筑波大学大学院博士課程
システム情報工学研究科修士論文

複数のプロトコルに対応する高性能な VPN サーバの設計と実装

登 大遊
修士 (工学)
(コンピュータサイエンス専攻)

指導教員　新城 靖

2013 年 3 月

概 要

インターネットを用いて遠隔地にあるコンピュータ同士を接続する VPN には、多くのプロトコルが存在する。パソコンだけではなくスマートフォンやタブレット端末にも VPN 通信機能が搭載されているが、デバイスや OS の種類によって、利用することができる VPN プロトコルが異なる。また、たとえばファイアウォールの透過が簡単であるが通信スループットが出にくいといった VPN プロトコルがあるなど、VPN プロトコルごとに長所・短所が存在する。

そのため、多数のユーザによる VPN 接続を受付けたいと考えるネットワーク管理者は、複数の VPN プロトコルをサポートしたい場合が多い。しかし、広く利用されているすべての VPN プロトコルをすべてサポートする VPN サーバプログラムは未だ存在していなかった。そのため、VPN プロトコルごとに VPN サーバプログラムが必要である。いくつかの VPN プロトコルをサポートするためには複数の VPN サーバプログラムを組み合わせて運用する必要があるが、組み合わせによりオーバーヘッドが発生しスループットが低下したり、管理が複雑になったり、セキュリティ機能が利用できなくなったりするといった問題点がある。

本研究は、1 個の VPN サーバプログラムで広く利用されている VPN プロトコルをできるだけ多くサポートする VPN サーバの設計と実装を行う。これにより、従来の複数 VPN サーバを組み合わせる手法における問題点を解決することができる。具体的には、SoftEther VPN, L2TP over IPsec, SSTP, OpenVPN (L2 モード), OpenVPN (L3 モード), EtherIP over IPsec および L2TPv3 over IPsec をサポートするマルチプラットフォームの VPN サーバを設計し実装する。これにより、従来手法では実現できなかったユーザ管理などのセキュリティ設定の共通化、IP アドレスの管理の簡素化などの管理性の向上ができること、および従来手法と比較して 2.3% ～ 11.2%程度のスループット高速が実現できることを明らかにする。

目次

第 1 章 序論

第 1 章では、本研究の背景、目的および構成について述べる。

1.1. 背景

インターネットなどのパブリックなネットワークを経由してプライベートネットワークにリモート接続する手法として、仮想プライベートネットワーク (VPN) がある。

VPN には 2 種類の利用方法がある。1 種類目は、リモートアクセスである。リモートアクセス型の VPN を利用すれば、たとえば会社の LAN に自宅や外出先などのコンピュータを、物理的には離れているにもかかわらず、仮想的には直接接続している状態とすることができる。リモートアクセス型の VPN の接続元のコンピュータ (VPN クライアント) の OS の種類によって使用することが可能な VPN プロトコルが異なる。たとえば、Windows では L2TP[1]、SSTP[2] および SoftEther VPN[3] が利用できるが、Mac OS X ではこれらの中で L2TP しか利用できない。Linux ではディストリビューションによって利用可能な VPN プロトコルが異なるが、たとえば CentOS では L2TP が簡単に利用できず、代わりに OpenVPN[4] を利用する必要がある。近年普及しているスマートフォンやタブレット端末などの OS である Apple iOS や Android などは OS の制限によりカーネルモードで動作する必要がある VPN クライアントソフトウェアを追加することができない場合が多く、ビルトインの PPTP および L2TP しか使用することができない。

もう 1 種類の VPN の利用方法として、拠点間接続がある。拠点間接続型の VPN を利用すれば、たとえば本社の LAN に支店の LAN を接続し、すべての支店の LAN と本社の LAN とを仮想的に 1 つの LAN とすることができる。それぞれの拠点にあるコンピュータ同士は、特別な設定をしなくても相互に通信することができるようになる。拠点間接続型の VPN プロトコルも複数存在し、拠点間接続型の VPN を実現するためのネットワーク装置やサーバの機種や OS によって利用可能な VPN プロトコルが異なる。Windows や Linux を拠点間接続 VPN 装置として利用する場合は SoftEther VPN や OpenVPN が使用できる。Cisco 社の VPN 対応ルータは L2TPv3[5] over IPsec をサポートしている一方、同等の目的のための全く異なるプロトコルである EtherIP[6] over IPsec しかサポートさていない VPN 対応ルータもある。

また、それぞれの VPN プロトコルにはそれぞれ特徴がある。たとえば L2TP は多くのスマートフォンなどでサポートされているが、UDP を用いて通信を行うために UDP の通信を禁止しているファイアウォールがある場合は使用できない。SSTP や SoftEther VPN は TCP を用いて通信を行い、HTTPS のポート番号を宛先として VPN 接続を行うので、ほとんどのファイアウォールを通過できるが、利用可能な OS が少なく、スマートフォンでは利用できない。

そのため、多数のユーザによる遠隔地からの VPN 接続をサポートしたいと考える企業のネットワーク管理者は、できるだけ多くの利用者による多様なネットワーク環境からの接続を受付ける必要がある。ところが、上記で挙げたような VPN プロトコルをすべてサポートする VPN サーバプログラムはこれまで存在しなかった。そこでネットワーク管理者は、1 台または複数台のサーバコンピュータ上で複数の種類の VPN サーバプログラムを同時に起動する必要があった。この場合、異なる VPN プロトコルのクライアント間で通信が発生すると、複数の VPN サーバプログラム間でプロセス間通信が発生するためのオーバーヘッドが発生し、パフォーマンスが低下してしまう。また、それぞれの VPN サーバプログラムごとに認証やアクセス制御などの設定を行う必要があり、管理が難しくなる。さらに、VPN 接続のクライアント側にあるコンピュータに対して IP アドレスを動的に払い出す場合、ある VPN プロトコルによっては DHCP を利用できるが、別の VPN プロトコルのためには予め定義しておいた IP アドレスプールから割当てなければならないといった制約があり、IP アドレスの使用効率が低下する。

上記のような問題を解決するために、1 個のプログラムだけで上記に挙げたすべての VPN プロトコルをサポートすることにより、通信速度を従来のように複数の VPN サーバプログラムを組み合わせる場合よりも高速化し、管理性も向上するような VPN サーバプログラムがあれば望ましい。本研究ではそのような VPN サーバプログラムの設計・実装を行う。

1.2. 目的

本研究の目的は、以下のような要求を満たす VPN サーバプログラムを設計・実装することである。

サポート対象 VPN プロトコル

単一の VPN サーバプログラムのプロセスで、以下の 7 種類の VPN プロトコルの接続を受付け、同時に処理することができる。異なる種類の VPN プロトコルのクライアント同士でも通信を可能とする。

- SoftEther VPN
- L2TP over IPsec
- SSTP
- OpenVPN (L2 モード)
- OpenVPN (L3 モード)
- EtherIP over IPsec
- L2TPv3 over IPsec

ユーザ管理などのセキュリティ設定の共通化

すべての VPN プロトコルについて、単一のユーザ認証設定を適用することができる。たとえば、あるユーザを作成すると、そのユーザは SoftEther VPN でも L2TP でも OpenVPN でも接続できるようにすることが望ましい。またユーザごとのポリシーを設定して通信内容を規制する場合も、ポリシーを 1 回設定するだけですべての VPN プロトコルに対して適用されることが望ましい。

IP アドレスの管理の簡素化

VPN クライアントに対して動的に IP アドレスを割当てる場合は、VPN プロトコルの種類にかかわらず、共通の IP アドレス割当の仕組みを用いることが望ましい。VPN プロトコルによっては DHCP に対応しているものもあるが、DHCP に対応していない VPN プロトコルもある。DHCP 非対応の VPN プロトコルを用いた VPN クライアントが接続してきた場合であっても、DHCP へのリクエスト変換を行い、DHCP サーバを用いて IP アドレス割当を行えるようにすれば、IP アドレスの管理が簡素化できる。

VPN プロトコル間のレイヤの差異を吸収

VPN プロトコルの中には内部でレイヤ 2 (Ethernet) をカプセル化して VPN 通信の対象としているものもあれば、内部で PPP を用いるために、レイヤ 3 (IP) を VPN 通信の対象としているものもある。上記の各目的を実現するためには、このような VPN プロトコルのレイヤの差異を VPN サーバ側で吸収し、共通のポリシーを適用し、共通の IP アドレス管理の手法を利用可能としなければならない。そのために、VPN サーバ内ではすべての VPN プロトコルをいったん Ethernet レイヤにレイヤ変換する。つまり、Ethernet を VPN サーバ内における共通のバスとして採用する。

マルチプラットフォーム

実装する VPN サーバプログラムは、できるだけ多くの主要な OS 上で動作するのが望ましい。OS の違いに関わらずその差異を吸収し、同様に動作することが望ましい。

1.3. 構成

第 2 章では本研究の関連研究について述べる。VPN を実現するための手法やプロトコルについて、それぞれの特徴や問題点を述べる。

第 3 章では従来の手法を用いて複数 VPN プロトコルをサポートする場合の問題点について述べる。従来手

法では複数の VPN プロトコルを同時にサポートする VPN サーバを設置する場合、複数の VPN サーバプログラムを同時に動作させ、プログラム間でパケットの受渡しを行う必要がある。そのため、通信のオーバーヘッドが生じ、管理が複雑になり、事前に確保しておく必要がある IP アドレスプールの使用効率も低下している。また、VPN ユーザへのセキュリティポリシーの適用やユーザ間の通信グループの分離が困難である。これらの問題点を解決するためには、新たな VPN サーバプログラムの設計と実装が必要である。

第 4 章では本研究の提案手法を詳述し、実装に向けての設計を述べる。本研究で実装する VPN サーバプログラムは、1 個のプロセスで複数の VPN プロトコルをサポートする。多様な種類の VPN プロトコルの通信を受付け、各プロトコルに応じて適切な処理を行い最終的に共通のフォーマットとして Ethernet フレームを内部的に生成する。この Ethernet フレームを用いて複数の VPN セッション間のパケットのスイッチングを行う。そのため、レイヤ 3 (IP) を通信の対象とする VPN プロトコルであっても、本 VPN サーバプログラム内で透過的にレイヤ 2 (Ethernet) にレイヤ変換を行うため、仮想的な IP スタックを備える。複数の VPN セッション間で IP アドレス管理、セキュリティ機能、ユーザ認証機能および通信グループの分離機能を実現し、第 3 章で述べた従来手法の問題点の解決を試みる。

第 5 章では VPN サーバプログラムの実装について述べる。本研究では既存の SoftEther VPN Server プログラムを拡張して新しい VPN サーバプログラムを実装する。そこで、まず既存のプログラムの内部構造について説明し、本研究のためにプログラムをモジュール化して実装する方法について述べる。実装においてプログラムを容易にすることとパフォーマンスを向上させることを両立させる方法についても述べる。

第 6 章では従来手法と本研究における手法との VPN 通信性能を比較するため、実装した VPN サーバプログラムを用いて実験を行う。実験の方法としては、多数のテストユーザをインターネットで募集してプログラムをテストしてもらう方法と、実験環境を構築してスループット等の性能を測定する方法の 2 種類を行う。実験の結果としては、第 3 章で提示した問題点がすべて解決されていることを確認した。従来手法では解決することができなかったそれぞれの問題を解決することができ、パフォーマンスも向上したことを説明する。

第 7 章では本研究の結論をする。

第 2 章 関連研究

第 1 章では、本研究の背景、目的および構成について述べた。第 2 章では、本研究に関連するネットワークプロトコルや、プログラムを記述する上における従来手法について述べる。

2.1. 既存の LAN およびリモートアクセスプロトコル

2.1.1. Ethernet

Ethernet[7] は LAN 内で広く用いられているレイヤ 2 の通信プロトコルである。Ethernet ではフレームと呼ばれるパケットに、宛先 MAC アドレス、送信元 MAC アドレス、レイヤ 3 プロトコル番号および通信対象となるレイヤ 3 の通信プロトコルのパケット (IP など) を格納して送信する。単一の Ethernet のネットワークには、何台でもコンピュータを接続することができる。通常、Ethernet スイッチ、L2 スイッチまたはスイッチング HUB と呼ばれるデバイスをネットワーク上に配置する。それぞれのスイッチは MAC アドレス学習を行い、あるポートから受取ったフレームを宛先 MAC アドレスに従って適切なポートにのみ伝送する。MAC アドレスは、ネットワークに参加した物理的なコンピュータのネットワークインターフェイスを識別するためのものであり、物理アドレスとも呼ばれる。

ほぼすべての OS が Ethernet に対応している。一般的なコンピュータのほかにも、スマートフォンやタブレット端末などの小型デバイスでも有線または無線 (Wi-Fi) で Ethernet に接続することができる。

Ethernet は本来、物理的に隣接したコンピュータ同士を接続するために設計されたプロトコルである。そのため、単一のビル内やキャンパス内でネットワークを構築するために利用されることが多い。しかし、最近は Ethernet をそのまま WAN で利用するための広域イーサネットサービスも提供されている。また、Ethernet を TCP/IP 上にカプセル化して伝送する方式の VPN プロトコル (後述の SoftEther VPN、OpenVPN、Ether IP および L2TPv3) も存在する。これらの手法を用いると、離れた拠点から Ethernet のネットワークに参加することが可能である (図 1)。

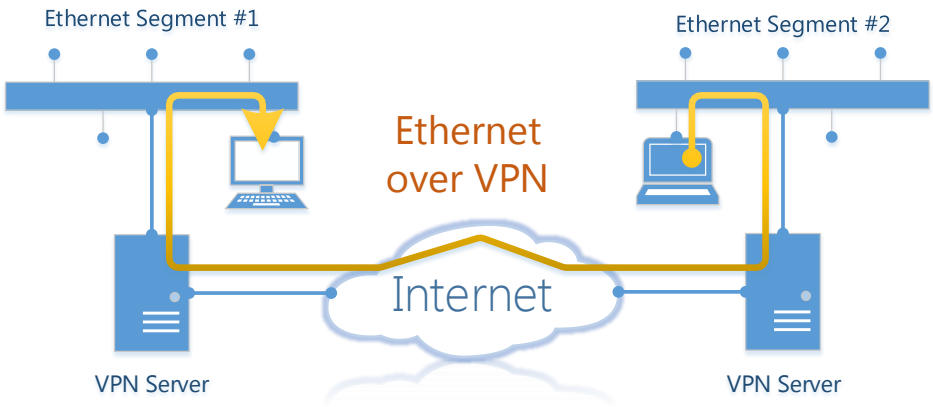


図 1. 離れた拠点同士の Ethernet セグメントを VPN で接続

2.1.2. PPP

電気通信回線を用いて 2 台のコンピュータ同士を接続するためのプロトコルとして、PPP (Point-to-Point Protocol)[8] がある。2 台のコンピュータの両方を音響カプラやモデム、ISDN インターフェイスを介して電気通信回線に接続する。一方を着信側、もう一方を発信側として回線を確立し、PPP のネゴシエーションが完了すると、互いに通信が可能になる。

PPP ではその名称通り 1 個の PPP ネットワークに 2 台のコンピュータしか参加することができない。これは、2 台のコンピュータをシリアルポートでクロス接続したり、Ethernet で 2 台のコンピュータをクロスケーブルで接続したりしているのと同等の状況である。そのため、各パケットを誰が受信すべきかを制御する必要はなく、Ethernet のような複数台のコンピュータを接続することを想定したプロトコルでは必須となるレイヤ 2 の物理アドレス (宛先、送

信元) は PPP パケット内には存在しない。PPP パケットは、ほぼレイヤ 3 (IP 等) のパケットそのままであり、IP 等のパケットを示すための 16 ビットのプロトコル番号が IP 等のパケットの直前に付加されているのみである。

1 個の PPP ネットワークには 2 台のコンピュータしか参加することができないが、そのコンピュータのうち一方または双方で IP ルータ機能を動作させれば、一方のコンピュータは相手先のコンピュータを経由してそのコンピュータの他のネットワークインターフェイスに接続している他のネットワークのコンピュータと通信することができる。この方法を活用し、社内の Ethernet で構成された LAN に、社員の自宅のコンピュータを、電話回線を介してリモートアクセスさせる目的のために PPP が広く利用されてきた。この場合、社内 LAN にも電話回線にも接続されているコンピュータが PPP のリモートアクセスゲートウェイとして使用される。1 台のコンピュータに何回線ものモデムを接続して同時に複数の社員からの PPP 接続をサポートする専用の組み込み型デバイスも開発された。

社内 LAN へのリモートアクセス手段として PPP を利用する場合、通常は PPP サーバと PPP クライアントとの間で、社内 LAN で使用されている IP サブネットとは重複しない別の IP サブネットを作成することになる。そして、PPP クライアントには社内 LAN のサブネットを宛先、PPP サーバをゲートウェイとするルーティングテーブルを設定し、同様に社内 LAN 上の各コンピュータには PPP クライアントを宛先、PPP サーバをゲートウェイとするルーティングテーブルを設定すれば、社内 LAN 内のコンピュータと PPP クライアントとは通信ができるようになる。

このように、PPP を用いて公衆網である電話回線や ISDN 回線などを用いて社内 LAN などのプライベートネットワークにアクセスすれば、公衆網を仮想的に社内 LAN の一部として使用することができる (図 2)。

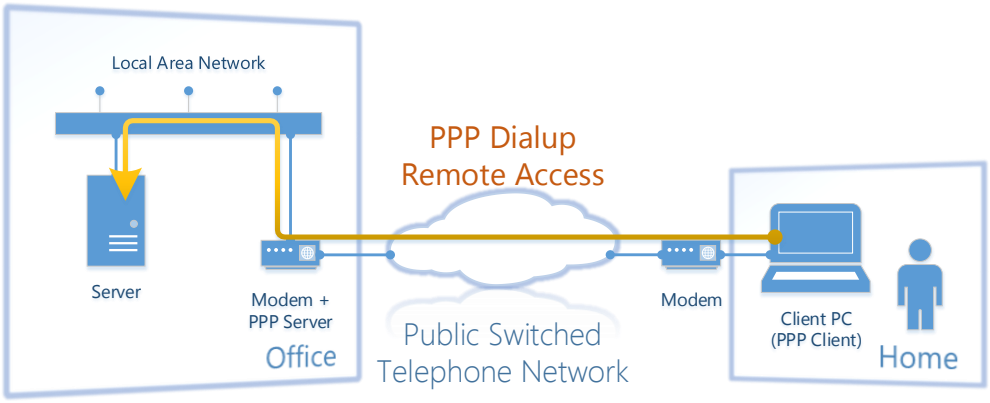


図 2. PPP ダイヤルアップ接続による LAN へのリモートアクセス

2.1.3. プロキシ ARP

PPP で前述のようにリモートアクセス型の VPN を実現する場合、通常はゲートウェイとなる PPP サーバが管理する PPP 側で使用する IP ネットワークと社内 LAN で使用される IP ネットワークとは別のサブネットに属することになる。たとえば社内 LAN では 192.168.10.0/24 を、リモートアクセス用には 192.168.20.0/24 を割当てるといった状態となる。この場合、社内 LAN 上のコンピュータとリモートアクセスのクライアントとが別々の IP ネットワークに属することになるため、管理が複雑になったり、クライアントサーバが同一の IP ネットワークに属していることを想定して開発されたプログラムが正しく動作しなかったりする原因になることがある。

そこで、上記の例で社内 LAN である 192.168.10.0/24 に、社内 LAN 上のコンピュータと同列に 192.168.10.0/24 上の IP アドレスのうち一部を PPP クライアントに割当てて利用させることができる技術としてプロキシ ARP[9] がある。プロキシ ARP を利用すれば、たとえば PPP クライアントが 192.168.10.100 というアドレスを使用している場合、PPP ゲートウェイは社内 LAN (Ethernet) 上のコンピュータからの 192.168.10.100 宛の ARP リクエストパケットに対して、自分自身が 192.168.10.100 というアドレスを持っているものとして ARP レスポンスパケットを送信する。すると、社内 LAN 上のコンピュータは 192.168.10.100 を宛先とする IP パケットを、当該 PPP ゲートウェイの MAC アドレスを宛先とする Ethernet フレームに格納して直接 Ethernet 上に送信することができるようになる (図 3)。これは社内 LAN 上のルーティングの設計が不要となり、LAN 上でのみ動作するプログラムをそのまま利用できるため、便利である。一部の PPP ゲートウェイソフトウェアにはプロキシ ARP が標準搭載されている。

プロキシ ARP を用いて PPP クライアントを社内 LAN の一部としてリモートから LAN に参加させる場合は、LAN 上で使用されている IP アドレスのうち一部をアドレスプールとして確保しておき、そのプールを PPP ゲートウェイで管理することが一般的である。たとえば、192.168.10.100 - 149 の合計 50 個の IP アドレスは PPP クライアント用として確保するという方法である。この場合、たとえば社内 LAN 上で DHCP を使用している場合はその DHCP サーバが PPP 用のプールとして確保されている範囲内の IP アドレスを DHCP クライアントに割当てないように設定するなどの注意を要する。

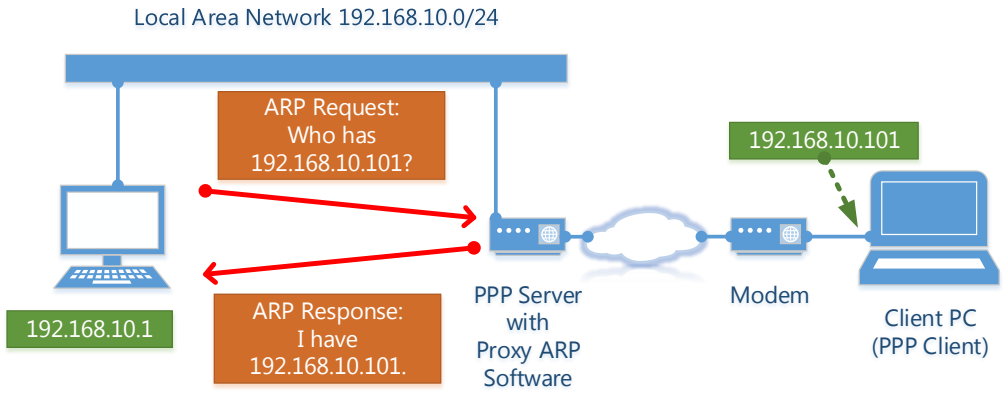


図 3. プロキシ ARP

2.1.4. L2TP

PPP は本来、電話回線上でのみしか利用できないプロトコルである。しかし、インターネット上で PPP を利用することができれば、インターネットを介して遠隔地から社内 LAN にリモートアクセス型の VPN を接続することができるため便利である。そのためには、PPP をインターネット上で利用できるプロトコルにカプセル化する必要がある。

