2020
秋号

百萬石
Hyakumangoku Vol.55

Published by MMM

部長挨拶

tonphy

tonphy@mma.club.uec.ac.jp

「百萬石」をお手に取っていただき、ありがとうございます。弊誌 (以下、部誌) は、電気通信大学の公認サークル MMA が新歓期 (春号) と弊学が開催する調布祭時期 (秋号) の年 2 回発行・頒布しているものです。MMA とは、”Microcomputer Making Association” の頭文字に由来しており、1975 年に創立されたサークルです。現在では計算機に関連することを中心に、ネットワークやセキュリティなどの分野でも活動しています。部誌には、MMA の部員が各々に行なっている活動を記事にしたものが掲載されています。内容は多岐に渡り、高度なプログラミングの話から旅行記など、MMA の活動の自由さが感じられるものとなっております。

MMA の活動状況を報告させていただきます。今年はコロナによるサークル活動自粛によって対面での活動が行えなかった為、前学期は新入生の入部手続きや、新入生向けの講習会などを行う事ができませんでした。そこで対応措置として、仮入部という形で新入生を受け入れ、新しく slack を作成したりオンラインで競技プログラミングの講習会を行ったりしました。特に競技プログラミングの講習会では、熟練度に合わせて 2 つの難易度に分けて講習をしました。既存部員も自分の技術を再確認できる良い機会だったのではないかと思います。また、正式な部員だけではありますが、部会もオンラインで開催しました。

後学期では仮入部員も部会に参加できるようにし、現在正式入部を行う為の手続きを行っています。また、サークル活動も徐々に解禁されている為、近いうちに元通りに MMA の運営が行えると考えております。

今後も MMA は技術が好きな人が集まる自由なサークルであり続けたいと思います。最後まで読んでいただきありがとうございました。

2020 年 11 月吉日

百萬石 Vol.55

Hyakumangoku Vol.55

2020 Autumn

MMA

目次

第 1 章　スーパーマリオ 64 スター 120 枚への道　ykm11　1
1.1　Preliminary 1
1.2　120 枚への道 3
1.3　まとめ 6

第 2 章　VPN のつくりかた　shosato　7
2.1　はじめに 7
2.2　環境 8
2.3　アプリケーションからのパケットを受け取る 8
2.4　LAN 環境下での双方向通信の仕組み 11
2.5　VPN からインターネットへ 13
2.6　インターネットから VPN へ 13
2.7　VPN サーバから VPN クライアントへ 14
2.8　VPN クライアントからアプリケーションへ 14
2.9　動作確認 15
2.10　おわりに 16
参考文献 16

第 1 章

スーパーマリオ 64 スター 120 枚への道ykm11

こんにちは，トリです．すこし肌寒く，秋の訪れを感じるような気候になってきましたがみなさんいかがお過ごしでしょうか．

さて，2020 年 9 月 18 日にスーパーマリオ 3D コレクションが発売されました．みなさんはもう既にクリアし，Real Time Attack（RTA, 海外では Speedrun が一般的）に挑戦していることかと思います．筆者はたまに Twitter に小技のプレイ動画を投稿しています．本稿では，筆者が考案した，スーパーマリオ 64 においてすべてのスター 120 枚を効率よく集めるルートの紹介をします．LBLJ やラプラス，ムシャーナなど[*1]個々の技については紹介しません（筆者ができません）．まだプレイしていない方はルートの参考にしてください．

1.1 Preliminary

1.1.1 コースの略記

マリオ 64 の各コース名は英語名の略記で表記されることが多いです．本稿でもそれに倣って略記を使います．各コースと略記の対応を表 1.1, 表 1.2 に示します．

[*1] ポケモンの名前が技名に採用されることが多いです

表 1.1　メイン 15 コース

ステージ名（日本語）	略記
ボムへいのせんじょう	BoB
バッタンキングのとりで	WF
かいぞくのいりえ	JRB
さむいさむいマウンテン	CCM
テレサのホラーハウス	BBH
やみにとけるどうくつ	HMC
ファイアバブルランド	LLL
あっちっちさばく	SSL
ウォーターランド	DDD
スノーマンズランド	SL
みずびたシティ	WDW
たかいたかいマウンテン	TTM
ちびでかアイランド	THI
チックタックロック	TTC
レインボークルーズ	RR

表 1.2　クッパ

ステージ名（日本語）	略記
やみのせかいのクッパ	BitDW
ほのおのうみのクッパ	BitFS
てんくうのたたかい！	BitS

その他の “ピーチの隠れスライダー”，“にじかけるはねマリオ”，“おさかなといっしょ” についてはそれぞれスライダー，にじかけ，おさかなと略すことにします．帽子ステージは VC(Vanish Cap)，WC(Wing Cap)，MC(Metal Cap) と略します．

1.1.2　各コースのスター枚数について

各コースのスターは 6 枚 +100 枚コインの計 7 枚です．これと隠れスター（スライダーやクッパステージの赤コインなどは隠れスター扱いとなります）15 枚を合わせて 120 枚となります．

1.2　120 枚への道

ルートを

1. 地下 in まで
2. 2 階 in まで
3. 3 階 in まで
4. 3 階, BitS

の区間に分け，それぞれの区間のルートを表で示してから解説を行います．

1.2.1　地下 in まで

表 1.3　地下 in まで

ステージ名	累計スター
BoB (1)	3
CCM	10
WF	17
スライダー (1)	18
WC	19
スライダー (2)	20
BoB (2)	24
BitDW	25
VC	26
BBH	33

