一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
IN2019-120(2020-03)

ワークフローにおけるICNを用いたファンクションチェイニング

金光　永煥[†,††]　花田　真樹[†††]　金井　謙治[††]　中里　秀則[††††]

† 東京工科大学コンピュータサイエンス学部
†† 早稲田大学理工学術院総合研究所
†††† 早稲田大学理工学術院基幹理工学部情報通信学科
†††† 東京情報大学総合情報学部

あらまし　近年のネットワーク仮想化の普及に伴い，ネットワーク機能の連携のためにファンクションチェイニングが用いられている．ネットワーク機能だけではなく，より包括的な処理の連携をサービスファンクションチェイニング（SFC）という．また，チェイニングは一般に逐次的な機能連携に用いられるが，今後，アプリケーションの複雑化に伴ってワークフロー型のSFCが必要になるものと考えられる．したがって，ワークフロー型のSFCにおいて，いかにしてファンクションを呼び出すかが課題である．そこで本稿では，ワークフロー型のSFCにおいて，ICNを用いたファンクション連携手法を提案する．集中管理によって予めスケジュールして配備されたファンクションに対する静的チェイニング手法と，各ノードで自律的にスケジュールしてチェイニングする動的チェイニング手法について述べる．また，実環境による実験により，双方の特性について明らかにする．

キーワード　NFV, Service Function Chaining，ICN，SFC，クラスタリング，スケジューリング，ワークフロー．

ICN-based Service Function Chaining for Workflows

Hidehiro KANEMITSU[†,††], Masaki HANADA[†††], Kenji KANAI[††], and Hidenori NAKAZATO[††††]

† School of Compute Science, Tokyo University of Technology.
†† Waseda Research Institute for Science and Engineering, Waseda University.
†††† Department of Communications and Computer Engineering, Waseda University.
†††† Department of Communications and Computer Engineering, Waseda University.

Abstract　With spreading of network virtualization technologies in the world, each network function is processed in cooperation with others to achieve Service Function Chaining (SFC). A generalized structure of such a chain is a workflow, that can be more critical structure when a complex-type structured application must be processed. Thus, how each service function should be processed efficiently is one important issue for practical workflow SFC. In this paper, we propose two variants of INC-SFC for workflow. The one is that a workflow SFC is supposed to be processed according to a static SF scheduling algorithm. The other is that each computational resource has its own neighbors in FIB, to which the target for the interest packet is sent. We also present the comparison results among them by an actual environment.

Key words　NFV, Function Chaining, ICN, SFC, Clustering, Scheduling, Workflow.

1. はじめに

今日のネットワーク仮想化（NFV: Network Function Virtualization）の発展に伴い，様々なネットワーク機能（VNF: Virtualized Network Function）が仮想環境上で配備され，そして可搬性によって移動可能となった[1], [2]．例えばルータ，ファイアウォール，ロードバランサ等がハードウェアの制約を意識せずに配備できることから，経済面や運用上でのコストにおいての利点がある．このようにハードウェア機能をソフトウェア上で実現することはSDN，OpenFlow等で可能であるが，一方で，ネットワークに限らず，汎用的に適用できるようになっている．例えばIoTデバイスからのセンシング情報の処理，動画等のマルチメディア処理，そして大容量のデータ解析を，それぞれ仮想化されたファンクションで処理できる．このようにVNFにとどまらず，より広範囲に使えるようなファンクションをサービスファンクション（本稿ではSFと呼ぶ）と

呼ばれる．そして SF どうしで入出力データの通信を行って処理を順次行うものをサービスファンクションチェイン（SFC）と呼ばれる [3], [4]．SFC を実現するためには SF の計算資源への割り当てが必要になるが，これにはいくつかの課題が存在する．例えば SF 間の通信時間の最小化，使用される計算資源数や SF インスタンス数の最小化，応答時間及び遅延の最小化等が挙げられる [5]〜[18].