PPP を UDP にカプセル化し、インターネット上で使用することができるようにするプロトコルとして、L2TP (Layer-2 Tunneling Protocol) がある (図 4)。L2TP では電話回線を用いた PPP と同様に、サーバとクライアントがある。L2TP サーバは UDP ポート 1701 で待ち受け、L2TP クライアントは任意のポートから L2TP サーバの当該ポートに対して L2TP ヘッダを含んだ UDP パケットを送信する。L2TP ではまず仮想的な電話回線を構築し、次にその仮想回線上で PPP リンクを確立する。一端 PPP リンクが確立されれば、電話回線を用いて PPP を使用する場合と同様に PPP 通信が可能になる。

L2TP は PPP を単に UDP にカプセル化したプロトコルであるため、暗号化機能を有していない。したがって、インターネット上で安全に通信を行うためには何らかの方法で L2TP の UDP パケットを暗号化することが望ましい。

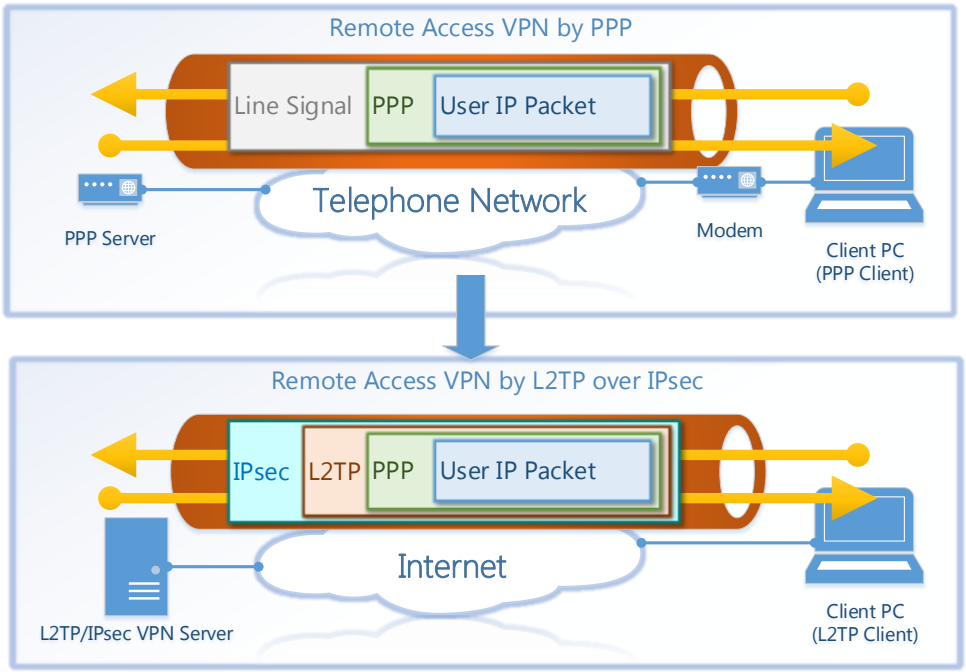


図 4. PPP をインターネット上で利用可能にした L2TP

2.1.5. IPsec

IPsec (Internet Protocol Security) [10] は、暗号化されていない IP パケットを暗号化し、インターネット上で安全に通信することを可能とするためのプロトコルである。IPsec は色々な規格から成るが、専ら ISAKMP (Internet Security Association and Key Management Protocol) あるいは IKE (Internet Key Exchange) というセッション確立と鍵交換のための制御用プロトコルと、ESP (Encapsulating Security Payload) というデータパケットの暗号化プロトコルとの 2 つを用いることが一般的である。

IPsec はまず IKE を用いてクライアント / サーバ間でワンタイムの鍵交換を行い、セッションを確立する[11]。確立されたセッションのことを SA (Security Association) と呼ぶ。SA はクライアント / サーバのメモリ上に暗号鍵とともに維持される (図 5)。

SA が存在する間は、一方からもう一方に対して送信されるデータは ESP を用いて暗号化される[12]。この際に SA として保持している暗号鍵を用いて暗号化を行う。暗号化されたパケットは ESP パケットとしてインターネットを経由してもう一端に届き、同様に暗号鍵を用いて復号化され、元のパケットに戻される。

IPsec は L2TP を暗号化してインターネット上で伝送するための暗号化レイヤとして、事実上の標準として使用されている。L2TP に対応したコンピュータ、スマートフォンやタブレット端末などでは、L2TP を使用しようとする場合は必ず IPsec が自動的に使用される仕組みになっているものが多い。このような利用方法を L2TP over IPsec と呼ぶ (図 4)。

IPsec は、IKE のために UDP ポート 500 を、ESP のために UDP ポート 4500 を使用する。ネットワーク上のルータ等のデバイスから見ると、IPsec の通信は単なる UDP の通信に見える。多くの NAT は IPsec を通過させることができるが、企業のファイアウォール等は UDP を遮断する設定となっていることがあり、この場合は IPsec を利用することができない。IPsec を利用することができない場合、ほとんどの OS では、L2TP を利用することができない。

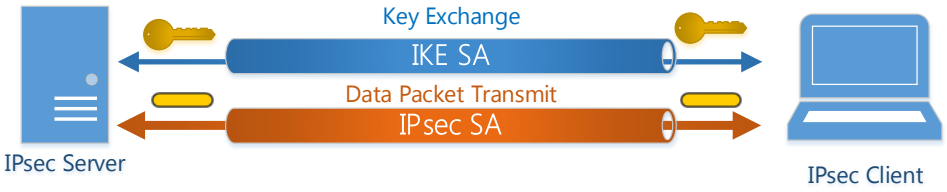


図 5. IPsec トンネルを構成する 2 種類の SA

2.1.6. SSTP

SSTP (Secure Tunneling Protocol) は、L2TP と同様に、PPP を IP パケットにカプセル化するプロトコルである[2]。L2TP を用いる場合は PPP を UDP にカプセル化し、さらにそれを IPsec によって暗号化して UDP で伝送する必要があるが、UDP を通過することができないファイアウォールの内側では使用することができない。また、HTTP プロキシサーバを経由して通信を行う必要がある環境でも使用することができない。

そこで、SSTP は代わりに TCP を用いて PPP をカプセル化する。この際に暗号化レイヤとして SSL (Secure Socket Layer) [13] を用いることにより、IPsec は不要となっている。そして、プロトコルの中身を検査する仕組みのファイアウォールによって遮断されることがないように、TCP の宛先ポート番号は 443 とし、Web サイトを閲覧するためのプロトコルである HTTPS (HTTP over SSL) と判別されにくいようになっている。このため、SSTP は L2TP over IPsec が使用できない環境でも使用できる (図 6)。

一方、SSTP で構築された VPN の内側で TCP を用いる場合は、TCP over TCP の形でデータが伝送されることになる。この場合、パケットロスが発生した場合にスループットが低下することが指摘されている[28]。また、SSTP は 2006 年に Microsoft 社によって提唱されたプロトコルであるため、対応しているクライアント側 OS は Windows 系しか存在せず、Mac OS X、スマートフォンおよびタブレット端末から利用することができない。

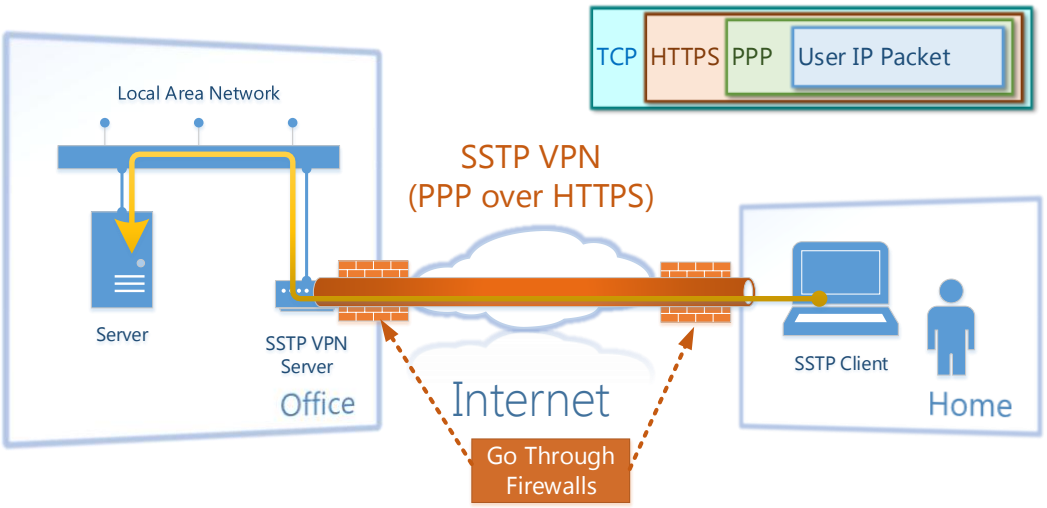


図 6. PPP over HTTPS によりファイアウォールを超えることができる SSTP

2.1.7. L2TPv3

PPP には前述のように、本来は 2 台のコンピュータ同士を接続するためのプロトコルである。PPP では、遠隔地のコンピュータを社内 LAN に透過的にリモートアクセスさせるために利用することはプロキシ ARP と組み合わせることにより一応可能となったが、2 箇所以上の LAN を透過的に VPN 接続することは困難である。2 箇所以上の LAN 同士を PPP リンクで接続し、それぞれの PPP ゲートウェイで、遠隔拠点で使用されている MAC アドレスの一覧を覚えておきそれらの MAC アドレスに対する ARP リクエストに対して応答する ARP プロキシを動作させれば可能であるが、管理が複雑になる。

そこで、PPP の代わりに Ethernet を直接カプセル化する L2TP の拡張プロトコルとして L2TPv3 が設計された[5]。L2TPv3 を使用すれば、ある Ethernet の物理的な LAN から発信される Ethernet フレームを、そのまま UDP にカプセル化し、別の拠点に伝送することができる。カプセル化された UDP パケットを受取った遠隔地では、UDP の中から Ethernet フレームを取り出し、これを物理的に LAN に対して発信する。

L2TPv3 により、あたかも仮想的な Ethernet 専用線を 2 拠点間に接続する場合と同様の通信を、インターネットを介して実現することができる。ただし、L2TPv3 は L2TP 同様に単純なカプセル化のプロトコルであるため、インターネット上で利用するためには暗号化のために IPsec と組み合わせて利用する必要がある (図 7)。

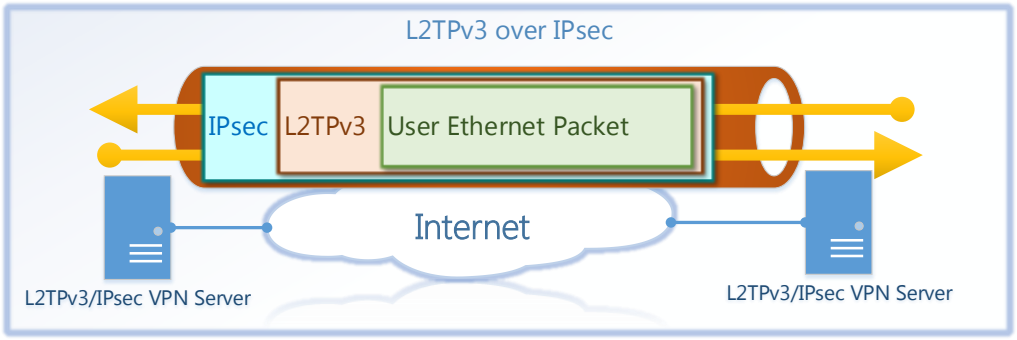


図 7. Ethernet フレームを伝送できる VPN プロトコルの 1 つである L2TPv3

2.1.8. EtherIP

L2TPv3 と類似のプロトコルに EtherIP (Ethernet over IP) がある[6]。EtherIP も Ethernet フレームを IP パケットにカプセル化して伝送するプロトコルであるが、L2TPv3 と異なり、セッション管理を行わないより単純なプロトコルである (図 8)。

EtherIP をインターネット上で利用するためには、L2TPv3 と同様に、暗号化のための IPsec と組み合わせる必要がある。

L2TPv3 と EtherIP は近い時期に別々に提唱されたプロトコルであるため、L2TPv3 に対応している VPN ルータは EtherIP に対応しておらず、EtherIP に対応している VPN ルータは L2TPv3 に対応していないといった状況である。

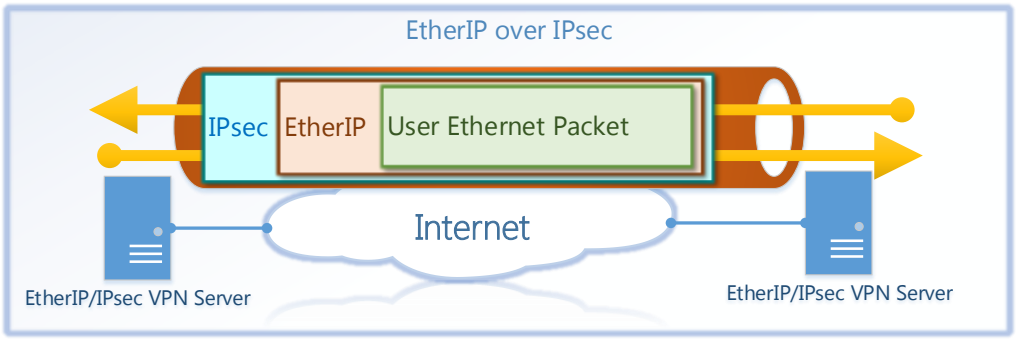


図 8. L2TPv3 と類似した VPN プロトコルである EtherIP

2.1.9. OpenVPN

OpenVPN はオープンソースの VPN ソフトウェアおよびそのプロトコルである (図 9) [4]。OpenVPN には L3 モードと L2 モードの 2 種類の動作モードがある。L3 モードと L2 モードを同時に動作させることはできず、どちらか一方のみを指定する必要があり、また異なるモード同士のサーバ / クライアント間は通信を行うことができないため、利用上は L3 モードと L2 モードの OpenVPN は別々のソフトウェアであると考えるべきである。

L3 モードの OpenVPN は、L2TP over IPsec とほぼ同様に、IP パケットを UDP にカプセル化して暗号化し伝送する。

L2 モードの OpenVPN は、L2TPv3 over IPsec または EtherIP over IPsec と同様に、Ethernet フレームを UDP にカプセル化し伝送する。

L3 モード、L2 モードのいずれも暗号化には IPsec によく似た仕組みが実装されているが、IPsec は使用されておらず、IPsec を併用する必要がない。また、SSTP と同様に、UDP の代わりに TCP にカプセル化することもできる。これにより UDP パケットを通過しないようなファイアウォールの内側からであっても利用することができる。しかし SSTP と異なり、通信は SSL セッションではなく独自のプロトコルが用いられている (独自のプロトコルのペイロードとして SSL が部分的に使用されている) ため、TCP コネクションのペイロードを検査することで例え宛先ポートが 443 であってもペイロードが SSL 以外の通信を遮断するようなファイアウォールを通過することはできない。

OpenVPN のクライアントプログラムは Windows、Linux および Mac OS X で利用できるが、システムソフトウェアを動作させることができないスマートフォンやタブレット端末では利用できない。

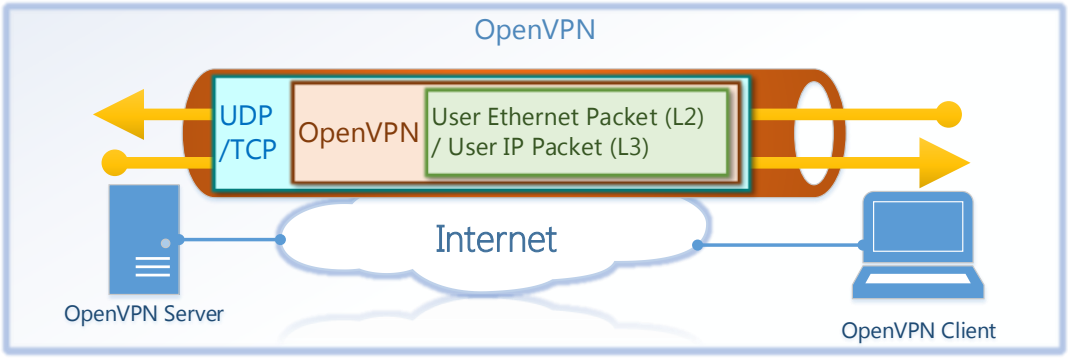


図 9. UDP の他に TCP を用いた伝送にも対応し L2, L3 の 2 モードを有する OpenVPN

2.1.10. SoftEther VPN

SoftEther VPN はレイヤ 2 の VPN ソフトウェアおよびそのプロトコルである (図 10) [3]。L2TPv3 や EtherIP のように、Ethernet フレームをカプセル化して伝送する。伝送には TCP コネクションを使用する。この際は SSTP と同様に SSL を用いるため、ペイロードが SSL 以外の通信を遮断するようなファイアウォールを通過することができる。

SSTP は 1 本の論理的な VPN セッションを 1 本の TCP コネクションで構築するが、SoftEther VPN は最大 32 本の TCP コネクションを確立してパケットをそれらの TCP コネクションに分散して伝送することができる。この手法により、ある TCP コネクションで遅延やパケットロスが発生した場合でも、別の TCP コネクションで素早く伝送を行うことができるため、スループットが向上している。

また、SoftEther VPN のプロトコルバージョン 4.0 においては、VPN サーバと VPN クライアントとの間で TCP のほかに UDP の伝送が可能であることが検出されれば、TCP の代わりに UDP にカプセル化して伝送するよう拡張されている。

SoftEther VPN は Ethernet フレームをカプセル化して伝送するため、拠点間接続型の VPN を構築することができるが、L2TP 等のレイヤ 3 の VPN プロトコルと同様にリモートアクセス型の VPN を構築することも可能である。

SoftEther VPN のクライアントプログラムは Windows および Linux で利用できるが、Mac OS X やスマートフォン、タブレット端末では利用できない。

SoftEther VPN は製品版が「PacketiX VPN」として市販されており、オープンソース版が「UT-VPN」として配布されている。

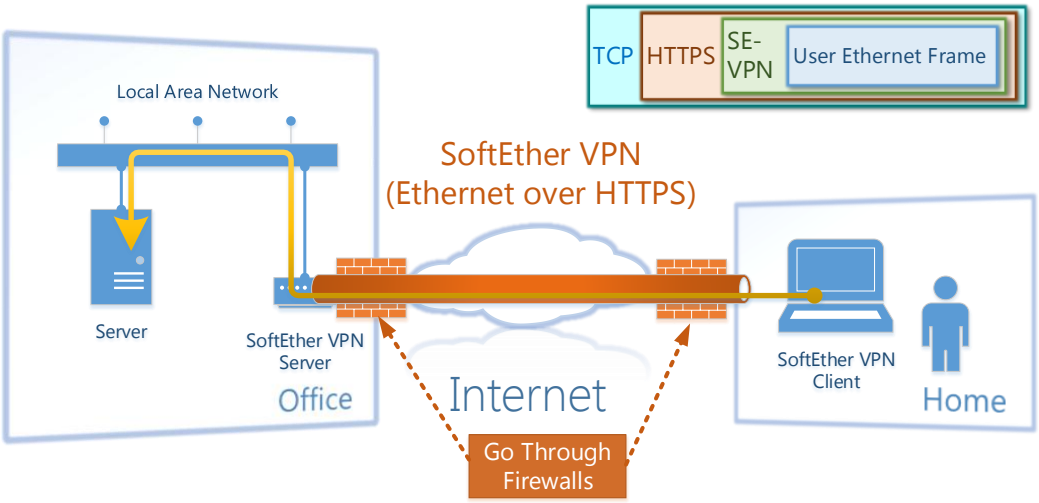


図 10. Ethernet フレームを HTTPS に乗せて VPN 通信する SoftEther VPN

2.2. その他の研究

2.2.1. The Click modular router

Click モジュール型ルータ[14] は、IP ルーティングに必要な各種処理を分解して “Elements” と呼ばれる各モジュールに実装し、それらのモジュール間におけるパケットの流れやステートの共有などのワークフローを専用言語で記述することが可能なルータの設計手法に関する研究である。通常モノリシックなプログラムとして各モジュール間が密に結合され実装されている IP ルータと異なり、Click ルータのモジュール間の結合は疎であるため、オリジナルなルータを実装したいと考えるユーザは簡単に希望する設計のルータを作成することができる。Click ルータのルーティング速度は、同等の処理を通常の Linux カーネルのルーティング速度と比較してちょうど 10%程度の速度低下が見られる程度であると報告されている。

本研究で実装する VPN サーバプログラムの実装においては、Click モジュール型ルータと同様に、パケットの処理の単位ごとにモジュールを作成してそのモジュール間でパケットの受渡しを行うことにより、複数のプロトコルが組み合わさっている複雑な VPN プロトコルに対する処理を行うことができるようにする。ただし、Click モジュール型ルータではユーザによるモジュール間の動的な結合や初期化パラメータを指定するための仕組みがあり、また各モジュール間における CPU のスケジューリングも独自に実装しているが、本研究の実装においてはモジュール間の結合はコンパイル時に定まる静的なものとし、CPU スケジューリングは OS の機能を利用する。

2.2.2. Apache Portable Runtime

Apache Portable Runtime (APR)[15] は、HTTP サーバである Apache を Windows、Linux、各種 UNIX などの異なる OS 上で移植可能にプログラミングするために用意された、OS の機能を抽象化し差異を吸収する C 言語用のライブラリである。OS によってマルチスレッド関係の API やソケット通信関係の API を代表とする差異があるが、APR を使用すると開発者はこれらの差異を意識することなる移植性の高いソースコードを記述することができる。本研究で実装する OS 抽象化レイヤは APR とよく似たモジュールであるが、APR では提供されていない Raw ソケット (UDP を用いない IPsec ESP の送受信に使用) や Etherner フレームソケット (OS にインストールされている LAN カードを用いて Ethernet フレームを送受信するために使用) をサポートする。