ゲーム開始時点で入れるコースは BoB のみです．RTA であれば LBLJ をキメて BitDW から攻略しますが，LBLJ はすぐに修得できるようなものではありません（3D コレクションではそもそも BLJ ができません）．BoB で “やまのうえのボムキング”, “はくねつノコノコレース”, “ワンワンのいぬごやで” の 3 枚を取得します．WC がないと BoB はクリアできないので，スター 3 枚取得で入れる隣の部屋の CCM に行きます．RTA でいきなり BitDW へ行くのはこれが理由ですね．

CCM，WF をクリアしたら右上の部屋からスライダーに入場します．スライダーのスターは通常クリアと 21 秒未満クリアの 2 枚があります．クリア後はロビーに戻されるので WC ステージに入ります（スター 10 枚取得で解禁）．再度スライダーステージへ行き 2 枚目のスターを取得します．

WC を取得したあとは BoB に戻ってクリアし，左上の部屋から BitDW に行きます．JRB の “ふきだすみずをくぐれ” は MC がないとスター取得が難しいです．最速泳ぎができるなら MC 取得前に攻略し

ますが，初心者向けではないので後に回しましょう．2 階 in の前に来ます．

BBH は VC がないとクリアできないので，一旦地下にいって VC を取得します．取得度，再度 VC ステージに入りポーズメニューを出して「コースをでる」を選択するとロビーに戻れるので時間短縮になります．BBH をクリアしましょう．

1.2.2　2 階 in まで

表 1.4　2 階 in まで

ステージ名	累計スター
ミップ (1)	34
SSL	41
LLL	48
HMC	55
MC	56
キノピオ (1)	57
ミップ (2)	58
DDD (1)	62
BitFS	63
DDD (2)	66
- 1 階 -	
おさかな	67
JRB	74

BBH クリア後は地下ステージを攻略します．スターを 15 枚集めると地下にミップ（うさぎ？）が出現し，捕まえるとスターを貰えます．SSL の入り口前で捕まえることで，SSL 入場がスムーズになります．SSL の 100 枚コインは “きょだいピラミッドのなぞ” で取得します．外で 60 枚以上のコインを集めておくと，ピラミッド内部で 100 枚にできます．SSL のあとは LLL に行きます．LLL の 100 枚コインは “かざんのパワースター” で取得します．外でコインを 80 枚集めておくと，火山の頂上への道中で 100 枚にできます．

HMC の “メタルでダッシュ” は MC がなくても，水中のスイッチをヒップドロップで押すことでスター取得が可能です．HMC クリア後は近くにいるキノピオに話しかけてスターを貰います．キノピオから貰えるスターは地下，2 階，3 階の計 3 枚です．

スター 50 枚取得によりミップが再度出現します．DDD へ行く前にスターを貰っておきましょう．DDD の “クッパのせんすいかん” のスターを取得するまで DDD の入口が BitFS の入口の手前にあり，BitFS に入れないようになっています．このスターを取得することで DDD の入口が遠くに移動します．入口が動いてからは，スター取得から DDD への再入場までが長くなってしまうので，DDD の入口が

近い位置にあるうちにスターを 4 枚取得します．DDD の入口が移動している様子を図 1.1，図 1.2 に示します（ゲームのスクリーンショットについては任天堂のガイドライン[*2]に従っています）．見にくいかもしれませんが，DDD の入口（波打っているところ）が動いていることがわかると思います．先に取得するスターは “うずしおのたからばこ”，“ふきだすみずをくぐれ”，“ボウシがそろったら” と “クッパのせんすいかん” です．言わずもがなですが，“クッパのせんすいかん” は 4 枚目に取得します．残りは BitFS クリア後に回収します．100 枚コインは赤コインと一緒に取得します．“マンタのおくりもの” は “クッパのせんすいかん” をクリアしないとマンタが出現しないので後に回しています．

DDD をクリアしたら BitFS に入りポーズメニューから「コースをでる」を選択，ロビーに戻り JRB へ行きます．おさかなは部屋に入って右のくぼみから入場できます．

図 1.1　DDD への入口が手前にある

図 1.2　クッパのせんすいかんのスター取得後

1.2.3　3 階 in まで

表 1.5　3 階 in まで

ステージ名	累計スター
WDW	81
THI	88
キノピオ (2)	89
TTM	96
SL	103

2 階に上がってすぐの絵から WDW に入り，THI，TTM，SL と反時計回りに攻略すると SL の部屋から出たとき目の前に 3 階への階段があります．2 階のコースでは，100 枚コインは赤コインと一緒に取得します．WDW では，ダウンタウンに赤コインと合わせてコインが 34 枚あるので，100 枚コインのときはダウンタウンに行く前にコインを 66 枚集めておきましょう．

[*2] https://www.nintendo.co.jp/networkservice_guideline/ja/index.html

1.2.4　3 階, BitS

表 1.6　3 階, BitS

ステージ名	累計スター
TTC	110
にじかけ	111
キノピオ (3)	112
RR	119
BitS	120
END	

3 階まで来たらあと少しです．まずは扉からまっすぐ進んで TTC をクリアしましょう．TTC の次は部屋の左側の小部屋からにじかけに行きます．反対側に RR の入り口がありますが，RR 入場前にキノピオからスターを貰っておきましょう．RR クリア時点でスターが 119 枚揃っているはずです．最後のクッパステージへ進みましょう．3 階の 100 枚コインは，TTC では “えっへんてっぺんドッスン”, RR では “てんくうのおやしき” で取得することをおすすめします．

