Vol.2018-IOT-40 No.40
2018/3/6

MPTCPにおける外部プログラムによるプライマリサブフローの切替

堀込 怜土[1,a)]　山井 成良[1,b)]　北川 直哉[1,c)]　大坐畠 智[2,d)]

概要：多人数が同時に無線ネットワークを使用する環境下において，多くの端末がWiFiとLTEなど複数のネットワークインタフェースを備えているとき，特定のAPにトラヒックが偏ってしまう問題が起こる．トラヒックの集中を緩和するため，筆者らは，MPTCPプロトコルで端末と接続されているVPNサーバを用いて，動的にトラヒック分散を行うことができるネットワーク構成を検討している．当該ネットワークで使用するVPNサーバは，各端末との通信で主に使用するサブフロー (プライマリサブフロー) をAPの接続状況をもとに選択することができる．しかし，プライマリサブフローを選択する機能をVPNサーバに実装することは困難であり，保守性に欠ける．そこで，本稿では，この問題を解決するため，外部プログラムによってプライマリサブフローを切り替える方法を提案する．このプログラム用いることで，VPNサーバプログラムを改変することなく，プライマリサブフローを切り替えることが可能となる．

キーワード：MPTCP，VPN，トラヒック分散

Change of Primary Subflow by External Program in MPTCP

Reido Horigome[1,a)]　Nariyoshi Yamai[1,b)]　Naoya Kitagawa[1,c)]　Satoshi Ohzahata[2,d)]

Abstract: When users' terminals with multiple wireless network interfaces such as WiFi and LTE have access to the Internet, traffic congestion often occurs since many of them use the same Access Point (AP) of WiFi network for example. To mitigate this congestion, we consider a network configuration that can perform dynamic traffic sharing with special VPN servers communicating with users' terminal through MultiPath TCP (MPTCP). These VPN servers have a function that can select a primary subflow for each terminal dynamically according to the current conditions of APs. However, implementation of this function into server program reduces software maintainability and increases implementation cost. In this paper, we propose a subflow changing method using an external program to solve these problems. With this program, the VPN server can change the primary subflows without modification.

Keywords: MPTCP，VPN，traffic sharing

1. はじめに

最近のスマートフォンやタブレットなどのモバイル端末には，WiFiとLTEなど複数のネットワークインタフェース (NIC) を搭載しているものが多い．複数のNICを利用し，LTEだけでなくWiFiも利用できる場合にはそちらを利用して通信を分散するといった機能を持つものも多い．また，複数のネットワークを同時に利用するマルチホーミング通信に対応する機器も存在する．マルチホーミング通信を実現する技術としてMPTCP (MultiPath TCP) があり，iPhoneなどのモバイル端末で利用可能である．これらの複数のネットワークを用いた通信では，スループットや

[1] 東京農工大学
Tokyo University of Agriculture and Technology
[2] 電気通信大学
The University of Electro-Communications
[a)] reidoh@net.cs.tuat.ac.jp
[b)] nyamai@cc.tuat.ac.jp
[c)] nakit@cc.tuat.ac.jp
[d)] ohzahata@is.uec.ac.jp

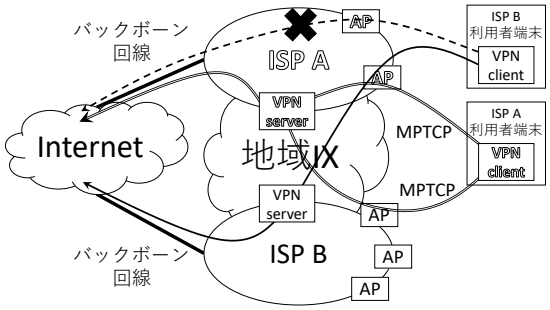

図 1 検討中のネットワーク構成

冗長性の向上が見込まれる反面，多くのデバイスの通信が特定のアクセスポイント (AP) に集中する問題が起こる．例えば，WiFi と LTE が利用可能な場合，WiFi が優先的に使われることが多く，WiFi の AP にトラヒックが偏ってしまう．この問題に対して，端末側で通信状況を把握し，AP を切り替える研究が行われているが [1]，AP を切り替える際にオーバヘッドが生じるという問題点がある．

