むものと，実際にファンクション処理を行うノードである AF Node 内に保持するものに分けられる．一方でチェインの終端機器はエンドノードと Gate Node と呼ばれる AFC の出入口となるノードに分けられる．本稿ではソフトウェアの変更ができない COTS デバイス上のアプリケーションに AFC を適用する目的に対して，ステートはヘッダ内への格納，終端機器は AFC Ingress，AFC Egress と呼ばれる出入口となるノードの利用を選択した．**図 4** に示すようにデータパケットは AF が適用される前後で AFC Ingress，AFC Egress を通過し，AFC ヘッダが挿入および除去される．

3.2 アーキテクチャ

本稿における AFC の構成要素について説明する．**図 5** はエンティティ構成の概要図である．AFC の設定が可能な領域を AFC domain と呼ぶ．AFC domain 内には，AFC Manager，AAA（Authentication, Authorization, and Ac-

Vol.2019-DPS-179 No.28
Vol.2019-MBL-91 No.28
Vol.2019-ITS-77 No.28
2019/5/24

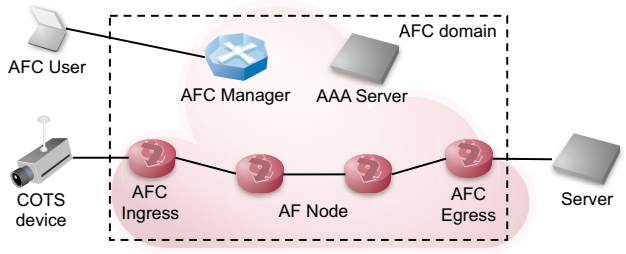

図 5　AFC for COTS devices アーキテクチャ

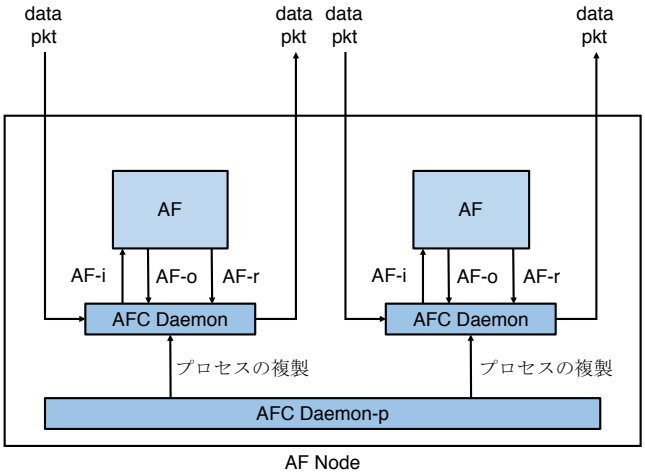

図 6　AF Node 内プロセス

counting）Server，AFC Ingress，AFC Egress，AF Node が存在し，これらに AFC User，COTS device，Server を加えた 8 種類のエンティティが存在する．AFC User は AFC domain 内に AF および AFC を設置するユーザである．AFC User は AFC に関する情報を AFC domain 内の AFC Manager に送信することで AFC を設定する．AFC Manager は AFC domain 内において AFC を管理するノードであり，AFC User から送られた情報を基に AF や AFC を設置，削除する．AAA Server は AFC User の認証認可を行うノードである．AFC Ingress と AFC Egress は AFC domain の境界に設置されるノードである．AFC を適用するデータパケットに関して，AFC Ingress は AFC domain への入口となり， AFC Egress は AFC domain の出口となる．同一のノードが AFC Ingress および AFC Egress として動作する場合もある． AF Node は AF の実行が可能なノードである．**図 6** は AF Node 内の詳細図である．AF Node には AFC Daemon-p と呼ぶデーモンプロセスが常に動作している．AFC User が AF の起動を要求すると，AFC Daemon-p は子プロセスとして AFC Daemon を起動し，その後は AFC Daemon がデータパケットの処理を行う．AFC Daemon は AF を起動するとともに，AF との間に入出力を持つ． この入出力には AF を適用するデータ受け渡しする AF-i，AF-o に加え，実行した AF の返り値を受け取る AF-r の 3 種類がある．COTS device は 既製品