1.3　まとめ

スーパーマリオ 64 でスターを 120 枚集めるルートを紹介しました．素人である筆者が考案したものですから，もっと良いルートがあるかもしれません．「このルートの方が速いよ！」といったコメントは大歓迎ですので，是非議論しましょう．ちなみに，2020 年 9 月 23 日時点で 16 枚 RTA の世界一位は日本の高校生です．120 枚のアジア一位も日本人です（世界だと 6 位です）．発売から 20 年以上経ってもプレイされ続けるマリオ 64 からまだまだ目を離せませんね．最後まで読んでいただき，ありがとうございました．

第 2 章

VPN のつくりかた

shosato

概要
Linux と C 言語でゼロから VPN のようなものを作ります

2.1 はじめに

VPN は Virtual Private Network のことで、別のネットワークにいる端末があたかも同じネットワークにいるかのように振る舞うようにする技術です。

VPN はクライアント・サーバ間で本当に送信したい IP パケットを暗号化し、それをデータとして TCP や UDP で送信します。もちろんこの TCP や UDP のデータを送信するのは IP パケットです[*1]。余分な IP ヘッダと TCP ヘッダが無駄ですがこれは致し方ない犠牲です。そして、TCP や UDP 経由で VPN サーバに到着した IP パケットは、そこから通常の IP パケットとして宛先に送信されます。

どこかのサイトにアクセスするとき、宛先がわからなければどうしようもないので IP パケットには必ず宛先アドレスが書かれていますが、ここは暗号化できないので、たとえ HTTPS 通信であっても盗聴者や ISP（インターネットサービスプロバイダ）にはどこにアクセスしているかわかってしまいます。VPN であれば本当に送信したい IP パケットを暗号化しているので ISP から見たら VPN サーバと何やら通信しているようにしか見えないのです。こうすることで ISP によるトラッキングからも逃れることができます。ただ、VPN サーバにはどこに接続しようとしているかわかるので、VPN プロバイダを利用してしまうと、信用する対象が ISP から VPN プロバイダに移るだけになってしまいます。そこで VPN サーバも自前で用意すればアクセスログも自分しか見れないので比較的プライバシーが確保されます。

この記事ではネットワークの学習のための VPN を作成しますので認証や暗号化は省きます（というのは建前で実装するのが面倒だっただけということは忘れてください）。実用したい人は OpenVPN[*2]を利用するのがいいと思います。この記事の VPN も OpenVPN を参考にしているのですんなり導入できるようになるはずです。なお、この記事で作るプログラムのソースコードは GitHub[*3]に置いてあります。

[*1] 簡単のために Ethernet ヘッダについては触れないことにします。
[*2] `https://openvpn.net/`
[*3] `https://github.com/shosatojp/microvpn`

2.2 環境

2.2.1 サーバ側

- Arch Linux (`5.8.14-arch1-1`)
- Conoha の VPS[*4]でグローバル IPv4 アドレスあり
- ネットワークインターフェースは `eth0`
- VPN のプライベート IP アドレスは `10.0.0.1` に設定
- TUN デバイスの名前は `tunserver` とします

2.2.2 クライアント側

- Ubuntu 20.04 LTS (`5.4.0-51-generic`)
- 一般家庭の LAN (`192.168.11.0/24`)
- ネットワークインターフェースは `eno1`
- VPN のプライベート IP アドレスは `10.0.0.2` に設定
- TUN デバイスの名前は `tunclient` とします

サーバ・クライアント間は UDP でポートは `1195` とします。図 2.1 が今回作る VPN の全体像です。次節以降適宜参照してください。

2.3 アプリケーションからのパケットを受け取る

まっ先に問題となるのが通常はインターネットに流れていくアプリケーションからのパケットをどうやって受け取るのかということです。Linux ではアプリケーションが TCP/UDP 通信した場合、カーネルによって IP パケットが作られます。すべてのパケットはカーネルが保持しているルーティングテーブルを参照して宛先が決められます。次のリストはデフォルトの状態のルーティングテーブルです。`192.168.11.0/24` は私の LAN で、`eno1` は PC の有線 LAN です（`eno1` のようなネットワークインターフェイスの名前は `ip addr` コマンドで調べることができます)。

```
$ route -n
Kernel IP routing table
Destination   Gateway      Genmask    Flags Metric Ref Use Iface
0.0.0.0        192.168.11.1  0.0.0.0        UG   100    0   0 eno1
192.168.11.0   0.0.0.0      255.255.255.0    U    100    0   0 eno1
192.168.11.1   0.0.0.0      255.255.255.255 UH   100    0   0 eno1
```

基本的にこの中から宛先アドレスが最も一致し、Metric が小さいものが選択されます。`0.0.0.0` はネットマスクが `0.0.0.0` ですので `0.0.0.0/0` ということですべての IP アドレスを示します。つまりす

