Vol.2019-DPS-180 No.18
Vol.2019-EIP-85 No.18
2019/9/20

5G コアネットワーク向けアプリケーション処理連接基盤

佐藤 友範[1,a)] 渡邊 大記[1,b)] 林 和輝[1,c)] 近藤 賢郎[1,d)] 寺岡 文男[2,e)]

概要：第 5 世代移動通信方式（5G）では超高信頼低遅延通信や大容量モバイル通信などの高機能な通信サービスが提供され，自動運転や高精細な拡張現実感 (AR) のようなサービスが普及すると考えられている．本稿はエッジサーバ，フォグサーバ，クラウドサーバを含むような 5G コアネットワークを 1 台の計算機のように見せる処理基盤として Application Function Chaining（AFC）を提案する．AFC は高精細 AR のようなアプリケーションを小機能（Application Function; AF）ごとに分割し，AF の連接によってアプリケーションを構成する．アプリケーションは Pub/Sub 方式または HTTP リクエストにより AFC を利用する．AFC 内ではアプリケーションメッセージ単位で AF が適用される．本稿では，プロトタイプ実装により AFC の基本性能を評価した．AFC の確立では AF の設置よりも AF の連接にかかる時間的オーバーヘッドが大きく，AFC 上でのデータ通信ではアプリケーションメッセージ長が 10KB 以上の場合で 90%以上の帯域使用率となった．

Application Function Chaining Infrastructure for a 5G Core Network

1. はじめに

次世代の無線通信システムである第 5 世代移動通信方式（5G）の導入が現在進められている．5G コアネットワークは以下の 3 つの通信サービスを要件としている．

- eMBB（enhanced Mobile Broadband）
- URLLC（Ultra-Reliable and Low Latency Communications）
- mMTC（massive Machine Type Communication）

5G により高機能な通信サービスが提供されることで，これらの要件を想定したサービスが普及することが考えられる．例えば，eMBB であればより高精細な VR（Vritual Reality）や AR（Augmented Reality)，URLLC であれば自動運転や遠隔医療，mMTC であれば大量のセンサを用いた IoT などが挙げられる．これらのサービスは，端末 (UE: User Equipment) からの入力を様々なファンクションが処理し，その結果を UE やデータベースに出力することで実現される．**表 1** に 5G におけるサービスと入力，処理，出力の例を示す．

表 1 5G におけるサービス例と入力，処理，出力

サービス	処理	内容
AR	入力	実際の映像，ユーザの視点，ユーザの操作
	処理	映像の加工
	出力	加工された映像・音声
自動運転	入力	自車の情報（現在位置，スピード，進行方向）
	処理	他車の情報や周辺情報の取得，計算
	出力	自車が次に取るべき行動
IoT	入力	（大量）センサからの（大量）データ
	処理	データの集約，データの加工
	出力	DB への収納

一方でネットワーク上でデータを分散処理する Edge/Fog/Cloud computing が近年注目を集めている．Edge/Fog/Cloud computing で前述したファンクションを実行することで，5G コアネットワーク全体を 1 つの計算機とみなして扱うことが出来る．また，データの入力 UE と出力 UE が異なる場合，データの通信路上で計算を行うことから，in-network computation あるいは計算と通信の融合とも言える．Edge/Fog/Cloud computing においては

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) latte@inl.ics.keio.ac.jp
e) tera@keio.jp

ファンクションの最適な配置が重要であり，具体的には低遅延を要求する処理はエッジサーバで実行することや，複雑な処理はクラウドサーバで実行することなどが挙げられる．

ネットワーク上のファンクションを複数連接してネットワークサービスを提供する技術として，サービスチェイニングが存在する．サービスチェイニングは主に通信事業者がファイアウォールなどのネットワーク機能を連接する用途で用いられる．このサービスチェイニングをそのまま一般的なサービスで利用するには問題点がある．従来の技術においてはどのようなファンクションを用いるかは通信事業者によって事前に指定されることを想定しており，利用者の視点からサービスのセマンティクスを理解してチェイニングすることは難しい．そのため，ファンクションの実行位置についても考慮されておらず，そのファンクションにとって最適な配置を実現することが出来ない．

小さな機能を複数組み合わせて高機能なサービスを提供する手法はアプリケーションのアーキテクチャにも見られる．昨今，WEB アプリケーションを代表する多くのアプリケーションがマイクロサービスアーキテクチャを採用している．マイクロサービスアーキテクチャはアプリケーションの機能を互いに独立する最小限の構成要素に分解し，それらが互いに通信することでアプリケーションを成立させるアーキテクチャである．分解された機能は独立してデプロイされ，RPC（Remote Procedure Call）で他の機能を呼び出すという点で Edge/Fog/Cloud computing とも相性が良い．しかし，計算資源を提供する事業者をまたぐと使うのが難しいという問題点があり，5G の高機能なネットワークを活用したより柔軟なサービスを提供するためには事業者の枠を超えたインテグレーションを考慮した仕組みが必要だと考える．