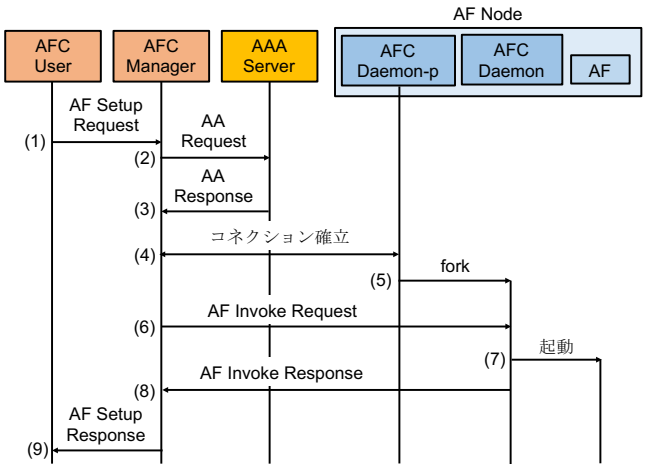

図 7　AF 設置のシーケンス (成功の場合)

の通信デバイスであり，パラメータ等の設定変更は可能であるがソフトウェアの変更はできないことを想定する．例としてはセンサデバイスや監視カメラなどが挙げられる．Server は COTS device の通信相手である．COTS device 上のアプリケーションは Server 上のアプリケーションと通信する．

4. 詳細設計

4.1 AF の設置シーケンス

AFC User は AFC で使用する AF を希望する AF Node に設置する．この作業を AF Setup と呼ぶ．**図 7** を用いて AF 設置のシーケンスを順を追って説明する．AFC User は AFC Manager とコネクションを確立し，AF Setup Request を AFC Manager に送信する（図 7（1））．AFC Manager は AF Setup Request を受信すると，AA（Authentication and Authorization）Request を AAA Server に送信する（図 7（2））．AAA Server は AA Request を受信すると，AFC User が AF を起動する権限を有するかを確認する．確認に成功すると AAA Server は AA Response を AFC Manager に送信する（図 7（3））．次に AFC Manager は AF Setup Request で指定された AF Node 上の AFC Daemon-p との間にコネクションを確立する（図 7（4））．AFC Daemon-p はコネクションを確立すると子プロセスである AFC Daemon を生成する（図 7（5））．結果として，AFC Manager と AFC Daemon の間にコネクションが確立し，以降の処理は AFC Daemon が行う．次に AFC Manager は AF Invoke Request を AFC Daemon に送信する（図 7（6））．AFC Daemon は AF Invoke Request を受信すると要求内で指定された実行形ファイルを起動し，AF とする（図 7（7））．その際，AFC Daemon は AF の標準入力と標準出力を自身の AF 入出力用ポートにそれぞれ接続しておく．また，AFC Daemon は AFC データパケットを送受信するためのポートと制御パケットを送受信する

Vol.2019-DPS-179 No.28
Vol.2019-MBL-91 No.28
Vol.2019-ITS-77 No.28
2019/5/24

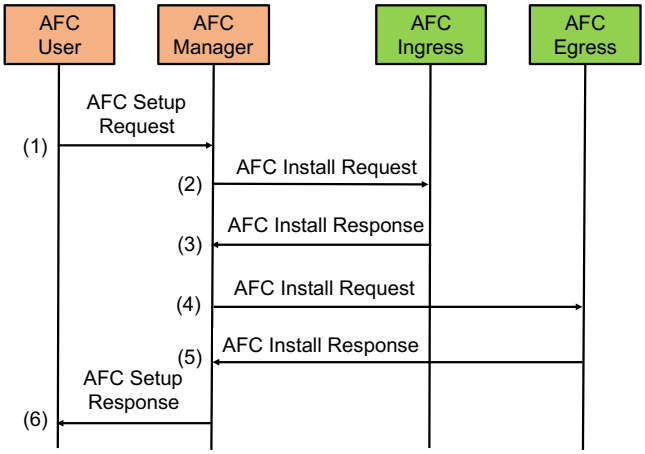

図 8　AFC 設置のシーケンス (成功の場合)