[*4] `https://www.conoha.jp/vps/`

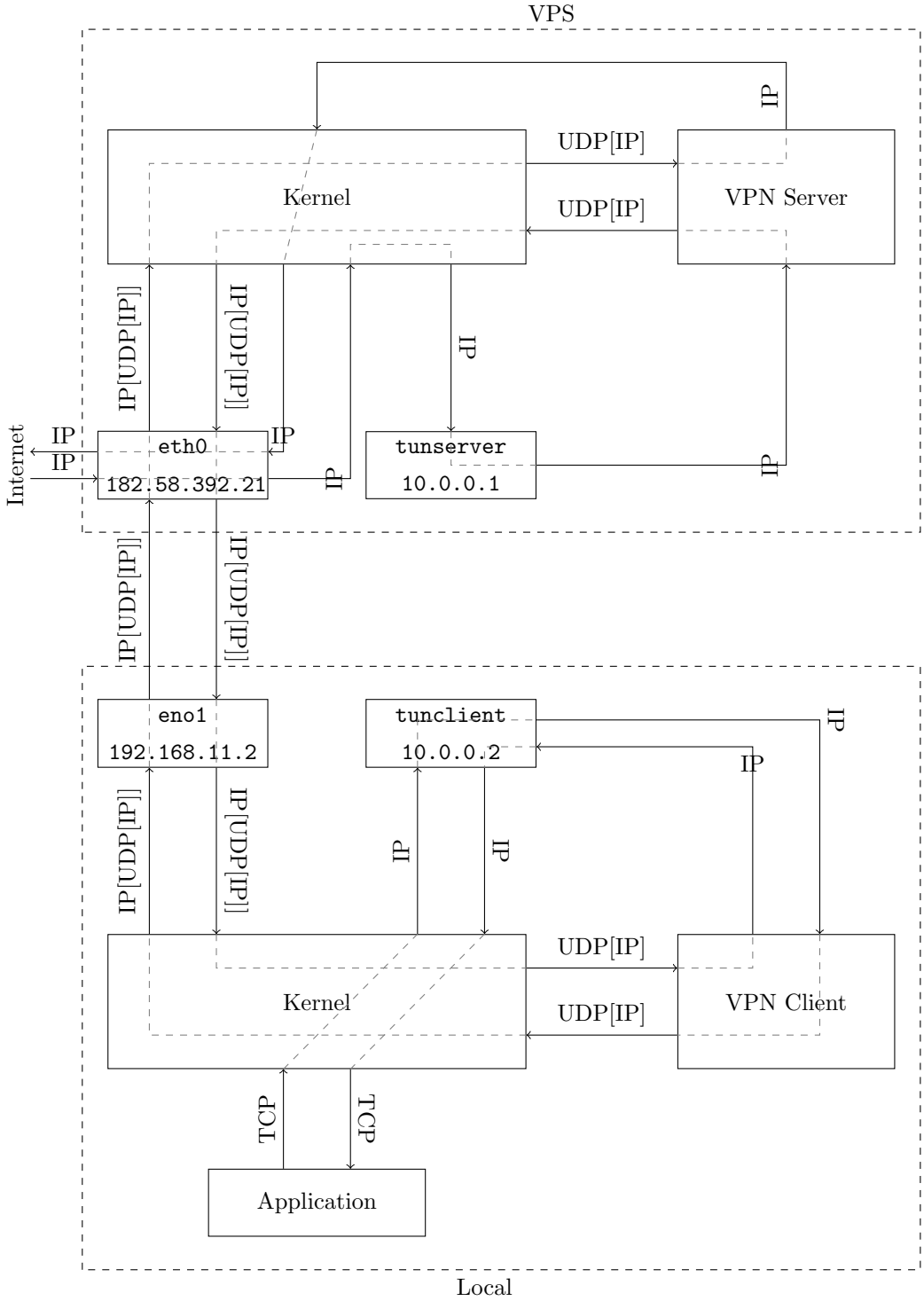

VPS
IP
UDP[IP]
Kernel
UDP[IP]
VPN Server
IP[UDP[IP]]
IP[UDP[IP]]
IP
Internet
IP
IP
eth0
182.58.392.21
IP
IP
tunserver
10.0.0.1
IP
IP[UDP[IP]]
IP[UDP[IP]]
eno1
192.168.11.2
tunclient
10.0.0.2
IP
IP
IP[UDP[IP]]
IP[UDP[IP]]
IP
IP
UDP[IP]
Kernel
UDP[IP]
VPN Client
TCP
TCP
Application
Local

図 2.1　パケットの流れ

べての通信は eno1 インターフェースから出ていくことになります（1 行目）。VPN クライアントではパケットを横取りして UDP 経由で送りたいので eno1 から出ていく代わりにプログラムに渡すことができれば良さそうです。

Linux にはそれを実現できる TUN という仮想ネットワークデバイスがあり、eno1 の代わりに使うことができます [7]。つまり、アプリケーションは TUN へパケットを送信し、帰ってきたパケットを TUN から受け取ることができるということです。これを利用することで各種アプリケーションからの IP パケットを VPN クライアントが受け取ることができるようになります。このデバイスは通常のファイルと同じように扱うことができ、ファイルを read()、write() で読み書きしたように、TUN を read() すると TUN に入ったパケットを読み取り、write() するとインターネットからパケットが送り返されたようになります。