本稿ではエッジサーバ，フォグサーバ，クラウドサーバを含むような 5G コアネットワーク向けのアプリケーション基盤を提案する．アプリケーションを AF（Application Function）と呼ぶファンクションに分割し，AF の連接（AFC: AF Chaining）によって一つのアプリケーションを構成する．この AF の連接で構成されるアプリを AF-Chained Application（AFC Application）と呼ぶ．また，AFC Application を利用する UE 上のアプリケーションを User Application と呼ぶ．図 1 は 5G ネットワークにおける AFC Application の例を示したものである．たとえば低遅延を要求する自動運転の処理はエッジサーバのみで行われる．高精細な画像を扱う AR では，軽い計算はエッジサーバが行い，複雑な計算はクラウドサーバが行う．大量センサからのデータは，通信路においてエッジサーバやフォグサーバによりデータの圧縮や統合が行われ，データベースに格納される．AFC が適用される領域を AFC domain と呼び，領域内の複数のサーバに AF が配置される．AFC の特徴の一つとして，製作者が異なる AF を組み合わせて利用できる点が挙げられる．なお，AF を最適なエッジ/フォグ/クラウドサーバに配置するための MANO（Management and Orchestration）は本稿の範囲外とする．

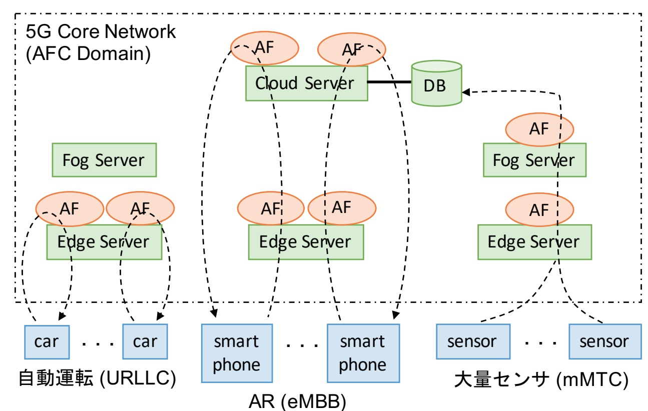

図 1　5G Core Network と AF-Chained Application

AFC Application へのデータの入出力は，5G コアネットワークへの Pub/Sub（Publish/Subscribe）方式，あるいは HTTP POST と HTTP GET により行い，5G コアネットワーク内では複数 AF の逐次処理または並列処理が行われる．AF の連接は直鎖状だけでなく，条件による分岐，合流，環状といった形状も実現可能にする．本稿では，AFC のプロトタイプ実装により基本性能を明らかにする．

2. 関連研究・関連技術

2.1 エッジ/フォグ/クラウドコンピューティング

ACACIA Mobile Edge Network [1] は LTE コア網に MEC（Mobile Edge Computing）を実現するシグナリングプロトコルを SDN（Software Defined Networking）や NFV（Network Functions Virtualization）で実現する研究である．ACACIA が対象とするのは，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．ACACIA は通信先を SDN で決定することにより、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という利点を持つ．

文献 [2] は低遅延性を要求する AR アプリケーションを提供することを目的として，エッジとクラウドが連携したシステムを利用するためのフレームワークを提案している．エッジおよびクラウドのリソースやネットワークの帯域といったパラメータにより，エッジとクラウドの適切なバランスが変化することが示されており，エッジ/フォグ/クラウドコンピューティングにおいてファンクションを自在に配置できることは重要な要求事項であると考えられる．

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

ⓒ 2019 Information Processing Society of Japan

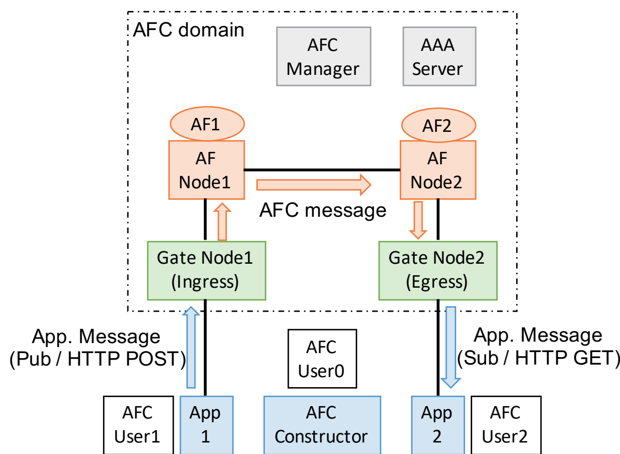

図 2　AFC アーキテクチャ

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 [4] や，IPv6 Segment Routing（SRv6） [5]，MPLS Segment Routing（SR-MPLS）[6] などで研究および標準化が進められている．

2.3　マイクロサービスアーキテクチャ

マイクロサービスアーキテクチャは分散型のシステムアーキテクチャとして，それまで多く採用されていた中央集中型である Service-Oriented Architecture に取って代わり注目されている．文献 [7] は両者の特徴や違いについて述べている他，サービス間の相互作用やリソースの考慮など様々なアプローチで近年研究がなされていることを示している．また，マイクロサービスアーキテクチャは分散型であることから Edge/Fog/Cloud computing とも相性が良い．文献 [8] はユースケースとして ShareLatex を用いて，実際にマイクロサービスアーキテクチャを採用しているアプリケーションを Edge/Core Hybrid な環境にデプロイした際の遅延時間について評価している．AFC は大きいサービスを小さな機能に分割するという点でマイクロサービスアーキテクチャと共通しているが，ファンクションの開発者や，計算資源およびネットワーク資源を提供する事業者の枠を超えたマイクロサービスのインテグレーションを考慮するという点で異なる．