ためのポートを生成する．AF の起動に成功すると，AFC Daemon は AF Invoke Response を AFC Manager に送信する（図 7 (8)）．AFC Manager は AF Invoke Response を受信すると AFC Daemon とのコネクションを切断し，AFC Manager は AF Setup Response を AFC User に送信する（図 7 (9)）．AFC User は AFC Manager から AF Setup Response を受信すると，AFC Manager との間のコネクションを切断する．これをもって，AF の設置が完了する．

4.2　AFC の設定シーケンス

AFC User は AF Setup により AF を設置した後，それらを組み合わせ AFC を設定する．この作業を AFC Setup と呼ぶ．**図 8** を用いて AFC 設定のシーケンスを順を追って説明する．

AFC User は AFC Manager との間にコネクションを確立し，AFC Setup Request を AFC Manager に送信する（図 8 (1)）．AFC Setup Request には AFC を適用するデータパケットを識別するためのマッチフィールドの情報が含まれており，データパケットの対応するフィールドの値がこのマッチフィールドの値とすべて一致した場合，そのデータパケットには AFC が適用される．マッチフィールドには 5 タプル（始点 IP アドレス，終点 IP アドレス，始点ポート番号，終点ポート番号，プロトコル）を使用する．AFC Manager は AFC Setup Request を受信すると AFC Ingress との間にコネクションを確立し，AFC Install Request を送信する（図 8 (2)）．AFC Install Request には先述したマッチフィールドの情報や AF Chain を構成する AF，それらが設置されている AF Node の情報などが格納されている．AFC Ingress は AFC Install Request を受信すると Request 内の情報を保存し，AFC Install Response を AFC Manager に送信する（図 8 (3)）．AFC Manager は AFC Ingress から AFC Install Response を受信すると AFC Ingress との間のコネクションを切断する．次に AFC

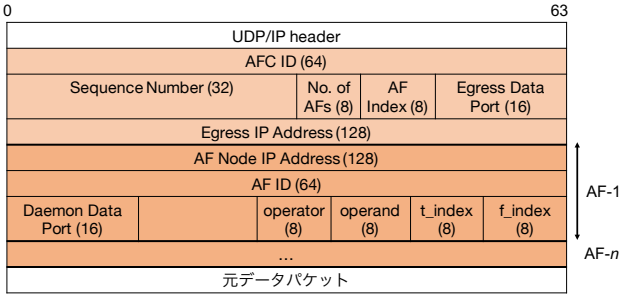

図 9　AFC のデータプレーンヘッダ

Manager は AFC Ingress 同様，AFC Egress に同内容の AFC Install Request を送信する（図 8 (4)）．AFC Egress も AFC Ingress 同様に Request 内の情報を保存し AFC Install Response を AFC Manager に送信する（図 8 (5)）．AFC Manager は AFC Egress から AFC Install Response を受信すると AFC Egress との間のコネクションを切断する．AFC Manager は両者の AFC Install Response の内容が成功であることを確認して AFC Setup Response を AFC User に送信する（図 8 (6)）．AFC User は AFC Manager から AFC Setup Response を受信すると，AFC Manager との間のコネクションを切断する．これをもって，AFC の設定が完了する．