TUN デバイスは次のようにして作成します。

```
// 希望する仮想デバイス名を入れる
int tun_alloc(char *dev) {
    struct ifreq ifr = {};
    int fd, err;

    if ((fd = open("/dev/net/tun", O_RDWR)) < 0) {
        printf("open()");
        return fd;
    }

    // IFF_TUN   : TUNを使う
    // IFF_NO_PI: 付加情報をつけないで生のパケットを読み書きする
    ifr.ifr_flags = IFF_TUN | IFF_NO_PI;
    if (*dev) {
        strncpy(ifr.ifr_ifrn.ifrn_name, dev, IFNAMSIZ);
    }

    if ((err = ioctl(fd, TUNSETIFF, (void *)&ifr)) < 0) {
        close(fd);
        perror("ioctl()");
        return err;
    }
    strncpy(dev, ifr.ifr_ifrn.ifrn_name, IFNAMSIZ);
    return fd;
}
```

デバイスを作成したらリンクアップします。C の system 関数で外部コマンドとして ip コマンドを呼び出しています。こうしているのは起動時に毎回このコマンドを打つのが面倒だからというだけです。

```
system("ip link set tunclient up");
```

次に TUN デバイスに IP アドレスを割り当てます。これは何でもいいですが、IPv4 のプライベート用 IP アドレス空間 [11] を使うのがいいでしょう。もし存在する IP アドレスに割り当ててしまうとそのサイトにアクセスできなくなります。ここではサーバに 10.0.0.1 を、クライアントに 10.0.0.2 を割り当てます（サブネットマスクは/24 としておきます）。

```
system("ip addr add 10.0.0.2 dev tunclient");
```

VPN クライアントプログラムがアプリケーションからのパケットを受け取るために、すべての通信が eno1 ではなく TUN に行くようにルーティングを行います。eno1 に行くエントリよりも Metric が小さくなるように TUN デバイスへの経路を追加します（クライアントプログラムで TUN デバイスを作成した後でないと設定できません）。ここではクライント側の TUN デバイスということで、インターフェース名は tunclient にします。

```
$ sudo ip route add 0.0.0.0/0 dev tunclient
# ip route で追加されたことを確認します。
```

これによりルーティングテーブルに次のエントリが追加されます。

```
Destination Gateway  Genmask   Flags Metric Ref Use Iface
0.0.0.0     0.0.0.0  0.0.0.0   U     0      0   0 tunclient
```

ただ、ひとつだけ問題があります。この設定では VPN サーバ宛の通信まで TUN に入ってしまいますから、VPN クライアントから VPN サーバに送ったパケットが無限ループしてしまいます。そこで VPN サーバ宛の通信は直接 eno1 から出るようにします。182.58.392.21 がサーバアドレスで、192.168.11.1 が LAN のゲートウェイ（ルータ）です。

```
$ sudo ip route add 182.58.392.21 via 192.168.11.1 dev eno1
```

```
// TUNでアプリケーションから受信
int datalen;
if ((datalen = read(tunfd, data, sizeof(data))) <= 0) {
  perror("read tunfd");
} else {
  // UDPでサーバに送信
  if (sendto(sockfd, data, datalen, 0,
        (struct sockaddr *)&server_addr,
        sizeof(server_addr)) < 0)
    perror("write sockfd");
}
```

2.4 LAN 環境下での双方向通信の仕組み

通信の両端がグローバル IP アドレスであれば特に何もしなくても通信できますが、少なくとも片方がルータ内にいて、プライベート IP アドレスしか持っていない場合にはひと捻り必要です（プログラムやコマンドでどうこうするわけではないです）。

サーバにはグローバル IP アドレスが割り当てられていて、クライアントは家庭内 LAN にいる環境を想定します。サーバのグローバル IP アドレスがわかるので LAN 内の端末から出たパケットはサーバに到達することができます。しかし、その逆のサーバからクライアントの場合、通常はひとつの家庭にひとつのグローバル IP アドレスが割り当てられているので、家庭内 LAN を外から見ると LAN 内のどの端末も同じグローバル IP アドレスを持つように見えます。つまり、サーバから出たパケットは LAN を目指しますが、家庭内 LAN の入り口までしか行けず、その先どの端末に行けばいいのかわからなくなってしまいます。これではもちろん双方向に通信することができないので、IP マスカレードと呼ばれる仕組

みが用いられます。これはまず間違いなくどの家庭用ルータにも実装されていて、LAN 内の送信元のプライベート IP アドレスとポート番号と、外から見たグローバル IP アドレスと任意のポート番号を変換するようにできています [9]。ここでパケットそのものを書き換え、その対応を覚えておくことで、一定のポート番号を使っている限りサーバからの通信を LAN 内の各端末に振り分けられるようになります。

しかしこの場合、サーバからクライアントへ通信を開始することはできません。まだクライント側のルータがポートの対応を知らないからです。ただ、クライアントが先にサーバへの通信を開始すれば、ルータがプライベート IP アドレスとポート番号の対応を覚えているので、サーバからクライアントへの通信も通ります。この覚えておく時間はルータによって違うと思いますが、Linux のカーネルパラメータのデフォルト値を見たところ UDP の場合 30 秒となっていました [8]。おそらくルータの OS にも Linux に似たようなものが使われているはずですのでだいたいこのくらいでしょう。つまり、30 秒間通信が途切れるとルータは対応付けを忘れてしまうのでサーバからのパケットを受け取れなくなってしまうということになります。今回の場合は PC で発生するほぼ全ての IP パケットをやり取りしているので 30 秒間全く通信がないということはまずないでしょう。

```
$ cat /proc/sys/net/netfilter/nf_conntrack_udp_timeout
30
```