3.　AFC アーキテクチャ

3.1　AFC の構成要素

本稿が提案する AFC のアーキテクチャを図 **2** に示す．AFC User は AFC Application の利用者であり，AFC を

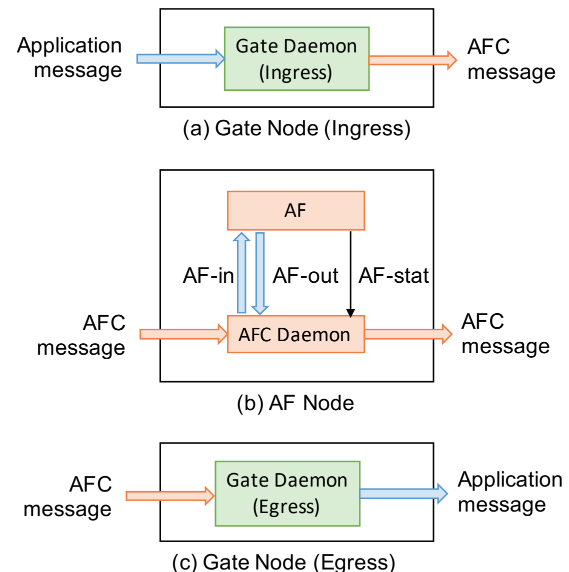

図 3　GateNode / AF Node の構造

利用するアプリケーションである User App を実行する．User App が動作するノードを App Node と呼ぶ．App Node は 5G システムにおける UE に相当する．この例では AFC User1 が User App1 を実行し，AFC User2 が User App2 を実行している．AFC Constructor は AFC の設置や削除を行う．この例では AFC User0 が AFC Constructor を実行している．User App が AFC Constructor を兼ねてもよい．AFC が適用される領域を AFC domain と呼ぶ．たとえば，1 つの 5G コアネットワークが AFC Domain に相当する．AFC domain には AFC Manager と AAA（Authentication, Authorization, and Accounting） Server が存在する．AFC Manager は AF や AFC などの設置や削除を管理する．AAA Server は AFC User の認証，認可およびアカウンティングを行う．このアカウンティングでは，AFC User が計算資源をどの程度消費したかを記録し，AFC User への課金に利用する．AFC domain と App Node の境界には Gate Node を配置する．User App からアプリケーションメッセージを受信する Gate Node を Ingress とも呼ぶ．一方，User App へアプリケーションメッセージを送信する Gate Node を Egress とも呼ぶ．1 台の Gate Node が 1 つの AFC に関する Ingress と Egress の双方として動作してもよい．5G では，コアネットワークと無線リンクの境界に置かれる RAN という装置が Gate Node に相当する．AFC domain 内で AF が動作するノードを AF Node と呼ぶ．エッジサーバ，フォグサーバ，クラウドサーバが AF Node に相当する．

3.2　Gate Node と AF Node の詳細

図 **3** は Gate Node および AF Node 内の詳細図である．Gate Node では Gate Daemon と呼ばれるプロセスが動作している．Gate Daemon は AFC ごとに生成される．

ⓒ 2019 Information Processing Society of Japan

Ingress はアプリケーションメッセージを入力とし，これにAFC ヘッダを付加した AFC メッセージを出力する（図 3 (a)）．Egress は AFC メッセージを入力とし，これから AFC ヘッダを除去して最終的に User App に送信するアプリケーションメッセージを出力する（図 3（c））．AF Node には AFC Daemon と呼ばれるプロセスが動作している（図 3（b））．AF の起動要求があると，AFC Daemon は AF を起動するとともに，起動した AF との間に 3 つの入出力パスを持つ．アプリケーションメッセージを受け渡しする AF-input と AF-output，および AF からの状態出力である AF-status である．AFC Daemon は AF の状態出力によって次にアプリケーションメッセージを送信する AFC Daemon を切り換えることができる．（5.3 節参照）

3.3 User App と AFC のインタフェース

User App による AFC 利用には Pub/Sub（Publish/Subscribe） 方式もしくは HTTP リクエストを用いる．送信側の User App は，利用する AFC を指定して Publish または HTTP リクエスト（POST）によりアプリケーションメッセージを Ingress に送信する．一方，受信側の User App は，利用する AFC を指定して Subscribe または HTTP リクエスト（GET）により Egress からのアプリケーションメッセージの受信を要求する．1 つの User App が送信側と受信側を兼ねてもよい．

3.4 AFC の形状

図 4 に AFC で扱うことが出来るチェイニングの形状を示す．直鎖状（図 4（a））の他，AF 実行時の状態出力による条件分岐（図 4（b））や合流（図 4（c））を持つことが出来る．また，同じ AF を複数配置することによる並列処理（図 4（d））や環状のチェイニング（図 4（e））も可能である．