2.2.3. BSD mbuf

mbuf[29]は主に BSD OS のネットワークスタック内でパケットバッファを効率良く扱うために利用されているデー

タ構造である。

あるユーザデータをネットワークで送受信する場合、送信したいデータ本体 (ペイロード) に対して複数のヘッダを順番に付けてから送信する (受信側では逆にヘッダを順番に取り外す) 処理が必要になる。各ヘッダの追加、削除を行う際に毎回新しいメモリ領域を確保してデータをコピーするとメモリの消費量が増え、オーバーヘッドも増加してしまう。mbuf は任意のデータの直前、中間および直後に任意の別のデータを連結させることができる。この場合、連結されたデータは実際にはメモリブロック上では連続していないが、mbuf の提供するユーティリティ関数によって連続したデータを読み出すことが可能である。また、TCP を用いてデータを送信する場合は送信対象のデータを分割して送信キューに入れておき、相手方に正しく到達したことが確認されるまでこれを保持することとなる。逆に受信側においては分割されて届いたデータ (届く順番は保証されていない) を、受信キューの先頭からある一定サイズ分の連続したデータが揃うまでメモリ上に保持しておく必要がある。このような処理を TCP セグメンテーションと呼ぶ。TCP セグメンテーションの効率的な実現のためにも mbuf が使用されている。

mbuf を参考にして開発されたものとして、Linux カーネル内の sk_buff[30]および Windows Vista 以降のカーネル内の NET_BUFFER[31]がある。sk_buff は mbuf とよく似ているが、Ethernet フレームがコンピュータの LAN カードによって受信された時点における正確なタイムスタンプを保持するなどの機能拡張が行われている。mbuf や sk_buff では構造体のメンバとしては保持するデータの仮想メモリ上のポインタ (通常は非ページメモリ領域内のポインタ) が格納されているが、これと比較して、NET_BUFFER では直接ポインタを格納せず、MDL (Memory Descriptor List) という物理メモリと仮想メモリの両方のアドレスを含んだマッピングテーブルへのポインタが格納されている。

本研究において実装する VPN サーバプログラムの VPN プロトコルモジュールは、複数のプロトコルレイヤを順番に経由してパケットが処理されるため、mbuf や sk_buff、NET_BUFFER の一部で用いられている手法を用いて高速化を図る。

第 3 章 複数の VPN プロトコルをサポートする際の問題点

第 2 章では、本研究に関連するネットワークプロトコルや、プログラムを記述する上における従来手法について述べた。第 3 章では、既存の複数の VPN プロトコルを比較し、複数の VPN プロトコルを同時にサポートするためには従来の複数の VPN サーバプログラムが必要であることを述べる。そして、複数の VPN サーバプログラムを組み合わせて使用する場合に生じる問題点について述べる。

3.1. VPN プロトコルの比較

VPN プロトコルには第 2 章で述べたように色々な種類があり、それぞれ長所、短所が存在する。それらをまとめるとのようになる。

表 1. VPN プロトコルの比較

	L2TP/IPsec	SSTP	PPTP	OpenVPN (L3 / L2)	L2TPv3/IPsec	EtherIP/IPsec	SoftEther VPN
カプセル化対象レイヤ	L3 (IP)	L3 (IP)	L3 (IP)	L3 (IP) / L2 (Ethernet)	L2 (Ethernet)	L2 (Ethernet)	L2 (Ethernet)
伝送可能プロトコル	IP のみ	IP のみ	IP のみ	IP のみ	すべての Ethernet 対応プロトコル	すべての Ethernet 対応プロトコル	すべての Ethernet 対応プロトコル
物理伝送プロトコル	IPsec (UDP)	HTTP over SSL (TCP)	GRE	独自プロトコル (TCP, UDP)	IPsec (UDP)	IPsec (UDP)	HTTP over SSL (TCP, UDP)
使用ポート	UDP 500/4500	TCP 443	TCP 1723	TCP 1194 UDP 1194	UDP 500/4500	UDP 500/4500	TCP 443 UDP 動的
クライアント PC への IP 割当	PPP IPCP	PPP IPCP	PPP IPCP	独自プロトコル	DHCP	DHCP	DHCP
HTTP プロキシ通過	×	○	×	○	×	×	○
厳しいファイアウォール通過*	×	○	×	×	×	×	○
VPN サーバ用および拠点間 VPN 接続用対応デバイス	Windows Linux Mac OS X Cisco ルータ等	Windows のみ	Windows Linux Mac OS X Cisco ルータ等	Windows Linux Mac OS X	Cisco ルータ SEIL ルータ	FreeBSD NEC ルータ	Windows Linux Mac OS X FreeBSD Solaris
リモートアクセスクライアント側対応デバイス	Windows Linux Mac OS X iPhone, iPad Android	Windows のみ	Windows Linux Mac OS X iPhone, iPad Android	Windows Linux Mac OS X Android	×	FreeBSD	Windows Linux
スループット測定結果†	560Mbps	1,105Mbps	―	90Mbps / 86Mbps	―	―	1,104Mbps

* 「厳しいファイアウォール」とは、TCP/IP パケットのヘッダに記載されているプロトコル番号やポート番号などを検査し許可されている値を含むパケット以外をフィルタするような単純な処理だけではなく、ペイロードに含まれる通信内容までも検査することで、ポート番号を変更して通信を行おうとする通信を遮断することができる能力を有するファイアウォールである。

† Intel Xeon E3-1230 3.2GHz, Windows Server 2008 R2 (x64) で測定。詳細は第 6 章および付録を参照。

ある VPN プロトコルは多くのクライアントデバイスから使用することができるがファイアウォールを通過しにくく、別の VPN プロトコルはファイアウォールを通過しやいすが対応していないクライアントデバイスが存在するといった状況になっていることがわかる。

そのため、ネットワーク管理者は多種多様なクライアントデバイスやネットワーク環境をすべてサポートするためには複数の VPN プロトコルをサポートする必要がある。これ 1 つだけで良いという VPN プロトコルは存在しない。

3.2. VPN サーバプログラムの比較

VPN を構築するためには、VPN サーバプログラムが必要である。代表的な VPN サーバプログラムが対応している VPN プロトコルをまとめるとのようになる。

表 2. 代表的な VPN サーバプログラムと対応している VPN プロトコル

	L2TP/IPsec	SSTP	PPTP	OpenVPN	L2TPv3/IPsec	EtherIP/IPsec	SoftEther VPN
Microsoft Routing and Remote Access Service (Windows Server に付属)	○	○	○	×	×	×	×
Mac OS X Server	○	×	○	×	×	×	×
OpenVPN	×	×	×	○	×	×	×
Cisco IOS	○	×	○	×	○	×	×
NEC IX Router OS	×	×	×	×	×	○	×
IIJ SEIL Router OS	○	×	○	×	○	×	×
PacketiX VPN	×	×	×	×	×	×	○
UT-VPN	×	×	×	×	×	×	○

現在は VPN サーバとして上記のような様々なプログラムが存在しているが、それぞれ対応している VPN プロトコルが異なり、1 つの VPN サーバプログラムですべての VPN プロトコルに対応しているものは 1 つも存在しない。

3.1 節で述べたように、これ 1 つだけをサポートすれば良いという VPN プロトコルは存在しないため、ネットワーク管理者は 2 つ以上の VPN プロトコルをサポートする必要がある。そのため、複数の VPN プロトコルをサポートする場合は、複数の VPN サーバプログラムを同時に使用する必要が生じる。

3.3. 複数の VPN サーバプログラムの組み合わせ

従来手法では、2 つ以上の異なる VPN プロトコルを用いる VPN クライアントデバイスからの接続を同時にサポートする必要がある場合、それぞれの VPN プロトコルをサポートする 2 台以上の VPN サーバを別々に構築して同一のネットワークに接続するか、2 つ以上の VPN サーバプログラムを組み合わせて 1 台の VPN サーバを構築するかのいずれかの手法を選択する必要がある。

たとえば、2 台の VPN クライアントがあり、1 台目は Windows であるため SSTP で、2 台目は Mac OS X であるため OpenVPN (L3) で、1 台の VPN サーバに同時に接続したい状況を想定する。この場合、2 種類の両方の VPN プロトコルをサポートする理想的な VPN サーバプログラム (図 11) は存在しないため、2 種類の VPN サーバプログラムを同時に使用する必要がある。

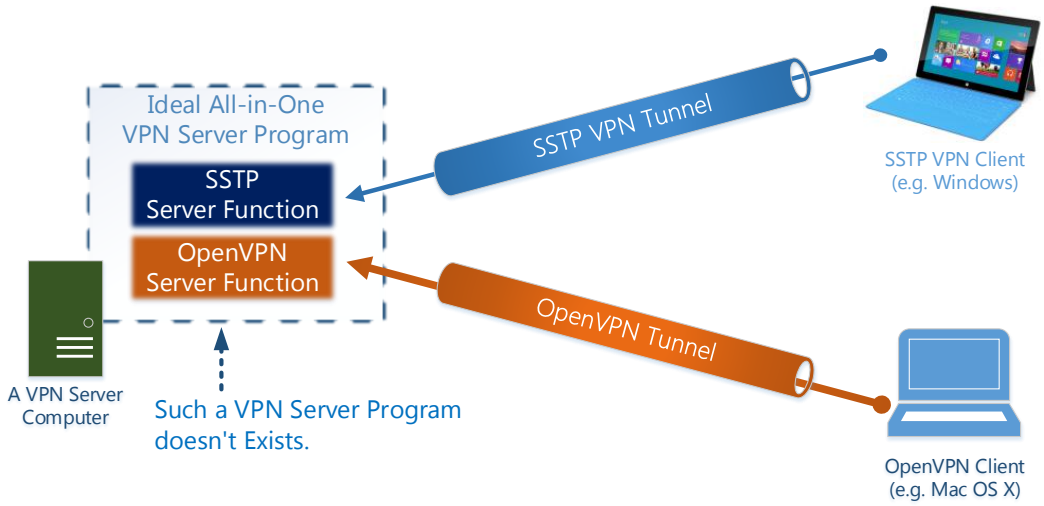


図 11. 複数の VPN プロトコルに対応した理想的な VPN サーバプログラム

2 種類の VPN サーバプログラムをそれぞれ 1 種類ずつ、合計 2 台の VPN サーバで動作させることが可能である。たとえば 1 台目のサーバコンピュータには SSTP のために Windowxs Server 2008 R2 上で動作する Microsoft Routing and Remote Access Service (RRAS) をインストールする。もう 1 台のサーバコンピュータには OpenVPN のために OpenVPN Server プログラムをインストールする。このようにすれば、SSTP クライアントは 1 台目の、OpenVPN クライアントは 2 台目の VPN サーバコンピュータにそれぞれ接続することができる。2 台の VPN サーバコンピュータが両方とも社内 LAN に接続されていれば、それぞれの VPN サーバに接続された VPN クライアントはいずれも社内 LAN にアクセスでき、また、VPN クライアント同士も相互に通信できる。

ただし、2 台以上の VPN サーバコンピュータを使用する手法では、物理的にコンピュータの台数が複数台必要になり、コンピュータ本体のコスト、管理コスト、それぞれのコンピュータ用に VPN 接続を待ち受けるための IP アドレスのコストが必要となる。

上記における 2 台の VPN サーバコンピュータの役割を 1 台の VPN サーバコンピュータに集約することが可能な場合がある。1 台の VPN サーバコンピュータ上で複数の VPN サーバプログラムを同時に動作させることにより、サーバの台数が節約でき、待ち受け用 IP アドレスも 1 個で足りるため、コストを低くすることができる。例えば Windows Server 2008 R2 上では Microsoft RRAS も OpenVPN Server も同時にインストールすることができ、互いに競合しないため、同時に起動することができる。そして、VPN サーバコンピュータ内の IP ルーティングを適切に設定することにより、2 台の VPN サーバコンピュータに分離する方式と同様の構成を 1 台の VPN サーバコンピュータで実現することができる (図 12)。

ただし、1 台のサーバコンピュータ上で同時に動作させることができない 2 つ以上の VPN プロトコルを同時に使用することはできない。たとえば、SSTP サーバの動作には Windows Server が必要であるが、L2TPv3 の動作には Cisco IOS 等のルータ専用 OS が必要である。この場合は必ず役割を 2 台以上の VPN サーバコンピュータ (1 台は一般的なコンピュータ、もう 1 台はルータ専用ハードウェア) に分離する必要がある。

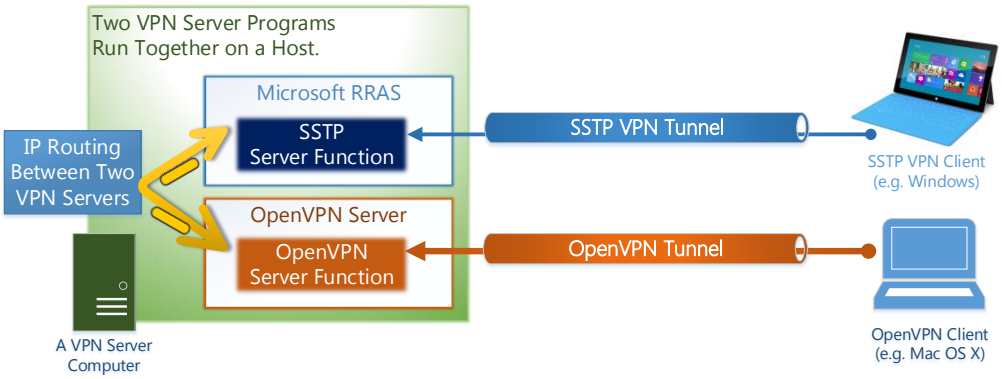


図 12. 2 個の VPN サーバプログラムを 1 台のコンピュータで動作させて OS 内部でリンクする方法

3.4. 通信のオーバーヘッドの発生

1 台の VPN サーバコンピュータ上で 2 個の異なる VPN サーバプログラムを同時に動作させ、2 種類の VPN クライアントからの VPN セッションを同時に処理する場合、当該 VPN クライアント間の通信には、プログラムの境界を越える際にオーバーヘッドが発生する (図 13)。

1 個の VPN サーバプログラムに 2 台の VPN クライアントが接続し、VPN クライアント間で通信が行われる場合、2 本の VPN セッション間のパケットデータの受け渡しは同一の VPN サーバプログラムのプロセス内で行われる。これはプロセス内の同一の仮想メモリ空間に属する単純なメモリコピーである。メモリコピーの回数は、VPN サーバプログラムのコードを工夫することにより削減することも可能である。

一方、2 個の VPN サーバプログラムにそれぞれ 1 台ずつ VPN クライアントが接続し、VPN クライアント間で通信が行われる場合、2 本の VPN セッション間のパケットデータの受け渡しのためには、OS の提供するルーティング機能またはブリッジング機能によって仲介される。この際、一方の VPN サーバプログラムが VPN クライアントから受取った IP パケット、または Ethernet フレームを、仮想的な L2 デバイス (tap または仮想 LAN カード) または L3 デバイス (tun または ppp アダプタ) を介して書き込み、OS に渡すことになる。OS は仮想 L2 デバイスまたは L3 デバイスからパケットを受け取り、カーネル内の IP ルーティングまたは Ethernet ブリッジプログラムによってもう一方の VPN サーバプログラムに渡すことになる。このような複雑な処理にはオーバーヘッドが発生する。プロセスと OS との間の境界を越える際にメモリコピーが必ず発生する。また、OS に対してパケットを渡す際に必ず特権の使用が必要である。そして、受け取り側のプロセスは OS からの新しいパケットの到着を待機するために非同期のイベント待ちを行うが、そのような同期機構を使用するためのオーバーヘッドも発生する。

したがって、2 つ以上の異なる VPN サーバプログラムを 1 台の VPN サーバコンピュータ内で動作させることは、通信パフォーマンスを考慮すると最適ではない。通信パフォーマンスが低下せず、かつ 1 台の VPN サーバコンピュータで複数の VPN プロトコルをサポートする新たな VPN システムが必要である。

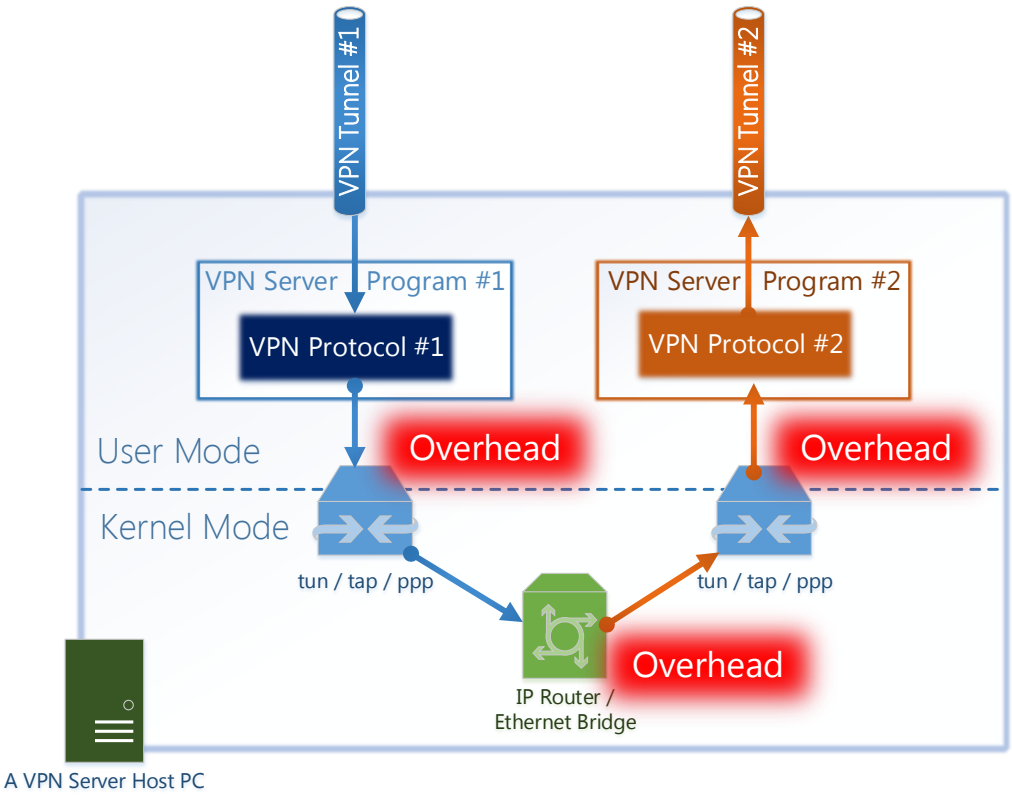


図 13. 2 個の別々の VPN サーバプログラム間の通信はオーバーヘッドが発生する

3.5. VPN 通信グループ間の分離が不能

同一の VPN プロトコルのみを使用する場合は、管理者は通信グループを定義し、同一の通信グループに所属している VPN クライアント間でのみ通信を許可することができる。たとえば、1 台の VPN サーバコンピュータ内で 2 個の OpenVPN サーバプログラムを同時に起動し、4 台の OpenVPN クライアントのうち 1 台目と 2 台目が OpenVPN サーバ #1 に、3 台目と 4 台目が OpenVPN サーバ #2 に VPN 接続することができる。この場合、1 台目と 2 台目は相互に通信でき、3 台目と 4 台目も相互に通信できる。1 台目と 2 台目、3 台目と 4 台目はそれぞれ通信グループを構成している。異なる通信グループ間の通信は一切できない。これにより、たとえば 1 台の VPN サーバコンピュータで、2 社以上の複数の会社に対して互いに分離され通信が行えないことが保証される形の VPN サーバ機能を提供することができる。

一方、1 台の VPN サーバコンピュータで OpenVPN プロトコルと L2TP/IPsec プロトコルの 2 種類のプロトコルを同時にサポートするために OpenVPN サーバプログラムと L2TP/IPsec サーバプログラムとを同時に動作させ、2 台の OpenVPN クライアントが OpenVPN サーバに、2 台の L2TP/IPsec クライアントが L2TP/IPsec サーバにそれぞれ接続する状態を考える。このとき、OpenVPN クライアント#1 と L2TP/IPsec クライアント#1 で通信グループ#1 を、OpenVPN クライアント#2 と L2TP/IPsec クライアント#2 で通信グループ#2 を構成し、通信グループ間の通信は禁止したい場合がある。しかし、3.4 で述べたように、OpenVPN サーバプログラムと L2TP/IPsec サーバとの間のパケットの受け渡しには OS の提供する IP ルーティング機能が必要である。Windows や Linux などの標準的な OS の IP ルーティング機能は単一のインスタンスしかない。つまり、OS 内で複数の IP ルータを仮想的に作成することができない。そのため、2 個の OpenVPN サーバプログラムと、2 個の L2TP/IPsec サーバプログラムとは、ただ 1 つの IP ルーティング機能を使用してリンクされることになる。すると、異なる通信グループ間で IP 通信ができてしまい、セキュリティを維持することができない。

この場合、OS のパケットフィルタ機能 (iptables 等) を用いて、2 個の OpenVPN サーバと 2 個の L2TP/IPsec サーバ合計 4 個の間の仮想 tun インターフェイス (または ppp インターフェイス) 間のパケットフィルタリングを行

い、異なる通信グループ間の IP 通信を禁止することは可能である。しかし、2 つの通信グループでそれぞれ使用したい IP サブネットが重複した場合にはこれを解決する方法がない。OS が提供する IP ルーティング機能は 1 つだけであり、それぞれの仮想 L3 インターフェイス間で重複する IP ネットワークが存在する場合には正しく動作しない。

このように、複数の VPN サーバプログラムを組み合わせて利用する場合、VPN クライアントを複数の通信グループに分離して互いに通信を不可能にすることは、現在の OS の機能では難しい。仮想マシン (VM) を用いて 1 台のコンピュータ上に複数個の OS を動作させれば、OS の数だけ IP ルーティング機能のインスタンスを動作させることができるが、通信グループの数だけ VM の作成と OS の起動が必要となりメモリを消費し、また一層複雑な管理が必要となる。

1 台の VPN サーバコンピュータ上で違いに分離された複数の通信グループを作成する場合において、通信グループの数が大量となった場合でも、管理が簡単で消費するリソースの量が少ない新たな VPN システムが必要である。

3.6. ユーザ認証、パケットフィルタ設定、ポリシー設定の複雑化

複数の異なる VPN サーバプログラムを運用する場合、それぞれの VPN サーバプログラムでユーザ認証の設定を行う必要がある。ある社員が L2TP/IPsec でも OpenVPN でも SoftEther VPN でも接続してくる可能性がある場合、ネットワーク管理者は 3 種類の VPN サーバプログラムにそれぞれ当該社員のアカウントを登録しておく必要がある。(ただし、Microsoft RRAS や SoftEther VPN は RADIUS 認証または Active Directory 認証をサポートしているため、RADIUS サーバまたは Active Directory ドメインを用意すればアカウントの登録は 1 回で済む。しかし、認証サーバの設置のためのコストが必要となる。)