上記の SFC は，いずれも IP によって互いの SF の位置を特定し，そして処理結果を送信することを前提としている．ネットワーク構造が動的に変動するような状況，例えば仮想マシン (VM) の追加や削除が頻繁に行われるような場合，追加された計算資源が SF の割り当て対象とはならないか，もしくは SF の割当先にデータが届かないという結果となる．そこで，ネットワーク構造の動的な変動に対応した SF の割り当てを行うことが，今後の多様化する状況において SFC を行うために必要である．また，従来の SFC ではチェイン構造，すなわち直線上の依存関係のみを想定しており，SF 間の処理順は常に同じである．そのため，スケジューリングは考慮する必要がなく，SF の割り当て先のみに焦点が当てられていた．しかしながら，SFC の構造を一般化すると，非循環有向グラフ（DAG: Directed Asyclic Graph）として表現できる．この DAG を模した構造はワークフロー呼ばれており，このワークフロー構造での SFC では実行順が決まっていないためにスケジューリングが必要となる．

今後の多様化するネットワーク上で汎用的な SFC を行うために，本稿では情報指向ネットワーク（ICN: Information-Centric Networking）に基づくワークフロー SFC を実現する手法を述べる．ワークフロー SFC のスケジューリングを集中的に行う場合と各計算資源（ノード）が自律的に行う場合のスケジューリング，そして各場合におけるチェイニング手法を述べる．そして実験により，各手法の特性について述べる．

2. 想定モデル

本稿における処理対象のファンクション集合，及び想定する実行環境について述べる．

2.1 システムモデル

$N = \{n_1, n_2, \dots\}$ を複数のホストマシンの集合とし，各 CPU コアを $M_i = \{m_{i,1}, m_{i,2}, \dots\}$，$n_i \in N$ とする．$m_{i,j}$ の MIPS 値と通信帯域幅をそれぞれ $\alpha_{i,j}$， $\beta_{i,j}$ とする．各 CPU コア，すなわち $m_{k,l} \in n_k$ において，$m_{k,l}$ は 1 以上の vCPU へ割り当てられるものとし，各 vCPU を $c_{k,l,m}$ とする．ここで HT が有効か無効かによって $1 \leq m \leq 2$ であるものとする．そして vCPU の集合を C_{vcpu} とする．また，「ノード」をここでは VM と定義し，一つの IP アドレスが付与された計算資源の単位とする．すると各ノードは 1 以上の vCPU を保持することになる．

2.2 サービスファンクション

ネットワーク上で，1 以上の SFC を処理する状況を想定する．SFC をグラフ $S = (V, E)$ と定義し，ここで V を SF 集合，E を SF 間の先行関係を決める有向辺と定義する．i 番目の SF を $v_i \in V$，v_i から v_j へのデータ転送を $e_{i,j}$ とする．w_i を v_i の仕事量，$e_{i,j}$ のデータサイズを $d_{i,j}$ とする．SFC において，各 SF は，全ての先行 SF からのデータが到着するまで実行開始できないものとする．$pred(v_i)$，$suc(v_i)$ をそれぞれ v_i の先行 SF 集合，後続 SF 集合とする．$pred(v_i) = \emptyset$ であれば，v_i は START SF とし，逆に $suc(v_i) = \emptyset$ であるとき，v_i を END SF と呼ぶ.

2.3 コストモデル

$c_{k,l,m}$ 上で v_i を実行したときの処理時間を

$$T_p(v_i, c_{k,l,m}) = \frac{w_i}{\alpha_{k,l,m}}, \tag{1}$$

とし，また，$m_{k,l}$ から $m_{p,q}$ へサイズ $d_{i,j}$ のデータを送信するのに要する通信時間を

$$T_c(d_{i,j}, L_{k,p}) = \frac{d_{i,j}}{\min\{\beta_k, \beta_p\}} = \frac{d_{i,j}}{L_{k,p}}, \tag{2}$$

とする．ここで，β_k は n_k の通信帯域幅である．vCPUs 間のデータ転送が同一ホスト内であれば，その通信時間はネットワーク上の通信に比べて無視できるものとし，0 と想定する．