3.5 想定する環境

上記で提案した AFC アーキテクチャは以下のような環境を想定している．AFC Domain は自ドメインで利用可能な AF の仕様や使用料を Web 上などで公開する．AFC Domain が AF を作成してもよいし，一般ユーザが作成した AF を AFC Domain が自ドメインに登録してもよい．AFC Constructor は公開された AF の情報を基に AFC を構成し，AFC App を作成する．AFC Constructor の実行者（図 2 における AFC User0）は AFC Domain に属していてもよいし，一般ユーザでもよい．作成された AFC App の仕様や使用料もまた Web 上などで公開する．AFC User は公開された AFC の仕様を参照し，これを利用する．User App を作成，実行してもよいし，AFC Domain や他のユーザが作成した User App を利用してもよい．User App を実行した AFC User には，AFC App の使用状況に応じて

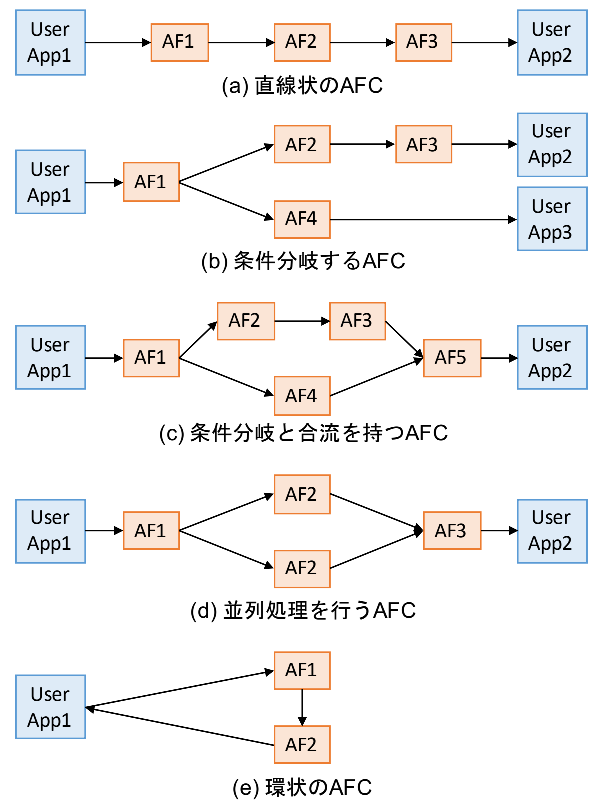

図 4 AFC の形状

課金され，利用料は AF の作成者，AFC の作成者，AFC Domain に配分される．

4. AFC の確立

図 2 に示す直線状 AFC を確立する場合を考える．AFC の確立は以下の 4 つのステップからなる．

(1) AFC Constructor は AFC Manager を介して Gate Node1 と 2 にそれぞれ Gate Daemon1 (Ingress) と 2 (Egress) を起動する．
(2) AFC Constructor は AFC Manager を介して AF Node1 と 2 にそれぞれ AF1 と 2 を起動する．
(3) AFC Constructor は AFC Manager を介して Gate Node1→AF Node1→AF Node2→Gate Node2 という直線状 AFC を構成する．AFC を構成する各ノード間には TLS コネクションが確立されるため，メッセージの盗聴や改竄は行われない．
(4) AFC User1 と 2 は AFC Manager を介して上記で作成した AFC のセッションキーを取得する．

ステップ 3 で AFC を構成する際には，AF 間の接続関係と次に適用する AF を動的に決定するための条件が各 AFC Daemon に設定される．ステップ 1 から 3 により，AFC Constructor は作成した AFC の利用権限を示すセッションキーを AFC Manager と共有する．AFC を利用するのは User App1 と 2 なので，User App1 と 2 は AFC Manager を介して AFC のセッションキーを取得する．その際，AFC