4.3　データパケットへの AFC の適用

COTS デバイス上のアプリケーションから送信されたデータパケットに AFC を適用する際の処理を示す．まず，AFC のデータプレーンパケットのヘッダフォーマットについて**図 9** を用いて説明する．なお，AF Node 間の通信には UDP/IP を用いる．AFC を適用する前のデータパケットを元データパケットと呼ぶ，元データパケットは AFC Ingress に到達すると，AFC を実現するための新たな UDP/IP ヘッダと AFC ヘッダでカプセル化される．AFC ヘッダには AF Chain に含まれる AF の個数や AFC Egress といった固定長部分の後に，各 AF についての情報が AF の個数だけ続いている．データ転送に際して注目すべきフィールドについて説明する．`No. of AFs` フィールドには AF Chain に含まれる AF の個数が指定されており，この数だけ可変長部分である AF の情報が固定長部分の後に続く．`AF Index` フィールドには現在処理中の AF のインデックスが格納されている．AF のインデックスは AFC ヘッダに格納されている順番に 0 から設定される．`Egress IP Address` フィールドおよび `Egress Data Port` フィールドには AF を適用し終えた際にデータパケットを送信する AFC Egress の IP アドレスと待ち受けるポート番号が指定されている．`AF Node IP Address` および `Daemon Data Port` には適用する AF が存在する AF Node の IP アドレスと AFC Daemon が待ち受けるポート番号が指定されている．それに続く 4 つのフィールドは分岐処理のための関係演算の情

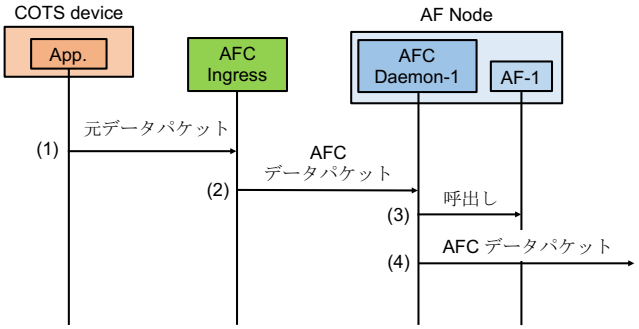

図 10 AFC 適用のシーケンス前半部分

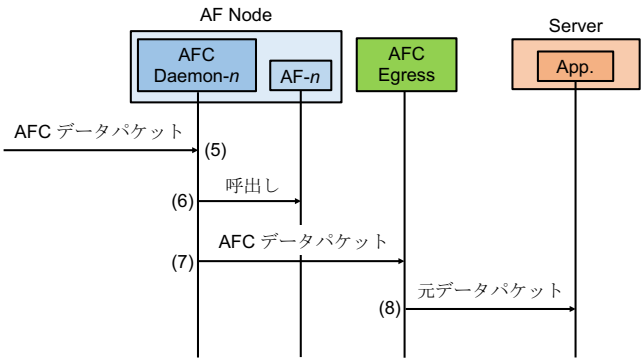

図 11 AFC 設置のシーケンス後半部分

報である．AF の返り値と `operand` フィールドの値に対し `operator` フィールドの関係演算子を適用し，結果が真であれば `t_index` フィールドの，偽であれば `f_index` フィールドで指定されたインデックスの AF が次に適用される．