3. ワークフローにおける SF スケジューリング

3.1 ワークフロースケジューリングの分類

従来の SFC では，そのほとんどが chain 構造，すなわち直線構造の SF 集合を想定している．そのような場合は各 SF の先行関係は常に決まっているため，各 SF の割り当て先を決定する問題，すなわち SF 配置問題のみを考えればよいことになる．しかしながら，より汎用的な構造である DAG，すなわちワークフロー SFC では，同時実行可能な SF が存在する場合がある．すなわち，各 SF がノード内の vCPU へ割り当てられた後，各 vCPU 内においてスケジューリングを行う必要がある．そのため，スケジューリングの基準によって応答時間が変動する．ワークフロー型ジョブのスケジューリングには，典型的にはマスタ側で環境情報とジョブ情報を参照した上で静的にスケジューリングを行うグローバルスケジューリングと，各 vCPU で個別にスケジューリングを行うローカルスケジューリングに大別される．最も使われているのはグローバルスケジューリングであり，その中で各仕事に対して特定の優先度に従ってスケジューリングを行うリストスケジューリング [20], [21]，そして仕事どうしを集約して通信の局所化を狙ったクラスタリングによるスケジューリング [19] がある．

3.2 スケジューリングの SFC への適用上の課題

一方で，SFC の場合では，予め定められた制約条件を満たすように SF を割り当てる，というものが多い．例えば SFC の処理に必要なネットワークフロー容量の最小化，処理容量の上限以下となるような SF 割り当て，COPEX, CAPEX 制約を満たす，等である．これらの条件を考慮に入れる際，予めネットワーク環境が固定であることを前提としている．しかしながら IoT センサ等，大量の情報送信端末が分散配置される状況では，動的に端末が増減するといったネットワークダイナミクスに対応した SF 割り当て基準が必要となる．その結果，各端末の資源情報を集中管理することは各 vCPU，ひいては各ノードの位置情報を頻繁に更新するという点で現実的とは言えず，各端末

が自律的に動作させ，かつ効率よく情報処理するためのチェイニングが必要となる．その場合，各ノードの位置情報を意識せずに SF を呼び出す仕組みが必要と考えられる．そこで，本稿ではワークフロー型 SFC において，ICN を用いたノードへの SF 割り当て手法，及びチェイニングを述べる．

4. ワークフローにおける ICN-SFC

4.1 ICN による利点

ここでは，ワークフロー型 SFC において ICN によるチェイニングを行う際のチェイニングパターンを述べる．ICN を SFC に適用することにより，例えばファンクション自体や，ファンクションの入出力データのキャッシュによる通信の分散化が可能となる．また，名前に基づいたルーティングにより，各ノードの位置に依らずにファンクション情報のみでアドレッシングが可能となる．一方，ICN を SFC に適用する上で様々なチェイニングパターンが考えられる．一つは，全計算資源の情報を一元管理してスケジューリングする集中管理である．そして他方では，各計算資源が隣接情報のみを保持して自律的にスケジューリングする自律管理である．いずれの手法も，スケジューリング結果に基づいて END SF から START SF の方向へ Interest パケットを送信して PIT エントリを保持させる．そして，START SF から処理して，処理結果を自身の PIT エントリに向けて送信する．次節からは，これら 2 手法について述べる．

4.2 集中管理に基づく ICN-SFC

ICN を用いた SFC のうち，集中管理による SFC について述べる．この場合は，集中管理を行うノード（Delegator），そして計算処理群（ノード群）が必要となる．まず，Delegator では当該 SFC に対してワークフロースケジューリングを行う．そして各ノード群，具体的には vCPU と SF とのマッピング情報，及び実行順に関する情報が出力される．その後，ICN に基づいたチェイニングが行われる．ICN によるチェイニングではまず，END SF の割り当て先 vCPU から START SF の割り当て先 vCPU に向かって Interest パケットを送信する．そして，ある SFv_i において，$suc(v_i)$ が割り当てられた全 vCPU からの Interest パケットを受け取った後，$pred(v_i)$ が割り当てられた各 vCPU に対して Interest パケットを送信する．そして START SF において全ての後続 SF からの Interest パケットが到着すると，今度は START SF から処理を開始する．そして処理結果は PIT の各エントリに相当する face を用いて送信する．図 1 に，ワークフロー型 SFC のチェイニング例を示す．図中，各図形は SF のタイプを示し，それぞれが別ノード内の vCPU に割り当てられており，互いにネットワーク上の通信が発生する状況を考える．すなわち，全ての Interest パケット及び Data パケットはネットワーク上で通信されるものとする．図 1(a) では END SF から START SF に向かって backtracking をする形で Interest パケットが送信される．これによって各 vCPU が属するノードの PIT エントリが埋まり，このエントリを参照することによって処理結果データが送信可能となる．図 1(b) のように，もし一部の後続 SF からの Interest パケットの到着