ⓒ 2019 Information Processing Society of Japan

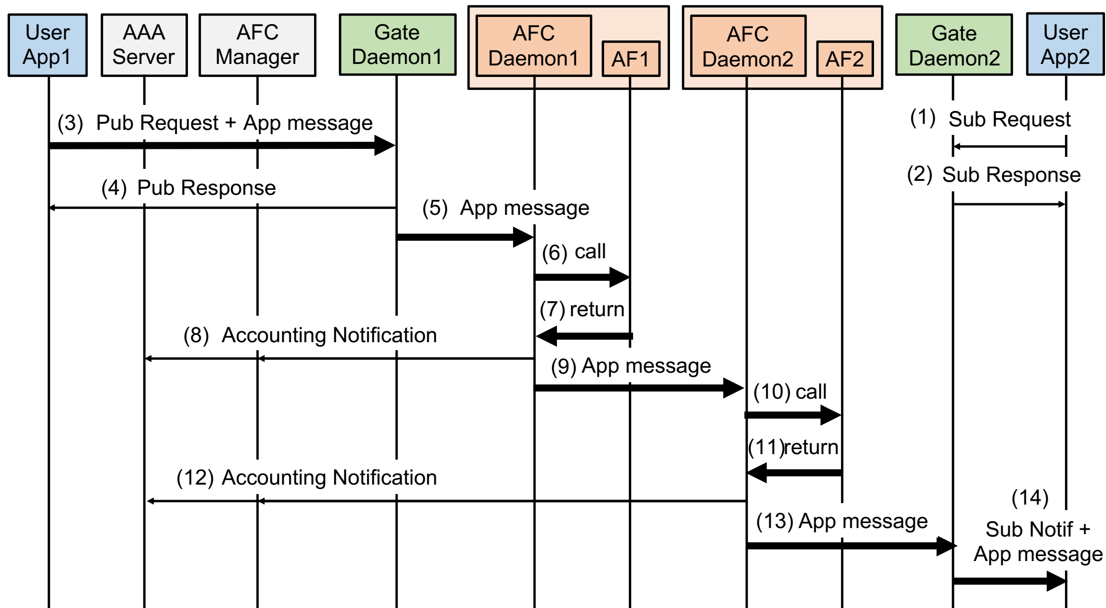

図 5　AFC でのデータ通信手順

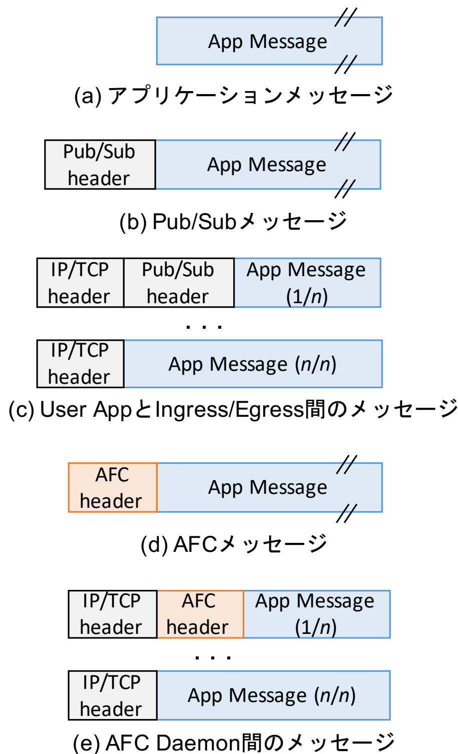

図 6　AFC 上でのメッセージ

Manager は User App1 と 2 の利用者である AFC User1 と 2 が，AFC Constructor が作成した AFC を利用する権限を有するかを，AAA Server に問い合わせて検証する．セッションキーは User App1 と 2，AFC Daemon1 と 2，Gate Daemon1 と 2 で共有される．たとえば AF1 と 2 および AFC は AFC Domain が提供し，AFC User1 と 2 がこの AFC を利用するような場合，AFC を作成する AFC User0 は AFC Domain に属し，AFC を利用する AFC User1 と 2 は AFC Domain と契約している一般ユーザとなる．

5. AFC でのデータ通信

5.1 データ通信手順

図 5 に AFC でのデータ通信手順を示す．この例では User App1 から送信されたアプリケーションメッセージ（図 6（a））が 2 つの AF で処理され User App2 へと届く．まず，User App2 は Egress である Gate Daemon2 とデータ用 TLS コネクションを確立し，Subscribe Request を送信する (図 5 (1))．Subscribe Request は Certificate フィールドを持ち，Gate Daemon2 はこの値から User App2 が AFC を利用する権限を有することを検証し Subscribe Response を User App2 に送信する（図 5（2））．

User App1 はアプリケーションメッセージに Pub/Sub ヘッダを付加した Publish Request（図 6（b））を Gate Daemon1 に送信する（図 5（3））．Publish Request が 1 つの TCP セグメントに収まらない場合には複数の TCP セグメントに分割して送信される（図 6（c））．Pub/Sub ヘッダには適用したい AFC の ID を格納するフィールドや AFC の利用権限検証用の Certificate フィールドが存在する．Gate Daemon1 は User App1 が AFC を利用する権限を有することを検証し Publish Response を User App1 に送信する（図 5（4））．

Gate Daemon1 はアプリケーションメッセージに AFC ヘッダを付加した AFC メッセージ（図 6（d））を作成し，AFC Daemon1 に送信する (図 5 (5))．その際，AFC メッセージが 1 つの TCP セグメントに収まらない場合には複数の TCP セグメントに分割して送信される（図 6（e））．

AFC Daemon1 は複数の TCP セグメントから AFC メッセージ（図 6（d））を再構成し，AFC メッセージから取

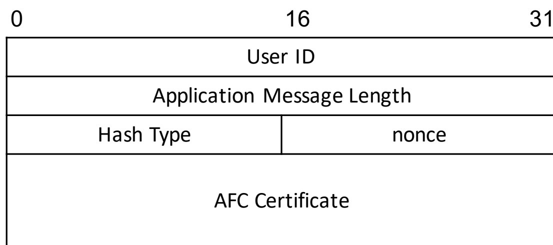

図 7 AFC ヘッダ

り出したアプリケーションメッセージ（図 6（a））を AF に入力用ポートから入力する（図 5（6））．AFC Daemon1 は AF の出力用ポートからアプリケーションメッセージを受信するとともに，状態出力用ポートから AF1 の終了状態値を得る（図 5（7））．AFC Daemon1 は AF1 の実行において使用した資源量を表す Accounting Notification を AAA Server に送信し，AAA Server は利用情報を記録する（図 5（8））．AFC Daemon1 はアプリケーションメッセージに AFC ヘッダを付加した AFC メッセージ（図 6（d），（e））を AFC Daemon2 に送信する（図 5（9））．