データパケットに AFC を適用する際のシーケンスを**図 10** および**図 11** を用いて説明する．COTS device 上のアプリケーションが元データパケットを送信する（図 10 (1)）．AFC Ingress はこの元データパケットを受信すると AFC Setup の際に指定されたマッチフィールドの値と受信したデータパケットのフィールドを比較する．その結果，この元データパケットとマッチする AFC を見つけると，AFC Ingress は元データパケットの先頭に IP ヘッダ，UDP ヘッダおよび AFC ヘッダを付加し，AFC データパケットを生成する．IP ヘッダでは，始点 IP アドレスフィールドに AFC Ingress の IP アドレスを設定する．終点 IP アドレスフィールドには，最初に適用する AF が存在する AF Node の IP アドレスを AFC Setup で保存した情報から設定する．UDP ヘッダでは，始点ポート番号フィールドには AFC Ingress のデータパケット用ポート番号を設定する．終点ポート番号フィールドには，最初に適用する AF に関する AFC Daemon のポート番号を AFC Setup で保存した情報から設定する．上記の AFC データパケットは IP ヘッダの終点アドレスと UDP ヘッダの終点ポート番号に従い 1 つ目の AFC Daemon である AFC Daemon-1 に到達する（図 10 (2)）．AFC Daemon-1 は AFC ヘッダの内容から，自身が適用する AF（ここでは AF-1 とする）を認識し，元データパケットの内容に AF-1 を適用する（図 10 (3)）．AFC Daemon-1 は AF-1 を適用した際の返り値と AFC ヘッダの AF-1 の部分にある `operator`，`operand`，`t_index`，`f_index` フィールドの値を用いて，次の AF のインデックスを動的に決定する．AFC Daemon-1 は AFC ヘッダの次の AF に関するフィールドを参照し，`AF Node IP Address` フィールドの値を IP ヘッダの Dst IP フィールドに，また `Daemon Data Port` フィールドの値を UDP ヘッダの Dst Port フィールドに設定する．さらに AF Index フィールドの値を次の AF のインデックスの値に設定する．AFC Daemon-1 はこの AFC データパケットを送信する（図 10 (4)）．この AFC データパケットは IP ヘッダの終点アドレスと UDP ヘッダの終点ポート番号に従い，次の AFC Daemon に到達する（図 11 (5)）．各 AFC Daemon でも AFC データパケットの AFC ヘッダの `AF Index` フィールドの値に従ってここで適用する AF を認識し，元データパケットに AF を適用する（図 11 (6)）．AFC データパケットの送信と AF の適用を繰り返すうちに，AFC ヘッダの `AF Index` フィールドの値が `No. of AFs` フィールドの値と等しくなった場合，次のノードは AFC Egress となる．そこで AFC Daemon は AFC ヘッダの `AFC Egress IP Address` フィールドの値を IP ヘッダの Dst IP フィールドに設定し，`Egress Data Port` フィールドの値を UDP ヘッダの Dst Port フィールドに設定する．AFC Daemon はこの AFC データパケットを送信し，IP ヘッダの終点アドレスと UDP ヘッダの終点ポート番号に従い，AFC Egress に到達する（図 11 (7)）．AFC Egress は受信した AFC データパケットの AFC ヘッダに含まれる AFC ID を AFC Setup の際に得た情報から検索し，転送すべき Server の IP アドレスとアプリケーションのポート番号を元データパケットを認識する．AFC Egress は AFC データパケットから元データパケットを取り出し，認識した Server 上のアプリケーションへ転送する．これにより元データパケットは Server 上のアプリケーションに到達する（図 11 (8)）．

5. 実装

実装したエンティティは AFC User，AFC Manager，AFC Ingress，AFC Egress，AFC Daemon の 5 つである．実装には C 言語を用いた．今回の実装では IAAA Server による認証認可は実装していない．

AF Setup および AFC Setup において，様々なメッセージをやり取りするが，これらのフォーマットはそれぞれ個別に決められている．各種制御メッセージと AFC ヘッダは構造体で定義されている．AFC domain 内のエンティティは AF や AFC の情報を双方向リストのテーブルによって保持する．それぞれのテーブルは構造体で定義されており，テーブル内のフィールドの多くは制御メッセージの同