サーバとクライアントのどちらもグローバル IP アドレスを持っていない場合、この手法を応用した UDP ホールパンチングというものを使いますが、VPN サーバはまずグローバル IP アドレスを持っていると思うので飛ばします。

プログラムは次のようになります。今回はクライアントから通信を開始するので問題ありません。

```
// クライアントから受信
int datalen;
if ((datalen = read(sockfd, data, sizeof(data))) <= 0) {
    perror("recv from client");
} else {
    // インターネットに送信
    struct sockaddr_in to = {
        .sin_family = AF_INET,
        .sin_addr.s_addr = ((struct iphdr *)data)->daddr,
    };
    if (sendto(eth, data, datalen, 0,
               (struct sockaddr *)&to, sizeof(to)) < 0) {
        perror("sendto internet");
    }
}
```

ただし、サーバやクライアントでファイアウォールが動いている場合には受信できないので、ポートを開けておきます。

```
$ sudo iptables -A INPUT -p udp --dport 1195 -j ACCEPT
# sudo systemctl reload iptables.service
```

2.5 VPN からインターネットへ

この IP マスカレードの仕組みを VPN サーバからインターネットにアクセスする場合にも使います。通常 VPN は複数のクライアントがいるので、さきほどの家庭内 LAN の各端末を VPN の各クライアントに、LAN を VPN に置き換えれば IP マスカレードの必要性はわかります。つまり、VPN のプライベート IP アドレスとポート番号と、サーバのグローバル IP アドレスと任意のポートを変換するということです。これはサーバ側で NAT（Network Address Translation）テーブルに手を加えればいいので、iptables[*5]を利用して次のようにします [5]。10.0.0.0/24 は VPN の各クライアントのプライベート IP アドレスで、eth0 はインターネットに接続されているネットワークデバイスです。

```
$ sudo iptables -t nat -A POSTROUTING -s 10.0.0.0/24 -o eth0 -j MASQUERADE
```

IP マスカレードを設定して、次のプログラムで IP パケットをインターネットに送信します。このとき送信元アドレスはクライアントの 10.0.0.2 のままです。

```
// クライアントから受信
int datalen;
if ((datalen = read(sockfd, data, sizeof(data))) <= 0) {
    perror("recv from client");
} else {
    // インターネットに送信
    struct sockaddr_in to = {
        .sin_family = AF_INET,
        .sin_addr.s_addr = ((struct iphdr *)data)->daddr,
    };
    if (sendto(eth, data, datalen, 0,
               (struct sockaddr *)&to, sizeof(to)) < 0) {
        perror("sendto internet");
    }
}
```

上の例でパケットには正しい宛先アドレスが書かれているのにもかかわらず、struct sockaddr_in to へコピーしています。無駄に見えますが仕様のようで、sendto() で宛先を指定しなければいけません。

2.6 インターネットから VPN へ

サーバ側の IP マスカレードにより VPN の各クライアントはインターネットへパケットを送信することができるようになったわけですが、インターネットから帰ってきたパケットは勝手には VPN 内に入ってくれません。ルーティングして VPN 内に入れる必要があります。IP マスカレードによって 10.0.0.0/24 のプライベートアドレスに戻されたパケットが VPN サーバプログラムで操作できればいいわけですから、クライアントの時と同じように TUN デバイスを使います。つまり、eth0 から入って

[*5] iptables の動きについては https://www.atmarkit.co.jp/ait/articles/0505/17/news131.html がわかりやすいです。

きた 10.0.0.0/24 宛のパケットを TUN に向けるということです。これを IP フォワーディングといいます。あとはクライアントのときと同様に IP アドレスの設定、リンクアップ、ルーティングを行います。

```
system("ip addr add 10.0.0.1 dev tunserver");
system("ip link set tunserver up");
system("ip route add 10.0.0.0/24 dev tunserver");
```

ただし、Linux でこのようなことをするときはルーティングだけでなく、明示的に IP フォワーディングを許可する必要があります。

```
# /etc/sysctl.conf
net.ipv4.ip_forward=1
# sudo sysctl -p で反映させます。
```

ここでもファイアウォールが効いているので、eth0 に入ってきたパケットが、tunserver に行くのを許可しておきましょう。

```
$ sudo iptables -A FORWARD -i eth0 -o tunserver -d 10.0.0.0/24 -j ACCEPT
```

2.7 VPN サーバから VPN クライアントへ

サーバ側で TUN デバイスに入ったパケットはプログラムで読み取れるようになりますから、これをクライアントへ転送すればいいのです。

```
// インターネットからパケット受信
int datalen;
if ((datalen = read(tunfd, data, sizeof(data))) <= 0) {
    perror("read tunfd");
} else {
    // クライアントに送信
    if (sendto(sockfd, data, datalen, 0,
               (struct sockaddr *)&client_addr,
               sizeof(client_addr)) < 0) {
        perror("sendto client");
    }
}
```

2.8 VPN クライアントからアプリケーションへ

やっとのことでクライアントに到着したパケットは前述の通り TUN デバイスに write() することでパケットがインターネットから入ってきたように振る舞います。あとはカーネルがアプリケーションにデータを渡してくれます。

```
// サーバからデータ受信
int datalen;
if ((datalen = read(sockfd, data, sizeof(data))) <= 0) {
    perror("read sockfd");
} else {
    // アプリケーションに送信
    if (write(tunfd, data, datalen) < 0)
```

```
        perror("write tunfd");
}
```