AFC Daemon2 でも AFC Daemon1 と同様にアプリケーションメッセージを AF に入力し（図 5（10）），アプリケーションメッセージと終了状態値を AF から受信する（図 5（11））．AFC Daemon2 が Accounting Notification を AAA Server に送信し（図 5（12）），利用情報を記録した後，AFC Daemon2 は AFC メッセージを Gate Daemon2 に送信する（図 5（13））．

Gate Daemon2 は複数の TCP セグメントから AFC メッセージ（図 6（d））を再構成し，AFC メッセージから取り出したアプリケーションメッセージに Pub/Sub ヘッダを付加した Subscribe Notification（図 6（b），（c））を User App2 に送信する（図 5（14））．

最終的に User App2 はアプリケーションメッセージ（図 6（a））を受信する．User App1 と 2 はそれぞれ Gate Daemon との間のデータ用 TLS コネクションを解放しデータ通信が完了する．

5.2 AFC ヘッダ

図 7 に AFC ヘッダのフォーマットを示す．User ID フィールドには AFC ユーザの識別子が格納される．Application Message Length フィールドにはアプリケーションメッセージ長が格納される．Hash Type フィールドには，AFC Certificate を作成するために使用するハッシュ関数の種類が格納される．nonce フィールドには乱数が格納される．AFC Certificate フィールドには，User App が AFC を使用する権限を有していることを示す証明書が格納される．

AFC Certificate は以下のように作成される．4 章で，User App は，AFC Daemon や Gate Daemon (Ingress/Egress)

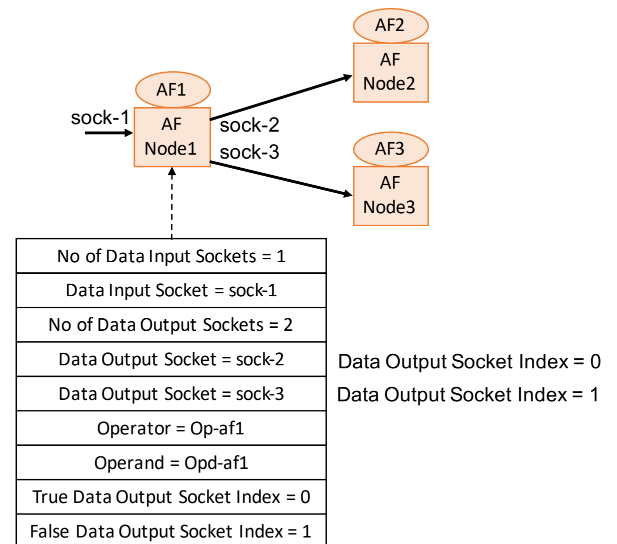

図 8 条件分岐の例

とセッションキーを共有することを述べた．User App は Publish Request を Ingress に送信する際，アプリケーションメッセージの内容にセッションキーを連結し，これにハッシュ関数を適用してハッシュ値を得る．Pub/Sub ヘッダには使用したハッシュ関数の種別と得られたハッシュ値を格納する．Publish Request を受信した Ingress は同様にしてハッシュ値を計算し，その結果が Pub/Sub ヘッダに格納された値と一致するかを確認する．Ingress は AFC Daemon に AFC メッセージを送信する際，同様にしてハッシュ値を計算し，ハッシュ関数の種別とハッシュ値を AFC ヘッダに格納する．

5.3 条件分岐

3 章で AFC は条件による分岐を持つことが出来ると述べた．条件分岐により次に適用する AF を動的に決定する際の処理を**図 8** を用いて説明する．AFC 確立の際に AF 間の接続関係と分岐条件は各 AFC Daemon の管理テーブルに保持される．AF 間の接続関係は通信の際に用いるソケットとしてテーブルに格納されており，順番にインデックス番号が付与されている．図の例では，sock-2 にはインデックス番号 0 が，sock-3 にはインデックス番号 1 が割り当てられている．分岐の条件は Operand フィールド，Operator フィールド，True Data Output Socket Index フィールド，および False Data Output Socket Index フィールドの 4 つからなる．AF を適用した際，AF の状態出力の値と Operand フィールドで示されたオペランドに Operator フィールドで示された関係演算子を適用し，結果が真であれば True Data Output Socket Index フィールドで示されたソケット（この例では sock-2）へ，結果が偽であれば False Data Output Socket Index フィールドで示されたソケット（この例では sock-3）へ AFC メッセージが送信さ

ⓒ 2019 Information Processing Society of Japan

れる．

6. 実装

本稿が提案する AFC アーキテクチャのプロトタイプを C 言語を用いて実装した．実装モジュールは User App，AFC Constructor，AFC Manager，AAA Server，Gate Daemon，AFC Daemon，ダミーの AF であり，コード量は約 2500 行である．今回は簡略化のため，エンティティ間のコネクションは TLS ではなく通常の TCP コネクションを用いた他，セッションキーや Certificate フィールドはそれらを模した定数の値を送信するのみとなっている．AFC domain 内のエンティティは AFC の情報を双方向リストのテーブルによって保持する．