Vol.2019-DPS-179 No.28
Vol.2019-MBL-91 No.28
Vol.2019-ITS-77 No.28
2019/5/24

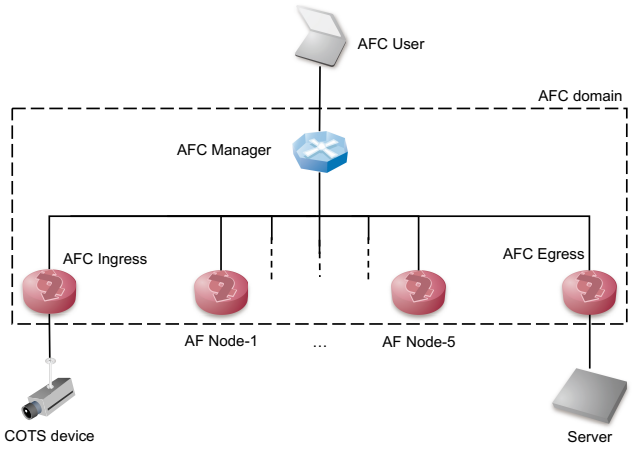

図 12　評価用トポロジ

表 2　PC のスペック

項目	内容
OS	Ubuntu 18.04 LTS
CPU	Intel（R）Core（TM）i3-6100, 3.50GHz
RAM	8GB
NIC	1Gbps × 5

名フィールドの値を参照している．各エンティティのプログラムにはテーブルの初期化や，テーブル内要素の検索，挿入，削除などテーブル操作用の関数群が実装されている．

6. 評価

6.1 評価内容

本節では基本的な性能を評価する．AF および AFC の設置に要する時間と共に，AFC を適用した際のスループットを評価する．

6.2 評価環境

今回の評価に用いるトポロジを図 12 に示す．AFC Manager，AFC Ingress，AFC Egress および 5 つの AF Node が AFC domain として一つのネットワーク上で接続されている．AFC domain 外のエンティティである AFC User は AFC Manager と接続されている．また，スループットを計測する際には COTS device および Server に見立てたマシンを用意し，それぞれ AFC Ingress，AFC Egress と接続した．それぞれの物理マシンの基本性能は表 2 に示すとおりである．また，今回の評価では AFC の基本性能を測るために単にデータを入出力する AF を用い，AF の返り値は常に 0 となっている．

6.3 AF 設置にかかるオーバーヘッド

AF の設置にかかるオーバーヘッドとして，1 つから 5 つの AF を別々の AF Node に設置した際にかかる時間を計測した．評価結果を図 13 に示す．計測はそれぞれ 10 回ずつ行い，図には平均値を示している．図 13 に示すとおり，AF の数が増えるにつれて AF 設置にかかる時間も大きくなることがわかる．設置にかかる時間は AF の数におおよそ比例しており，メッセージの配送にかかる時間を除くと，AF1 つにつき約 0.9 ms の時間を要する．しかしながら，実際の AFC の運用を想定した際，AF は AFC を適用する以前に 1 度だけ設置しその後再利用するのが一般的であるため，メッセージの配送と同程度であるミリ秒単位のオーダーであれば問題にならないと考えられる．