また、VPN サーバに接続している VPN クライアントが行うことができる通信プロトコルの種類 (ポート番号等) や宛先 (IP アドレス等) を規制するためのパケットフィルタやセキュリティポリシーを設定したいと考える場合、それらのセキュリティ設定の方法や実装されているセキュリティ機能は VPN サーバプログラムの種類によって大きく異なる。たとえば SoftEther VPN は一般的なファイアウォールに搭載されているような優先順位付きのアクセスリスト機能を有しているが、Microsoft RRAS は単純な IP パケットフィルタしか実装していない。OpenVPN にはそもそもパケットフィルタの機能が無く、通信を許可することができる宛先 IP サブネットを指定することができ、また VPN クライアント同士の通信を許可するか禁止するかを設定することができる程度である。

このように、異なる VPN プログラム間ではパケットフィルタやセキュリティポリシーの設定が統一できず、また実現できる機能も少ないため、ネットワーク管理者が解決しなければならない運用上の課題が増大する (図 14)。

ユーザ認証、パケットフィルタ設定、ポリシー設定を一元化し、VPN プロトコルの種類にかかわらず 1 回だけ設定を行えば済むような新たな VPN システムが必要である。

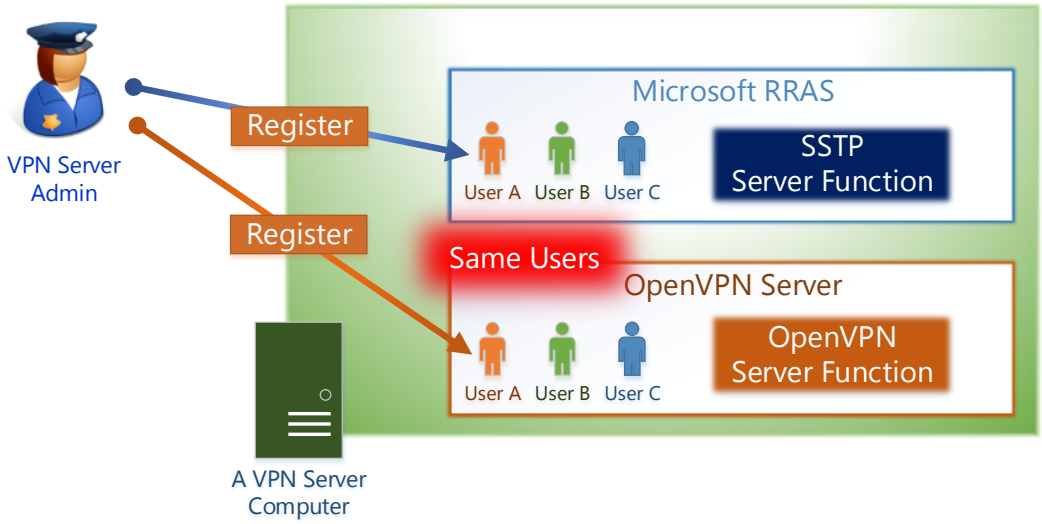


図 14. 複数の VPN サーバプログラムでそれぞれユーザ管理等の必要が生じ運用が複雑化

3.7. ログの形式、パケットログ機能の有無の相違

VPN サーバプログラムを運用する場合、トラブルシューティングや不正侵入を早期に発見するために VPN サーバプログラムが出力するログを読む必要がある。

また、企業などでコンピュータネットワークを運用する場合、ネットワーク上を流れるパケットのログを証拠としてディスクに蓄積しておき、何らかの事件や事故が発生した際などに証拠が必要となった場合に、当時の通信記録を検証することができるようにしておく必要がある。このような技術分野は「コンピュータ・フォレンジクス」と呼ばれている。コンピュータ・フォレンジクスを実現するための製品として、物理的なネットワーク上を流れるパケットログを記録する装置やソフトウェアは数多く存在ずが、それらの装置では物理的なネットワーク上を流れる暗号化された VPN パケットのペイロードを含むログを取ることができない。VPN パケットのログは、VPN サーバプログラムによって記録される必要がある。

複数の異なる VPN サーバプログラムを運用する場合、それぞれの VPN サーバプログラムが出力するログのフォーマットは全く異なる。たとえば Microsoft RRAS は Windows のイベントログに対してログを書き出す。SoftEther VPN はテキストファイルにログを書き出すか、syslog で送信するかを選択する。OpenVPN はテキストファイルにログを書き出す。同じようにテキストファイルにログが書き出される場合であっても、その書式や特徴は異なる。たとえば、重大なイベントが発生した場合のみネットワーク管理者のポケットベルに通報するようなスクリプトを作成したいと考える場合でも、それぞれの VPN サーバプログラムが出力するログの特徴を事前に吟味する必要が生じる。

ログの言語も VPN サーバプログラムによって異なる。Microsoft RRAS や SoftEther VPN は、現在の OS の言語設定に従ったログ (日本語 OS であれば日本語、英語 OS であれば英語) を出力できる (いずれも、ログの言語を変更することは容易である)。しかし OpenVPN は英語のログしか出力しない。この場合、たとえば日本人のネットワーク管理者は Microsoft RRAS や SoftEther VPN のログは理解しやすい日本語で読み、OpenVPN のログはやむを得ず英語で読むというという状況を希望する場合があるが、この場合はソフトウェアによって読むことができるログの言語が異なり、日常的な管理業務の負荷が大きくなる (図 15)。

さらに、出力することができるログの種類も VPN ソフトウェアによって異なる。Microsoft RRAS や OpenVPN は、パケットログを出力することができない。たとえば VPN を経由して VPN クライアントが社内 LAN に対して、あるいは他の VPN クライアントに対して不正な通信を行ったことが後から発覚した場合、パケットログがなければ通信を始動した VPN ユーザの特定が困難となることがある。SoftEther VPN はパケットログを出力することができ、出力したいログの種類を細かに設定することができる。会社で SoftEther VPN、Microsoft RRAS および

OpenVPN の 3 種類の VPN サーバプログラムが稼働している場合、たとえば不法行為を行おうとする社内の犯人は、パケットログが取られてしまう SoftEther VPN プロトコルを使用せず、代わりに Microsoft RRAS でサポートされている L2TP/IPsec または SSTP を使用するか、あるいは OpenVPN を使用するかを意図的に選択し、自身の不法行為の端緒や証拠ができるだけ残らないように画策することができる。

このように、ログの形式やパケットログの有無が相違する複数の VPN サーバプログラムを同時に運用することは、ネットワーク管理におけるリスクを増大させる。

SoftEther VPN のように豊富なパケットログを出力する機能を有し、かつ VPN プロトコルの種類にかかわらず保存されるログの書式や言語が統一されているような新たな VPN システムが必要である。

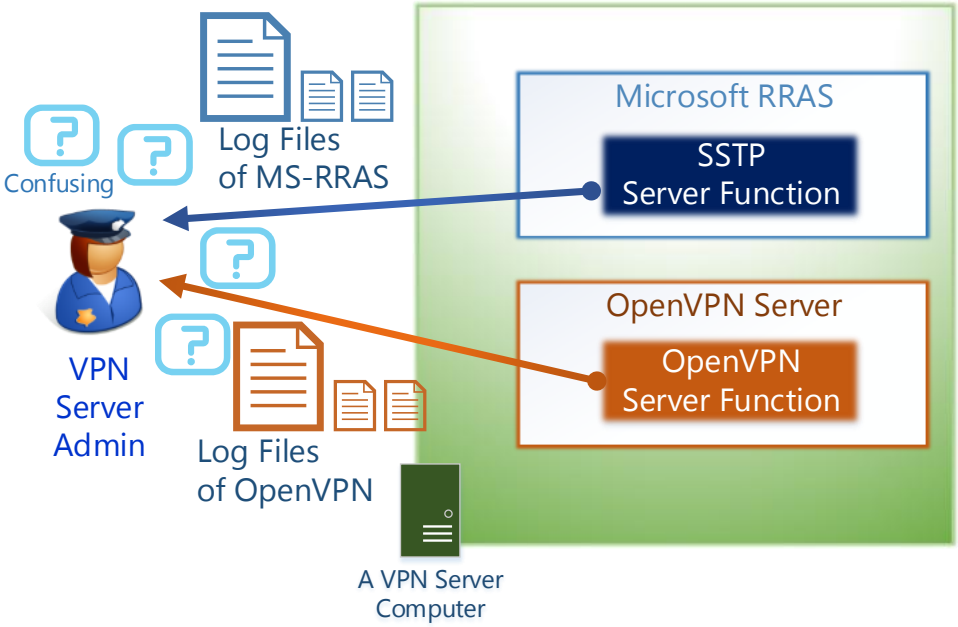


図 15. VPN サーバプログラムごとに出力されるログファイルの形式が異なる

3.8. VPN クライアント用 IP アドレスの管理の複雑化および使用効率の低下

遠隔地から社内 LAN に透過的にアクセスすることができるリモートアクセス型の VPN サーバを構築する場合、社内 LAN 上で使用されている IP アドレスと同一のサブネットに属する IP アドレスを VPN クライアントに対して動的に割り当てる必要がある (静的に割り当てることも可能であるが、管理がとても複雑になり、またいつ接続してくるか分からない VPN クライアントに静的 IP アドレスを割り当てることはアドレスの利用効率の低下を招く)。

複数の VPN サーバプログラムを同時に運用する場合、IP アドレスの管理の複雑化と、IP アドレスの動的割り当てにおける使用効率の低下が問題となる (図 16)。

社内 LAN では一般的にプライベート IP アドレスが使用される。プライベート IP アドレスであっても、アドレス空間にそれほど余裕がない場合がある。たとえば以前に 192.168.0.0/24 というサブネットで社内 LAN の構築を開始した場合、その後にこのサブネットを拡大する処理は、場合によってはすべてのネットワーク機器やコンピュータの設定を変更する必要があり大きな労力を要する。また、たとえば 10.0.0.0/8 の一部のサブネットを使用するようにアドレスを再設計したいと考えた場合でも、安物のネットワーク対応機器の中にはなぜか IP アドレスが 192.168 で始まるものしか受付けないというものがある。そのため、プライベート IP アドレスで運用されている社内 LAN であっても、多数のコンピュータが存在し、また多数の VPN クライアントが同時に社内 LAN にアクセスしようとする場合は、IP アドレスの使用効率 (使用密度) をできるだけ高めることが適切である。

しかし、VPN サーバプログラムによって IP アドレスを VPN クライアントに対して割り当てる方法は様々である。

Microsoft RRAS はレジストリに予め定義しておいた IP アドレスプールの範囲内から IP アドレスを割り当てると

いう方式である。予め IP アドレスプールを定義するためには、他の用途で絶対に使用しない IP アドレスブロックを決めておく必要があるため、たとえば VPN 接続してくる可能性があるクライアントが最大 100 台だからといって 128 個のブロックをプールに設定しても、10 台程度しか VPN 接続されていない状態では利用率は 10%未満となり、効率が悪い。

なお、Microsoft RRAS に DHCP プロキシエージェントをインストールすれば、社内 LAN に存在する既存の DHCP サーバから IP アドレスを割り当て、それを VPN クライアントに再割り当てするということも可能となっている。しかし、この場合は VPN クライアントが接続してきたときに必要最小限の個数の IP アドレス (通常は 1 個) を DHCP サーバに要求するのではなく、予め接続している可能性がある VPN クライアントの同時接続数を RRAS の VPN サーバに設定し、RRAS がその個数だけ予め DHCP サーバから IP アドレスを予約して取得する実装となっているので、IP アドレスプールを使用する方法と比較して利用効率はあまり変わらない。

OpenVPN の動的アドレス割り当ては、IP アドレスプールを使用する方法しか存在しない。しかも、Windows 版 OpenVPN サーバプログラムの実装では、1 個のクライアント用 IP アドレスをプール内から割り当てる場合に、合計 4 個もの周囲のアドレスを消費する仕様となっている。残りの 3 個の周囲のアドレスも使用することができないため、IP アドレスプール内の使用効率も大変悪い。

SoftEther VPN では、VPN クライアントに対して社内 LAN に既設の DHCP サーバから 1 個ずつ IP アドレスを要求しその IP アドレスを割り当てる。Microsoft RRAS のように、予め多数の IP アドレスを DHCP サーバに対して予約するようなことは行わずに、VPN クライアントが要求してくる度に 1 個ずつ DHCP サーバ上のプールを消費する。IP アドレスの使用効率は最も高い。

このように、複数の VPN サーバプログラムでは IP アドレスの割り当ての仕組みや効率が異なり、ネットワーク管理者はそれぞれの仕組みをよく理解する必要が生じる。そして、ある VPN クライアント用の IP アドレスとして現在どの IP アドレスが割り当てられているのかを把握するために、それぞれの VPN サーバプログラムのログを調査したり、IP アドレスプールの使用状況を閲覧したり、DHCP サーバの割り当て状況を閲覧したりするといった作業が必要になる。

これらの管理コストを低下させるため、複数の VPN プロトコルを用いる場合であっても、IP アドレスの動的割り当ては社内 LAN 上に既設の単一の DHCP サーバに一元化することができる新たな VPN システムが必要である。

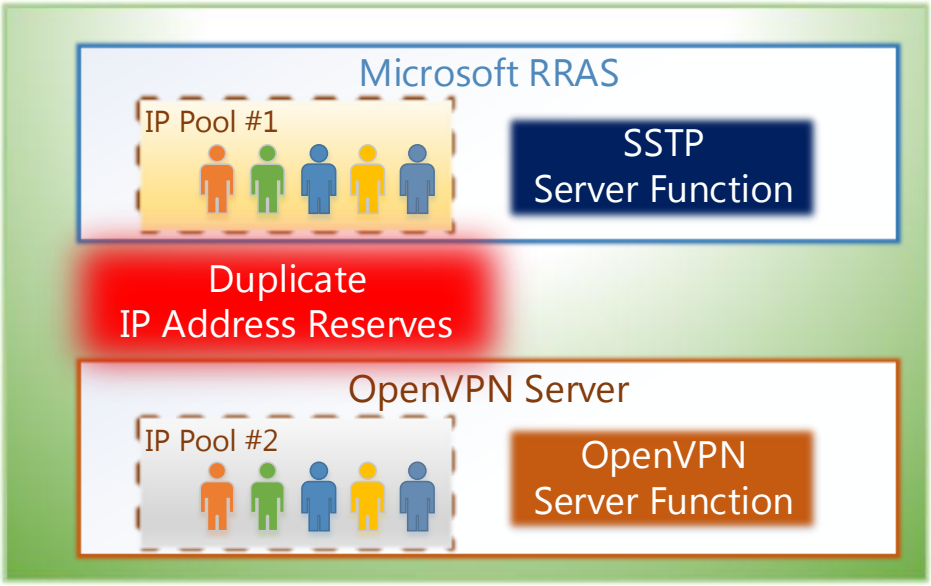


図 16. 複数の VPN サーバプログラムごとに IP アドレスプールを予約するため IP 使用効率が低下

第 4 章 設計

第 3 章では、複数の VPN プロトコルをサポートする VPN サーバコンピュータを構築するために複数の VPN サーバを組み合わせて使用する方法の問題点を述べた。第 4 章ではこれを解決するために、複数の VPN プロトコルを 1 個の VPN プロセスとして実装する方法を提示し、そのような VPN サーバプログラムの設計を行う。

4.1. 複数 VPN サーバプロトコルを 1 個のプロセスで実装

複数の VPN プロトコルに対応するため、複数の VPN サーバプログラムを同時に 1 台の VPN サーバコンピュータで動作させた場合、プログラム間のパケットの受け渡しが一端 OS を介して実施されることになる。この際に発生するオーバーヘッドを避ける手段には以下の 2 通りがある。

方法 1. 1 個のユーザモードプロセスにまとめる方法

OS を介したパケットの受け渡しによってオーバーヘッドが発生する原因は、OS の持つ IP ルーティングまたはブリッジング機能が持つキューに一端パケットを渡す必要があり、この際に生じるユーザモードからカーネルモードへの遷移やメモリコピーを避けることができないためである。

1 個のプロセス内にすべての VPN プロトコルの VPN サーバ機能を組み込めば、異なる VPN プロトコル間でパケットの受け渡しが発生したとしても、そのパケットのデータ本体はユーザモードのメモリ空間内でのみ受け渡され、メモリコピーの回数やカーネルモードへの遷移の頻度を最小限に減らすことができる。

方法 2. すべてカーネルモードで実装する方法

複数の VPN サーバプログラムを別々のプログラム (カーネルモードモジュール) として実装し、すべてカーネルモードで動作させる方法もある。この場合、たとえ OS の提供する IP ルーティングまたはブリッジング機能を用いて VPN サーバプログラム間でデータを受け渡す場合であっても、ユーザモード / カーネルモード間の遷移は発生しないため、オーバーヘッドを削減することができる。

すべてのプログラムをカーネルモードで実装する方法は、Windows Server の Microsoft RRAS ソフトウェアで採用されている。RRAS は L2TP/IPsec サーバ機能および SSTP サーバ機能の両方を提供している。この 2 つのサーバは別々のプログラムとして実装されている (L2TP/IPsec サーバ機能は「rasl2tp.sys」、SSTP サーバ機能は「rassstp.sys」というカーネルモジュールである) が、両方ともカーネルモードで動作し、Windows の持つ IP ルーティングスタックに対してカーネルモード内で直接接続されている。

しかし、プログラムをカーネルモードで実装する場合は開発の難易度が高くなり、また OS ごとに異なる実装が必要となるため移植性が低下する。同一の系列の OS であっても、バージョンが異なれば動作しなくなる可能性がある。そのため、本研究ではカーネルモードで実装する方法は採用しないことにし、1 個のユーザモードプロセスにまとめる方法を用いる (図 17)。

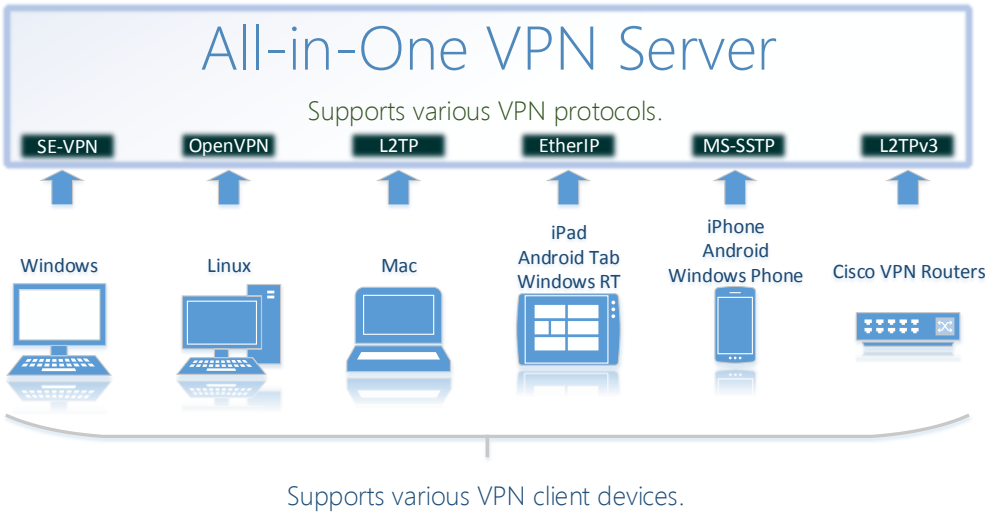


図 17. 本研究で実装する VPN サーバプログラムの概要

4.2. Ethernet を共通のバスとして使用

本研究で実装する新しい VPN サーバプログラムは、3.1 節の表にある PPTP 以外のすべての VPN プロトコルをサポートすることを目標とする。これらの VPN プロトコルの中には、レイヤ 2 (Ethernet) をカプセル化するものと、レイヤ 3 (IP) をカプセル化するものの両方が混在している。新しい VPN サーバプログラムは、レイヤ 2 VPN プロトコルで接続してきている VPN クライアントと、レイヤ 3 VPN プロトコルで接続してきている VPN クライアントとの間の通信を実現しなければならない。

1 個の VPN サーバプログラムで 2 種類の VPN プロトコルをサポートしようとする場合、両方のプロトコルのカプセル化対象レイヤが同一であれば、その 2 種類の VPN クライアント間の通信の実現は簡単である。たとえばレイヤ 3 (IP) 同士であれば IP ルーティング機能を実装し、レイヤ 2 (Ethernet) 同士であれば Ethernet スイッチング機能を実装する。そして、ルーティングまたはスイッチング機能を、異なる VPN プロトコル間に挟み込むようにして設置すれば、互いの VPN プロトコルの VPN クライアント間のパケットの送受信が可能となる。

一方、1 個の VPN サーバプログラムで、レイヤ 2 とレイヤ 3 の両方の VPN プロトコルのクライアントが混在し、互いに通信することができるようにするには、いずれかの VPN プロトコルのレイヤを、何らかの方法でもう一方の VPN プロトコルのレイヤに合わせなければならない。このために以下の方法が考えられる。

方法 1. すべてのレイヤ 2 (Ethernet) フレームを内部的にレイヤ 3 (IP) パケットに変換する

VPN サーバがレイヤ 2 の VPN プロトコルのクライアントから受信した Ethernet フレームのプロトコルの種類が IP である場合のみフレームを受け付け、Ethernet ヘッダを削除して中身の IP パケットを取り出せば、その IP パケットを他のレイヤ 3 の VPN プロトコルのクライアントにそのまま渡すことができる。他の VPN プロトコルのクライアントから受取った IP パケットには、Ethernet ヘッダを取り付けてレイヤ 2 の VPN プロトコルのクライアントに送信することができる。

この方法の利点は、実装が容易である点である。

この方法の欠点は、レイヤ 2 の VPN クライアント同士であっても IP パケットの通信しかできなくなる点である。Ethernet 上で動作する IP 以外のプロトコルが利用できなくなる。また、遠隔地の Ethernet セグメント同士を接続する拠点間 VPN 接続も利用できなくなる。

方法 2. すべてのレイヤ 3 (IP) パケットをレイヤ 2 (Ethernet) フレームに変換する

VPN サーバがレイヤ 3 の VPN プロトコルのクライアントから受信した IP パケットにダミーの Ethernet ヘッダを付加すれば、この Ethernet フレームを他のレイヤ 2 の VPN プロトコルのクライアントにそのまま渡すことができる。他の VPN プロトコルのクライアントから受取った Ethernet フレームからは、Ethernet ヘッダを削除してレイヤ 3 の VPN プロトコルのクライアントに送信することができる。

この方法の利点は、通信可能なプロトコルの種類が減らないことである。レイヤ 3 の VPN クライアントは IP パケットしか送受信できないが、レイヤ 2 の VPN クライアント同士は Ethernet 上で動作する任意のプロトコルが引き続き利用可能である。