Gate Node では Gate Daemon-p と呼ばれるプロセスが動作している．Gate Daemon の設置要求があると，Gate Daemon-p は AFC ごとに子プロセスとして Gate Daemon を `fork()` により生成する．AF Node においても，同様に AFC Daemon-p と呼ばれるプロセスが動作しており，AF の設置要求に応じて AFC Daemon を `fork()` により生成し，AFC Daemon は AF を `fork()` により生成する．また，AFC Daemon と AF の間の 3 つの入出力パスにはパイプを用いている．

7. 評価

本章では基本的な性能を評価する．Gate Daemon，AF および AFC の設置に要する時間と共に，AFC でのデータ通信におけるスループットを評価する．

7.1 評価環境

今回の評価に用いるトポロジを**図 9** に示す．AFC Manager，AAA Server，2 つの Gate Node および 4 つの AF Node が AFC domain として一つのネットワーク上で接続している．AFC domain 外のエンティティである 3 つの App Node は AFC Manager と TCP で接続しており，そのうち App Node1 は Gate Node1 と，App Node2 は Gate Node2 ともそれぞれ TCP で接続している．各マシンは物理マシンであり，それぞれの仕様は**表 2** に示すとおりである．今回の評価では AFC の基本性能を測るために単にアプリケーションデータを入出力するダミーの AF を用い，AF の状態出力は常に 0 となっている．AF は直鎖状に連接されており，分岐の設定では常に同じ AFC Daemon に AFC メッセージが送信されるようになっている．各評価の計測回数はそれぞれ 20 回であり，各評価結果の図では平均値とともに最大値および最小値を示す．

7.2 Gate Daemon と AF 設置のオーバーヘッド

図 10 に，それぞれ単一の Gate Daemon および AF 設置に要する時間を示す．どちらも 1.5 ミリ秒程度の時間を要する．この内，制御メッセージの配送にかかる時間は Gate Daemon 設置の場合で 5×RTT，AF 設置の場合で 6×RTT である．今回は RTT が極めて小さい環境で評価しており，実際のネットワークではネットワーク遅延による影響が支配的になると予想できる．また，今回の実装では複数の Gate Daemon や AF を設置する際に並列処理はせず 1 つずつ順番に処理していくため，設置するエンティティの数だけこの評価で得られた時間がかかると考えられる．

7.3 AFC 設置のオーバーヘッド

図 10 に，連接する AF 数を変化させた場合に AFC の設置に要する時間を示す．AF が 1 つの場合で 4.44 ミリ秒の時間を要し，制御メッセージの配送に 7×RTT の時間を要する．AF の数に比例して AFC の設置にかかる時間も増加していき，AF が 4 つの場合で 9.66 ミリ秒の時間を要する．AF の数が一つ増えると，制御メッセージの配送が 3×RTT の時間分増加する．今回は RTT が極めて小さい環境での