2.9　動作確認

図 2.2 は、作った VPN を使って Youtube で 4K 動画を再生中に、iftop を eno1 で開いてみた様子です。確かに VPN サーバとしか通信してませんね。速度もそこそこ出ているようです。レイテンシも気になるほどではありませんでした。

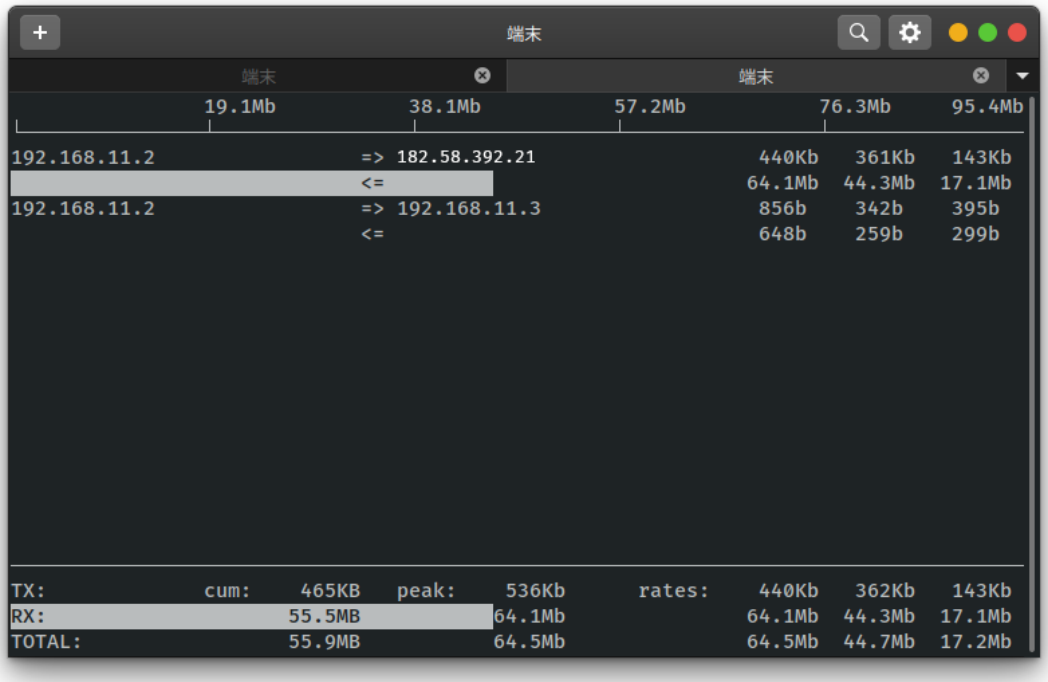


図 2.2　iftop の様子

これは VPN ありなしのときそれぞれの traceroute の結果です。住所がバレるので少し隠してありますが、家で契約してるプロバイダを経由していないことがわかります。

```
# VPNあり
$ sudo traceroute -i tunclient 1.1.1.1 -4 -I
traceroute to 1.1.1.1 (1.1.1.1), 30 hops max, 60 byte packets
 1  v182-58-392-21.fswu.static.cnode.io <- ConohaのVPS（VPNサーバ）
 2  unused-133-54-39-045.interq.or.jp <- Conohaが契約してるプロバイダ
 3  unused-133-54-39-001.interq.or.jp
 4  unused-133-54-39-082.interq.or.jp
 5  32.1.63.23
 6  one.one.one.one
```

```
# VPNなし
$ sudo traceroute -i eno1 1.1.1.1 -4 -I
traceroute to 1.1.1.1 (1.1.1.1), 30 hops max, 60 byte packets
 1  buffalo.setup <- 家のルータ
 2  tokyo00-n000.flets.iij.net <- プロバイダ入口
 3  TOKYO00-NTTeast1.flets.iij.net
 4  * * *
 5  * * *
 6  * * tky001ix11.IIJ.Net <- プロバイダ出口
 7  23.13.45.92
 8  one.one.one.one
```

2.10　おわりに

読んでいただきありがとうございます。この記事ではパケットの流れに沿って VPN を Linux と C 言語で自作しました。興味のある方は VPN サーバを構築してみるのはいかがでしょうか？

参考文献

[1] socket, JM Project, 2020/10/21,
https://linuxjm.osdn.jp/html/LDP_man-pages/man2/socket.2.html

[2] udp, JM Project, 2020/10/21,
https://linuxjm.osdn.jp/html/LDP_man-pages/man7/udp.7.html

[3] ioctl, JM Project, 2020/10/21,
https://linuxjm.osdn.jp/html/LDP_man-pages/man2/ioctl.2.html

[4] epoll, JM Project, 2020/10/21,
https://linuxjm.osdn.jp/html/LDP_man-pages/man7/epoll.7.html

[5] iptables, JM Project, 2020/10/21,
https://linuxjm.osdn.jp/html/iptables/man8/iptables.8.html

[6] getsockopt, JM Project, 2020/10/21,
https://linuxjm.osdn.jp/html/LDP_man-pages/man2/getsockopt.2.html

