Vol.2021-DPS-186 No.21
Vol.2021-CSEC-92 No.21
2021/3/15

ホワイトボックススイッチを利用したネットワークサービスプロバイダ視点での機能カスタマイズを実現するVNF基盤

田部 悠介[1,a)]　近藤 賢郎[2,b)]　寺岡 文男[3,c)]　金子 晋丈[3,d)]

概要：ネットワークプロバイダが運用するネットワークのカスタマイズを実現するために，プログラマブル VNF(Virtualized Network Function) をスイッチに実装する需要がある．本稿は，ホワイトボックススイッチを利用したネットワークプロバイダがカスタマイズ可能となる VNF 基盤を提案する．本基盤では，スナップパケットやフロー情報を VNF に処理させることで，DataPath をハードウェアで完結させることができ，ネットワークスループットの低下を回避する．また，本システムはネットワークプロバイダのユーザビリティに配慮したシステムを設計する．スイッチへの VNF 展開するためのライブラリにより，ネットワークプロバイダは最小限のプログラミング記述で第三者 VNF を自由にチェイニングが可能となる．本稿は，上記を満たすシステムを汎用サーバで実装・評価を行い，システムを妥当性を検証する．

VNF platform based on white box switches that enables functional customization from the perspective of network service providers

1. はじめに

ネットワークプロバイダは運用するネットワークをカスタマイズしたいという需要がある．カスタマイズしたい機能 (VNF) は統計情報の取得や解析，ロードバランシングや NAT などのネットワーク制御，DDoS 検知や URL フィッシング検知などのセキュリティ等が挙げられる．ネットワークプロバイダごとに独自の機能が実装される可能性があるため，VNF はプログラマブルである必要がある．つまり，ベンダ NOS(Network OS) 一体型のスイッチ上に，任意の VNF を展開することは困難である．

従来では，VNF をスイッチではなく汎用サーバ上にソフトウェアとして展開する．ソフトウェアで実装されるため VNF はプログラマブルであり，また，移植性が高いという利点がある．しかし，パケットをソフトウェア処理するため，DataPath が CPU・PCIe バスに及び，オーバヘッドが大きいという問題が生じる．これを解決するために，VNF をハードウェアオフローディングする手法も存在する．プログラマブルハードウェアで VNF を記述することで，VNF をハードウェア上に展開することが可能となる．DataPath はハードウェア上で完結するため，オーバヘッドを抑制することができる．しかし，ハードウェアでは複雑な L7 ベースの VNF(L7 VNF) を実装できないという問題がある．また，スイッチとコントローラを分離した SDN 手法による機能分離によるオーバヘッド回避手法も存在する．ネットワーク情報をスイッチからコントローラへ転送し，コントローラがそのデータを元にスイッチへフォワーディングルール (FWD ルール) を変更する命令を送信するものである．DataPath がスイッチ上で完結するためオーバヘッドを回避できるが，スイッチ・コントロー

[1] 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University, Hiyoshi, Kanagawa 223–8522, Japan
[2] 慶應義塾情報セキュリティインシデント対応チーム
Faculty of Science and Technology, Keio University, Hiyosi, Kanagawa 223–8522, Japan
[3] 慶應義塾大学理工学部
Faculty of Science and Technology, Keio University, Hiyoshi, Kanagawa 223–8522, Japan
[a)] al@inl.ics.keio.ac.jp
[b)] latte@itc.keio.ac.jp
[c)] tera@keio.jp
[d)] kaneko@dmc.keio.ac.jp

ラ間のネットワーク情報の転送コストが問題となる．

本稿では，ネットワークのオーバヘッドの抑制と不必要な通信コストの回避を念頭に置いた，プログラマブル VNF をスイッチ上に展開する基盤を提案する．本基盤はホワイトボックススイッチ上で動作するが，今回は汎用サーバで本基盤を実現するシステムを実装・評価を行い，システムの妥当性を検証する．

2. 関連研究

2.1 ハードウェアオフローディング

AccelNet[1]，ClickNP[2]，FlowBlaze[3] は，FPGA や NetFPGA に VNF を記述する方法を提供している．ハードウェア記述の抽象化を提供することで，ラインレートでの VNF 実行を維持しつつ，VNF 作成者が簡単に VNF を実装することを実現している．しかし，これらの手法はハードウェア処理の制約上， L7 のパケット処理を実装することはできない．

