Vol.2021-ARC-246 No.2
2021/10/11

# SDNコントローラにおける優先度付きキューを用いた高優先度パケットの処理高速化

高倉 玲央[1,a)]　八巻 隼人[1]　三輪 忍[1]　本多 弘樹[1]

**概要**：近年，インターネットやデータセンタにおいてネットワーク機器の一括管理や，より細粒度なネットワーク制御を実現するためにSDN（Software-Defined Networking）が活用されている．SDNにおいて，ソフトウェアで実装されるSDNコントローラの処理遅延が大きいことが課題として挙げられており，特に近年の遅延クリティカルなアプリケーションを利用する場合にこの問題が顕著となる．そこで，本稿ではSDNコントローラに優先度付きキューを実装し，このような遅延クリティカルなアプリケーションでの使用が想定されるToS値の高い高優先度なパケットの処理を高速化するシステムを提案する．パケットトレースを用いた評価では，高優先度と低優先度なパケットがランダムな順序で混在するトレースにおいて，高優先度パケットの処理を8スレッドに並列化することで，低優先度パケットの処理遅延増加を1.02倍に抑えつつ，高優先度パケットの処理遅延を0.56倍まで削減できることを示した．

## 1. はじめに

近年，ネットワーク機器における経路設定やロードバランシング，フィルタリングといったネットワーク制御に関する設定をソフトウェアにより行うSDN（Software-Defined Networking）が注目され，WAN（Wide Area Network）やデータセンタ，無線ネットワーク等において広く活用されている [1], [2], [3], [4], [5], [6]．SDNにより，ユーザやアプリケーションを識別してネットワーク機器の制御を変更するといった，従来の手動設定では不可能な柔軟性の高いネットワーク制御が可能となる．特に，パケットのヘッダ部分のみならず，通信内容を含んだペイロード部分まで解析するペイロード解析と，SDNを組み合わせることで，通信内容に応じた動的経路制御 [7] やセキュリティ [8]，ネットワークスライシング [9], [10] といった高度な機能の実現が期待されている．

SDNの課題として，ソフトウェア処理を行うSDNコントローラの処理遅延が挙げられる．SDNでは，単純なパケット転送機能を提供するSDNスイッチと，SDNスイッチを制御するSDNコントローラが物理的に分離されており，SDNコントローラはSDNスイッチを細粒度に制御するためにソフトウェアで実装される．そのため，SDNコントローラのパケット処理遅延は，$\mu$sオーダのSDNスイッチと比べて，msオーダと大きい．

一方，近年の通信では，AR/VR（Augmented/Virtual Reality）やロボットの遠隔操作等，リアルタイム性が重視される遅延クリティカルなアプリケーションが数多く登場している．このようなアプリケーションの通信における許容遅延は数〜100ms程度とされており，現在のSDN環境でこれら遅延クリティカルなアプリケーションを利用することはまだ現実的ではない．

そこで，本稿では，SDNコントローラに優先度付きキューを実装し，遅延クリティカルなアプリケーションの通信で想定されるToS値の高い高優先度なパケットの処理をマルチスレッドで処理することで高速化するシステムを提案する．本稿では，現在広く使われているオープンソースのSDNコントローラであるRyuに提案システムを実装した．低優先度，高優先度パケットが混在するパケットトレースを用いた評価では，提案システムにより，高優先度と低優先度なパケットがランダムな順序で混在するトレースにおいて，高優先度パケットの処理を8スレッドに並列化することで，低優先度パケットの処理遅延増加を1.02倍に抑えつつ，高優先度パケットの処理遅延を0.56倍まで削減できることを示した．

本稿は以下のように構成されている．第2章では，SDNの概要とその構成要素について詳述する．第3章で提案手法について述べる．第4章では評価方法および評価結果を示す．第5章で関連研究について述べ，第6章に結論をまとめる．

[1] 電気通信大学
The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[a)] takakura@hpc.is.uec.ac.jp