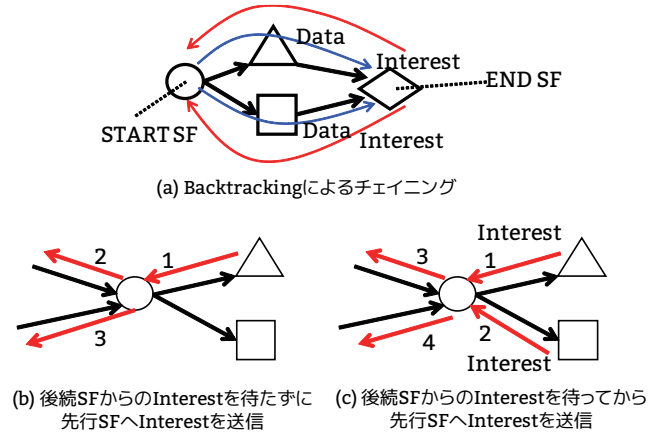


図 1　ICN-SFC の集中管理によるチェイニング

を待たずに先行 SF へ Interest パケットを送信した場合，処理結果の送信先として PIT 内に，Interest パケット未到着の後続 SF に対してデータ送信ができない可能性がある．すなわち，図 1(c) のように全後続 SF の割り当て先 vCPU からの Interest パケット受信を待つことにより，SF 処理後に全後続 SF の割り当て先 vCPU が属するノードへ処理結果を送信可能であることが保証される．このような集中管理に基づく ICN-SFC では，Delegator が予め全ノードの vCPU 情報を保持する必要がある．ネットワーク規模やノード構成が固定である場合はその性能はグローバルスケジューリングの基準に依存する．しかしながら，ノード数や vCPU 数の増減によって Delegator では各ノード内の vCPU 情報の取得タイミングによっては各 SF の割り当て先に，Interest パケット送信先が見つからないといった不整合が生じる可能性がある．

4.3 各ノードの自律管理に基づく ICN-SFC

4.3.1 Interest パケットの同一ノードへの到達保証の問題

この場合では，Delegator といった，集中管理ノードがいない状況において ICN-SFC を行うことを想定する．各ノードが持つ FIB 内の face から，Interest パケットの送信先を決める必要がある．すなわち Interest パケットの宛先は，各ノードが持つ FIB によって決まるため，複数 vCPU の Interest パケット送信先 SF が同一であったとしても，宛先 vCPU，ひいては宛先ノード自体が異なる場合がある．その結果，同一 SF を複数ノード内の vCPU 間で処理することになり，計算資源が枯渇してしまうという問題がある．図 2 に，それぞれ別ノード内の vCPU どうしが自律的に Interest パケットを送信する場合のチェイニングの例を示す．図 2(a) では，n_1 と n_2 がそれぞれ，ファンクション C の割当先としてそれぞれ n_3，n_4 としている．そのため，一つの SF が n_3，n_4 という，別ノード内 vCPU にて実行されることになる．一方，図 2(b) では，n_1 と n_2 がともに n_3 をファンクション C の割り当て先として選択しているため，ファンクション C は n_3 のみで処理される．

以上のことから，各ノードの vCPU で自律的にファンクションの割り当て先を決める場合は，一つの SF の割り当て先が同一となるような仕組みが必要となる．しかしながら，この場合は複数ノードから，アプリケーションレイヤにおける 1 ホッ