P4[4] は，パケット処理記述を抽象化を提供するプログラミング言語である．P4 で書かれた処理は，処理内容によって適切なハードウェアにマッピングされる．そのため，P4 プログラマはハードウェア記述を P4 という単一のプログラミング言語で記述でき，ハードウェア記述自体の負担と，ハードウェアごとの記述法の差異による負担を回避することができる．しかし，L7 処理は CPU にマッピングされてしまうので，DataPath が CPU を通ってしまいオーバヘッドが生じる可能性がある．

2.2 VNF フレームワーク

CLICK[5]，FLICK[6] は，VNF を記述するフレームワークである．CPU 処理の最適化やカーネル処理などの実装の難易度が高い処理をフレームワーク側が負担することで，VNF プロバイダは VNF の本質的なロジックのみの実装で VNF を作成できるようになる．このフレームワークで実装される VNF は，汎用 Linux マシンの DataPath 上で実行される．

LibVNF[7], [8] は，VNF チェインを考慮した VNF フレームワークである．こちらも，CLICK や FLICK と同様に，フレームワークで実装される VNF は，汎用 Linux マシンの DataPath 上で実行される．また，チェイニングは VNF プロバイダが決定するものとなっているが，VNF を使用する人がチェイニングを決定する仕組みになっていない．

2.3 フロー情報の近似

Elastic Sketch[9] や FCM-Sketch[10] は，フロー情報を圧縮する技術である．この技術は SDN において，フォワーディングルールを決定するコントローラへフロー情報を提供するために使用される．コントローラへの通信コストを削減しつつ，フロー情報の精度を高めるための技術となっている．しかし，誤差を許容するため，攻撃者のわずかな振る舞いを観測するようなセキュリティ運用に使用できない．

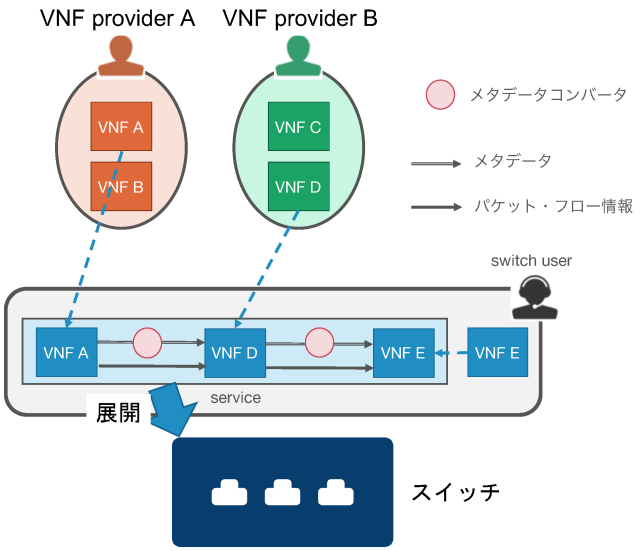

図 1 提案アーキテクチャの想定環境
Fig. 1 Assumed environment of the proposed architecture.

3. 設計

3.1 用語定義

提案アーキテクチャを利用する人物とその役割を**図 1** に示す．本稿が提案する手法を使用する人物として VNF プロバイダとスイッチユーザが挙げられる．VNF プロバイダは VNF 作成者のことを指す．スイッチユーザは提案システムを使用する人物である．提案システムで展開されるネットワーク機能をサービスと呼び，スイッチユーザは VNF を自由に組み合わせた (チェイニングした) ものをサービスとして自身が運用するスイッチに展開する．

3.2 要件

L7 VNF の展開とオーバヘッド回避を両立することが課題となる．オーバヘッドを抑制できるハードウェアオフローディングでは複雑な処理をする L7 VNF を記述できない．それに対して，L7 VNF をソフトウェアで実装したソフトウェアスイッチングでは，DataPath が CPU・PCIe バスに及ぶためオーバヘッドが大きい．そのため，オーバヘッドを回避しつつ，L7 VNF が実行できる仕組みが必要となる．

また，スイッチユーザによるカスタマイズ性の向上を要件に含める．VNF を展開するのはスイッチユーザなためである．，VNF 展開におけるスイッチユーザの自由度をあげるべきだ．そのため，VNF 作成において，スイッチユー