ⓒ 2021 Information Processing Society of Japan

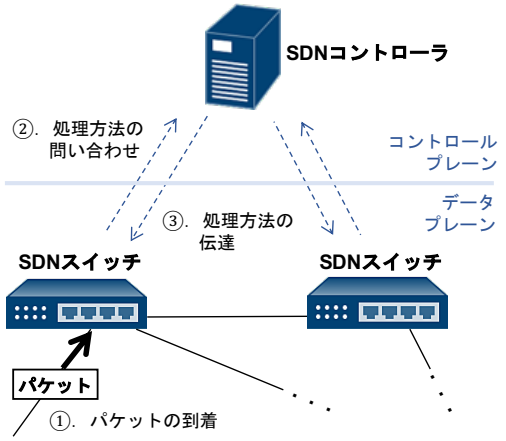


**図 1** SDN の概要

## 2. SDN

### 2.1 概要

SDN（Software-Defined Networking）は ONF（Open Networking Foundation[6]）により提唱された，ネットワーク管理用ソフトウェアによってネットワーク機器の動的な制御変更，またネットワーク自体の構成の変更を可能とするネットワークアーキテクチャである．なお，本稿では，SDN を実現するためのプロトコルとして同じく ONF が標準化を進める OpenFlow[11] を用いた SDN の実装を前提として，以降の説明を進める．

図 1 に SDN の概要を示す．SDN では，パケット転送を担うデータプレーンと，データプレーンを制御するコントロールプレーンが分離される．本稿では，各プレーンの機能を有する機器をそれぞれ SDN スイッチ，SDN コントローラと呼ぶ．パケットが SDN スイッチに到着すると，そのスイッチにおいて処理方法が定義されていないパケットは SDN コントローラへと送信され，処理方法の問い合わせがなされる．SDN コントローラは SDN スイッチから受信したパケットの処理方法を決定し，SDN スイッチへと伝達する．SDN スイッチは SDN コントローラより受信したパケットの処理方法に基づき，当該パケットと以降の同一処理が行われるパケットを転送する．

従来のネットワークの場合，ネットワーク機器の制御を変更するには当該ネットワーク機器に直接またはリモートで接続した上で，手動で設定を変更する必要があった．一方，SDN の場合は上述したように，SDN スイッチの設定を SDN コントローラを通じて変更できるため，複数項目や複数スイッチの一括での設定変更が可能であり，従来の手動設定による労力や設定ミスを削減することが出来る．また近年では，細部の設定変更が可能な SDN の利点を生かして，動的経路制御や DPI（Deep Packet Inspection）を利用した経路制御といった柔軟な制御の実現に SDN が用いられている．

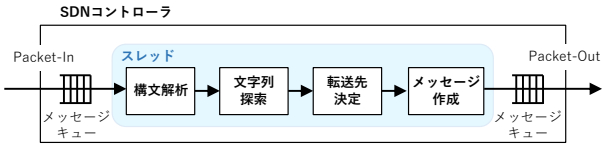


**図 2** SDN コントローラのアーキテクチャ

一般的に，SDN スイッチは OpenFlow 対応のベンダ製スイッチやベアメタルスイッチといった，高速なパケット転送ハードウェアを持つ機器が用いられるが，SDN コントローラは処理の複雑さからソフトウェアにより実装される．特に，SDN コントローラが DPI のようなパケットのペイロードにまで踏み込んだ複雑な処理を行う場合，$\mu$s オーダのパケット処理遅延である SDN スイッチに対し，SDN コントローラのパケット処理遅延は ms オーダと大きく，SDN コントローラの高速化が大きな課題となる．

### 2.2 SDN コントローラ

本節では SDN コントローラについて詳細を述べる．SDN コントローラのアーキテクチャを図 2 に示す．なお，本アーキテクチャはオープンソースの SDN コントローラとして広く利用されている Ryu[12] を基にしている．以下では，それぞれについて説明する．