この方法の欠点は、実装が複雑となることである。たとえば、VPN サーバ側から見た場合、レイヤ 2 の VPN クライアントには MAC アドレスが割り当てられているが、レイヤ 3 の VPN クライアントには MAC アドレスという概念がない。Ethernet フレームを VPN サーバが適切な VPN クライアントに対して受け渡しするためには MAC アドレスをもとにした Ethernet スイッチング動作が必要であるため、レイヤ 3 の VPN クライアントにも仮想的な MAC アドレスを割り当てる必要がある。また、IP を Ethernet 上で使用するためには ARP の送受信が必要であるが、レイヤ 3 の VPN クライアントは ARP の送受信を行わないため、ARP の状態を VPN サーバ側で管理する必要がある。すわなち、接続しているレイヤ 3 の VPN プロトコルのクライアントの個数分、仮想的な IP スタックのインスタンスを作成する必要がある。

方法 3. レイヤ 3 (IP) の VPN クライアント同士とレイヤ 2 (Ethernet) の VPN クライアント同士とを分離し、レイヤ間を接続モジュールで相互接続する

レイヤ 3 (IP) の VPN クライアント同士は従来のレイヤ 3 型 VPN サーバのように直接 IP ルーティングによってパケットを受け渡し、レイヤ 2 (Ethernet) の VPN クライアント同士は従来のレイヤ 3 型 VPN サーバのように直接 Ethernet スイッチングによってパケットを受け渡す。そして、レイヤ 3 の IP ルータがレイヤ 2 の Ethernet スイッチに対して接続し、すべてのレイヤ 3 VPN クライアントを代表して 1 個の MAC アドレスを持つ方法がある。

この方法の利点は、方法 2 と同様に通信可能なプロトコルの種類が減らないことである。また、方法 2 では接続しているレイヤ 3 の VPN クライアントの個数だけ仮想的な IP スタックを作成する必要があるが、この方法では仮想的な IP スタックはレイヤ 3 の IP ルータがレイヤ 2 の Ethernet スイッチに対して接続する点の 1 個のみで良い。

この方法の欠点は、実装が方法 2 よりも複雑となる可能性があることである。確かにパケットの受け渡しを行うだけであれば実装はそれほど難しくないかも知れないが、本研究で解決したいと考えている課題として、他にもパケットログを共通化したり、パケットフィルタやセキュリティポリシーなどのセキュリティをすべての VPN クライアントに対して VPN プロトコルにかかわらず一括して適用したりしたいという要求がある。パケットの交換処理としてレイヤ 2 とレイヤ 3 の 2 種類のモジュールを用意すると、これらのセキュリティ機能でパケットをモニタリング、遮断する仕組みを 2 個用意する必要が生じる。

本研究では、上記の 3 種類の方法の利点と欠点を考慮した結果、実現することができる機能が最も多く、かつ実装が用意である方法 2 を用いることにする。

したがって、VPN サーバは仮想的な Ethernet スイッチを持ち、レイヤ 2 の VPN プロトコルの VPN クライアントが接続してきた場合はその VPN クライアントを仮想 Ethernet スイッチに直接接続する。レイヤ 3 の VPN プロトコルの VPN クライアントが接続してきた場合は、直接仮想 Ethernet スイッチに接続することができないため、間にレイヤ 3– レイヤ 2 のレイヤ変換装置を挿入して仮想 Ethernet スイッチに接続する。このように、仮想 Ethernet スイッチは、VPN クライアントのプロトコルの種類にかかわらず常に接続中の VPN クライアントをレイヤ 2 (Ethernet) の VPN クライアントとして扱う (図 18)。

本研究で実装する VPN サーバプログラムの仮想 Ethernet スイッチのことを「仮想 HUB」、仮想 HUB に対して現在接続されている VPN クライアント 1 個ずつに対して仮想 HUB 側で接続される仮想的な接続ポートを「VPN

セッション」と呼ぶ。

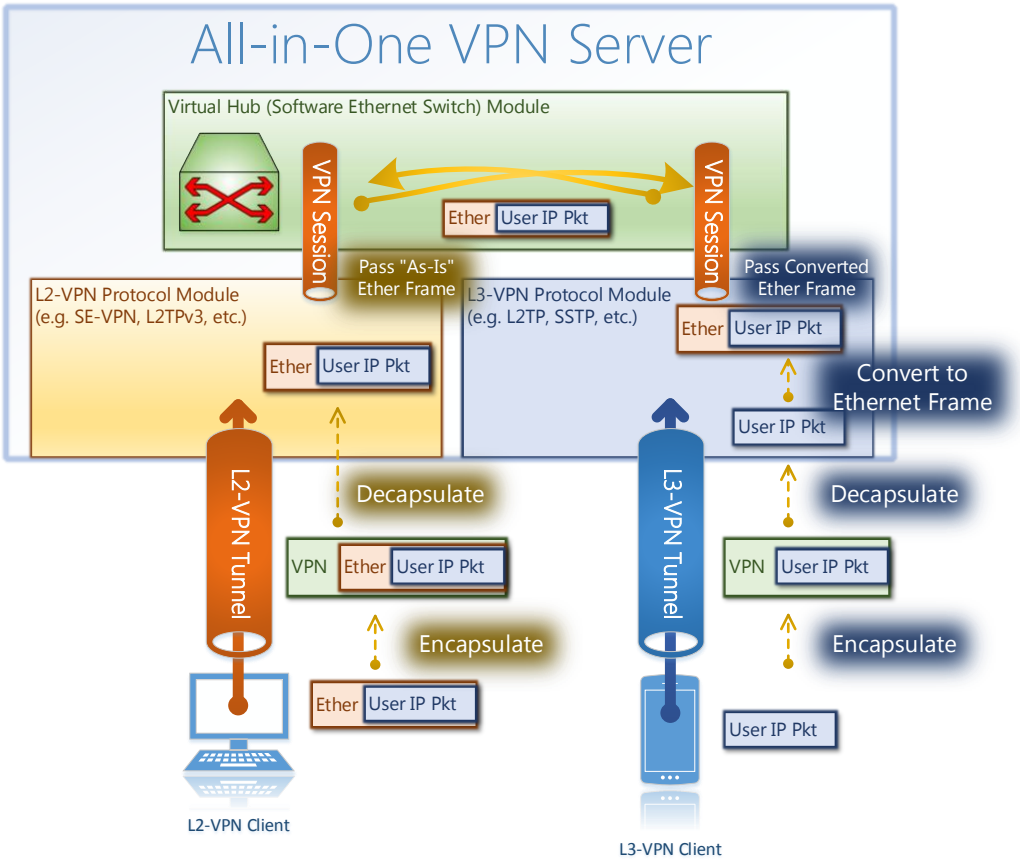


図 18. L3 VPN プロトコルは一旦 L2 (Ethernet) にレイヤ変換してから仮想 HUB で取り扱う

4.3. 仮想 HUB におけるレイヤ 2 VPN セッションの扱い

仮想 HUB に複数のレイヤ 2 の VPN セッションが接続されている場合は、仮想 HUB は物理的な Ethernet スイッチと同等のスイッチング処理を行う (図 19)。VPN セッションごとに送信元 MAC アドレスを学習し、ある VPN セッションが Ethernet フレームを仮想 HUB に対して送信しようとした場合は、そのフレームの宛先 MAC アドレスを、仮想 HUB が保持している MAC アドレス学習用のテーブルから検索し、一致する VPN セッションに対してそのフレームを受け渡す。MAC アドレス学習用のテーブルに存在しない MAC アドレスを宛先としている Ethernet フレームの場合、または宛先 MAC アドレスがブロードキャストアドレスの Ethernet フレームの場合は、現在接続されているすべての VPN セッションに対してブロードキャストする。このような MAC アドレス学習用のテーブルを FDB (Forwarding Database) と呼ぶ。

仮想 HUB は、後述のパケットログやパケットフィルタ、セキュリティポリシーなどのために必要な場合を除き、ある VPN セッションから別の VPN セッションに対して送信される Ethernet フレームの内容を解析せずに、単純に適切な VPN セッションに対してスイッチングするレイヤ 2 の処理のみを行う。仮想 HUB は IP ヘッダを解釈してルーティングするといったレイヤ 3 の処理は一切行わない。

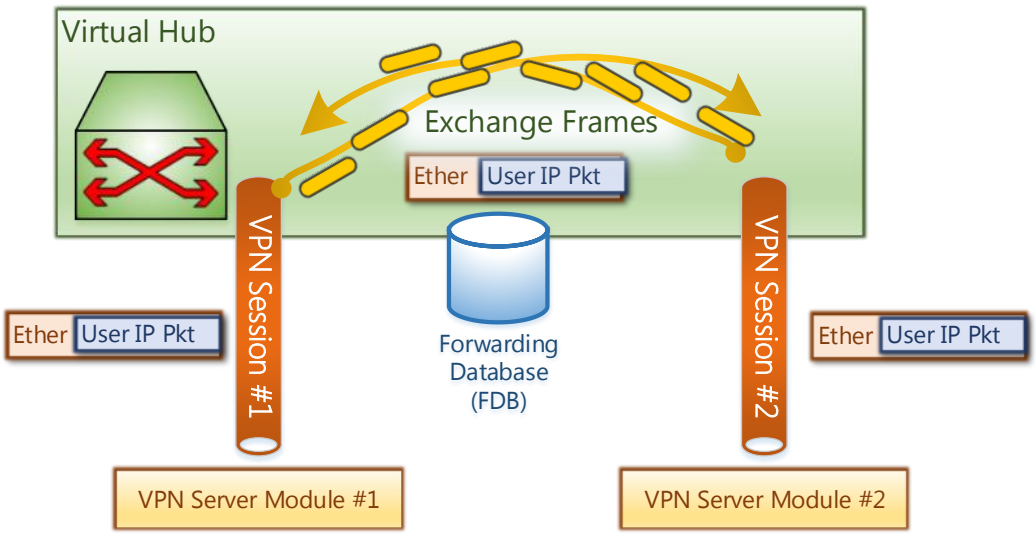


図 19. 仮想 HUB は複数 VPN セッション間の MAC アドレス学習と Ethernet フレームの交換を行う

4.4. 仮想 HUB におけるレイヤ 3 VPN セッションの扱い

レイヤ 3 VPN セッションは仮想 HUB に直接接続することができない。L2TP/IPsec や SSTP、OpenVPN (L3) といった VPN プロトコルが仮想 HUB に接続してきた場合には、VPN サーバプログラムはその VPN プロトコルのレイヤ (L3) を仮想 HUB が扱うことができるレイヤ (L2) に変換するためのレイヤ変換を行う。

レイヤ変換器が、新しいレイヤ 3 VPN セッションが作成される度に 1 個ずつ作成される。各レイヤ変換器は、対応するレイヤ 3 VPN セッションが切断される際に自動的に破棄される。

レイヤ変換器は、Ethernet ヘッダの追加・削除、ARP リクエスト / レスポンスの送受信、DHCP リクエストの送信 / レスポンスの受信処理を行う。これらの処理は、レイヤ 3 VPN セッションとして接続している VPN クライアントのユーザには認識されず、透過的に行われる (図 20)。

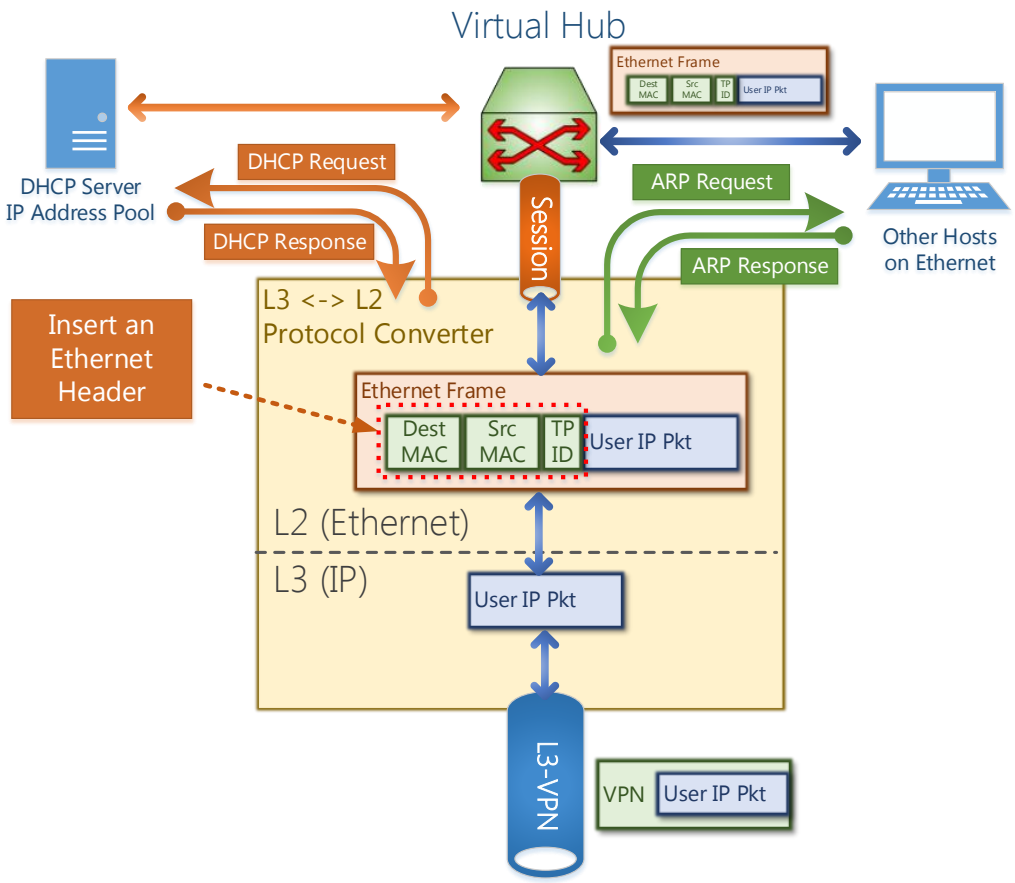


図 20. L3-L2 レイヤ変換器による Ethernet ヘッダ処理、ARP 処理および DHCP 処理

4.5. レイヤ変換器の持つパラメータと DHCP サーバとの通信

レイヤ変換器は、初期化、動作中、解放の 3 つの状態がある。初期化処理によって、レイヤ変換器の持つ一時 MAC アドレスや IP パラメータが決定される。

レイヤ変換器が初期化処理で設定するパラメータにはのようなものがある。

表 3. レイヤ変換器の持つ IP パラメータの一覧

一時 MAC アドレス (例: CA-01-23-45-67-89)	ユニークな MAC アドレスであり、レイヤ変換器の初期化時に乱数を用いて生成されるが、一端破棄された後は使い回される。
IP アドレス (例: 192.168.0.123)	レイヤ変換器が仮想 HUB 内で通信に使用する IP アドレス。この IP アドレスは、初期化処理が完了したときに接続してきた VPN クライアントに VPN プロトコルを通じて通知し、VPN クライアントのコンピュータの TCP/IP スタックが使用する IP アドレスにもなる。
サブネットマスク (例: 255.255.255.0)	レイヤ変換器が属する仮想 HUB 内における IP ネットワークの範囲を示す。この範囲内の IP アドレスに対しては直接 ARP で MAC アドレスを解決して通信する。この範囲外の IP アドレスに対しては、デフォルトゲートウェイを経由して通信する。
デフォルトゲートウェイ (例: 192.168.0.254)	サブネット外の IP アドレスに対して IP パケットを送信しようとするときは、デフォルトゲートウェイの IP アドレスに対して ARP 解決を行い、その MAC アドレスに対して IP パケットを送信する。
DHCP サーバドレス (例: 192.168.0.254)	仮想 HUB 内で上記の IP アドレス、サブネットマスクおよびデフォルトゲートウェイを決定するために、DHCP サーバに対して DHCP リクエストを送り IP アドレスの割り当てを受ける。IP アドレスを払い出した DHCP サーバのアドレスを記憶する。
DHCP リース期限 (例: 240 分)	DHCP アドレスの払い出しを受けたときに同時に指定されるリース期限である。リース期限が切れる前に、自動的にリースを更新することにより同一の IP アドレスが他のコンピュータに使用されることを防止する。
DNS サーバドレス (2 個) (例: 192.168.0.1)	DNS サーバドレスはレイヤ変換器によって直接的に参照されることはないが、接続してきた VPN クライアントに VPN プロトコルを通じて通知し、VPN クライアントのコンピュータの DNS クライアントがリゾルバとして使用する。
WINS サーバドレス (2 個) (例: 192.168.0.2)	WINS サーバドレスはレイヤ変換器によって直接的に参照されることはないが、接続してきた VPN クライアントに VPN プロトコルを通じて通知し、VPN クライアントのコンピュータの Windows ファイル共有クライアントがリゾルバとして使用する。

IP パラメータを自動的に決定するために、レイヤ変換器は仮想 HUB を経由して同一セグメント上に既設の DHCP サーバに対して DHCP リクエストを送信する。DHCP レスポンスが返ってきた場合は、それ以降は DHCP サーバからリースされた IP アドレスをレイヤ変換器の IP アドレスとして使用する。DHCP サーバから IP アドレスを取得することに失敗した場合は、レイヤ変換器の初期化は失敗し、接続しようとしている VPN クライアントはエラーによって切断される。

レイヤ変換器は、動作中は Ethernet ヘッダの追加・削除や ARP の送受信、DHCP パケットの送受信を行う。また、DHCP サーバから払い出された IP アドレスのリース期限が切れないようにするため、自動的にリースの更新を行う。

解放処理では、レイヤ変換器は DHCP サーバに対して、リースされていた IP アドレスの解放を依頼してからレ

イヤ変換器自身の動作を停止させる。

この仕組みにより、VPN サーバプログラムは DHCP サーバに対して常に必要最小数の (実際に接続している) レイヤ 3 VPN クライアントの IP アドレスの個数だけをリクエストすることができ、3.8 節で述べた、従来のレイヤ 3 VPN サーバが共通して持っていた IP アドレスプールの使用効率の低下問題を解決する。

4.6. レイヤ変換器による Ethernet ヘッダの追加・削除および ARP の送受信

一時 MAC アドレスについて

各レイヤ 3 VPN セッションに関連付けられているすべてのレイヤ変換器は、それぞれ、仮想 HUB を通じて Ethernet フレームを送受信することができるようにするためにユニークな一時 MAC アドレスを持つ。一時 MAC アドレスは、レイヤ 3 VPN セッションが仮想 HUB に接続しようとする際に自動的に作成されるレイヤ変換器によって乱数を元に作成され、レイヤ変換器が削除される際まで使用される。ただし、毎回異なる乱数を生成すると、レイヤ 3 VPN セッションが接続・切断を繰り返した場合、後述する仮想 DHCP クライアントが既設の DHCP サーバの IP アドレスプールを使い果たしてしまう可能性があるため、同じ MAC アドレスを使い回すことが望ましい。この場合、いかなる時点でも複数のレイヤ変換器が重複した一時 MAC アドレスを保有してしまうことを防止しなければならない。

VPN クライアントからのパケットの受信処理

レイヤ 3 VPN セッションのクライアント側から IP パケットが仮想 HUB に対して送信された場合、レイヤ変換器は IP パケットに Ethernet フレームを追加する。この際、送信元 MAC アドレスフィールドにはレイヤ変換器が持つ一時 MAC アドレスを書き込む。宛先 MAC アドレスフィールドには、IP ヘッダの宛先 IP アドレスとして指定されている IP アドレスに対応する MAC アドレスを書き込む。

ARP の処理

宛先 IP アドレスに対応する MAC アドレスを決定する方法は少し複雑である。通常のコンピュータの OS の TCP/IP スタックが行っている ARP の送受信やルーティングテーブルに従ったフォワーディングといった処理と同等の処理をレイヤ変換器が行う必要がある。レイヤ変換器は、宛先の IP アドレスが自身の所属する IP サブネット範囲内である場合は直接 ARP により MAC アドレスを解決しようと試み、解決された場合は当該 MAC アドレスを L2 における宛先とする。IP アドレスがサブネットの範囲外である場合は、デフォルトゲートウェイを通じて IP パケットをフォワーディングする必要がある。この場合は、デフォルトゲートウェイとして指定されている IP アドレスの MAC アドレスを、ARP を用いて解決し、その MAC アドレスを L2 における宛先とする。通常の TCP/IP スタックと同様、レイヤ変換器は ARP の学習結果を ARP テーブルとしてキャッシュする。また、仮想 HUB に接続されている他のコンピュータからレイヤ変換器に割り当てられている IP アドレスの MAC アドレスの解決を要求する ARP リクエストを受信した場合は、直ちに ARP レスポンスを返信する。

これらの ARP の処理は、2.3 節で述べたプロキシ ARP とほぼ同様である。

VPN クライアントに対するパケットの送信処理

仮想 HUB を経由して他のコンピュータからレイヤ変換器に割り当てられている IP アドレスを宛先とした IP パケットを含む Ethernet フレームを受取った場合は、レイヤ変換器は当該 Ethernet フレームから Ethernet ヘッダ部分を削除し、VPN クライアントに渡す。

4.7. パケットログ、パケットフィルタ、セキュリティポリシーの共通処理

仮想 HUB は、内部を流れる Ethernet フレームに対して、レイヤ 2 の VPN セッション、レイヤ 3 の VPN セッションの両方に対してレイヤの違いを考慮することなく共通のセキュリティ処理が可能である (図 21)。

パケットログ

仮想 HUB にはパケットログの出力機能を付ける。パケットログとは、流れるパケットのヘッダ部分を解釈して重要なヘッダの値をテキスト形式で列挙し、テキストファイルに 1 パケット 1 行の形式で保存する機能である。

3.7 節で述べたように、従来の複数の VPN サーバプログラムを組み合わる手法の場合は、それぞれの VPN サーバプログラムで個別にパケットログ機能を実装しなければならない。一方、本研究の手法の場合は仮想 HUB にパケットログ機能を 1 つ実装すれば、VPN プロトコルの種類にかかわらず同一内容のパケットログを出力することができる。

パケットフィルタ

仮想 HUB にはパケットフィルタ機能を付ける。パケットフィルタとは、流れるパケットのヘッダ部分を解釈して、予め設定したパケットフィルタルールにある条件式にヘッダ部分の値が一致する場合にそのパケットを通過または破棄する動作を行うフィルタである。

パケットフィルタ機能により、Ethernet ヘッダ (L2)、IP ヘッダ (L3)、TCP/UDP ヘッダ (L4) の各フィールドの値を条件式として判定することができる。パケットログと同様、本研究の手法であればパケットフィルタ機能も仮想 HUB に 1 つのみ実装すれば、VPN プロトコルの種類にかかわらず同一の動作が保証される。