そこで，我々の研究チームは**図 1** のようなネットワーク構成を検討している．図 1 において ISP A, B はローミングを行っており，ISP B の利用者端末は ISP A の AP を経由して ISP B 内の VPN サーバとの間で MPTCP による接続が可能になっている．また，ISP A の利用者端末のように ISP A, B の両者の AP に同時にアクセスできる場合には，両方の AP を同時に利用する．これにより，全てのアプリケーションにおいて，雑音など周囲の環境の影響を受けやすい無線ネットワークの部分をマルチホーム化できる．MPTCP に対応した HTTP プロキシを用いることで，HTTP 通信においてサーバが MPTCP に対応していなくてもマルチホーミング通信を行えるようにする研究 [2] があるが，本ネットワーク構成においては宛先のサーバがどのような種類のサーバであってもマルチホーム化できるため，より汎用性が高い．

本ネットワーク構成において，VPN サーバは各利用者端末がどのネットワークを使用しているかを全て把握できる．把握したネットワークの使用状況をもとに，VPN サーバで各利用者端末との通信を行う経路を切り替えれば，端末が使用する AP が切り替わり，特定の AP への通信の集中を緩和することができる．

MPTCP における各経路には優先度があるため，優先度を変更できれば経路の切り替えが可能である．経路の優先度を，MPTCP を利用するアプリケーションから変更する機能はすでに提案されている [3]．しかし，その機能を利用するためにはアプリケーションに経路を変更する機能を追加する必要がある．VPN サーバのような複雑なプログラムにこの機能を追加するのは困難であり，また，保守性に欠ける．そこで，本稿では，MPTCP を利用するアプリケーションプログラムとは異なる，外部のプログラムによって経路を変更するシステムを提案する．提案するシステムを用いれば，アプリケーションプログラムを改変することなく，使用する経路を切り替えることが可能となる．

本稿では，まず，2 章で提案システムに関連する技術について述べる．次に，3 章で提案システムについて述べる．さらに，4 章で提案システムの動作確認の結果を示し，5 章でまとめと今後の課題を述べる．

2. 関連研究

2.1 MPTCP

MPTCP は，マルチホーミング通信を実現するプロトコルである [4]．MPTCP は TCP を拡張したものであり，RFC6824[5] で仕様が定められている．複数の TCP コネクションを同時に利用することで，スループットや通信の冗長性の向上を計っている．MPTCP が商用で使われている代表的な例として iOS の Siri がある [6]．MPTCP で扱われる個々の TCP コネクションはサブフローと呼ばれる．

MPTCP 以外のマルチホーミング通信を実現するプロトコルとして SCTP[7] があるが，SCTP を導入するためには既存の TCP 対応のアプリケーションを SCTP に対応するよう変更する必要がある．一方，MPTCP は TCP を拡張したものであるため，カーネルが MPTCP に対応していればアプリケーションの変更をせずに利用できる．検討しているネットワークでは TCP を用いることを想定しているため，本研究では MPTCP を用いる．

2.1.1 path manager

MPTCP では端末が複数の NIC を搭載している際に，NIC をどのように使用してサブフローを生成するかを選択する必要がある．Linux 上の MPTCP では，サブフローの生成の仕方は path manager によって管理される．現在 (バージョン 0.92) の MPTCP では，path manager として default，fullmesh，ndiffports，binder のいずれかを選択可能である．default を選択した場合，送信元は確立要求の際に通信先に IP アドレスを通知せず，新たなサブフローは生成されない．fullmesh を選択した場合，通信元と通信先のすべてのインタフェースの組合せのサブフローが生成される．ndiffports を選択した場合，1 組の IP アドレスについて異なる送信元ポート番号を使用することで複数のサブフローが生成される．binder を選択した場合，Loose Source Routing に基づいた経路選択が行われる．

2.1.2 scheduler

Linux 上の MPTCP では，各サブフローで通信を行う際，どのサブフローを使用するかを管理する機能があり，scheduler と呼ばれる．現在の MPTCP では，schduler として default，roundrobin，redundant のいずれかを選択可能である．default を選択した場合，RTT (Round Trip Time)

が最も小さいサブフローを用いて通信を行う．roundrobin を選択した場合，ラウンドロビン方式で使用するサブフローを選択する．redundant を選択した場合，すべてのサブフローで同じデータを重複して送信する．