**Packet-In メッセージ**

前節で述べたように，SDN スイッチが処理方法について未定義のパケットを受信した場合，SDN スイッチは当該パケットの転送処理を問い合わせるために，SDN コントローラへと Packet-In メッセージを送信する．このメッセージには，処理を問い合わせたパケットのメタデータが含まれている．

**メッセージキュー**

SDN コントローラから受信したメッセージ，また SDN コントローラへ送信するメッセージは，メッセージキューに格納されて処理される．通常，SDN スイッチはパケット到着順に Packet-In メッセージを SDN コントローラへ送信するため，メッセージキューに格納される Packet-In メッセージの格納順も前後することはない．

**メインスレッドでの処理**

メインスレッドでは，メッセージの解析を行いパケットのメタデータを取り出し，文字列探索等の操作を行って転送先を決定したあと，Packet-Out メッセージの作成と送信を行う．従来の SDN コントローラでは，シングルスレッドにてメッセージキューの先頭から逐次処理を行うため，メッセージキューが貯まっている場合には，処理が開始されるまでに大きな遅延が生じることある．

**Packet-Out メッセージ**

SDN コントローラが Packet-In メッセージを受信した場合，メッセージ内のパケットのメタデータを元にして，

ⓒ 2021 Information Processing Society of Japan

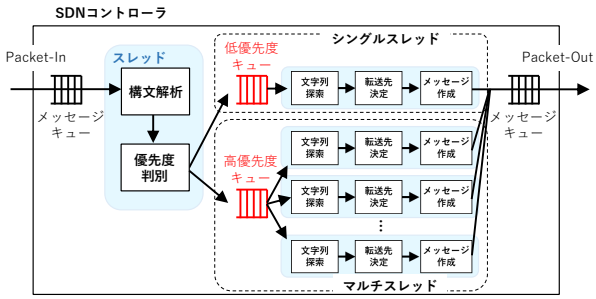


**図 3** 提案手法における SDN コントローラのアーキテクチャ

当該パケットの転送先を決定する．この転送先を含めた Packet-Out メッセージを SDN スイッチへ送信することで，SDN スイッチはパケットの転送処理を行う．

## 3. 提案手法

### 3.1 概要

本稿では，既存の Ryu コントローラに対して，処理するパケット情報を低優先度/高優先度の 2 種類の優先度付きキューに振り分けるシステム，また，優先度付きキューからパケット情報を取り出し，高優先度パケットに対してはマルチスレッドで並列に処理をおこなうシステムを提案する．図 3 に提案システムのアーキテクチャを示す．

提案手法では優先度付きキューを実装し，パケットのメタデータの参照，転送先の決定，Packet-Out メッセージの作成と送信については，メインスレッドの後段に存在する複数のサブスレッドにて処理を行う．このとき，低優先度パケットについては 1 つのサブスレッドにて，高優先度パケットについては複数のサブスレッドにて処理を行うため，高優先度パケットの処理遅延を削減することが可能である．

### 3.2 優先度付きキュー

提案手法におけるコントローラでは，メインスレッドは Packet-In メッセージを受信すると，メッセージを解析して内部にあるパケットのメタデータを取り出し，パケットが低優先度もしくは高優先度のどちらであるかを判別して，対応した優先度のキューにメタデータを格納する．そのため，メインスレッドでは文字列探索と転送先の決定，Packet-Out メッセージの作成と送信は行わない．

Ryu は Python ベースのソフトウェアなので，優先度付きキューには Python の queue モジュールの同期キューを用いて実装した．Python の同期キューでは，マルチスレッド処理で同じキューへのアクセスがあった場合に，スレッドセーフを保つようにアクセスに対してロックを提供している．そのため，後述する処理システムにて複数のスレッドでのアクセスの際にデータの整合性が無くなることはない．

メインスレッドでは，パケットの IP ヘッダ中の ToS（Type of Service）フィールドを参照することで，当該パケットが低優先度もしくは高優先度のどちらであるかを判別する．ToS フィールドは 8 ビットで構成され，それぞれのビットの役割は RFC791, RFC1349, RFC2474 にてそれぞれ定義されている．提案手法では，RFC2474 にて ToS フィールドから再定義された DS フィールドに非対応の機器を考慮して，RFC791 と RFC1349 にて優先度として定義されている先頭 3 ビット（DS フィールドでは優先度を表す 6 ビットの上位 3 ビット）の IP プレシデンスの値を，優先度を表す ToS 値として参照する．