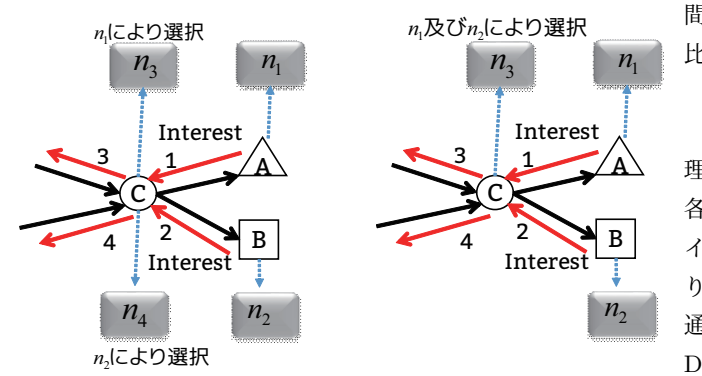


図 2　ICN-SFC の自律的なチェイニング

プで同一ノードに Interest パケットが到達する保証は無い．したがって，複数ホップにて必ず同一 vCPU へ到達するような Interst パケットのルーティングが必要となる．

4.3.2　Interest パケット転送先決定の条件

次に，FIB エントリである face リストから Interest パケットの転送先を選択する際の課題を述べる．各 FIB エントリから選択する場合，その選択基準として考えられるのは各 face に対応するノード（vCPU）の性能情報と，SFC 情報である．このうち vCPU の性能情報はそれが属するノードの FIB エントリに基づいて参照される．そのため，複数の SF どうしが共通の先行 SF を保つ場合，共通の先行 SF の割り当て先を決める際に同一 vCPU に到着する保証はない．したがって，複数の Interest パケットが同一 vCPU に到着するように，1 ホップごとに同一 vCPU に近づくような仕組みが必要である．

4.3.3　同一 vCPU へ到達保証する Interest ルーティング

そこで，ID に基づいて Interest パケットを転送するための方法を述べる．ここで v_p，v_q，v_r において，それぞれ $v_q, v_r \in suc(v_p)$ とする．そして v_q, v_r が割り当てられた vCPU をそれぞれ $c(v_q)$，$c(v_r)$ とし，v_p の割り当て先を決める状況を想定する．ここで $c(v_q)$，$c(v_r)$ は互いに異なるノードに属するものとする．SF 及び vCPU から得られる ID 値をそれぞれ $H(V)$，$H(C_{vcpu})$ とする．$c(v_q)$ が属するノードの FIB に $H(v_x) < H(c(v_q)) < H(v_y)$ となるような v_x, v_y が属しており，$c(v_r)$ が属するノードの FIB に $H(v_{x'}) < H(c(v_q)) < H(v_{y'})$ となるような $v_{x'}$，$v_{y'}$ が属しているものとする．それぞれの vCPU において $H(c(v_q))$ に最も近い値をもつ FIB エントリに対して Interest パケットを送信することにより，1 ホップ毎に $H(c(v_q))$ の距離がより近い vCPU に Interest パケットが到達する．もし最も近い vCPU が FIB エントリではなく自身の vCPU であれば，その vCPU が v_q の割り当て先となる．このような自身の vCPU の ID よりも小さいエントリと大きなエントリを FIB に保持させることにより，1 ホップ毎に SF の ID に近づけることが可能となる．

5.　予備的実験

本実験では，ワークフローエンジンを実環境用に実装した上で，特定の SFC を用いてスケジューリングし，そして応答時間を ICN-SFC における集中管理の場合と自律管理の場合とで比較した．

5.1　実験環境

図 3 に，集中管理による ICN-SFC 実行環境を示す．集中管理の場合は，Delegator 内で SFC の情報（SF 間の依存関係，各 SF の仕事量，実行コマンド，出力データの Path 及びファイルサイズ）を JSON ファイルに記述して配備した．また，割り当て候補となる全計算資源の情報（vCPU の MIPS，VM の通信帯域幅）を JSON ファイルに記述して配備した．各 SF は Docker コンテナを想定しており，SF の割当先にて当該コンテナが無ければ，コンテナリポジトリから取得する．その上で，我々が開発した SF スケジューリングアルゴリズムである SF-CUV [22] を用いてスケジューリングを行った．SF-CUV は SF どうしを積極的に集約して割り当てることによって互いの通信時間を局所化しつつ，割り当て後のスケジューリングによって応答時間を最小化することを目的とした手法である．