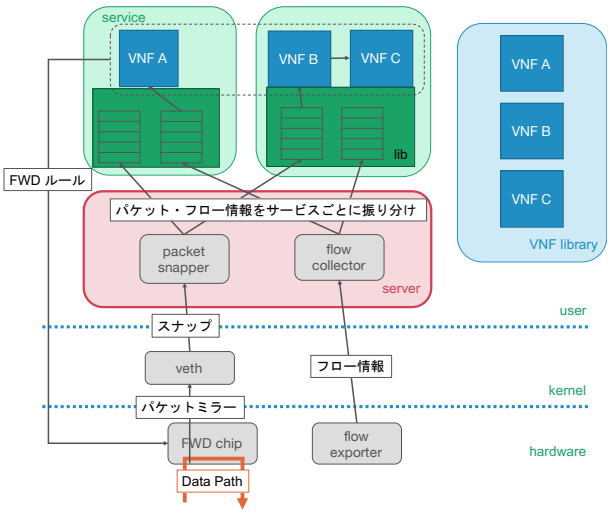

図 2 提案アーキテクチャ
Fig. 2 Proposed architecture

ザが任意の VNF をチェイニングできるような仕組みが必要である．また，第三者が作成した VNF を呼び出し，パケット・フロー情報を入力するといった記述は，スイッチユーザにとって負担となる．そのため，スイッチユーザがサービス定義に必要な VNF の選択・チェイニングの指定のみの記述でサービスを実装できるような仕組みが必要である．

3.3 提案アーキテクチャ

本稿はホワイトボックススイッチ上で動作する VNF 基盤を提案する．本稿が提案するアーキテクチャを**図 2** に示す．ホワイトボックススイッチにより，Linux が動作する汎用マシンの側面と専用チップでパケット処理するスイッチの側面の 2 つの側面を 1 台の筐体で担うことができる．本アーキテクチャで展開される VNF は汎用マシンの部分で実行する．つまり，VNF は Linux 上で展開されるためソフトウェア実装が可能となり，L7 VNF の展開が可能となる．ここで，DataPath がソフトウェア実装された VNF に及ぶとオーバヘッドが問題となる．本アーキテクチャでは，DataPath をフォワーディングチップ (FWD chip) で完結させ，FWD chip が処理したパケットのスナップやフロー情報を VNF に渡す仕組みにする．DataPath が VNF に及ばないため，ネットワークに流れるパケットにおけるネットワークスタック処理を回避しオーバヘッドを抑制する．また，VNF に入力するパケットを先頭 150Byte のスナップパケットないしは軽量なフロー情報とすることでメモリコピー処理を削減する．これらの仕組みは 1 台のホワイトボックススイッチで行われるため，スナップパケット・フロー情報の転送コストを回避する．

3.4 VNF サーバ

VNF が処理するパケット・フロー情報を取得する実装はスイッチユーザ・VNF プロバイダにとって負担となる．入力されたデータからネットワークロジックを決定するという本質的なロジックと関係ないロジックであるからだ．また，それぞれの VNF あるいはサービスを実装するスイッチユーザ・VNF プロバイダが各々パケット・フロー情報を取得するロジックを実装することは重複してしまう．

そこで，本アーキテクチャでは VNF サーバを導入する．VNF サーバは FWD chip から転送されたスナップパケット・フロー情報を取得し，展開されているサービスへ振り分ける．これにより，スナップパケット・フロー情報の取得の部分のロジックを VNF サーバが負担するため，スイッチユーザ・VNF プロバイダは VNF における本質的なロジックのみを記述すればよくなる．packet snapper では，各パケットから指定した長さ分だけを取得する．本稿が想定する VNF はアプリケーションヘッダを処理するため，先頭から 150Byte が必要である．そのため，packet snapper に 150Byte を指定する．MTU を 1500Byte とすると VNF が処理するパケットは 150Byte となり，メモリコピーを 10 分の 1 にすることができる．flow collector では，flow exporter から転送されたフロー情報を取得する．フローのバイト数やカウントなどの軽量なトラフィック情報を VNF で処理可能となる．