2.1.3 サブフローの優先度

MPTCP において各サブフローには優先度を設定することができる．優先度として，backup モードと active モードを設定できる．backup モードのサブフローは，他に active モードサブフローが存在する場合には使用されない．backup モードのサブフローは通常使用されないため，利用料金などの面から優先的な使用を避けたいサブフローがある場合に有用である．

2.2 サブフローの優先度を変更する機能

現在の MPTCP では，MPTCP 対応の iproute コマンド [8] によって NIC 単位でサブフローの優先度を変更することができる．しかし，各サブフローの優先度を変更することはできない．そこで，文献 [3] において，各サブフローの優先度を変更する機能が提案されている．提案されている機能は，setsockopt 関数に優先度を変更したいサブフローの識別番号と優先度を示す値を渡すことで，当該サブフローの優先度を変更できるものである．文献 [3] に示されている設定の例を以下に示す．

```
struct mptcp_sub_prio {
        __u8 id;
        __u8 low_prio;
};
struct mptcp_sub_prio flow_prio = {5, 1};
setsockopt(clientSocket, IPPROTO_TCP,
        MPTCP_SET_SUB_PRIO,
        &flow_prio,
        sizeof(flow_prio));
```

まず，mptcp_sub_prio 構造体に，優先度を変更したいサブフローの識別番号 5 と優先度を示す値 1 を格納する．優先度を示す値は，1 が backup モード，0 が active モードを表す．そして，setsockopt 関数に，優先度を変更する機能に割り当てられた MPTCP_SET_SUB_PRIO という定数と mptcp_sub_prio 構造体を渡す．この例では，識別番号 5 のサブフローの優先度を示す値を 1 に設定している．これにより，識別番号 5 のサブフローは backup モードとなる．当該機能を追加するためのパッチはメーリングリストで公開されている．この機能を用いて，主に使用するサブフロー (プライマリサブフロー) を active モードにし，その他のサブフローを backup モードにすることにより，プライマリサブフローを切り替えることが可能である．

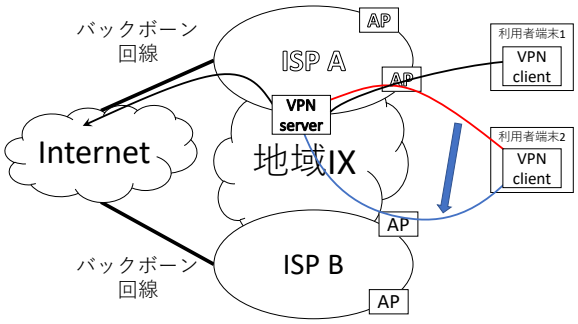

図 2 検討中のネットワーク構成でのトラヒック分散

3. 提案手法

3.1 検討中のネットワーク構成でのトラヒック分散

検討中のネットワークにおいて，トラヒック分散を行う様子を図 2 に示す．図 2 では，ISP A の利用者端末は，ISP A の VPN サーバと MPTCP 通信を行っており，ISP A,B それぞれの AP を経由する 2 つのサブフローを用いている．そして，ISP A の AP を経由するサブフローがプライマリサブフローである．一方，ISP B の利用者端末は，ISP B の VPN サーバと TCP 通信を行っており，ISP A の AP を経由する 1 つの経路を用いている．この場合，ISP A の利用者端末と ISP A の VPN サーバとの通信において，ISP B の AP を経由するサブフローをプライマリサブフローとすることで，各 AP の利用の偏りが小さくなる．これを実現するためには，ISP A の VPN サーバに，ISP A の利用者端末のプライマリサブフローを切り替える機能が必要である．

3.2 提案手法の概要

VPN サーバによって，利用者端末プライマリサブフローを切り替えることは，2.2 節で説明した機能を利用すれば可能である．しかし，この機能を使用するためには，VPN サーバ内で setsockopt 関数を呼びだす必要があるため，VPN サーバの改変が必要となる．VPN サーバのような複雑なプログラムを改変するのは困難であり，また，保守性に欠けるため，他の方法を考える必要がある．