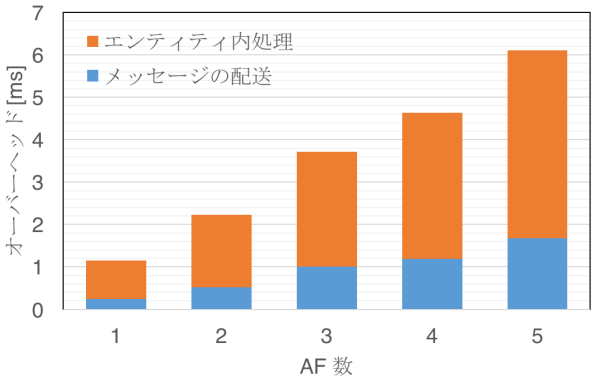

図 13　AF Setup にかかる時間

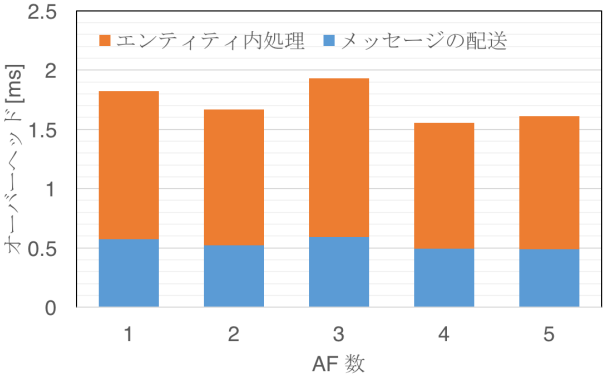

図 14　AFC Setup にかかる時間

6.4 AFC 設置にかかるオーバーヘッド

AFC の設置にかかるオーバーヘッドとして，1 つから 5 つの AF を含む AFC を設置した際にかかる時間を計測した．評価結果を図 14 に示す．計測はそれぞれ 10 回ずつ行い，図には平均値を示している．図 14 に示すとおり，AFC 設置の際には含まれる AF の数に関わらず 1.5~2 ms の時間を要し．メッセージの配送にかかる時間を除くと，内部での処理にかかる時間は約 1.4 ms 程度である．これは AFC Setup の場合には AF Node に関わらず AFC Manager，AFC Ingress，AFC Egress のみで処理されるためと考えられる．AFC のステートをヘッダで保持することにより，AFC の設置については低オーバーヘッドで処理することができ，このことから一度 AF の設置さえされていれば，複数の AF を含むアプリケーションにおいても

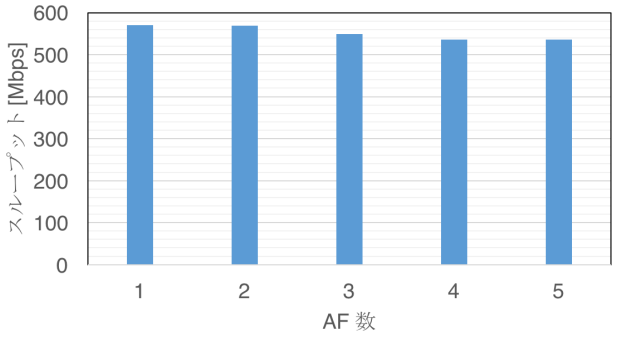

図 15 AFC 上のデータ転送のスループット

低遅延で対応することが出来ることや，AF の構成，マッチフィールドの条件といった設定の頻繁な変更にも対応することが出来ると言える．

6.5 AFC を適用した際のスループット

データパケットに AFC を適用した際のスループットを，経由する AF 数は 1 つから 5 つに変化させて計測した．これらの AF はそれぞれ別々の AF Node で起動され，元データパケットのペイロードは 1,024 バイトで固定されている．評価結果を**図 15** に示す．計測はそれぞれ 10 回ずつ行い，図には平均値を示している．1Gbps の NIC を利用しているため，帯域使用率は AF が 1 つの場合で約 55%である．AF の数が増えると少しずつ低減していき AF が 5 つの場合で約 52%となった．ステートをヘッダに保持していることから，ヘッダが帯域使用率をある程度占め，また AF の数が増えるとその分ヘッダの大きさが増加し帯域使用率を下げることが避けられない．AF を 5 つ含む AFC の場合，1 パケットにつき 192 バイトの AFC ヘッダが付加される．このことから，複数の AF Node による負荷分散を考慮する際にはコンピューティング資源よりもネットワーク資源を重視する必要があると言える．

7. おわりに

筆者らは，アプリケーションレベルのチェイニングを実現する通信機構である Application function Chaining（AFC）を提案しており，本稿では AFC Ingress や AFC Egress といった出入口となるノードを利用することで，Commercial off-the-shelf（COTS）デバイスのような，パラメータ等の設定変更は可能であるがソフトウェアの変更は出来ないデバイスに対しても AFC の適用を可能にした．