3.5 メタデータコンバータ

VNF 間の通信媒体をメタデータと呼び，メタデータを用いて VNF チェイニングの利便性を向上する．第三者のロジックが組み込まれた VNF の処理結果を自身の作成した VNF に入力するといったことが可能となる．しかし，VNF チェイニングの決定は，VNF プロバイダではなくスイッチユーザが行うため，メタデータのフォーマット差異の問題が発生する．VNF 作成時には，チェイニングする前後の VNF を決定できず，チェイニングを考慮したメタデータフォーマットを定義できないためである．

そこで，メタデータコンバータを導入する．これは，ある VNF の出力したメタデータを別の VNF が利用できるようなフォーマットに変換する仕組みである．これにより，VNF ごとによるメタデータフォーマットの差異をメタデータコンバータが吸収することができるため，チェイニングできる VNF の種類を増加する．チェイニング決定はスイッチユーザが行うため，VNF 間のメタデータコンバータの挙動はスイッチユーザが定義する．以上により，スイッチユーザによる自由な VNF チェイニングが可能となる．

3.6 サービスライブラリ

スイッチユーザが VNF チェイニングとそれに伴うメタ

データコンバータの挙動定義を指定したものがサービスとなる．サービスを実装するスイッチユーザのユーザビリティが課題となる．VNF へスナップパケット・フロー情報入力，メタデータ入出力の管理などのロジックを一から実装するのはスイッチユーザにとって負担であるだ．

本稿では，スイッチユーザのサービス実装負担を最小限にするサービスライブラリを提供する．サービスで定義されることは，VNF チェイニングとメタデータコンバータの挙動である．つまり，これらを定義する API のみをスイッチユーザが記述するような仕組みにすることで，必要最低限の記述でサービスを実装することが可能となる．

4. 実装

本節では，3 節で述べた設計を元にしたシステム実装について説明する．後述するように，ミラーリングの設定が上手くいかなかったが，本節ではホワイトボックススイッチ上で動くと仮定して実装の説明をする．

4.1 実装環境

本システムは SONiC[11] という NOS の上で動作する．SONiC は Linux ベースの OS であるため，ソフトウェア実装の VNF が容易に実行することが可能である．また，多くのホワイトボックススイッチ機器が SONiC に対応している．SONiC 前提としたシステムにより，本システムに対応可能なプラットフォームを多くし移植性を高めている．

packet snapper でパケットをスナップするために，FWD chip は SONiC 上の仮想インターフェース (veth) にミラーリングする．全ての物理インターフェースに入ってくる (rx) パケットを veth にミラーリングする．しかし，本稿はミラーリングの設定が上手くいかなかったため，汎用マシンでの評価をする．

flow exporter は SONiC は sFlow[12] が対応しているため，sFlow を使用する．本システムでは，sFlow は全ての物理インターフェスをキャプチャし，SONiC 上で稼働する VNF サーバの flow collector にフロー情報を転送するように設定をする．

4.2 VNF サーバ

VNF サーバは，スナップパケット・フロー情報を取得し，起動中のサービスに展開する役割を果たす．本システムでは，VNF サーバはデーモンプロセスとして SONiC 上で実行される．本稿では，VNF サーバを C++で実装した．

4.2.1 packet snapper

packet snapper は veth からスイッチに来たパケットをスナップする役割を果たす．スナップとは先頭から指定した長さ分だけのパケットをキャプチャすることである．本稿では，LiBPCAP[13] ライブラリを用いて VNF サーバの中にパケットをスナップする処理を実装した．スナップする長さを 150Byte としている．これは，L7 VNF の大部分はアプリケーションヘッダのみを必要とし，150Byte 程度あれば十分なためである．

スナップしたパケットをサービスと共有するために，共有メモリを使用する．共有メモリを利用することで，VNF サーバ・サービス間のスナップパケットの転送を高速にする．また，スナップパケットは最大 150Byte であるため，固定長のメモリを用意しやすいのも共有メモリを採用した理由である．各サービスプロセスが共有メモリを用意し，VNF サーバが共有メモリにスナップパケットをコピーする．共有メモリ上をリングバッファとして扱う仕組みを用意して，VNF サーバがバッファへ push し，サービスがバッファから pop する．