一方，自律管理の場合は，図 3 において，Delegator が END SF を処理することを前提とする．そして，FIB エントリ数を 10 とし，SF である v_k の ID 値を以下のように定義した．

$$H(v_k) = \frac{w_k}{\max\left\{\max_{v_j \in pred(v_k)}\{d_{j,k}\}, \max_{v_l \in suc(v_k)}\{d_{k,l}\}\right\}}. \quad (3)$$

そして，vCPU である $c_{p,q,r}$ の ID 値を以下のように定義した．

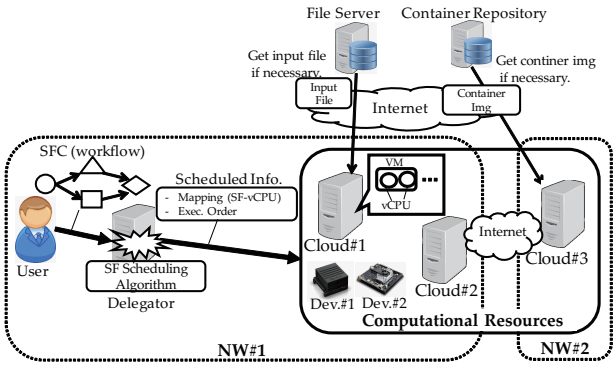
$$H(c_{p,q,r}) = \frac{\alpha_{p,q,r}}{\beta_p}. \quad (4)$$

そして，v_k の割り当て先となる Interest パケットの送信先は，FIB エントリの中から (4) が (3) に最も近いものを選択して，その face へ送信した．一方で，Interest 転送中に当該 SF がすでに保持していればその vCPU が処理する，という方針とした．

次に，実験に用いた SFC について述べる．図 4 に，SFC の構造を示す．図の SFC は画像処理を行うが，それぞれの仕事量は，1000MI は 1.0GHz と単純化した想定の下，事前の計測時間に基づいて割り当てた．また，B は画像分割数であり，この値が大きいほどより多くの計算資源を要することになる．ここで同一の図形は同一コマンドを意味し，SF 間のデータサイズは事前の計測時間に基づいて割り当てた．

5.2　実験結果

表 1 に，画像分割数 $|B|$ を変動させた場合の応答時間の比較結果を示す．この実験では，事前に Docker コンテナを未配備な状況での応答時間と，Docker コンテナを配備済みである状況での応答時間を比較した．また，表中，集中管理による応答時間を 1.000 とした場合の割合値を自律管理の応答時間とした．この結果では，Dokcer 配備を行わない状況では集中管理による ICN-SFC の応答時間が総じて良いという結果となった．また，画像分割数の大小による特徴は見られなかった．一方，Docker 配備済みである場合では，いずれの画像分割数において自律管理による応答時間の方が良い，という結果となった．これは，Interest パケット転送時に，すでに Docker コンテナが配備済みであるかどうかの検査を行っているためであると考えられる．一方，集中管理では，スケジューリングアルゴリズム SF-CUV



Hardware	Parameter	Value
NW#1 Router#1	Bottleneck BW to external hosts	100Mbps.
	Internal BW	1Gbps
Cloud#1	# of VMs	15VMs
	# of vCPUs	2vCPUs x 4VMs 4vCPUs x 11VMs
	vCPU Freq.	{2.0, 2.5, 3.0} GHz.
	BW to Router #1	1Gbps
Cloud#2	# of VMs	6VMs
	# of vCPUs	2vCPUs x 4VMs 4vCPUs x 2VMs
	vCPU Freq.	{1.0,2.0, 2.5} GHz.
	BW to Router #1	1Gbps
Dev.#1	Type	NVIDIA Jetson AGX Xavier (2.26 GHz x 8Cores, 16GB RAM)
Dev.#2	Type	NVIDIA Jetson TX2 (2.0 GHz x 6Cores, 8GB RAM)

Hardware	Parameter	Value
NW#2 Router#2	Bottleneck BW to external hosts	100Mbps.
	Internal BW	1Gbps
Cloud#3	# of VMs	8VMs
	# of vCPUs	4vCPUs x 4VMs 4vCPUs x 2VMs
	vCPU Freq.	{1.0, 3.0, 3.5} GHz.
	BW to Router #2	1Gbps

図 3　SFC 実行環境.