### 3.3 優先度別の処理システム

優先度付きキューにエンキューされたパケットのメタデータは，メインスレッドの後段にある複数のサブスレッドにて取り出され，ペイロードデータの文字列探索と Packet-Out メッセージの作成/送信が行われる．マルチスレッド機構については，Python の threading モジュールを用いて実装した．Python のマルチスレッドでは，スレッド間で共有するメモリに対するアクセスについて，データの整合性は保証されていない．

今回の実装では，優先度付きキューに関して複数のスレッドでのアクセスが想定されるが，前述の通り優先度付きキューとして用いている同期キューにはスレッドセーフを保つためのロックが提供されているため，データの整合性は保証される．

また，マルチスレッドにおける問題点の一つとして，スレッドの立ち上げと立ち下げにかかるオーバヘッドの大きさが挙げられる．今回の実装では，コントローラの起動時にあらかじめサブスレッドの立ち上げを行い，立ち上げたサブスレッドをプールしておく方法を選んだ．この方法の利点として，優先度付きキューにデータが格納されるたびにサブスレッドの立ち上げを行う必要が無く，処理を行わないときはサブスレッドは待機状態となるためオーバヘッドを減らすことが出来る．

ペイロードデータの文字列探索機構として，今回は最もシンプルな探索アルゴリズムであるナイーブ法（力任せ法）にて実装した．この探索アルゴリズムでは，探索する文字列を先頭から順にパターンと一致するか確認し，一致しなかった場合，1 文字ずらして再度パターンと一致するか確認する．このアルゴリズムの場合，探索する文字列長を $n$，パターン長を $m$ とすると，ワーストケースで $O(nm)$ の計算量となり，同じ文字列探索アルゴリズムである KMP 法（クヌース・モリス・プラット法）の $O(n)$ や BM 法（ボイヤー・ムーア法）の最良の場合の $O(n/m)$ よりも劣る．

ⓒ 2021 Information Processing Society of Japan

表 1　評価に用いたマシンの詳細

| | 仕様 |
|---|---|
| OS | Ubuntu 18.04.5 LTS |
| CPU | Xeon E5-1620 v4 3.50GHz，4 コア |
| メモリ | 66GB |

## 4. 評価

提案手法の有効性について，Ryu コントローラに提案手法を実装したうえで，高優先度/低優先度パケットを含めたパケットトレースを作成し，以下の 2 項目について評価を行った．

(1) 高優先度パケットを処理するサブスレッド数の違いによる処理遅延
(2) 文字列探索におけるパターン数の違いによる処理遅延

本章では，まず評価方法について述べ，次に評価結果について考察する．

### 4.1 評価方法

本評価では，1 台のマシン上に仮想ネットワークを構築し，評価環境を作成した．マシンの詳細は表 1 に示す．

仮想ネットワーク内には 2 つのエンドポイント，OpenFlow スイッチ，Ryu コントローラの計 4 ノードを配置した．ネットワークトポロジーはスイッチに他の 3 ノードが接続するスター型である．図 4 に評価方法の概要図を示す．

評価に用いるトレースは高優先度パケットを 500 パケット，低優先度パケットを 500 パケットの計 1000 パケットを含めたトレースを作成した．1 パケットのサイズは一般的な MTU（Maximum Transmission Unit）サイズである 1500 バイト（TCP ペイロード長 1448 バイト）とし，高優先度パケットの場合は ToS 値を 5 に，低優先度パケットの場合は ToS 値を 0 に書き換えた．RFC791 より，5 の場合は優先度クリティカル, 0 の場合は優先度標準となる．また，トレースは①高優先度パケット 500 が前半に，低優先度パケット 500 が後半に配置されたトレース High_Low，②低優先度パケット 500 が前半に，高優先度パケット 500 が後半に配置されたトレース Low_High，③高優先度パケットと低優先度パケットが混在するトレース Random の計 3 種類を用意した．