そこで，2.2 節で述べた機能がどのように動作するかカーネルのソースコードを解析した．その結果，サブフローの優先度を変更するとき，low_prio および send_mp_prio の値を変更していることがわかった．low_prio および send_mp_prio については 3.3 節で詳しく述べる．何らかの方法でこれらの値を変更すれば，VPN サーバを改変することなくプライマリサブフローを切り替えることができる．そこで，VPN サーバとは異なるプログラム（外部プログラム）によりカーネル内で管理されている各サブフローの low_prio と send_mp_prio の値を書き換えることにより，プライマリサブフローを変更する方法を提案する．

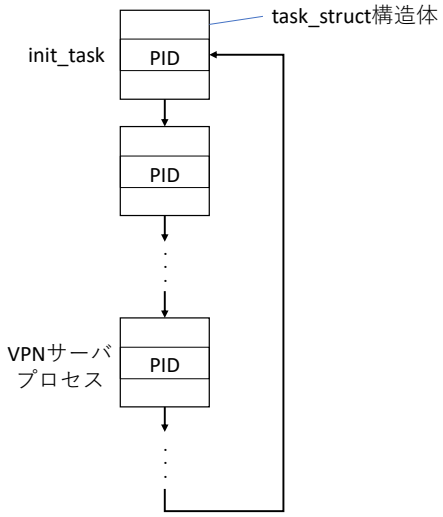

図 3　VPN サーバプロセスに対応する task_struct 構造体へのアクセス

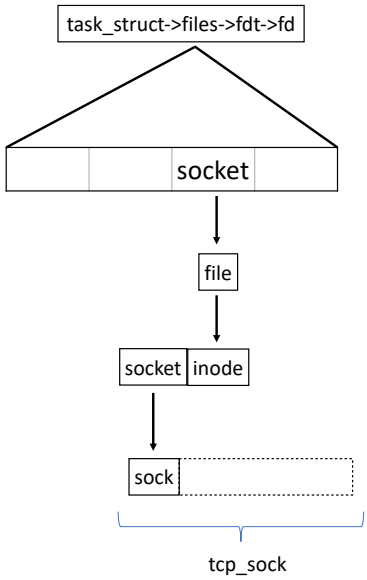

図 4　task_struct 構造体からメタソケットへのアクセス

3.3　Linux カーネルにおける MPTCP の構造

前節で述べた提案手法を Linux 上で実装した．Linux における MPTCP の実装は活発に進められており，また，MPTCP の導入が容易であることから Linux を選択した．

提案手法を実現するためには，VPN サーバプログラムが使用するサブフローに関する low_prio および send_mp_prio へアクセスする必要がある．カーネルのソースコードを解析した結果，low_prio および send_mp_prio へ，以下の 3 つの手順でアクセス可能であることがわかった．

(1) VPN サーバプロセスのプロセス ID (PID) から VPN サーバプロセスの情報を格納している task_struct 構造体へアクセスする
(2) VPN サーバプロセスの情報を格納している task_struct 構造体から，VPN サーバプロセスが開いている TCP ソケット (メタソケット) へアクセスする
(3) メタソケットから，low_prio および send_mp_prio へアクセスする

ここで，task_struct 構造体はプロセスの情報を格納するための構造体で，1 つのプロセスの情報は 1 つの task_struct 構造体に格納されている．また，MPTCP のソケットはアプリケーションからは TCP ソケットとして扱えるようになっており，その TCP ソケットはメタソケットと呼ばれる．

本節では，これら 3 つの手順について詳しく説明する．

3.3.1　PID から task_struct 構造体へのアクセス

PID をもとに VPN サーバプロセスの情報を格納している task_struct 構造体へアクセスする方法を述べる．VPN サーバプロセスの情報を格納している task_struct へアクセスする様子を図 **3** に示す．task_struct 構造体には pid メンバが存在し，プロセスの PID が格納されている．Linux では，task_struct 構造体が双方向リストで連結されている．この双方向リストは，init_task と呼ばれる task_struct 構造体を先頭にして循環している．そのため，init_task から双方向リストをたどると，全てのプロセスを走査することができる．したがって，init_task から PID を確認しつつプロセスを参照していけば，VPN サーバプロセスの PID をもつ task_struct 構造体へアクセスできる．