4.2.2 flow collector

flow collector は flow exporter から転送されたフロー情報を取得するが，capture と dump の 2 つの部分からなる．capture・dump 両方とも，NFDUMP[14] が提供するツールを使用する．

capture では，NFDUMP が提供する sfcapd というツールを使用する．flow expoter から転送されるフロー情報はバイナリ形式であるが，sfcapd はバイナリデータをそのままファイルに保存するツールである．sfcapd は終了シグナル (SIGINT など) を送るまで実行し続けるプロセスであるため，VNF サーバとは別にデーモンプロセスとして起動する．あらかじめ指定したディレクトリに 2 秒ごとにファイルに書き出すように設定する．

dump では，NFDUMP が提供する nfdump というツールを使用する．nfdump はフロー情報のバイナリファイルから人間が理解できる文字列に変換する．sfcapd が保存したファイルを nfdump の入力として，出力結果をサービスに与える仕組みにする．ここで，sfcapd が保存するファイルは複数であり，ファイル名も一定ではないため，nfdump が入力とするファイルを監視する必要がある．そのため，VNF サーバは sfcapd が保存するディレクトリを監視するように実装する．新しいファイルがあれば，そのファイルを引数として nfdump を実行するような処理を VNF サーバに組み込んでいる．

VNF サーバ・サービス間のフロー情報の共有は FIFO を使用する．スナップパケットとは異なり，nfdump によって出力されるフロー情報は文字列であり可変長であるためである．可変長のデータを共有メモリで管理することが困難であるため，共有媒体を FIFO にしている．

4.3 サービスライブラリ

サービスライブラリはサービスを実装するためのライブラリである．実装の要求事項は，サービスを記述するスイッチユーザの負担を最小限にすることである．本節では，これを満たすためのサービスライブラリの実装手法と，

```
class pkt_vnf {
public:
  /* パケット処理ロジック */
  virtual void
    handle_packet
    (const snap_packet *, const void *in, void *out) = 0;

  /* チェインする時に,コンバータ処理に必要 */
  virtual size_t in_pos(int) = 0;
  virtual size_t in_size(int) = 0;
  virtual size_t out_pos(int) = 0;
  virtual size_t out_size(int) = 0;
}
```

図 3 スナップパケット処理 VNF の抽象クラス
Fig. 3 Snap packet processing VNF abstract class.

サービスライブラリの使用方法について説明する．

4.3.1 サービスライブラリ実装

3.6 節で述べたように，サービスのロジックを記述することはスイッチユーザにとって負担となる．スイッチユーザがやるべきことは，サービスとして組み込む VNF チェイニングとメタデータコンバータの挙動を決定すること，つまり，サービスの仕様定義のみである．このため，サービスの使用定義のみをスイッチユーザに記述されるライブラリが必要である．

これを実現するために，サービスライブラリで実装すべきロジックは以下の通りとなる．

- 共有メモリからスナップパケットを取得．
- FIFO ファイルからフロー情報を取得．
- VNF にスナップパケット・フロー情報を入力して呼び出し．
- メタデータを保存するメモリの確保．

展開する VNF はサービスライブラリ内で実行されるが，VNF の選択はスイッチユーザが行うため問題が生じる．サービスライブラリは具体的な VNF を想定した記述ができないことである．

この問題を解決するために，プログラミングのインターフェスという仕組みを利用している．本稿はインターフェースとして，**図 3** のような抽象クラスを定義する．この抽象クラスはサービスライブラリ実装で使用される関数シグネチャ (関数名，引数の型) を定義している．VNF プロバイダはこの抽象クラスをオーバライドしたクラスを VNF として提供する．オーバライドする際に，図 3 で示した 5 つの関数の実装し，サービスライブラリはこの実装された関数が実行される．これにより，サービスライブラリ実装時は，抽象クラス型変数とその変数の関数を利用して記述が可能となる．そして，サービスライブラリ実行時は，その抽象クラス型変数にスイッチユーザが指定した具体 VNF クラスが代入される．このようにして，サービスライブラリ制作におけるロジック実装とサービスライブラリ使用におけるロジック指定の分離を実現する．ロジック実装はスイッチユーザにとって負担となる実装で，ロジック指定とはスイッチユーザが展開する VNF の選択をすることである．