セキュリティポリシー

パケットフィルタはパケットごとに 1 個ずつ、通過・破棄を判定する、ステートレスなフィルタである。しかし、場合によっては VPN セッション単位で状態を持つステートフルなフィルタを用意したほうが良い場合もある。このようなステートフルなフィルタとして、セキュリティポリシー適用機能を搭載する。

セキュリティポリシー適用機能は、VPN セッションごとに、その VPN セッションが確立されてから現在までに送受信されたパケットの特徴を記録しておき、その記録された状態をもとに、それ以降のパケットの送受信を許可するか否かを判断する仕組みである。代表的なものとして ARP Spoofing[16] 防止フィルタと、DHCP Snooping ファイルタがある。

ARP Spoofing 防止フィルタは、ある VPN クライアントがすでに使用している IP アドレスと同一の IP アドレスについて、別の VPN クライアントが勝手に「自分がその IP アドレスを使っている」と主張する ARP レスポンスを送信することを禁止するフィルタである。本研究の手法では VPN セッション間のパケットの受け渡しはすべて仮想 HUB がレイヤ 2 (Ethernet) のヘッダおよび FDB を見て行っている。もし偽の ARP レスポンスの発信を許容してしまうと、ある VPN クライアントが使用している IP アドレスを別の VPN クライアントが横取りして通信妨害・盗聴することが可能となってしまう。このような攻撃は、単純なスイッチング HUB として動作するだけの仮想 HUB では検出・防止することができない。そこで、仮想 HUB の VPN セッションに対して ARP Spoofing 防止フィルタを適用することにより、仮想 HUB のスイッチングモジュールにフレームが渡される直前で、このような危険なフレームを破棄することができるようにする。

DHCP Snooping フィルタは、ある VPN クライアントが DHCP サーバから割り当てを受けた IP アドレス以外の IP アドレスを勝手に使用することを防止するフィルタである。このフィルタにより、ネットワーク管理者が意図した IP アドレスのみが VPN 接続してきた VPN クライアントに割り当てられる。本来、レイヤ 2 の Ethernet セグメントでは IP アドレスは自己申告制となっており、「自分はこの IP アドレスを使っている」と主張したノードがその IP アドレスを先取することになっている。不特定多数のユーザが遠隔地から認証なしで接続することを可能にするような

VPN サーバを構築したい場合は、VPN クライアントが自分の IP アドレスを任意に決めてしまえることは管理上の問題を招くため、DHCP サーバによって割り当てられた IP アドレスのみを使用させたい場合がある。このような場合に DHCP Snooping フィルタを使用する。

本研究の手法の場合、このような VPN セッションごとに状態を持つ必要があるセキュリティポリシー機能は 1 つのみ実装すれば良く、一端実装してしまえば、VPN プロトコルの種類の違いにかかわらず、すべての VPN クライアントに対してそのセキュリティポリシーが適用されることが保証される。

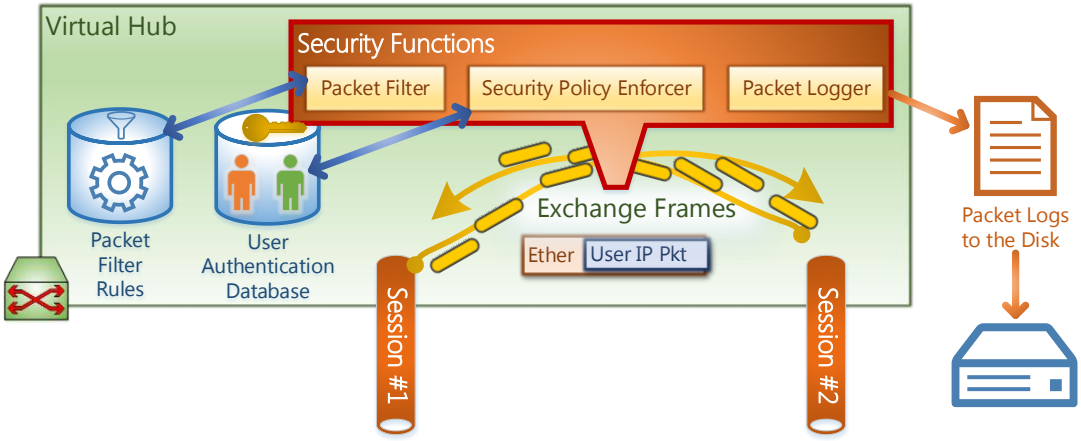


図 21. 仮想 HUB のセキュリティ機能

4.8. ユーザ認証設定の一元管理

VPN プロトコルごとにユーザ認証のプロトコルは異なる。SoftEther VPN の場合は SHA (Secure Hash Algorithm) による CHAP (Challenge-Handshake Authentication Protocol) 認証、L2TP/IPsec および SSTP の場合は MS-CHAPv2 (Microsoft CHAP Extension, Version 2) または PAP (Password Authentication Protocol) 認証、OpenVPN の場合はクリアテキストのパスワードを暗号化されたチャネルを用いて送受信する方式による認証、L2TPv3/IPsec および EtherIP/IPsec の場合は IKE 内における ID ペイロードを用いた認証が一般的に利用されている。

これらの異なる認証メカニズムを吸収し、ユーザ認証を仮想 HUB に一元化するために、仮想 HUB にはユーザ認証データベースを置く。そして、ユーザ認証データベースを用いてユーザ認証を行うためのアダプタを VPN プロトコルごとに用意する。たとえばユーザ A が仮想 HUB のユーザ認証データベースに登録されていれば、このユーザ A は SoftEther VPN、L2TP/IPsec、SSTP または OpenVPN のいずれの VPN プロトコルであっても VPN サーバに接続することができる。これにより、第 3 章で述べたような複数の VPN サーバプログラムを組み合わせる従来手法では実現できなかったユーザ認証設定の一元管理を可能にする (図 22)。

ユーザ認証データベースにおけるユーザの認証方法として、ユーザごとのパスワードのハッシュをデータベースに記録する方式のほかに、外部の RADIUS (Remote Authentication Dial In User Service) または Active Directory サーバのユーザ認証データベースを用いる方式も利用可能とする。この場合も、ネットワーク管理者は一度仮想 HUB に RADIUS または Active Directory を使用するための設定を行うだけで、すべての VPN プロトコルのユーザがそれらの外部ユーザ認証を用いて VPN サーバに接続することができるようになる。

外部認証サーバを用いたユーザ認証を使用するためには、2 種類のモードを用意する。個別のユーザごとに外部認証サーバ側におけるユーザ ID をマッピングしておき、当該ユーザとしてログイン要求を出した VPN クライアントから受け取った認証データをそのユーザ ID と組み合わせて外部認証サーバに提示するモードと、いかなるユーザ ID であっても自動的にそのまま ID と認証データとを外部認証サーバに提示するモードである。両方のモードを組み合わせて使用することも可能とする (この場合、マッピング表が存在するユーザの場合はマッ

ピングの結果のユーザ ID が、それ以外のユーザについては VPN クライアントから提示されたそのままのユーザ ID が使用される)。

ユーザ認証データベースでは、個別のユーザ、またはユーザを複数まとめたグループ単位で、4.7 節で述べたセキュリティポリシーを設定することができる。また、パケットフィルタ機能のルールの一部として、パケットの送信元または宛先のユーザ名またはグループ名を指定することができる。このように、通信内容に関する高度な規制を、認証されたユーザごとに異なる設定として適用することが可能である。

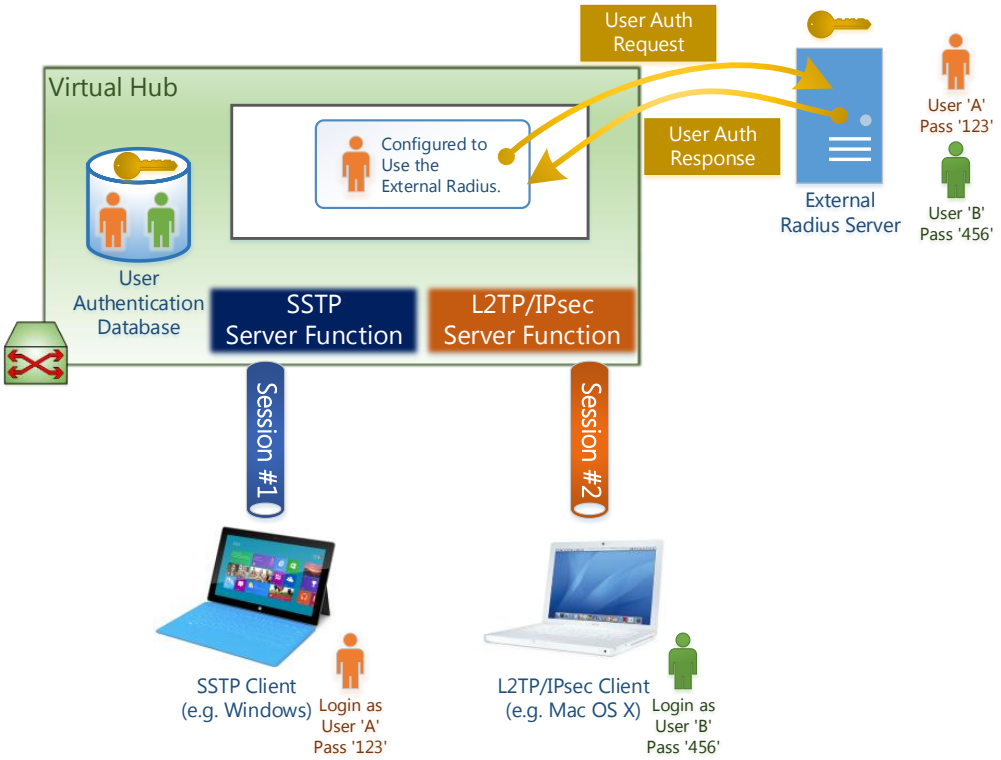


図 22. VPN プロトコルの種類にかかわらず一元化されたユーザ認証が利用可能

4.9. 複数の仮想 HUB の作成

ネットワーク管理者は、VPN サーバプログラム内に複数の仮想 HUB を作成することができる。各仮想 HUB はそれぞれがレイヤ 2 (Ethernet) のブロードキャストドメインを構成する。

1 台の机の上に複数の物理的な Ethernet スイッチが設置されている場合であっても、異なったスイッチに接続されているコンピュータ同士は通信ができないのと同様に、同一の VPN サーバプログラム上であっても、異なる仮想 HUB 間では通信を行うことができない。そのため、1 台の物理的な VPN サーバコンピュータを用いて複数の仮想 HUB を作成し、それぞれの仮想 HUB の管理権限をそれぞれ 1 人ずつの VPN グループ管理者に委任することにより、多数の VPN グループを 1 台のコンピュータでホストすることができる。これは、1 台の物理的なコンピュータ上で複数個の VM を起動し、それぞれの VM を各ユーザが自由に使用するのと同様である。

VPN サーバプログラム内では、仮想 HUB をメモリが許容する限り何個でも作成することができるようにする。仮想 HUB は名前で識別される。4.8 節で述べたユーザ認証データベースや 4.7 で述べたセキュリティ設定は仮想 HUB ごとに独立しているため、たとえば仮想 HUB「coins」と仮想 HUB「accc」があるとき、異なる仮想 HUB に接続されている VPN クライアント同士は一切通信ができない仮想 HUB「coins」に登録されているユーザ a は a@coins という ID で、仮想 HUB「accc」に登録されているユーザ b は b@accc という ID で VPN 接続を行う。

この仕組みにより、本研究の手法は、3.5 で述べた VPN 通信グループ間の分離が不能であるという従来手法の問題点を解決する (図 23)。

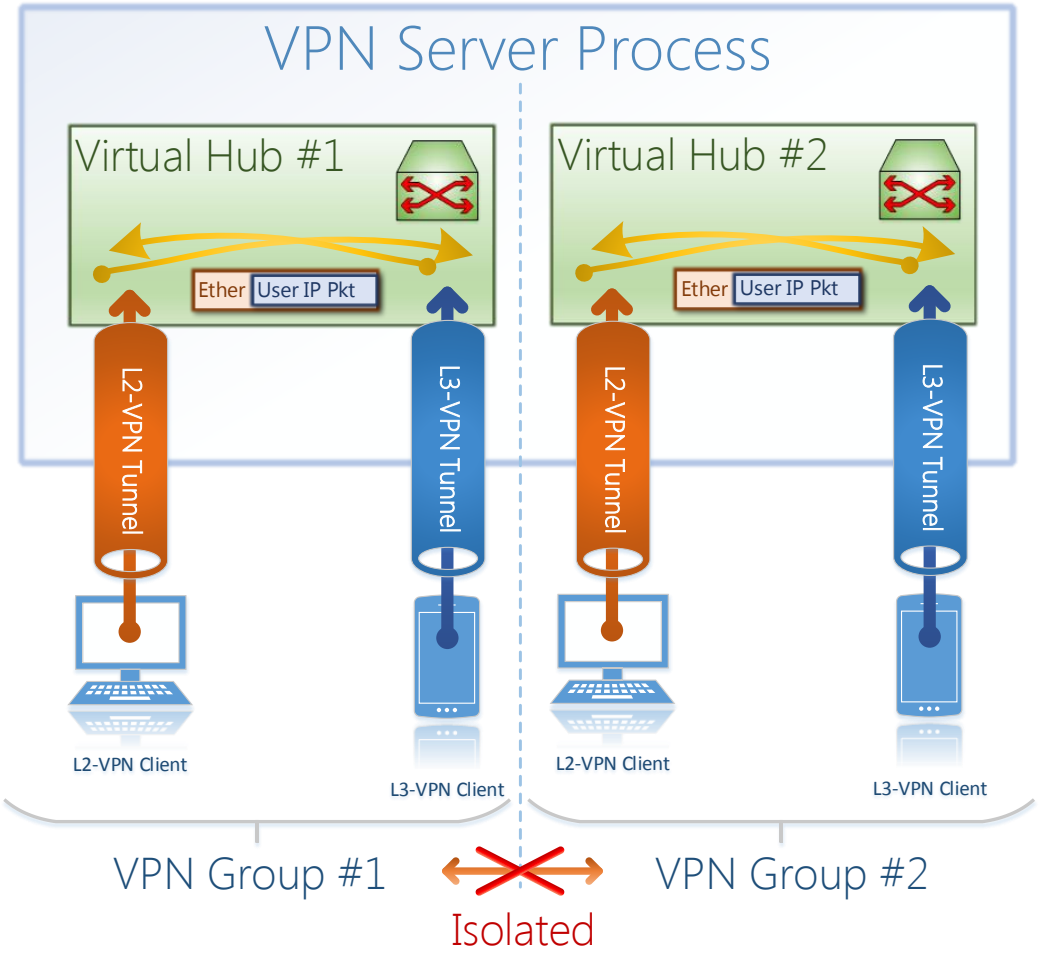


図 23. 仮想 HUB 間は分離されており相互に通信不可能

第 5 章 実装

第 4 章では、複数の VPN プロトコルを 1 個のプロセスでサポートする VPN サーバプログラムの設計について述べた。第 5 章では、第 4 章で設計した VPN サーバプログラムを実装する方法について述べる。

5.1. 既存の VPN サーバプログラムの拡張

第 4 章で述べた設定を実装するためには、フルスクラッチで新しいプログラムを開発する方法と、既存のプログラムを拡張する形で新しいプログラムを開発する方法の 2 種類がある。全く新しくプログラムを開発する方法は時間がかかるので、本研究では既存のプログラムを拡張する方法を選択する。

本研究で実装したいプログラムは VPN サーバプログラムなので、表 2 にあるような既存の VPN プロトコルのうちいずれかをサポートする拡張する既存の VPN サーバプログラムを拡張することが望ましい。一般的なコンピュータ上で動作する既存の VPN サーバプログラムの中には以下のようなものがある。

Microsoft Routing and Remote Access (RRAS) サービス

Windows Server に搭載されている VPN サーバ機能である。RRAS を拡張するためには、RRAS のソースコードが必要である。しかし、Microsoft 社のポリシーによりソースコードは一般に公開されておらず、秘密保持契約を締結して公開を受けた場合でもコードの改変が禁止されているため、本研究では使用できない。

OpenVPN Server プログラム

OpenVPN, Inc. が開発して販売している OpenVPN サーバプログラム、またはそのオープンソース版 (GPL ライセンス) を拡張する方法がある。そこで OpenVPN サーバプログラムの機能を調査したところ、4.7 節で述べたパケットログ、パケットフィルタ、セキュリティポリシーや 4.8 節で述べたユーザ管理の仕組みがなく、4.9 節で述べた通信グループの作成機能もない。これらの機能を新たに OpenVPN サーバプログラムに追加することは難易度が高い。

SoftEther VPN Server プログラム

SoftEther VPN Server はソフトイーサ株式会社が開発して販売されている PacketiX VPN Server、およびオープンソース版 (GPL) として配布されている UT-VPN Server の 2 つで共通で利用されている VPN サーバエンジンである。SoftEther VPN Server には 4.7 節で述べたパケットログ、パケットフィルタ、セキュリティポリシーや 4.8 で述べたユーザ管理の仕組みがすでに搭載されており、仮想 HUB を何個でも作成することができる 4.9 節で述べた通信グループの作成機能も実装されている (図 24)。

上記のうち OpenVPN Server と SoftEther VPN Server の 2 つを比較検討した結果、本研究で実装したい機能を新たに追加するためには SoftEther VPN Server を拡張したほうが簡単であるということが分かった。

そこで、本研究では既存の SoftEther VPN Server のバージョン 3.0 を拡張し、新たにバージョン 4.0 という名称で実装することにする。以下では本研究で実装するプログラムの名称を「SoftEther VPN Server 4.0」または「VPN Server」と標記する。

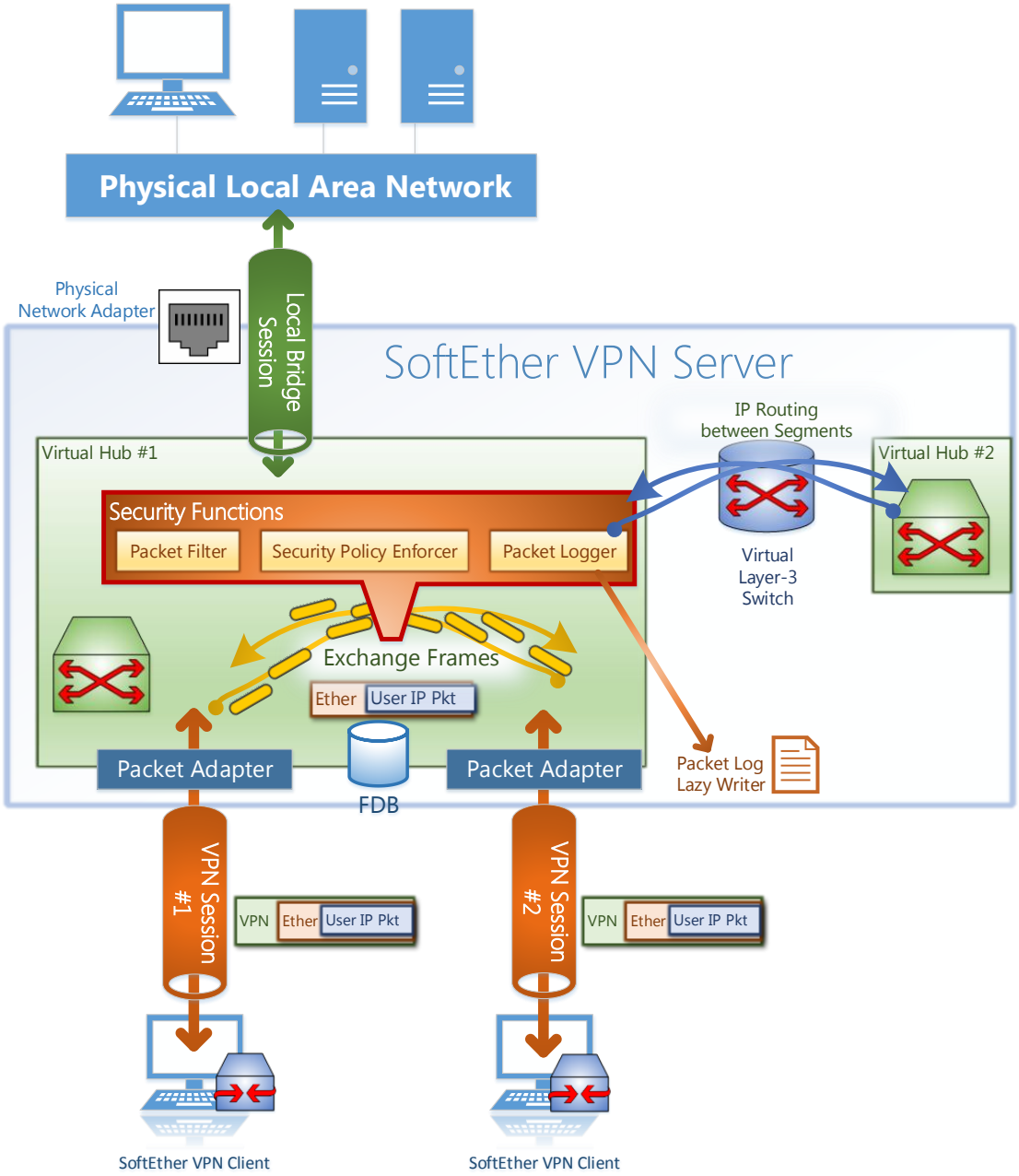


図 24. 既存の SoftEther VPN Server バージョン 3.0 の主要機能

5.2. SoftEther VPN Server 3.0 の内部構造

本研究の実装方法を検討するために、まず、既存の SoftEther VPN Server 3.0 の内部構造について述べる。

全体

SoftEther VPN Server 3.0 は複数のモジュールが組み合わさって実装されている。

「仮想 HUB」は複数の VPN クライアントからの VPN セッションを受付け、VPN セッション間のパケットの送受信のためのスイッチングを行うモジュールである。VPN セッションの作成、削除は VPN クライアントの増減に伴い動的に行われる。この際には「Packet Adapter」というモジュールが生成され、仮想 HUB に接続される。Packet Adapter は VPN クライアントとの間では VPN プロトコルを用いてパケットを受渡しし、仮想 HUB との間ではメモリ上のキューを用いてパケットを受渡しする中間モジュールである。Packet Adapter の VPN クライアント寄りの部

分に、VPN プロトコルを送受信するためのモジュールがある。

仮想 HUB 上を流れるパケットすべてを監視し、定義されたセキュリティポリシーに従ってフィルタリングを行ったり、パケットログを作成したりするためのセキュリティポリシーモジュールがある。セキュリティポリシーはユーザ認証データベースモジュールと関連付けられている。これらのモジュールに対するデータの読み書きは、VPN サーバの管理者によって RPC モジュールを介して指示される。