3.3.2　task_struct からメタソケットへのアクセス

task_struct 構造体からメタソケットへアクセスする様子を図 **4** に示す．まず，VPN サーバプロセスの情報を格納している task_struct 構造体の files メンバ (files_struct 構造体)，files_struct 構造体の fdt メンバ (fdtable 構造体)，fdtable 構造体の fd メンバとたどる．fd メンバはプロセスが開いているファイルの情報を持つ構造体 (file 構造体) へのアドレスの配列である．次に，配列の中からファイルの種類がソケットのものにアクセスする．続いて，アクセスした file 構造体の f_inode メンバ (inode 構造体) の直後に配置されている socket 構造体へアクセスする．その際，f_inode メンバのアドレスの値から socket 構造体の大きさを引くことで該当の socket 構造体のアドレスを取得することができる．アクセスした socket 構造体の sk メンバ (sock 構造体) がメタソケットを表している．Linux では，sock 構造体にソケットの基本情報が含まれており，その下に IP，TCP のような上位プロトコルの情報が付随している．そのため，sock 構造体を TCP ソケットを表す tcp_sock 構造体の大きさで読み込むことで，TCP ソケットとして取り出すことができる．

3.3.3　メタソケットから low_prio および send_mp_prio へのアクセス

次に，取得したメタソケットから low_prio と send_mp_prio へアクセスする様子を図 **5** に示す．メタソケットは mptcp_cb 構造体をメンバに持つ．mptcp_cb 構造

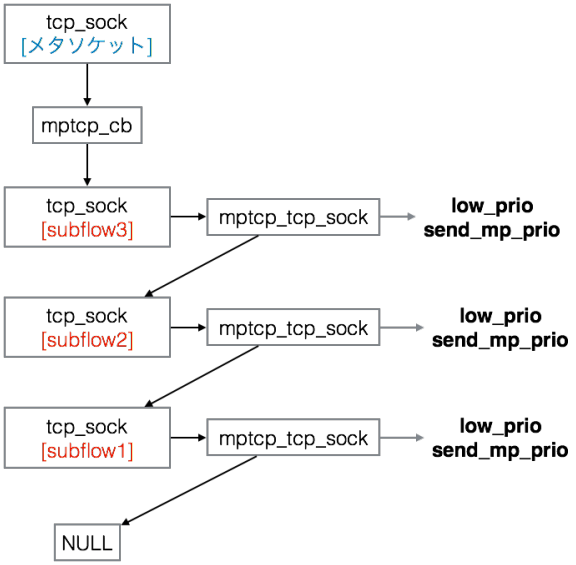

図 5　MPTCP ソケットの構造

体から，最後に生成されたサブフローに対応する tcp_sock 構造体へアクセスできる．各サブフローに対応する tcp_sock 構造体は mptcp_tcp_sock 構造体をメンバに持っている．mptcp_tcp_sock 構造体は各サブフローの MPTCP に関する情報を保持しており，low_prio と send_mp_prio の値も保持している．

3.4　実装方法

実装環境は以下の通りである．

- Linux ディストリビューション：Ubuntu16.04
- Linux バージョン：4.4.83
- MPTCP バージョン：0.92

提案システムを実装するためには，カーネル内部の low_prio および send_mp_prio を書き換える必要がある．そこで，物理メモリ上の各変数に対応する値を書き換えることとした．Linux において物理メモリにアクセスするため，/dev/mem ファイル [9] を利用した．/dev/mem ファイルはメモリを表すデバイスファイルである．/dev/mem ファイルに対して読み書きを行うことで，メモリ内部の値を操作することができる．Linux での x86_64 アーキテクチャのメモリマップ [10] を参考にし，メモリへのアクセスを行った．

デフォルトの設定で，/dev/mem を介してメモリ上の low_prio および send_mp_prio が格納されているアドレスへアクセスしようとしたところ，Linux のメモリに対する保護機能により，アクセスすることができなかった．そこで，/dev/mem にアクセスするため，カーネルのコンパイルの際にカーネルのソースファイルが置かれているディレクトリ内のファイル「.config」内の CONFIG_STRICT_DEVMEM を無効にし，カーネルを再コンパイルした．これにより，low_prio および send_mp_prio が格納されているアドレスへのアクセスが可能となった．