4.3.2 サービスライブラリ API

表 1 はサービスライブラリで定義された API である．

service_init は最初にスイッチユーザが呼び出す API である．string 型の引数にはサービス名を渡す．サービス名はスイッチユーザが自由に定義できるが，同時に同じサービス名を持つサービスの起動はできない．

register_pkt_vnf と *register_flow_vnf* はサービス上で実行したい VNF を指定するための API である．もし，チェイニングを実現したい場合は，この API を複数呼び出すことで実現する．実行される VNF の順番は，API に引数として *pkt_vnf* または *flow_vnf* の型の引数に VNF クラスを渡した順番と一致する．ここで，size_t [][3] はコンバータの定義である．呼び出される API の 1 つ前の VNF の出力結果と現在呼び出されている API で指定された VNF のメタデータのやりとりを指定する．この引数は 2 次元配列となっており，3 つの size_t の型を持つ配列の配列を API に渡す．3 つの型を持つ配列は順に，現在の VNF の入力構造体メンバのインデックス，1 つ前の出力構造体メンバのインデックス，データ変更の関数アドレス，を指定する．この配列を変数毎に定義することで，前方の VNF の出力構造体のうちどのメンバが，後方の VNF の入力構造体のメンバにコピーするかを指定することができる．データ変更の関数アドレスは void(*)(void *) という型の関数を size_t にキャストして渡す．この関数は，前方のメタデータを必要に応じて，加工するためにスイッチユーザが定義するものである．これにより，コピーするメンバごとにこの長さ 3 の配列を定義し，それらを配列として API に渡す．また，外側の配列は可変長となっているので，最後の要素は 0, 0, 0 という配列を指定する必要がある．

service_run はサービスロジックが起動する API である．この API 内で，無限ループ処理が走り，パケットやフロー情報の取得，VNF の実行が行われる．

スイッチユーザはこれら API を使用することで，サービスを最小限の記述量で実装することができる．

5. 評価

5.1 スナップパケット処理 VNF のスループット

本節では VNF がパケットを処理する速度について検証した．汎用マシン上に提案システムをインストールし，提案システムを検証した．汎用マシンの環境を表 2 に示す．

本システムでは，DataPath 上のパケットのスナップパケットをサービスが所有するキューに保存し，サービスがそのキューからスナップパケットを取り出して VNF にわたす仕組みとなっている．つまり，ネットワークスループットと VNF のスループットは独立している．そのため，

表 1　サービス API
Table 1　service API.

API	引数の型	内容
service_init	string	サービスの初期化・サービス名の定義
register_pkt_vnf	pkt_vnf *, size_t [][3]	パケット処理用 VNF の登録
register_flow_vnf	flow_vnf *, size_t [][3]	フロー情報処理用 VNF の登録
service_run		サービスの実行

表 2　評価環境
Table 2　evaluation environment.

項目	内容
OS	Ubuntu 18.04.5 LTS (Bionic Beaver)
CPU	Intel(R) Xeon(R) Bronze 3106 CPU @ 1.70GHz, 8 core
Memory	8GB
NIC	Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection

VNF スループットがネットワークスループットと異なる場合，VNF でパケットが処理されていないこととなる．本節では，ラインレートでの速度を計測するために，汎用マシン 2 台を用いて計測した．

計測には Iperf3[15] コマンドを使用し，受信側の汎用マシンに提案システムをインストールして検証した．マシン間のリンクは 10Gpbs であるのに対して，Iperf3 の実測値は 9.85Gbps であり，10 秒間パケットを流した．この時のパケットサイズの MTU は 9000 Byte であった．

チェイニングを 1 とした時に同時に実行したサービス数に対する VNF スループットの実データ量，MUT 換算量の結果をそれぞれ，図 4，5 に示す．この評価では，全てのサービスのチェイニング VNF は 1 である．ここで，VNF で処理するパケットはスナップパケットであることに注意されたい．MTU が 9000 Byte である時，スナップする長さは 150 Byte であるため，VNF が実際に処理するデータ量を 150 Byte とすると，MTU 換算における処理したパケットのデータ量は 9000 Byte となる．実際にスイッチに流れるパケットとのデータ量を比較するため，この節におけるスループットは MTU 換算における値となる．図 5 に示した図の縦軸は，ラインレートで 10 秒間パケットを流した時に実際に処理できたパケットサイズの合計である．ネットワーク上に流れたパケットは 11.5GByte であるため，サービス数が 5 つになると処理できないパケットが発生することがわかった．サービスごとにキューを用意しそれぞれにスナップパケットをコピーするという実装であるため，サービス実行数が多くなるほどキューへのコピーコストが大きくなってしまう．そのコストが処理速度低下を招いたと考えられる．また，図 4 から，サービス数が少ない場合でも，サービス間の処理データ量が微妙に違うことから，全てのパケットを処理しきれているわけではないということがわかった．