作成したトレースを仮想ネットワーク上の片方のエンドポイントからもう片方のエンドポイントへと送信する．エンドポイント間にはスイッチが存在し，スイッチは受信したパケットを Packet-In メッセージとして Ryu コントローラへと送信する．Ryu コントローラではトレースの 1 パケット目の Packet-In メッセージを受信した時刻と，高優先度/低優先度パケットの 500 パケット目の Packet-Out メッセージを送信した時刻を計測し，その差をそれぞれの優先度の処理遅延として評価を行う．

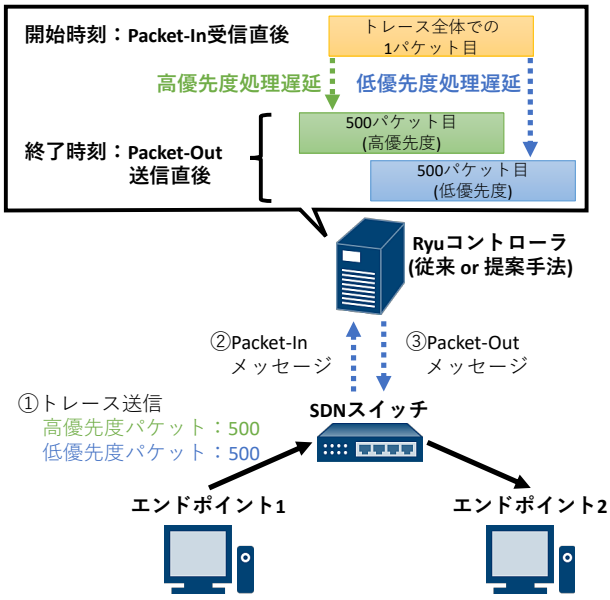


図 4　評価方法の概要図

### 4.2 高優先度パケットを処理するサブスレッド数の違いによる処理遅延

本評価では，図 2 に示したような従来の Ryu コントローラと，図 3 に示したような提案手法を実装した Ryu コントローラを用意し，提案手法の Ryu コントローラ内の高優先度パケットを処理するサブスレッド数を 1, 2, 4, 8 スレッドとした場合の処理遅延を 5 回ずつ測定し，それぞれの平均を求めた．なお，従来・提案手法のコントローラ共に，ペイロードデータに対して 10 バイトのパターンを 20 個探索することを想定し，文字列探索を行っている．図 5, 6, 7 にそれぞれ測定結果を示す．

提案手法において，処理遅延の削減効果がワーストケースになると予想されるのは，高優先度パケットが低優先度パケットより先に到着する場合である．これは，従来のコントローラがパケットを到着順に処理する，つまり高優先度パケットをシングルスレッドにて優先的に処理するのに対して，提案手法では 1 スレッド分を低優先度パケットの処理に充てているからである．

図 5 の結果では，トレースとして High_Low を用いており，高優先度パケット 500 が先に，低優先度パケット 500 が後に到着する．従来のコントローラの場合，シングルスレッドにて高優先度パケット 500 を処理し終えた後に低優先度パケット 500 を処理するため，高優先度パケットの処理遅延は 2526ms，低優先度パケットの処理遅延は 5034ms と，処理遅延がほぼ 1：2 の割合となっている．

一方，提案手法の場合，サブスレッド数を増やすと高優先度パケットの処理遅延は削減されるが，最良の場合でも 8 スレッド時の 2965ms であり，従来のコントローラと比べて約 1.17 倍の処理遅延となった．低遅延パケットの処理遅延については，8 スレッド時で 5238ms であり，従来