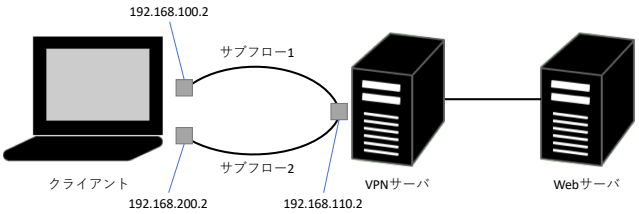

図 6　動作確認で用いるネットワーク構成

サブフローの優先度を変更する方法を説明する．まず，コンパイルされたカーネルオブジェクトのシンボルテーブルを調べ，3.3.1 節で述べた init_task のアドレスを調査した．次に，init_task から 3.3 節で述べた手順で low_prio と send_mp_prio を取得した．low_prio には，サブフローの優先度を表す値が格納されており，backup モードのときは 1，active モードのときには 0 となっている．send_mp_prio には，サブフローの優先度を変更したことを通信相手に伝えるかどうかを表す値が格納されており， backup モードのときは 1，active モードのときには 0 となっている．サブフローを backup モードにする場合には，low_prio と send_mp_prio の両方に 1 を書き込む．サブフローを active モードにする場合には，low_prio に 0，send_mp_pri に 1 を書き込む．

4.　動作確認

動作確認として，クライアントが VPN サーバに接続されている状態で通信を行い，通信途中に外部プログラムによりプライマリサブフローを切り替える実験を行った．実験を行う際，path manager と scheduler は，文献 [11] を参考にして，それぞれ fullmesh，default とした．VPN サーバプログラムには OpenVPN を用いた．

実験で用いるネットワーク構成を**図 6** に示す．クライアントと VPN サーバが MPTCP を用いた OpenVPN プロトコルで接続されている．そして，VPN サーバが Web サーバと接続されている．Web サーバには実験に使用するためのファイルが置いてある．クライアントは，サブフロー 1 のインタフェースに 192.168.100.2，サブフロー 2 のインタフェースに 192.168.200.2 の IP アドレスを持つ．VPN サーバは，192.168.110.2 の IP アドレスを持つ．このネットワーク構成で，クライアントが Web サーバから 20M バイトのファイルをダウンロードしている途中で．**図 7** のようにプライマリサブフローを変更した．

挙動を確認するため，以下に示した tcpdump コマンドをクライアント上で実行し，クライアントの両インタフェースでやりとりされたパケットを観測した．

```
tcpdump -i any port 1194
```

any によってすべてのインタフェースを指定し，OpenVPN が使用する 1194 番ポートに限定することで，OpenVPN に関する通信のみ観測するようにした．

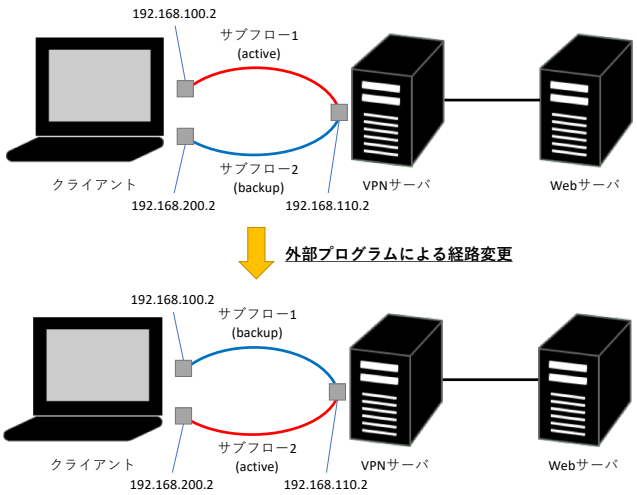

図 7　プライマリサブフローを変更する様子

```
16:44:14.889298 IP 192.168.100.2.38456 > 192.168.110.2.1194:
16:44:14.928097 IP 192.168.110.2.1194 > 192.168.100.2.38456:
16:44:14.928118 IP 192.168.100.2.38456 > 192.168.110.2.1194:
16:44:14.928312 IP 192.168.110.2.1194 > 192.168.100.2.38456:
```