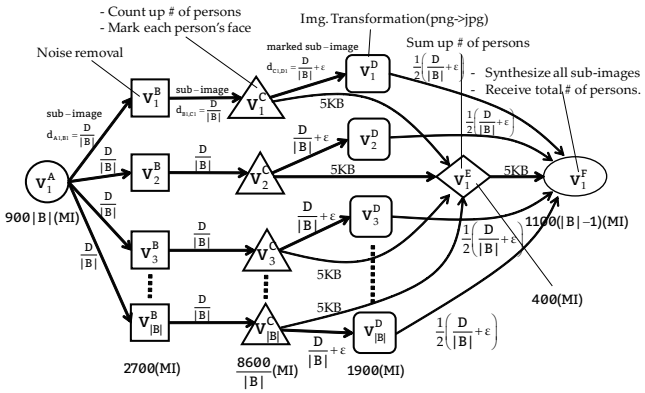


図 4　実験に用いた SFC の構造

表 1　実 験 結 果

	Docker 配備なし		Docker 配備あり	
$\|B\|$	集中管理	自律管理	集中管理	自律管理
2	1.000	1.237	1.000	0.889
5	1.000	1.321	1.000	0.913
10	1.000	1.228	1.000	0.773
20	1.000	1.194	1.000	0.681
30	1.000	1.293	1.000	0.662

による SF 割り当てを行っている．これはすでに Docker コンテナ配備済みかどうかの判断は行わない．よって，配備済みコンテナをいかに再利用するかが，応答時間の最適化する上で重要になるものと考えられる．

6.　まとめと今後の課題

本稿では，ICN 基づくワークフロー SFC を実現する手法，すなわち集中管理に基づく手法と各ノードが自律的に管理する手法を述べた．いずれも ICN の Interest パケットを backtrcking によって送信し，戻り経路としてファンクション実行結果を PIT エントリに向けて送信する，というものである．実験の結果，ファンクションであるコンテナの配備済みかどうかの判断が，応答時間を最小化する上で重要であることが分かった．

今後の課題としては，より実用的な状況における実験，並びに自律管理による ICN-SFC において，Interest パケットの効率の良いルーティング手法の検討が挙げられる．

謝辞　本研究成果は，戦略的情報通信研究開発推進事業 (国際標準獲得型)「スマートシティアプリケーションに拡張性と相互運用性をもたらす仮想 IoT-クラウド連携基盤の研究開発 (Fed4IoT)」の支援を受けている．また，科学技術研究費（基盤研究（C））:「クラウド間連携と仮想化ファンクション集約による計算資源の有効利用に関する研究」による一部支援を受けている．

文　　献

[1] R. Mijumbi, J. Serrat, J-L. Gorricho et al., “Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Commun. Surv. & Tutorials*, vol. 18, no. 1, pp. 236–262, Sep., 2016.

[2] A. Belbekkouche, M.M. Hasan, and A. Karmouch, “Resource Discovery and Allocation in Network Virtualization," *IEEE Commun. Surv. & Tutorials*, vol. 14, no. 4, pp. 1114–1128, Feb., 2012.

[3] D. Bhamare, R. Jain, M. Samaka, et al., “A survey on service function chaining," *J. Netw. Comput. Appli.*, Vol. 75, pp. 138–155, Sep., 2016.

[4] P. Quinn and J. Guichard, “Service Function Chaining: Creating a Service Plane via Network Service Headers," *Computer*, vol. 47, no. 11, pp. 38–44, Nov. 2014.