ⓒ 2021 Information Processing Society of Japan

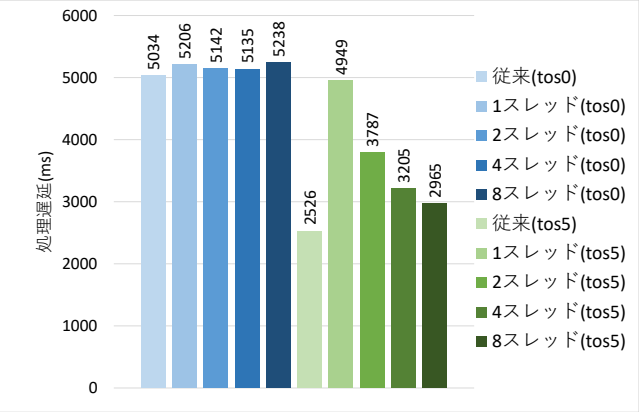


**図 5** High_Low トレースでの処理遅延

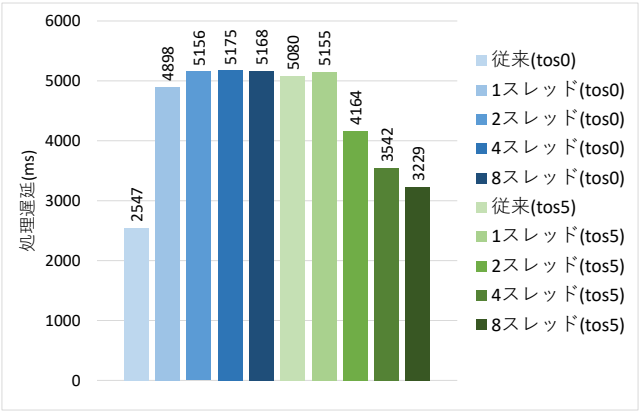


**図 6** Low_High トレースでの処理遅延

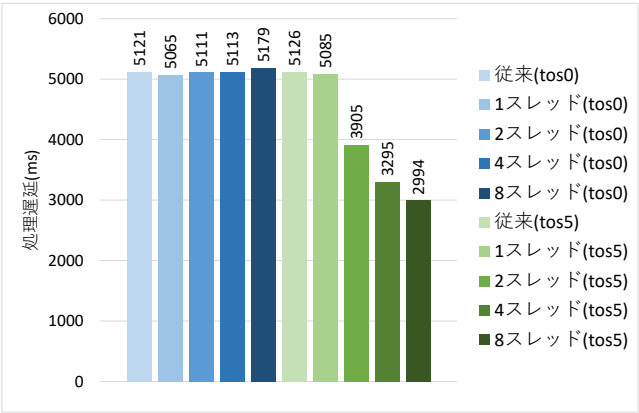


**図 7** Random トレースでの処理遅延

のコントローラとほぼ変わらない処理遅延となった．

提案手法において，処理遅延の削減効果がベストケースになると予想されるのは，高優先度パケットが低優先度パケットより後に到着する場合である．これは，従来のコントローラがパケットを到着順に処理する，つまり低遅延パケットをシングルスレッドにて優先的に処理してしまい，高優先度パケットの処理開始までに遅延が発生するのに対して，提案手法では優先度別にキューに振り分け，高優先度パケットをマルチスレッドで処理するからである．

**表 2** 文字列探索パターン数による処理遅延 (ms)
（括弧内の値は従来に対する比率）

| | | パターン数 | | |
|---|---|---|---|---|
| | | 1 | 20 | 50 |
| 従来 | 低優先度 | 424 (1) | 5121 (1) | 12321 (1) |
| | 高優先度 | 425 (1) | 5126 (1) | 12334 (1) |
| 提案 | 低優先度 | 483 (1.14) | 5179 (1.01) | 12563 (1.02) |
| | 高優先度 | 327 (0.76) | 2994 (0.58) | 7076 (0.56) |

図 6 の結果では，トレースとして Low_High を用いており，低優先度パケット 500 が先に，高優先度パケット 500 が後に到着する．従来のコントローラの場合，メインスレッドにて低優先度パケット 500 を処理し終えた後に高優先度パケット 500 を処理するため，高優先度パケットの処理遅延は 5080ms，低優先度パケットの処理遅延は 2547ms と，処理遅延がほぼ 2：1 の割合となっている．