サービス数を 1 として，VNF スループットの実データ

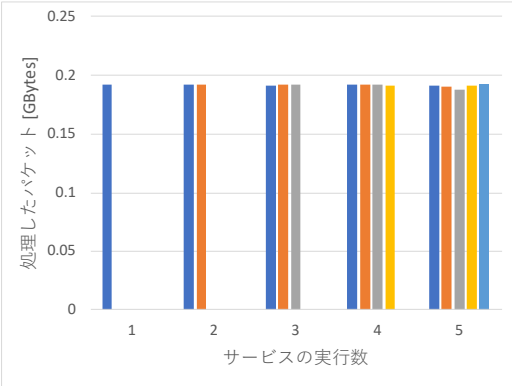

図 4　サービス数に応じた VNF スループット (実データ量)
Fig. 4　VNF throughput (actual data volume) according to the number of services

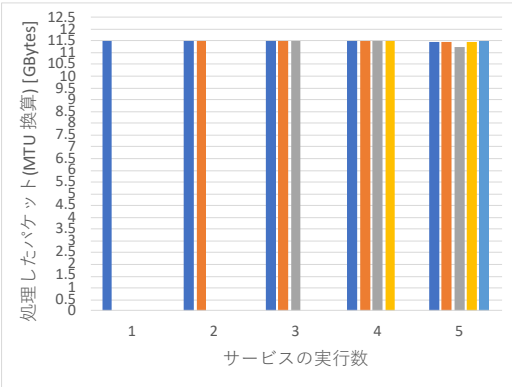

図 5　サービス数に応じた VNF スループット (MTU 換算データ量)
Fig. 5　VNF throughput according to the number of services (MTU conversion data amount)

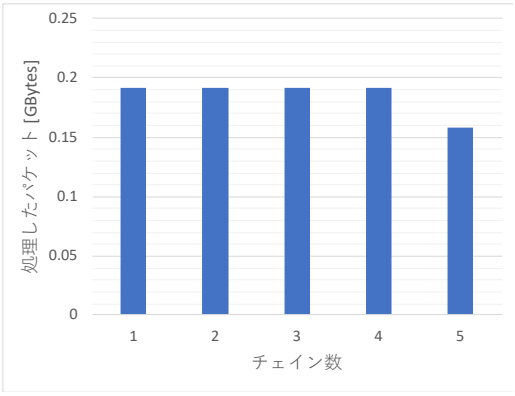

図 6 チェイニング数に応じた VNF スループット (実データ量)
Fig. 6 VNF throughput (actual data volume) according to the number of chains

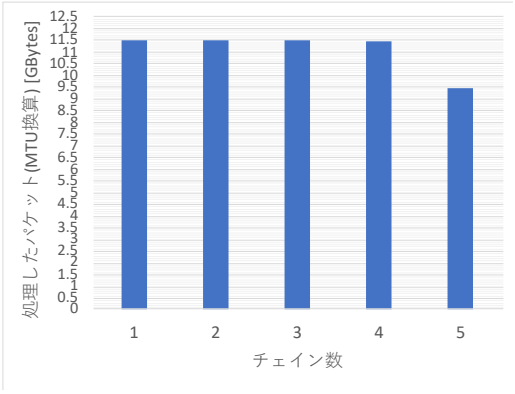

図 7 チェイニング数に応じた VNF スループット (MTU 換算データ量)
Fig. 7 VNF throughput according to the number of chains (MTU conversion data amount)

量，MUT 換算量のそれぞれチェイニング数との関係を表したものを，図 4，5 に示す．

11.5 GByte ほぼ全てを処理できたのはチェイニング数が 3 のサービスまでとなった．また，チェイニング数 5 では大幅に処理速度低下がみられ，流れるパケット速度が大きくなるほど，チェイニングによるコストの影響が大きくなるのではないかと考えられる．チェイニング間ではメタデータコンバータによる処理が発生するが，メタデータコンバータによるメタデータのコピーによるオーバヘッドが大きい点が問題だと考えられる．