[5] L. Wang, Z. Lu, X. Wen, et al., “Joint Optimization of Service Function Chaining and Resource Allocation in Network Function Virtualization," *IEEE Access*, vol. 4, pp. 8084–8094, Nov., 2016.

[6] M. C. Luizelli, W. L. C. Cordeiro, L. S. Buriol, et al., “A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining," *Comput. Commun.*, vol. 102, 67–77, Nov., 2016.

[7] T-W. Kuo, B-H. Liou, K. C-J Lin, et al., “Deploying Chains of Virtual Network Functions: On the Relation Between Link and Server Usage," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1562–1576, Aug., 2018.

[8] M. Ghaznavi, N. Shahriar, S. Kamali, et al., “Distributed Service Function Chaining," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11. pp. 2479–2489, Nov., 2017.

[9] D. Bhamare, M. Samaka, A. Erbad, et al., “Multi-Objective Scheduling of Micro-Services for Optimal Service Function Chains," *in Proc. IEEE ICC 2017 SAC Symp. Cloud Commun. Netw. Track*, pp. 1–6, May, 2017.

[10] M. T. Beck and J. F. Botero, “Scalable and coordinated allocation of service function chains," *Comput. Commun.*, vol. 102, pp. 72–88, Oct. 2016.

[11] Y. Sang, B. Ji, G. R. Gupta, et al., “Provably Efficient Algorithms for Joint Placement and Allocation of Virtual Network Functions," *in Proc. IEEE Int. Conf. Comput. Commun. (IEEE INFOCOM 2017)*, pp. 1–9, May, 2017.

[12] S. Sahhaf, W. Tavernier, M. Rost, et al., “Network service chaining with optimized network function embedding supporting service decompositions," *Comput. Netw.*, vol. 93, pp. 492–505, Oct., 2015.

[13] H. A. Alameddine, S. Sebbah, and C. Assi, “On the Interplay Between Network Function Mapping and Scheduling in VNF-Based Networks: A Column Generation Approach," *IEEE Trans. Netw. Serv. Managem.*, vol. 14, no. 4, pp. 860–874, Dec., 2017.

[14] Z. Li and Y. Yang, “Placement of Virtual Network Functions in Hybrid Data Center Networks," *in Proc. IEEE Int. Symp. High-Performance Interconnects*, pp. 73–79, Aug., 2017.

[15] S. Khebbache, M. Hadji, and D. Zeghlache, “Virtualized network functions chaining and routing algorithms," *Comput. Netw.*, vol. 114, pp. 95–110, Jan., 2017.

[16] V. Eramo, E. Miucci, M. Ammar et al., “An Approach for Service Function Chain Routing and Virtual Function Network Instance Migration in Network Function Virtualization Architectures," *IEEE/ACM Trans. Netw.*, Vol. 25, No. 4, pp. 2008–2025, Aug., 2017.

[17] O. Soualah, M. Mechtri, C. Ghribi et al., “An Effcient Algorithm for Virtual Network Function Placement and Chaining," *in Proc. IEEE Annu. Consumer Communications & Networking Conference (CCNC)*, pp. 647–652, Jan., 2017.

[18] X. Lin, D. Guo, Y. Shen et al., “DAG-SFC: Minimize the Embedding Cost of SFC with Parallel VNFs," *in Proc. Int. Conf. Parallel Processing (ICPP 2018)*, 10 pages, Aug. 2018.

[19] H. Kanemitsu, M. Hanada, and H. Nakazato, “Clustering-based Task Scheduling in a Large Number of Heterogeneous Processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 11, pp. 3144–3157, Nov., 2016.

[20] H. Topcuoglu, S. Hariri, and M. Y. Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, 2002.

[21] H. Arabnejad and J. G. Barbosa, “List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table," *IEEE Trans. Parallel Distrib. Syst.*, Vol. 25, No. 3, pp. 682–694, 2014.

[22] H. Kanemitsu, K. Kanai, J. Katto, and H. Nakazato, “Function Clustering Algorithm for Resource Utilization in Service Function Chaining," *in Proc. IEEE Int. Conf. Cloud Comput. 2019 (IEEE CLOUD2019)*, pp. 193-195, July, 2019.