図 8　プライマリサブフロー変更前

```
16:45:18.496169 IP 192.168.200.2.43622 > 192.168.110.2.1194:
16:45:18.496434 IP 192.168.110.2.1194 > 192.168.200.2.43622:
16:45:18.496445 IP 192.168.200.2.43622 > 192.168.110.2.1194:
16:45:18.533030 IP 192.168.110.2.1194 > 192.168.200.2.43622:
```

図 9　プライマリサブフロー変更後

プライマリサブフローを変更する前のコマンドの結果を図 8 に示す．図 8 より，プライマリサブフロー変更前は 192.168.100.2 と 192.168.110.2 の通信が行われている．つまり，サブフロー 1 が使われている．次にプライマリサブフローを変更した後の図 9 では，192.168.200.2 と 192.168.110.2 の通信が行われている．つまり，サブフロー 2 が使われている．以上の実験結果から，通信中に，提案方式を実装したプログラムにより low_prio および send_mp_prio の値を書き換えると，実際に使用されるサブフローが変更されることが確認できた．

5. おわりに

複数の NIC を持つ端末が多く存在する環境では，特定の AP に通信が集中する問題がある．その問題に対し，端末を VPN サーバに接続し，端末と VPN サーバとの通信で主に使用する経路を管理できるネットワーク構成について述べた．．本ネットワーク構成を実現するためには，VPN サーバにおいて端末との通信で主に使われる経路を変更する機能が必要であるが，既存の技術を用いた方法では VPN サーバプログラムの改変が必要であった．そこで，MPTCP において外部プログラムによってプライマリサブフローを切り替えるシステムを開発し，動作確認を行った．動作確認の結果，実際にプライマリサブフローが切り替わることが確認できた．

今後の課題として，提案したシステムを活用し，複数のクライアントのプライマリサブフローを調整することで，トラヒックを分散する機能を実装することを考えている．また，その機能を利用して実際にトラヒック分散を行い，特定の AP への通信の集中が緩和されるかを調べる実験を行う．さらに，各クライアントの通信の利用状況に応じて，プライマリサブフローを切り替えるかどうかをクライアントごとに判断することも考えている．

謝辞

本研究の成果の一部は科学研究費助成事業：基盤研究（C）（17K00118）「アクセスクラウド：複数の無線系ネットワークを活用した公平かつ高速な通信」の助成による．

参考文献

[1] 河田真宏，玉井森彦，安本慶一：仮想化ネットワークインターフェースを用いたトリガ駆動に基づく Wi-Fi アクセスポイントの動的負荷分散方式，研究報告モバイルコンピューティングとユビキタス通信 (MBL)， Vol. 2013, No. 28, pp. 1–8 (2013).

[2] Pollalis, C., Charalampou, P. and Sykas, E.: HTTP data offloading using multipath TCP proxy, *CIT/IUCC/DASC/PICOM, 2015 IEEE International Conference on*, pp. 777–782 (2015).

[3] Mondal, A., K, A. and Shailendra, S.: Enhanced Socket API for MPTCP-Controlling Sub-flow Priority, *arXiv preprint arXiv:1707.03585* (2017).

[4] ip networking lab: MultiPath TCP, `http://www.multipath-tcp.org`.

[5] IETF: RFC6824: TCP Extensions for Multipath Operation with Multiple Addresses, `https://tools.ietf.org/html/rfc6824`.

[6] Apple, I.: Use Multipath TCP to create backup connections for iOS, `https://support.apple.com/en-us/HT201373`.

[7] IETF: RFC4960: Stream Control Transmission Protocol, `https://tools.ietf.org/html/rfc4960`.

[8] multipath tcp: iproute-mptcp, `https://github.com/multipath-tcp/iproute-mptcp`.

[9] Linux: MEM(4), `http://man7.org/linux/man-pages/man4/mem.4.html`.

[10] Linux: x86_64 メモリマップ，`https://github.com/torvalds/linux/blob/v4.10/Documentation/x86/x86_64/mm.txt`.

[11] Wang, K., Dreibholz, T., Zhou, X., Fa, F., Tan, Y., Cheng, X. and Tan, Q.: On the path management of multi-path tcp in internet scenarios based on the nornet testbed, *Advanced Information Networking and Applications (AINA), 2017 IEEE 31st International Conference on*, IEEE, pp. 1–8 (2017).