仮想 HUB

SoftEther VPN Server 3.0 には複数の仮想 HUB を作成することができる。それぞれの仮想 HUB は、Ethernet スイッチと同様に MAC アドレス学習を行う FDB を持ち、仮想 HUB に現在接続されているエンティティは「VPN セッション」と呼ばれる。

仮想 HUB には色々な種類の VPN セッションを接続することができる。SoftEther VPN Client をインストールしたクライアントコンピュータが SSL ベースのトンネルを VPN Server との間で接続し、そのトンネルの VPN Server 側の終点が VPN Server 内で仮想 HUB に接続している形が VPN セッションの基本形である。これ以外にも、たとえば VPN Server を動作させるサーバコンピュータの物理的な LAN カードと仮想 HUB との間でブリッジを作成する「ローカルブリッジ機能」がある。他にも、仮想 HUB 内で仮想的な NAT や DHCP サーバを動作させる「SecureNAT 機能」、仮想的な IP ルータを動作させる「仮想レイヤ 3 スイッチ機能」などがある。これらの機能はモジュールとして実装されており、仮想 HUB に対して VPN セッションの形でリンクしている。VPN サーバプログラムの内部でこのようなリンクを動的に作成、削除することができるようにするため、「Packet Adapter」というモジュールが用意されている。

Packet Adapter は、物理的なスイッチング HUB などでスイッチング用 IC に対してスター型に配線されている、LAN ポート側にある PHY (PHYceiver chip) と呼ばれる A/D コンバータ IC のような役割を仮想 HUB で実現したものである。物理的なスイッチング HUB では、10Base-T、100Base-TX、1000Base-T、1000Base-SX、1000Base-LX などの複数の異なる種類の LAN ポートが存在し、それぞれに対応した PHY が用意されている。この PHY はスイッチング用 IC に対しては共通のデジタル信号規格でパケットの受渡しを行い、LAN ポートに接続された機器に対してはインターフェイス固有の電気的規格でパケットの受渡しを行う。

仮想 HUB に複数の VPN セッションが接続されているとき、仮想 HUB はそれぞれの VPN セッションの Packet Adapter から受取ったパケットに対して、ヘッダを解析する。パケットフィルタ、セキュリティポリシーの設定が VPN セッションに対して適用されている場合は、それらのポリシーを適用してパケットを破棄することがある。パケットが破棄されなかった場合は、パケットログの設定によってパケットのヘッダがログファイルに記録される (ログファイルへの記録はディスクへの書き込み操作を伴うため、メモリ内に一端キューイングされ、「遅延ライタ」という別スレッドで動作するプログラムによって遅延書き込みされる)。これらの前処理が完了した後、仮想 HUB はそのパケットの宛先の MAC アドレスを仮想 HUB 内の FDB と照合し、一致する宛先 VPN セッションの受信キューに挿入する。これは、物理的なスイッチがポート間のパケット交換を「ストア・アンド・フォワード」により受け渡しすることと同様である。

仮想 HUB は上記のような処理に必要な情報、およびユーザ認証等の設定を保持するために、状態を持っている。状態のうち保存する必要のあるもの (たとえば仮想 HUB 単位、ユーザ単位のトラフィックなどの統計データ等) は定期的にディスクに保存され、VPN Server が停止した場合は次回起動時にロードされる。

VPN Client からの VPN セッションの受け付け

SoftEther VPN Client をインストールしたクライアントコンピュータは、SoftEther VPN Server 3.0 が動作するサーバコンピュータの TCP ポートに対して VPN セッションを確立しようとする。この際、VPN Server 側では管理者が設定した TCP ポートが Listen 状態となっている。複数の TCP ポートで Listen することができる。VPN Client はそのポートのいずれかを宛先としてまず TCP コネクションを確立し、次にその TCP コネクション内で SSL セッショ

ンを確立する。SSL セッションが確立したら、HTTP/1.1 POST メソッドを用いて認証データを VPN Server に送信し、VPN Server はユーザ認証を行う。この際に、パスワード本体をネットワーク上に流さないようにするために SHA を用いたチャレンジ・アンド・レスポンス認証を行う。

VPN Server 側では、ユーザ認証を実施し成功した場合は、仮想 HUB のユーザ認証データベースにユーザごとに登録されているセキュリティポリシーをロードし、そのセキュリティポリシーを設定した新しい VPN セッションオブジェクトを作成する。VPN セッションオブジェクトには Packet Adapter インターフェイス (実装上はコールバック関数を持つ関数ポインタ) を登録する。この際に登録されるコールバック関数は、VPN Client との間のパケットの受け渡しを行うための通信モジュールに登録される。

VPN 通信中のパケットの送受信

仮想 HUB に複数の VPN セッションが接続されているとき、仮想 HUB は VPN セッション間のパケットの受け渡しを担当する。一方、VPN セッションが物理的なネットワークを経由して遠隔地にある VPN Client との間でトンネリング通信を行うための処理は、Packet Adapter が行う。

仮想 HUB の行うべき処理も、Packet Adapter が行うべき処理も、いずれもそれぞれ単体で見れば単純なループ処理である。そして、そのループ内では簡単な状態遷移の処理が行われる。たとえば、SSL の内部で実装されている VPN 通信処理では、まず次のパケットのバイト数が整数値で記載されたビットがネットワーク上を流れ、次にパケットの本体が流れる。VPN Server の実装ではこの処理の効率を高めるために非同期ソケットを用いている。非同期ソケットを用いる場合は、非同期ソケットの送受信 API の呼出元のプログラムがどこまで送受信が完了したのかといった状態を覚えておく必要がある。このような状態管理が発生するプログラムを書くとき、2 つの異なる目的のプログラムを同時に実装しようとするとプログラムが複雑になり、バグが多くなる。そこで、仮想 HUB 側の処理と VPN セッション (Packet Adapter) 側の処理とを分離し、モジュール化している。モジュール化によりバグが発生する可能性が低下するだけではなく、将来、Packet Adapter のインターフェイスを実装する他の種類の VPN プロトコルを実装したいと思ったときに素早くそのような拡張が可能であるという利点が生じる。

RPC による管理とモニタリング

1 個の VPN Server には複数の仮想 HUB があり、それぞれの仮想 HUB にはユーザ認証データベースの中に複数のユーザがある。そして、それらのユーザのいずれかによって認証された VPN セッションが仮想 HUB に接続されており、仮想 HUB は FDB 内のエントリを VPN セッションに関連付けて管理している。仮想 HUB にはユーザ認証データベースの他にも、パケットフィルタの設定、パケットログの設定などのデータベースや、これまでの通信統計データを記録するためのデータベースが含まれている。

ネットワーク管理者はこれらの状態をモニタリングしたり、データベースを管理したり (たとえばユーザの追加、削除など)、動作中の仮想 HUB に対して能動的な命令を出したり (たとえば接続中の VPN セッションの強制切断など) するタスクを実行することが必要な場合がある。このようなタスクを受け付けるために、VPN Server は Remote Procedure Call (RPC) インターフェイスをネットワークに対して公開している。

VPN Server に対して RPC でリクエストを出す場合は、命令名とパラメータを送信する。VPN Server はリクエストを処理し、結果としてエラーコードを返す。結果には追加のデータが含まれる場合がある。

複数のネットワーク管理者が多数の管理端末から同時に 1 台の VPN Server に接続することができる。RPC によるリクエストを送信するためには、最初に VPN Server に管理モードという特別なセッションで接続する必要がある。管理モードには「VPN Server 全体の管理モード」と「仮想 HUB ごとの管理モード」の 2 種類がある。VPN Server 全体の管理モードは root 権限のようなものであり、すべての管理操作が可能である。仮想 HUB ごとの管理モードは、あらかじめ VPN Server 全体の管理者から権限を委任されている個別の仮想 HUB しか管理することはできない。

RPC のリクエストには、結果がすぐに返るものと、少し時間がかかるものの 2 種類がある。たとえば、既存のユー

ザの作成・削除などの命令は、メモリ内でのみ処理が完結する処理である。ユーザの作成命令が届いた場合は、まずどの仮想 HUB に対するものであるかを判別し、動作中の仮想 HUB のオブジェクトを取得する。そのオブジェクトの中のユーザ認証データベースを編集するために、データベース全体をロックする。そしてユーザオブジェクトを新規作成してデータベースに登録する。この際、同一 ID のユーザがすでに存在していないかどうかなどのチェックを行う。データベースに登録が完了したらロックを解除し、RPC 呼出元に対して成功のコードを返す。

VPN セッションの削除のように、実行するためにネットワーク通信を伴う RPC リクエストもある。この場合は、RPC はそのリクエストが完了するまでブロックする。リクエストが完了するまでの間は同じ管理モードのセッションから届いた別の RPC リクエストはキューに入れられ、処理中のリクエストが完了するまで一時待機させられる。また、別々の管理モードのセッションから届いた RPC リクエスト同士が競合する場合は、いずれか早く到着したものが処理されるまで、残りのリクエストはキューに入れられ待機されられる。

参照ポインタによるガベージコレクション

前述したような処理を行うために VPN Server プログラムは内部で動的に多数のオブジェクトを作成する。これらのオブジェクトの解放忘れがあると、メモリリークが発生するだけではなく、ソケットやカーネルモードのオブジェクトのハンドルなどを維持したままとなっているためリソースリークが発生する。たとえば、VPN Server では管理者は仮想 HUB をいつでも作成し、削除することができる。仮想 HUB を 1 個作成してすぐに削除するという処理を何度繰り返されたとしても、その完了後はメモリやリソースは全く消費していない状態にする必要がある。

しかし、VPN Server 内部のオブジェクトの依存関係はとても複雑である。たとえば仮想 HUB のオブジェクトがある場合、ある実行中の RPC のリクエストがその仮想 HUB を参照しており、別の VPN セッションもその仮想 HUB を参照していることがある。この RPC のリクエストが完了するか、VPN セッションが切断されるかのどちらが先に発生するかは分からない。この状態で、管理者が VPN Server から仮想 HUB を削除する操作を別の RPC リクエストとして呼び出したとする。このとき、仮想 HUB が VPN Server の持っている仮想 HUB のリストから削除される瞬間に仮想 HUB オブジェクトを解放すると、仮想 HUB を参照していた RPC リクエストも、VPN セッションも無効なポインタに参照してクラッシュしてしまう可能性がある。これを避けるために、仮想 HUB の削除処理はその仮想 HUB を現在参照している他のすべての処理が終了するまで待機するというロジックを書くこともできるが、プログラムが複雑になりバグが生じる可能性が高くなる。

そこで、VPN Server には参照ポインタによる簡易的なガベージコレクションを組み込んである。オブジェクトの構造体のヘッダに参照ポインタがあり、カウンタが 0 になると初めてクリーンアップ処理が呼ばれ、オブジェクト内の共有されている可能性があるリソースが解放される。参照ポインタの増加、不要となった場合の解放 (参照ポインタの減少) はすべて明示的に書く必要がある。参照ポインタはスレッドセーフである。

OS 抽象化レイヤ

前述の様々な処理をプログラムする場合に、1 度プログラムを書けばできるだけ多くの OS でそのまま動作することが望ましい。そこで、VPN Server は最下層に OS 抽象化レイヤを設けている。OS 抽象化レイヤは OS の提供するシステムコールや API を呼び出す。OS 抽象化レイヤを介することなく、VPN Server の機能を実装するプログラムコードから直接システムコールや API を呼び出すことはできる限り行わない方針で実装されている (図 25)。

OS 抽象化レイヤは、Windows、Linux、Mac OS X、FreeBSD、Solaris の 5 種類の OS 用に実装されている。Windows とそれ以外の UNIX 互換系 OS とは大きく異なるが、UNIX 互換系 OS でもそれぞれ細かな点で違いがある。特にソケット関係と物理的な LAN カードへの低レベルでの読み書き関係のシステムコールが異なる。

OS 抽象化レイヤがあることにより、SoftEther VPN Server はソースコードをビルドする際にマクロ定義を用いて切替えるだけで、上記 5 種類のいずれの OS 用にもビルドすることができる。

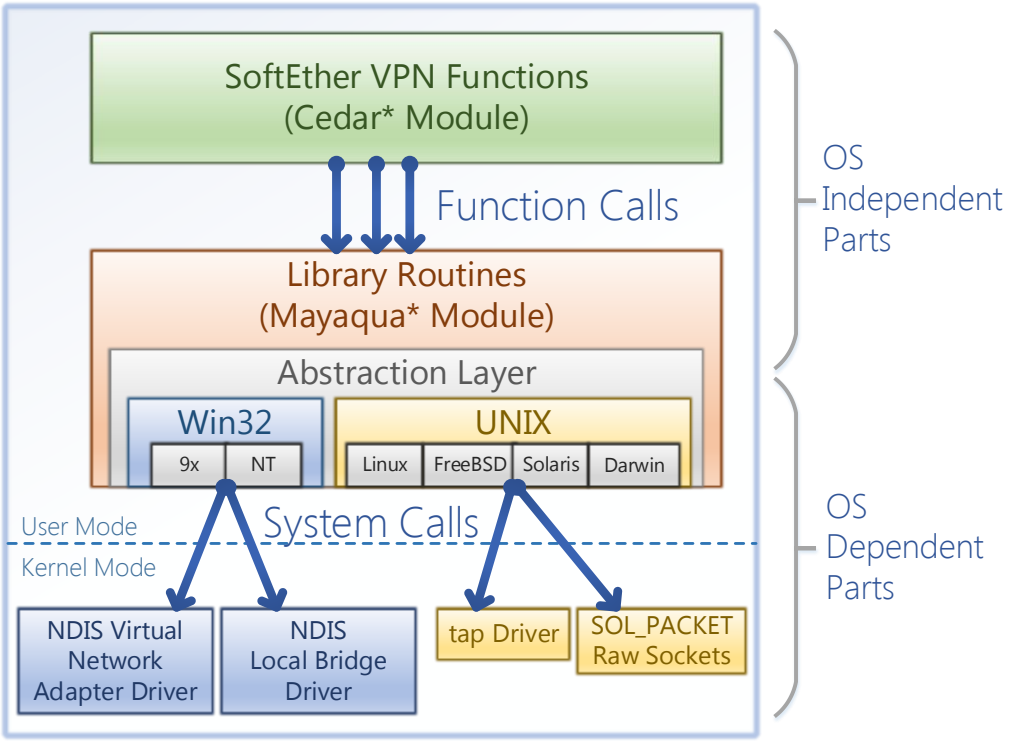


図 25. SoftEther VPN Server の内部構造‡

5.3. SoftEther VPN 4.0 で実装する VPN プロトコルモジュール

本研究では、L2TP/IPsec、SSTP、OpenVPN (L3)、OpenVPN (L2)、L2TPv3/IPsec および EtherIP/IPsec の 6 種類の新しい VPN プロトコルを VPN Server に実装する。

VPN Server は従来のバージョン (2.0 および 3.0) では SoftEther VPN Protocol (SSL ベースの Ethernet over TCP/IP) にのみ対応している。このバージョンに新たに前記 6 種類の VPN プロトコルを追加するためには、できるかぎり既存のコードを書き換えないことが望ましい。そのために、これらの新しい 6 種類の VPN プロトコルそれぞれについて「VPN プロトコルモジュール」を実装することにする (図 26)。

VPN プロトコルモジュールは、VPN クライアントからの VPN 接続要求を受け付け、ユーザ認証およびネゴシエーションを行い、これらが成功した後は VPN 接続が切断されるまでの間、仮想 HUB に対して VPN セッションとしてを確立し維持する。リンクが維持されている間は、VPN クライアントからのパケットは VPN プロトコルモジュールによってカプセル化以上されて仮想 HUB に投入される。仮想 HUB からその VPN クライアント宛として出てきたパケットは VPN プロトコルモジュールによってカプセル化されて VPN クライアントに伝送される。

VPN プロトコルモジュールの動作を詳述する。

段階 1. 新しい VPN クライアントの受付

自身が処理することが可能な VPN プロトコルによる新しい VPN クライアントが接続してくるまで待機する。新しい VPN クライアントが接続してきた場合は、新しいコネクションオブジェクトをメモリ上に作成し、VPN クライアントのエンドポイント情報を記録する。以後、同一の VPN クライアントからパケットが届いた場合はエンドポイント情報が一致するコネクションオブジェクトを検索することができるようになる。

新しい VPN クライアントの接続要求の受付処理が開始され、コネクションオブジェクトが作成された後も、次の新しい VPN クライアントが接続してくることを待機するために受付処理は継続する。

---

* 「Cedar」、「Mayaqua」はモジュールの名称である。

段階 2. VPN プロトコルに基づくネゴシエーションとユーザ認証

VPN プロトコルに基づくネゴシエーションを実施する。たとえば L2TP/IPsec、L2TPv3/IPsec および EtherIP/IPsec ではまず IPsec によるネゴシエーションが必要である。2.5 で述べたように、IPsec では SA を作成し、以後は SA の内部でデータを送受信する。詳しくは SA には IKE SA と IPsec SA の 2 種類がある。IKE SA は鍵交換を行うためのトンネルであり、IPsec SA は鍵交換が完了した後のユーザパケットを送受信するためのトンネルである。

IPsec が必要なプロトコルでは、IPsec によるネゴシエーションが完了した後に VPN プロトコル固有のネゴシエーション処理を行う (IPsec が不要なプロトコルでは、IPsec のネゴシエーションは省略される)。VPN 固有のネゴシエーションは、プロトコルによって異なる。共通的には、使用するセッション ID を取り決めたり、キープアライブパケットの送信間隔、タイムアウト秒数などを決定したりする。

ネゴシエーションが完了したら、VPN プロトコルごとに固有の手順でユーザ認証を行う。L2TP および SSTP では PPP を用いてユーザ認証を行い、PPP の内部では MS-CHAPv2 や PAP を用いる。OpenVPN ではトンネル内に流れる暗号化されたユーザ名とパスワードを用いる。L2TPv3 および EtherIP では、IPsec の SA ですでに交換された ID ペイロードを用いて認証を行う。

VPN プロトコルモジュールは、VPN クライアントからのユーザ認証データを受付けるが、単独ではユーザ認証を行うことはできない。ユーザ認証データベースは接続先として指定された仮想 HUB が保有しており、仮想 HUB に対する照会が必要である。仮想 HUB に対してユーザ認証データを転送し、仮想 HUB が認証の成功・失敗を決定する。仮想 HUB のユーザ認証データベースの設定によっては、ユーザは RADIUS または Active Directory などの外部認証サーバによって認証されることになっている。

このような複雑な認証方式について、VPN プロトコルモジュールは知る必要がない。VPN プロトコルモジュールは、VPN プロトコルの規約に従って VPN クライアントに対してユーザ認証データを要求し、VPN クライアントから応答されたユーザ認証データの生の値を仮想 HUB に渡すだけでよい。認証方法として VPN クライアントから PAP のデータが提示された場合、PAP にはユーザ名とパスワードのクリアテキストが含まれるので、これをそのまま仮想 HUB に渡す。仮想 HUB は当該データをそのまま外部認証サーバに渡すことで認証が完了する。しかし、大半の VPN クライアントはセキュリティを高めるためにデフォルトでは PAP の使用を拒否し、代わりに MS-CHAPv2 の使用を要求してくる。この場合、VPN プロトコルモジュールはまず MS-CHAPv2 の規約に従い乱数 (チャレンジ) を生成し、これを VPN クライアントに渡す[§]。VPN クライアントは対応するクライアントレスポンスを VPN プロトコルモジュールに渡す。VPN プロトコルモジュールは、先般送信したチャレンジと受け取ったクライアントレスポンスの組をそのまま仮想 HUB に渡す。仮想 HUB は当該データをそのまま外部認証サーバに渡す。この際に、仮想 HUB は Windows NT 4.0 の SAM (Security Account Manager) を用いて通信する方式、Windows 2000 以降の Active Directory を用いて通信する方式、および RADIUS プロトコルにおける RFC2548 (Microsoft Vendor-specific RADIUS Attributes) を用いて RADIUS 通信を行う方式の 3 方式をサポートする。外部認証サーバからはサーバーレスポンスを仮想 HUB に返すので、仮想 HUB はこれを VPN プロトコルモジュールに渡し、VPN プロトコルモジュールは VPN クライアントに渡す。このようにして MS-CHAPv2 の認証が可能になる (MS-CHAPv2 を用いたユーザ認証の場合は、CHAP 認証を用いた場合と異なり、外部認証サーバ上でユーザごとのパスワードの平文を保持しておく必要がなく、パスワードの MD4 ハッシュ値のみで足りる) 。

なお、VPN クライアント側で MS-CHAPv2 による認証を要求した場合で、外部認証サーバが MS-CHAPv2 に対応していない旨の応答を仮想 HUB から受けた VPN プロトコルモジュールは、一旦認証を拒否し、改めて PAP を使用して再度認証するよう VPN クライアントに対して要請する。

仮想 HUB は、ユーザ認証に成功したときには VPN セッションオブジェクトを新たに作成する。VPN セッションオブジェクトは VPN プロトコルモジュールに渡される。

[§] MS-CHAPv2 の乱数 (チャレンジ) は、RADIUS および Active Directory のいずれの場合でも認証サーバを呼び出す側 (本実装の場合は VPN サーバプログラム) が生成する必要がある。

段階 3. レイヤ変換器の作成と IP パラメータの VPN クライアントへの通知 (レイヤ 3 プロトコルのみ)

VPN プロトコルモジュールは、レイヤ 3 (IP) を対象とする VPN プロトコルの場合 (L2TP, SSTP, OpenVPN (L3) の場合)、ユーザ認証が完了したら直ちに VPN プロトコルモジュールは 4.4 節から 4.6 節で述べたレイヤ変換器のインスタンスを作成する。レイヤ変換器は仮想 HUB から渡された VPN セッションを上位レイヤ、VPN プロトコルモジュールを下位レイヤとする上下 2 個のインターフェイスを持つ。

レイヤ変換器を作成すると、VPN プロトコルモジュールはレイヤ変換器に対して IP アドレスの取得を要求する。IP アドレスはレイヤ変換器が DHCP クライアントとなって仮想 HUB 上の既設の DHCP サーバに対して要求される。DHCP サーバからの IP アドレスの割り当てを受けることができなかった場合、またはタイムアウトが発生した場合は、VPN 接続を中断する (VPN クライアントにはエラーコードが返される)。DHCP サーバからの IP アドレスの割り当てを受けることができた場合は、VPN クライアントに対して、IP パラメータ (4.5 節) が通知される。