[7] Universal TUN/TAP device driver, 2020/10/21,
https://www.kernel.org/doc/Documentation/networking/tuntap.txt

[8] /proc/sys/net/netfilter/nf_conntrack_* Variables, 2020/10/21,
https://www.kernel.org/doc/Documentation/networking/nf_conntrack-sysctl.txt

[9] Cisco ASA シリーズファイアウォール CLI コンフィギュレーションガイド, 2020/10/21,
https://www.cisco.com/c/ja_jp/td/docs/sec/firewall/asa5500-xnext-generationf/cg/004/asa-firewall-cli/nat-overview.html

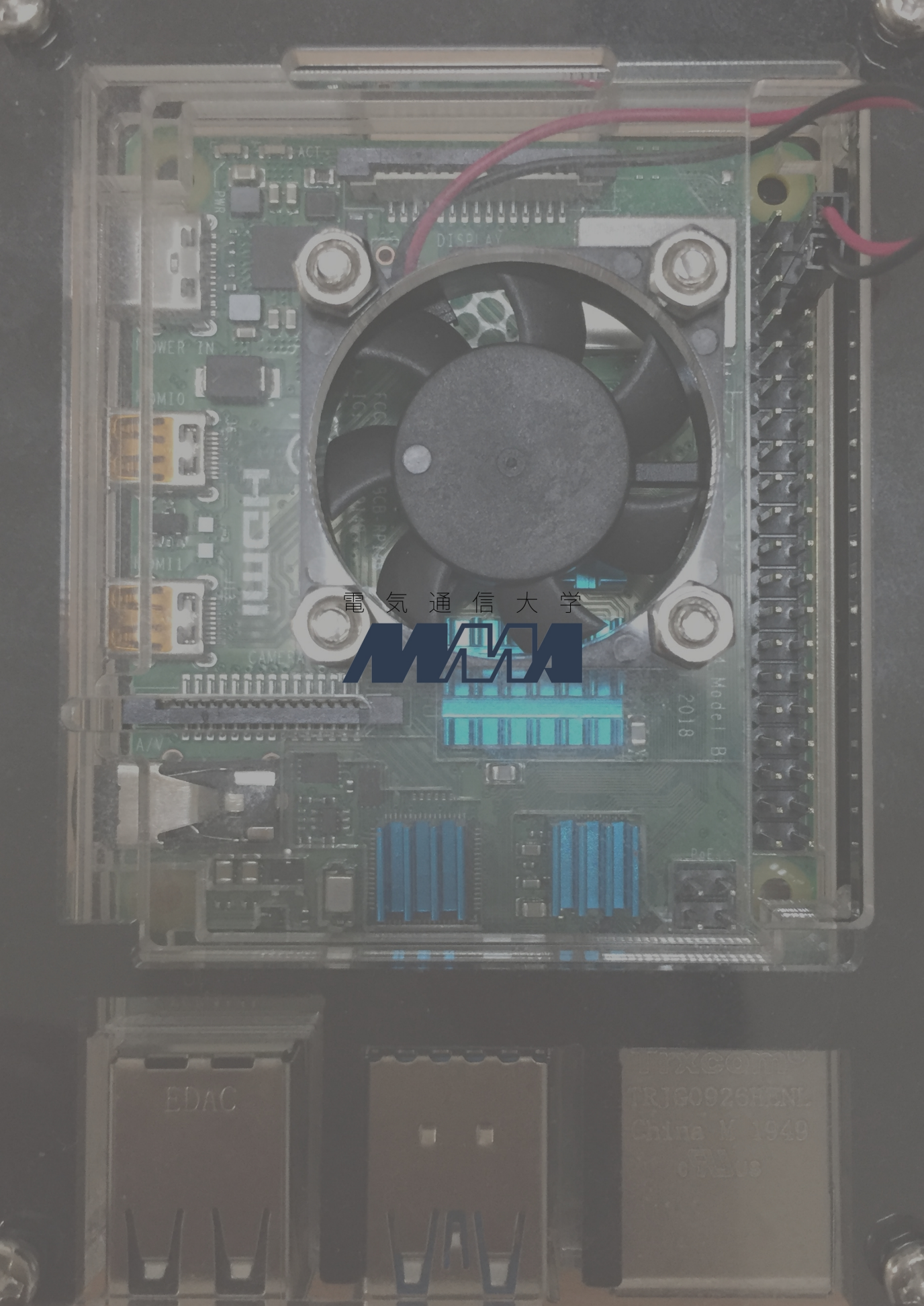
[10] Special-Use Domain Names, 2020/10/21, https://tools.ietf.org/html/rfc6761

[11] INTERNET PROTOCOL, 2020/10/21, https://tools.ietf.org/html/rfc791

百萬石 2020 -秋- © 電気通信大学 MMA

2020 年 10 月 30 日　初版第一刷発行　【本書の無断転載を禁ず】

著　者　ykm11, shosato
表　紙　torippy
編集者　torippy, shosato
発行者　電気通信大学 MMA
発行所　電気通信大学 MMA 部室
〒182−8585 東京都調布市調布ヶ丘 1−5−1 サークル会館 208 号室
https://www.mma.club.uec.ac.jp/
印刷所　電気通信大学 MMA 部室
製本所　電気通信大学 MMA 部室


電 気 通 信 大 学
EDAC
TXCOM
TRJ0926EENL
China M 1949