一方，提案手法の場合サブスレッド数を増やすと，高優先度パケットの処理遅延は削減され，最良の場合では 8 スレッド時の 3229ms であり，従来のコントローラと比べて約 0.64 倍の処理遅延となった．低遅延パケットの処理遅延については，8 スレッド時で 5168ms であり，従来のコントローラの約 2.03 倍の処理遅延となった．

トレースとして Random を用いた結果が図 7 である．このトレースでは，高優先度パケット 500 と低優先度パケット 500 がトレース全体に混在しており，不規則に到着する．従来のコントローラの場合，高優先度パケットの処理遅延は 5126ms，低優先度パケットの処理遅延は 5121ms と，処理遅延がほぼ同じ値となっている．一方，提案手法の場合，サブスレッド数を増やすと高優先度パケットの処理遅延は削減され，最良の場合では 8 スレッド時の 2994ms であり，従来のコントローラと比べて約 0.58 倍の処理遅延となった．低遅延パケットの処理遅延については，8 スレッド時で 5179ms であり，従来のコントローラとほぼ変わらない処理遅延となった．

### 4.3 文字列探索におけるパターン数の違いによる処理遅延

本評価では，図 2 に示したような従来の Ryu コントローラと，図 3 に示したような，提案手法において高優先度パケットを処理するサブスレッド数を 8 スレッドで実装した Ryu コントローラを用意し，ペイロードデータに対して行う文字列探索のパターン数を 1, 20, 50 個とした場合の処理遅延を 5 回ずつ測定し，それぞれの平均を求めた．なお，トレースは 4.1 節の評価にて提案手法の有効性が高かったトレース Random を用いており，従来，提案手法のコントローラ共に文字列探索でのパターン長は 10 バイトである．表 2 に測定結果を示す．

表 2 より，1 パターンの文字列探索の場合では，提案手法において，低優先度パケットの処理遅延が従来と比べて 1.14 倍，高優先度パケットの処理遅延が従来と比べて 0.76

倍となった．一方，50 パターンの文字列探索の場合では，8 スレッドでの提案手法において，低優先度パケットの処理遅延が従来と比べて 1.02 倍，高優先度パケットの処理遅延が従来と比べて 0.56 倍となった．

パターン数を増やすにつれて，低優先度パケットの処理遅延の増加を抑えつつ，高優先度パケットの処理遅延を大きく削減できていることが確認できる．これは，パターン数が少ない場合においては，メインスレッドによるパケットのメタデータの優先度付きキューへの格納や，サブスレッドによる Packet-Out メッセージの作成/送信といった，文字列探索以外の処理のオーバヘッドが大きく影響していることが理由として考えられる．

## 5. 関連研究

NIDS の処理負荷削減を目的とした，SDN コントローラにてペイロードデータに対して文字列探索を行い，動画フローの判別を行う手法 [13] が提案されている．この手法では，SDN コントローラにてパケットペイロード内の HTTP ヘッダを参照して，パケットフローのコンテンツの種類を判別することにより，SDN スイッチを制御して動画フローを NIDS へ転送しないことで NIDS の処理負荷を削減している．

## 6. おわりに

近年，インターネットやデータセンタにおいてネットワーク機器の一括管理や，より細粒度なネットワーク制御を実現するために SDN（Software-Defined Networking）が活用されている．SDN の実現において，ソフトウェアで実装される SDN コントローラの処理遅延が大きいことが課題として挙げられており，特に近年の遅延クリティカルなアプリケーションを利用する場合にこの問題が顕著となる．そこで，本稿では SDN コントローラに優先度付きキューを実装し，このような遅延クリティカルなアプリケーションでの使用が想定される ToS 値の高い高優先度なパケットをマルチスレッドで処理することで高速化するシステムを提案した．