本稿では，この AFC for COTS devices のアーキテクチャを各エンティティ毎に Linux 上で実装し，単にデータを入出力する AF を用いて，AF および AFC の設置にかかる時間と AFC を適用した際のデータ転送のスループットを，AF の数を変化させながら評価した．AF の設置では，AF の数に比例するものの，メッセージの配送にかかる時間と同程度のオーダーで処理されることを示した．AFC の設置では，ステートをヘッダで保持することにより，AF の数に関わらずほぼ一定の時間で処理出来ることを示した．一方でスループットの計測ではヘッダによって帯域使用率は AF が 1 つの場合でも約 55%に留まることを示した．

本稿の執筆段階では，同一の AF が複数の AF Node に設置されていた際に，どの AF Node で処理するかを動的に決定することが出来ない．今後は AFC Manager によって各 AF Node の負荷を監視し，それによって AF を実行する AF Node を動的に決定するといった動作が考えられる．また，AFC は通信路の一部として提供されるため，本来であればデバイスからサーバに向けて送信されたパケットを AFC Ingress でキャプチャするべきであるが，今回の実装では簡単のためデバイスから直接 AFC Ingress にデータパケットを送信している．デバイスからサーバに向けて送信されたパケットをどのように AFC 側でキャプチャするかが実装上の課題となっている．

参考文献

[1] Hu, Y. C., Patel, M., Sabella, D., Sprecher, N. and Young, V.: Mobile Edge Computing A key technology towards 5G, ETSI White Paper 11, European Telecommunications Standards Institute (2015).
[2] Reznik, A., Arora, R., Cannon, M., Cominardi, L., Featherstone, W., Frazao, R., Giust, F., Kekki, S., Li, A., Sabella, D., Turyagyenda, C. and Zheng, Z.: Developing Software for Multi-Access Edge Computing, ETSI White Paper 20, ETSI (2017).
[3] Halpern, J. and Pignataro, C.: Service Function Chaining (SFC) Architecture, RFC 7665, IETF (2015).
[4] 渡邊大記：アプリケーションレベルのサービスチェイニングを実現する通信機構の設計と実装，修士論文，慶應義塾大学大学院理工学研究科 (2018).
[5] 渡邊大記，近藤賢郎，寺岡文男：通信と計算の融合に向けたアプリケーション処理連接基盤，信学技報，IA2018-37, Vol. 118, pp. 9–16 (2018).
[6] Cho, J., Sundaresan, K., Mahindra, R., Van der Merwe, J. and Rangarajan, S.: ACACIA: Context-aware Edge Computing for Continuous Interactive Applications over Mobile Networks, CoNEXT '16, pp. 375–389 (2016).
[7] Quinn, P., Elzur, U. and Pignataro, C.: Network Service Header (NSH), RFC 8300, IETF (2018).
[8] Previdi, S., Filsfils, C., Leddy, J., Matsushima, S. and Voyer, D.: IPv6 Segment Routing Header (SRH), Internet-Draft draft-ietf-6man-segment-routing-header-18, IETF (2019).
[9] Filsfils, C., Previdi, S., Bashandy, A., Decraene, B., Litkowski, S. and Shakir, R.: Segment Routing with MPLS data plane, Internet-Draft draft-ietf-spring-segment-routing-mpls-19, IETF (2019).
[10] Qazi, Z. A., Tu, C.-C., Chiang, L., Miao, R., Sekar, V. and Yu, M.: SIMPLE-fying Middlebox Policy Enforcement Using SDN, *SIGCOMM Comput. Commun. Rev.*, Vol. 43, No. 4, pp. 27–38 (2013).
[11] Nakamura, R., Okada, K., Saito, S., Tanahashi, H. and Sekiya, Y.: FlowFall: A Service Chaining Architecture with Commodity Technologies, *ICNP*, pp. 425–431 (2015).