5.2 スイッチユーザのユーザビリティ

本節では，サービス実装の容易さを検証することでスイッチユーザのユーザビリティを検討した．表 3 は，サービスライブラリを使った時と，使わなかった時に実装に必要なコード行数 (LoC: Line of Code) を示している．チェインする時は，1 変数を編集すると仮定した場合のコード行数である．

パケット処理 VNF，フロー処理 VNF 両方の場合において，80 % を超えるコードの削減を実現した．このことから，サービスライブラリの提供によりスイッチユーザのサービス実装を容易にすることが明らかとなり，スイッチユーザのユーザビリティ向上を実現した．

6. おわりに

本稿は，ホワイトボックススイッチを利用したカスタマイズ可能な VNF 基盤を提案した．VNF の実行状態であるサービスを，実装するためのライブラリを提供しスイッチユーザのユーザビリティを向上した．しかし，ホワイトボックススイッチの実装とスナップパケット処理のスループットに課題が残る結果となった．

参考文献

[1] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking: Smartnics in the public cloud. *In Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, 2018.

[2] Bojie Li, Kun Tan, Layong Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. *In Proceedings of the 2016 ACM SIGCOMM Conference*, pages 1–14, 2016.

[3] Salvatore Pontarelli, Roberto Bifulco, Marco Bonola, Carmelo Cascone, Marco Spaziani, Valerio Bruschi, Davide Sanvito, Giuseppe Siracusano, Antonio Capone, Michio Honda, et al. Flowblaze: Stateful packet processing in hardware. *In Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 531–548, 2019.

[4] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[5] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[6] Abdul Alim, Richard G Clegg, Luo Mai, Lukas Rupprecht, Eric Seckler, Paolo Costa, Peter Pietzuch, Alexander L Wolf, Nik Sultana, Jon Crowcroft, et al. Flick: Developing and running application-specific network services. *In Proceedings of the 2016 USENIX Annual Technical Con- ference (USENIX ATC 16)*, pages 1–14, 2016.

[7] Priyanka Naik and Mythili Vutukuru. libvnf: A framework for building scalable high performance virtual network functions. *In Proceedings of the 8th Asia-Pacific Workshop on Systems*, pages 1–8, 2017.

[8] Priyanka Naik, Akash Kanase, Trishal Patel, and Mythili Vutukuru. libvnf: building virtual network functions made easy. *In Proceedings of the ACM Symposium on Cloud Computing*, pages 212–224, 2018.

[9] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uh-

表 3 サービス実装における LoC
Table 3 LoC in service implementation.

項目 (数字はチェイニング時の数)	LoC (ライブラリ未使用)	LoC (ライブラリ使用)	削減率
VNF(パケット) x 1	126	11	91.3%
VNF(パケット) x 2	137	20	85.4%
VNF(パケット) x 3	148	27	81.8%
VNF(フロー) x 1	134	11	91.8%
VNF(フロー) x 2	146	20	86.3%
VNF(フロー) x 3	158	27	82.9%

lig. Elastic sketch: Adaptive and fast network-wide measurements. *In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 561–575, 2018.

[10] Cha Hwan Song, Pravein Govindan Kannan, Bryan Kian Hsiang Low, and Mun Choon Chan. Fcm-sketch: generic network measurements with data plane support. *In Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pages 78–92, 2020.

[11] Azure：SONiC, 入手先 ⟨https://github.com/Azure/SONiC/wiki⟩ (参照 2021-2-15)

[12] IETF:sFlow 入手先 ⟨https://tools.ietf.org/html/rfc3176⟩ (参照 2021-2-15)

[13] TCPDUMP&LiBPCAP：Libpcap 入手先 ⟨https://www.tcpdump.org/⟩ (参照 2021-2-15)

[14] NFDUMP：nfdump 入手先 ⟨https://github.com/phaag/nfdump⟩ (参照 2021-2-15)

[15] Iperf：iPerf3 入手先 ⟨https://iperf.fr/iperf-download.php⟩ (参照 2021-2-15)