高優先度と低優先度のパケットの配置が異なる 3 種類のパケットトレースを用いた評価では，高優先度と低優先度のパケットがランダムな順序で混在するトレースにおいて，高優先度パケットの処理を 8 スレッドに並列化することで，低優先度パケットの処理遅延増加を 1.01 倍に抑えつつ，高優先度パケットの処理遅延を 0.58 倍まで削減できることを示した．

さらに，上記の条件において文字列探索を行うパターン数を 20 個から 50 個に増加させた場合に，低優先度パケットの処理遅延増加を 1.02 倍に抑えつつ，高優先度パケットの処理遅延を 0.56 倍まで削減できることを示した．

**謝辞** 本研究は JSPS 科研費 JP21K17730 の助成を受けたものである．

### 参考文献

[1] Shirmarz, A. and Ghaffari, A.: Performance issues and solutions in SDN-based data center: a survey, *The Journal of Supercomputing*, Vol. 76, No. 10, pp. 7545–7593 (2020).

[2] Amin, R., Reisslein, M. and Shah, N.: Hybrid SDN Networks: A Survey of Existing Approaches, *IEEE Communications Surveys & Tutorials*, Vol. 20, No. 4, pp. 3259–3306 (2018).

[3] Bizanis, N. and Kuipers, F. A.: SDN and virtualization solutions for the Internet of Things: A survey, *IEEE Access*, Vol. 4, pp. 5591–5606 (2016).

[4] Hong, C.-Y., Mandal, S., Al-Fares, M., Zhu, M., Alimi, R., B., K. N., Bhagat, C., Jain, S., Kaimal, J., Liang, S., Mendelev, K., Padgett, S., Rabe, F., Ray, S., Tewari, M., Tierney, M., Zahn, M., Zolla, J., Ong, J. and Vahdat, A.: B4 and after: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google's Software-Defined WAN, *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, Association for Computing Machinery, pp. 74–87 (2018).

[5] Yousaf, F. Z., Bredel, M., Schaller, S. and Schneider, F.: NFV and SDN—Key Technology Enablers for 5G Networks, *IEEE Journal on Selected Areas in Communications*, Vol. 35, No. 11, pp. 2468–2478 (2017).

[6] ONF: Open Networking Foundation, , available from ⟨https://www.opennetworking.org/⟩ (accessed 2021-09-06).

[7] Bellavista, P., Giannelli, C. and Montenero, D. D. P.: A Reference Model and Prototype Implementation for SDN-Based Multi Layer Routing in Fog Environments, *IEEE Transactions on Network and Service Management*, Vol. 17, No. 3, pp. 1460–1473 (2020).

[8] Islam, M. J., Mahin, M., Roy, S., Debnath, B. C. and Khatun, A.: Distblacknet: A distributed secure black sdn-iot architecture with nfv implementation for smart cities, *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, IEEE, pp. 1–6 (2019).

[9] Ordonez-Lucena, J., Ameigeiras, P., Lopez, D., Ramos-Munoz, J. J., Lorca, J. and Folgueira, J.: Network slicing for 5G with SDN/NFV: Concepts, architectures, and challenges, *IEEE Communications Magazine*, Vol. 55, No. 5, pp. 80–87 (2017).

[10] Barakabitze, A. A., Ahmad, A., Mijumbi, R. and Hines, A.: 5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges, *Computer Networks*, Vol. 167, p. 106984 (2020).

[11] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J.: OpenFlow: Enabling Innovation in Campus Networks, *SIGCOMM Comput. Commun. Rev.*, Vol. 38, No. 2, pp. 69–74 (2008).

[12] Ryu: Ryu SDN Framework, , available from ⟨https://ryu-sdn.org/⟩ (accessed 2021-09-06).

[13] 高倉玲央，八巻隼人， 三輪忍，本多弘樹：OpenFlow を用いた動画フローの非ミラーリングによる NIDS 処理負荷の削減，電子情報通信学会技術研究報告，Vol. 119, No. 343, pp. 51–56 (2019).

ⓒ 2021 Information Processing Society of Japan