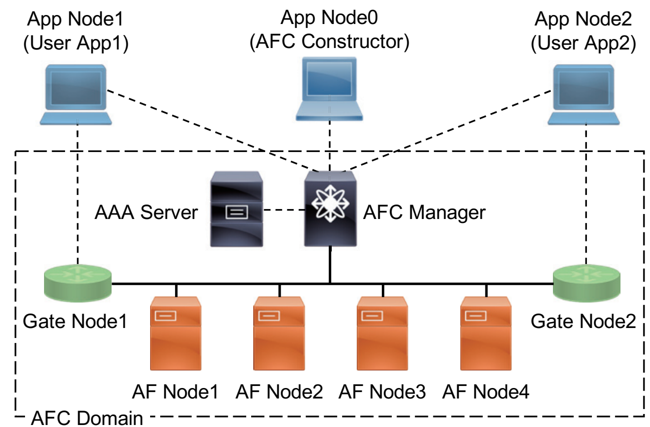

図 9 評価用トポロジ

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

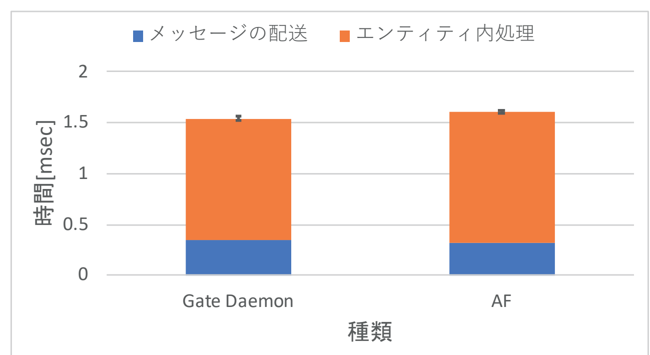

図 10 Gate Daemon と AF の設置にかかる時間

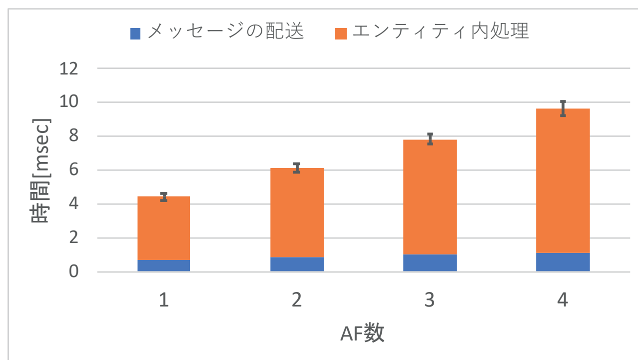

図 11　AFC の設置にかかる時間

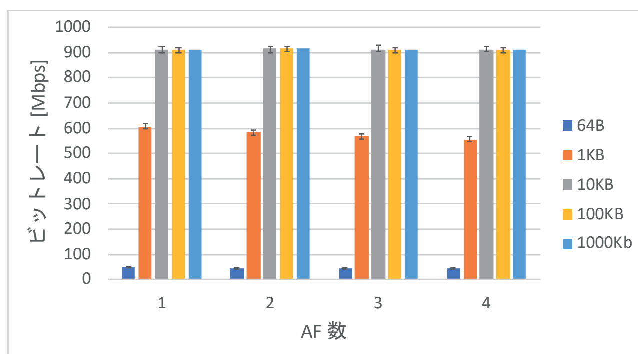

図 12　データ転送スループット

評価であるが，AF の数が増加するとエンティティ内での処理が全体の処理時間に与える影響も大きくなっていく．

7.4　データ通信スループット

本稿での AFC の設計では，AFC Daemon と AF は別プロセスとして実行されるため，AF の実行の際には AFC Daemon から AF へ，AF から AFC Daemon へ，という 2 回のアプリケーションメッセージのデータコピーが生じる．そこでこのようなデータコピーのオーバヘッドを測定するため，AFC でのデータ通信スループットを 5 つのメッセージサイズ（64B，1KB，10KB，100KB，1000KB）の場合で，経由する AF 数を 1 つから 4 つに変化させながら測定した．64B は IoT などのセンサ値の通信を想定しており，1000KB は AR などでの高精細画像の通信を想定している．**図 12** に測定結果を示す．ここで示した値は AFC ヘッダなどを含まないアプリケーションメッセージのみの値である．経由する AF 数が 1 つのとき，メッセージサイズが 64B の場合には 48Mbps，メッセージサイズが 1KB の場合には 607Mbps となった．またメッセージサイズが 1KB の場合，AF の数が増加するとスループットが 1 つにつき約 15Mbps 低下した．メッセージサイズが小さい場合には AF の影響が大きいことが分かった．メッセージサイズが 10KB よりも大きい場合には物理リンクの 90%程度のスループットとなっており，経由する AF の数が増加してもスループットへの影響が小さいことが分かった．

8.　おわりに

本稿ではエッジサーバ，フォグサーバ，クラウドサーバを含むような 5G コアネットワーク向けのアプリケーション連接基盤である Application Function Chaining（AFC）を提案した．アプリケーションの機能を AF と呼ぶ小機能に分割し，AF の連接によって一つのアプリケーションを構成する．AF の連接は直鎖状だけでなく，条件による分岐，合流，並列処理，環状といった形状も可能である．アプリケーションの利用には Publish/Subscribe 方式もしくは HTTP リクエストを用い，アプリケーションメッセージ単位で AF を適用する．AF はメッセージの入出力とは別に状態出力を持ち，この値に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．

本稿では一連の処理に関するプロトタイプを実装し，データを入出力するだけの AF を用いて基本性能を評価した．実際のネットワーク遅延を考慮すると，Gate Daemon や AF の設置よりもチェイニングを実際に確立する AFC 設置の時間的オーバーヘッドが大きいことが示された．AFC でのデータ通信に関してはメッセージサイズが 10KB よりも大きい場合に物理リンクの 90%以上の帯域使用率となり，AF の実行によるスループットへの影響が小さいことを示した．今後の展望としては，計算資源やネットワーク資源を考慮した AF のオーケストレーションや，複数の入出力を持つ柔軟な形での AF の実行，連接などが挙げられる．

参考文献

[1] 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).
[2] Maheshwari, S., Raychaudhuri, D., Seskar, I. and Bronzino, F.: Scalability and Performance Evaluation of Edge Cloud Systems for Latency Constrained Applications, *SEC*, pp. 286–299 (2018).
[3] Halpern, J. and Pignataro, C.: Service Function Chaining (SFC) Architecture, RFC 7665, IETF (2015).
[4] Quinn, P., Elzur, U. and Pignataro, C.: Network Service Header (NSH), RFC 8300, IETF (2018).
[5] 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).
[6] 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).
[7] Cerny, T., Donahoo, M. J. and Trnka, M.: Contextual Understanding of Microservice Architecture: Current and Future Directions, *SIGAPP Appl. Comput. Rev.*, Vol. 17, No. 4, pp. 29–45 (2018).
[8] Tato, G., Bertier, M., Rivière, E. and Tedeschi, C.: ShareLatex on the Edge: Evaluation of the Hybrid Core/Edge Deployment of a Microservices-based Application, *MECC*, pp. 1–6 (2018).

ⓒ 2019 Information Processing Society of Japan