レイヤ 2 (Ethernet) を対象とする VPN プロトコル (L2TPv3, EtherIP, OpenVPN (L2) の場合) については、レイヤ変換器は不要なため、作成されない。

段階 4. VPN 通信処理

段階 3 までが完了すると、VPN 通信処理が可能な状態となる。

レイヤ 2 (Ethernet) を対象とする VPN プロトコルの場合は、VPN クライアントから受信してカプセル化解除された Ethernet フレームは仮想 HUB に対して VPN セッションオブジェクトに関数ポインタとして登録された Packet Adapter を経由して直接渡される。仮想 HUB から VPN セッションオブジェクトを経由して受取った Ethernet フレームは、カプセル化して VPN クライアントに送信される。

レイヤ 3 (IP) を対象とする VPN プロトコルの場合は、VPN クライアントから受信してカプセル化解除された IP パケットはレイヤ変換器により Ethernet フレームに変換されてから仮想 HUB に対して VPN セッションオブジェクトを経由して渡される。仮想 HUB から VPN セッションオブジェクトを経由して受取った Ethernet フレームは、レイヤ変換器により IP パケットに変換されてからカプセル化して VPN クライアントに送信される。

パケットの送受信がないときでも、VPN プロトコルモジュールは VPN クライアントからのキープアライブに対して応答したり、一定時間ごとにキープアライブを VPN クライアントに対して送信したりして、VPN クライアントとの間のトンネルが途中にある NAT などによって破棄されないように (もし破棄されたり、VPN クライアントがクラッシュした場合などはこれを検出して VPN プロトコルモジュール側でもセッションを切断するように) メンテナンスする。

段階 5. VPN 切断処理

VPN プロトコルモジュールは、仮想 HUB 側から VPN セッションが切断されようとしているか、または VPN クライアント側から VPN セッションが切断されようとしているかのいずれかのイベントが発生すれば、直ちに切断・解放処理を行う。

仮想 HUB に対しては、レイヤ 3 (IP) を対象とする VPN プロトコルの場合は、レイヤ変換器を通じて DHCP サーバに対して IP アドレスを返却してから VPN セッションオブジェクトを解放する。レイヤ 2 (Ethernet) を対象とする VPN プロトコルの場合は特に何もせず VPN セッションオブジェクトを解放する。

VPN クライアントに対しては、プロトコルごとに定められた手順により論理的なトンネルを切断してから、プロトコルモジュール側の当該 VPN クライアントに対応したコンテキストのメモリを解放する。

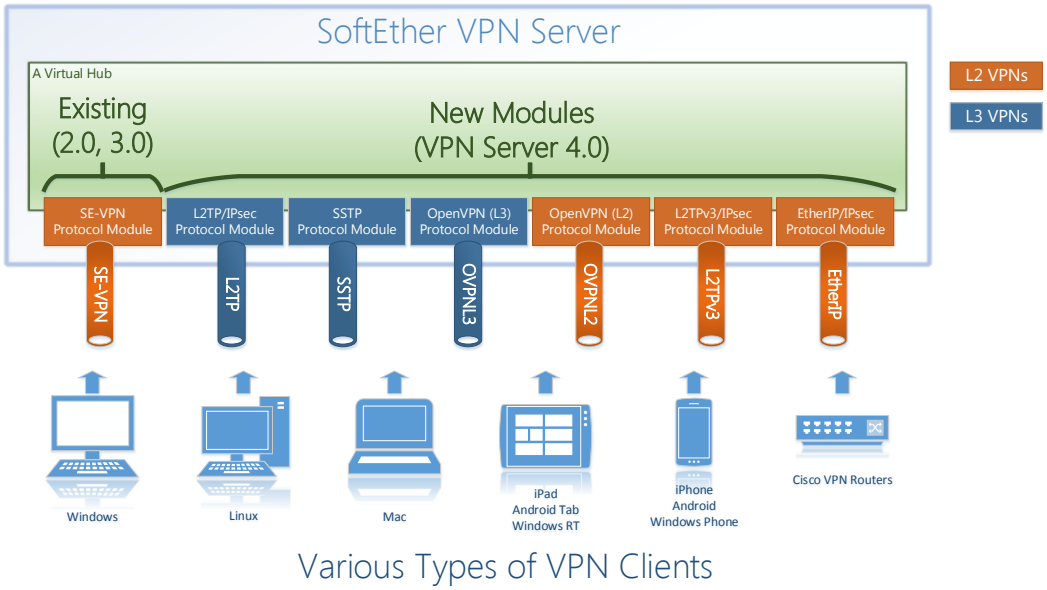


図 26. 既存の VPN モジュール 1 個と本研究で実装する VPN モジュール 6 個

5.4. サブモジュール化

VPN プロトコルモジュールは VPN プロトコルの種類ごとにそれぞれ実装する必要がある。しかし、実際にはそれぞれの VPN プロトコルが必要とする処理は以下のように共通のものがある。そこで、VPN プロトコルモジュールの全体をいくつかの共通的なサブモジュールに分割し、各モジュール間を結合する手法により、共通の処理を統合し、全体のコード量を削減する。

VPN プロトコルモジュールを構成するサブモジュール間の階層関係を図 27 に示す。

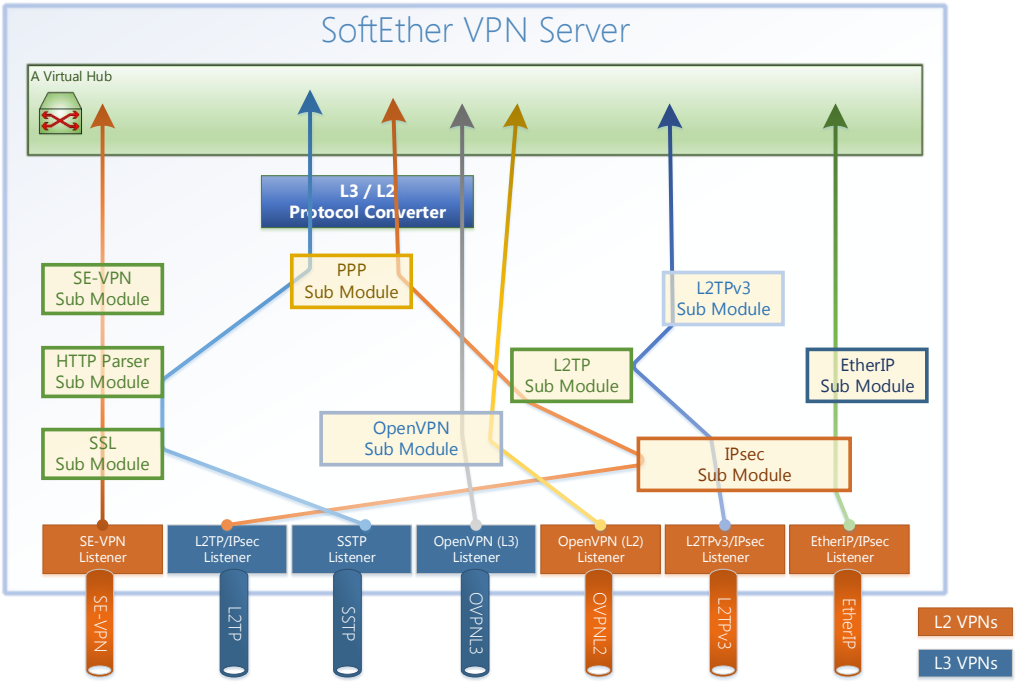


図 27. VPN プロトコルのサブモジュール間の階層構造

IPsec サブモジュール

IPsec は L2TP/IPsec、L2TPv3/IPsec および EtherIP/IPsec の 3 つの VPN プロトコルで使用されている。IPsec 処理をそれぞれのモジュールごとに実装するのは大きな労力を必要とする。そこで、IPsec パケットの送受信や

暗号化、IKE の状態管理などの処理を IPsec サブモジュールによって一元的に実装する。

今回の実装では、IPsec サブモジュールは IPsec バージョン 2 (IKE バージョン 1) にのみ対応する。IPsec バージョン 3 (IKE バージョン 2) は、VPN クライアント側の対応デバイスが少ないため実装しないことにした。

IPsec サブモジュールは、以下の拡張的な機能に対応する。

- RFC 3947 Negotiation of NAT-Traversal in the IKE[18]
- draft-ietf-ipsec-nat-t-ike[19]
- RFC 3706 A Traffic-Based Method of Detecting Dead Internet Key Exchange (IKE) Peers[20]

IPsec サブモジュールは、最大 30,000 本の SA を管理することができるように実装する。

IPsec の通信においては、まず UDP ポート 500 を用いてネゴシエーションが行われる。この際、IPsec サブモジュールはできる限り VPN クライアントに対して RFC 3947 Negotiation of NAT-Traversal in the IKE または draft-ietf-ipsec-nat-t-ike の規格に基づく ESP over UDP カプセル化をするよう要求する。VPN クライアントがこれに応じた場合は、IPsec のデータパケットの送受信は UDP ポート 4500 で行われる。そのため、すべての IPsec の通信は標準的な UDP ソケットのみを用いて行うことができる。

しかし、もし VPN クライアントがいずれの ESP over UDP カプセル化規格にも対応していない、または明示的にこれらの使用を拒否した場合 (たとえば物理ネットワークが IPv6 の場合) は、IPsec のデータパケットの送受信は UDP が使用されず、代わりに IP プロトコル番号 50 の IP パケットが使用される。この場合は OS に対して Raw IP ソケットの作成を要求する必要があり、少し複雑なプログラムを必要とする。

L2TP サブモジュール

L2TP/IPsec および L2TPv3/IPsec はいずれも、暗号化レイヤの上に L2TP レイヤがあり、L2TP の処理を必要とする。L2TP は 1 本の UDP などで構成されたデータグラムベースのコネクション上で、従来の電話回線やダイヤルアップ回線のような仮想的なトンネルを複数本構築するためのプロトコルである。

L2TP は一見すると最初に単純なネゴシエーションを行い、ネゴシエーションの結果トンネルが確立されればその後は各トンネルの先頭に簡単なヘッダを付加するだけであり、実装が容易であるように感じられる。しかし、実際に実装をしようとすると、当初の想定に反して複雑である。まず、L2TP のパケットを送受信するために「L2TP トンネル」を確立する必要があるが、L2TP トンネルの確立は TCP における 3 ウェイ・ハンドシェイク (SYN, SYN+ACK, ACK) と同様に、Start-Control-Connection-Request, Start-Control-Connection-Reply, Start-Control-Connection-Connected の 3 種類のメッセージを合計 3 回送受信させる必要がある。この際に、メッセージが消失した場合に備え再送タイマを用いた自動再送を行う必要がある。L2TP トンネルが確立した後でも、VPN クライアントは 2 本目、3 本目の L2TP トンネルを確立することができる規格になっている。実際のところ、1 本の IPsec トンネル内に 2 本以上の L2TP トンネルを確立する VPN クライアントはこれまで知られていないが、プロトコルの規格上はそのようなことができるから、サーバを実装する側としてはそのような可能性を考慮して複数の L2TP トンネルに対応しなければならない。

L2TP トンネルが確立されると、次はその L2TP トンネルを用いて再度 3 ウェイ・ハンドシェイクのためのメッセージ (Incoming-Call-Request, Incoming-Call-Reply, Incoming-Call-Connected) とメッセージに付随するパラメータを用いて、「L2TP セッション」を確立する必要がある。この際には L2TP トンネルが実装する保証されたメッセージ到達機構を利用する必要がある。つまり、TCP に実装されているような「シーケンス番号」、「受信ウインドウサイズ」、「再送タイマ」と同様の処理を L2TP でも実装しなければならない。

規格上、「L2TP セッション」は 1 本の「L2TP トンネル」内に複数本確立することができる。これについても、1 本の「L2TP トンネル」内に複数本の L2TP セッションを確立するような VPN クライアントはこれまで知られていないが、プロトコルの規格上はそのようなことができるから、サーバを実装する側としてはそのような可能性を考慮して複数の L2TP セッションに対応しなければならない。

L2TP トンネルが確立されれば、ようやく L2TP を用いて上位レイヤ (PPP フレーム、L2TPv3 のフレームなど) を送受信することができるようになる。L2TP はトンネルが確立されるまでの手順がとても複雑であり、サブモジュール化して実装することが必須である。

IPsec や PPP などに関する実装上参考になる解説は、日本語・英語で書籍や Web サイトなどで手に入れることができる。一方、L2TP に関する実装上参考になる解説は、英語でさえもほとんど無い。そこで、信頼できる資料として RFC に頼ることとなる。しかし、実際には Cisco 社の VPN 対応ルータが VPN クライアントとして接続してくる際に投げられる L2TP パケットには RFC にないフィールドが多数存在し、それらを適切に処理しなければならないといったことが判明した。そこで本研究では L2TP を実装する上では実際の L2TP 対応クライアントの挙動を解析しながら進めることにした。

PPP サブモジュール

L2TP/IPsec と SSTP は、暗号化レイヤの上に PPP レイヤが載っているため、PPP の処理を必要とする。PPP については L2TP と異なりそれほど複雑な処理は必要ないと当初は考えた。しかし、実際には実装には下記のような労力を要した。

PPP はまず簡単なネゴシエーションを行い、使用するユーザ認証プロトコルを決定する。ほぼすべての L2TP/IPsec や SSTP 対応の VPN クライアントで使用できる PPP 上でのユーザ認証プロトコルは PAP (Password Authentication Protocol)[21] と MS-CHAPv2 (Microsoft Challenge and Response Authentication Protocl Version 2.0)[22] の 2 種類がある。PAP の実装はとても簡単である。しかし、PAP はパスワードが復号化可能な形でネットワーク上を流れるのでセキュリティ上良くない。そのため、Windows に付属の VPN クライアントでは PAP がデフォルトで無効化されている。また、Windows Mobile に付属の VPN クライアントには PAP が実装されていない。これらの理由から、MS-CHAPv2 の実装は必須である。

MS-CHAPv2 を VPN Server 側でサポートするためには、仮想 HUB のユーザ認証データベースを拡張する必要がある。従来のバージョンの VPN Server では、ユーザのパスワードは SHA によって暗号化されて格納されていた。しかし、MS-CHAPv2 ではパスワードは平文または MD4 によって暗号化されて保存されていることが要求される。平文でパスワードを保存することはセキュリティ上危険なため、仮想 HUB では新たにユーザのパスワードを MD4 で保存することとした。ところが、古いバージョンの VPN Server ですでに作成されているユーザがある場合は、今回の拡張で実装された VPN Server 4.0 にアップグレードしたときに古いバージョンのユーザは SHA で暗号化されたパスワードしか保持しておらず、MS-CHAPv2 認証をサポートすることができない。この場合はやむを得ず、仮想 HUB のログに、ユーザのパスワードを再設定するように指示するメッセージを記録することで管理者による運用で対応することにした。

VPN Server ではユーザ認証データベースにおいて、ユーザごとのパスワードを内部で保持せずに、外部の RADIUS サーバまたは Active Directory ドメインコントローラにユーザ認証を委任する設定も可能である。L2TP または SSTP を用いて接続してきたユーザが入力したユーザ名と暗号化されたパスワードを RADIUS サーバに対して送信し、結果を受け取ることは容易に実現できた。しかし、Active Directory ドメインコントローラ、または VPN Server を実行している Windows 2000 以降の Windows の LSA (Local Security Authority) に対して MS-CHAPv2 認証データを渡して認証を行ってもらう方法が不明であった。MSDN (Microsoft Developer Network) のドキュメントを調べたり、インターネット上で情報を収集したりしても方法が不明である。しかし、Microsoft 製の RRAS サーバや RADIUS サーバは確かに MS-CHAPv2 認証を LSA に対して要求していることが分かっている。そこで Windows Server 2003 に付属の「iassam.dll」および「raschap.dll」を逆アセンブルして動作を解析した結果、LsaLogonUser[23] という LSA の API に渡す構造体に、ドキュメントで公開されていない隠された方法で MS-CHAPv2 認証データを渡すことにより Windows に認証を要求することができることが判明した。本研究ではこの方法を用いた。

上記のような方法で PPP におけるユーザ認証が完了した後は、IPCP (IP Control Protocol)[23] を用いて IP アドレスを決定する。IP アドレスはレイヤ変換モジュールを経由して仮想 HUB から DHCP サーバ経由で確保しても

らう方法と、PPP クライアントが明示的に希望する IP アドレスの使用を要求する方法の 2 種類をサポートする。ただし、後者の方法はユーザごとのセキュリティポリシーで禁止することを可能とする。

OpenVPN サブモジュール

L2TP/IPsec の場合は、暗号化は IPsec、VPN 通信は PPP (L2TP 上の仮想トンネル) が処理することになっている。これと比較して、OpenVPN の場合は、暗号化も VPN 通信も両方とも OpenVPN, Inc. が策定した独自規格により処理される。

OpenVPN には L2、L3 の 2 つの動作モードがある。L2、L3 のモードの実装上の違いは少ないため、2 つのモードは単一の OpenVPN サブモジュールで処理することにする。また、OpenVPN は UDP、TCP のいずれの下位プロトコルでも通信が可能となっている。今回の実装では両方に対応する。

OpenVPN サブモジュールは、届いた OpenVPN パケットを解釈する。OpenVPN プロトコルは IPsec の IKE を参考にして設計されている。まず、VPN クライアントと VPN サーバとの間で 1 本のセッションを確立する。セッションの確立時には、SSL の仕組みを部分的に用いて鍵交換やユーザ認証を行う。

本来、SSL は TCP の上で動作させるものである。しかし、OpenVPN は当初は UDP プロトコルのみをサポートしていた。UDP はデータグラム型の伝送しかできず、データが途切れなくかつ順序に変更なく送受信されることを想定して設計されている SSL をそのまま伝送することができない。そこで、OpenVPN は UDP 上に TCP によく似たシーケンス番号、確認応答および再送のメカニズムを独自に実装している。この独自の再送可能プロトコル上にセグメント化された SSL ストリームを載せることで、OpenVPN は SSL を UDP 上で使用している。その後、OpenVPN は UDP のみではファイアウォールの通過が難しい環境のために、TCP をサポートした。TCP のサポートのため、OpenVPN は UDP パケットを単に順番に連結してそれを TCP コネクション上にストリームとして流す方針で実装を行った。そのため、本来 TCP 上にそのまま流せば良い (SSL over TCP) SSL のストリームが、SSL over OpenVPN-Reliable-Protocl UDP over TCP のように冗長な形で流れることになる。

1 本のセッションは、8 本のチャネルにより構成される。通常は 1 本しか使用されない。1 本のチャネルではパケットを送信する度に「パケット ID」と呼ばれるカウンタが 1 ずつインクリメントされていき、これが 0xF0000000 (10 進数で 4026531840) を超えた場合に次のチャネルが使用されるという仕組みになっている。1 パケットあたり 64 バイトの VPN 通信データが約 257Gbytes 流れたときにパケット ID が 0xF0000000 を超える可能性があるため、今回の実装では複数本のチャネルを正しくサポートする必要がある。

チャネルが確立されるときには、ユーザ認証を行う必要がある。OpenVPN は PPP のようにチャレンジ・アンド・レスポンス認証をサポートしておらず、クリアテキストのパスワードが前述した SSL ストリーム上を流れる形で VPN Server に届く。したがって、VPN Server はそのパスワードを単純に仮想 HUB に渡してユーザ認証を行ってもらえば良く、RADIUS や Active Directory を用いる場合であっても PPP のような複雑な処理は不要である。

5.5. モジュール間のパケットデータの受け渡しのためのプロセス内パイプ

今回の実装では、複数のサブモジュール間でパケットデータが流れることになるため、サブモジュール間をリンクする必要がある。

各サブモジュールのリンクの疎密の度合いは、パフォーマンスと拡張性に大きな影響を与える。密にリンクする場合、コード量は少なくなりパフォーマンスも向上するが、コードが分かりにくくなり、拡張性が低下したり、バグの原因となったりする。一方、疎にリンクする場合、コードが分かりやすくなり拡張性が向上するが、冗長となり、パフォーマンスも低下する。そのため、バランス良くリンクを設計することが肝要である。

OS の提供するパイプやソケットペアのオーバーヘッド

OS はプロセス間通信を行うためにパイプやソケットペアを提供する。UNIX 用プログラムではパイプやソケット

ペアは複数のプロセス間の連携に広く使われている。パイプを用いてモジュール間のパケットの受け渡しをすれば、モジュール間の結合は疎になり、相互依存性が低下する。しかし、今回の VPN Server の実装では 1 個のプロセス内ですべての処理を行うため、OS の提供するプロセス間通信の機構を用いることは冗長となる。パイプに何らかのメッセージを書き込む際にシステムコールが 1 回呼び出され、読み出す際にももう 1 回呼び出されることになるためである。また、パイプ内のキューはカーネルが管理するメモリ上に置かれているため、書き込み、読み出しの際に合計 2 回、メモリコピーが発生する。

Tube

そこで、今回の実装では OS の提供するパイプと似た機能を持つ「Tube」と呼ばれる仕組みを実装し、モジュール間のパケットの受け渡しに使用することにした。Tube は書き込まれたデータを順番に読み出すことができるキューの構造をしている。TCP ソケットと似ているが、UDP ソケットのように書き込まれたデータは自動的に結合されることなく、分割されたままでキューに格納される。さらに、UDP ソケットと異なり、TCP ソケットのようにデータの順序と到達性を保証する。

Tube に対してデータ 1 個を書き込む場合は、書き込むデータのバッファのアドレスとサイズを指定する。書き込む際に、バッファの内容をコピー (クローン) してからそのバッファを書き込むか、バッファをクローンせずに書き込むかを指定することができる。クローンしたバッファを書き込む場合はクローンに必要なオーバーヘッドが生じるが、書き込み側はそのバッファを再利用することができる。バッファをクローンせずに書き込む場合、書き込み側はそのバッファを二度と利用することはできないが、コピーが発生しないため高速である。

2 つのモジュール間でパケットを受け渡しする場合、両方のモジュールが同一のスレッドで動作している場合と、別々のスレッドで動作している場合の 2 通りがある。同一のスレッドで動作している場合は、Tube では特に同期メカニズムを用いる必要はない。同一スレッド内のあるコードが書き込みを実行した後に別のコードが読み出しを試行した際に単純にキューからデータを読み出せば良いためである。一方、別々のスレッドで動作している場合は、Tube に対してデータが書き込まれたことを、Tube が読み出し可能状態になることを待機しているもう一方にスレッドに通知する必要がある。この際には、OS の提供するスレッド間の同期機能が使用される (OS 間の違いは 5.2 節で述べた OS 抽象化レイヤによって吸収される)。2 つのモジュールが別々のスレッドで動作している場合のみ、スレッド間の同期やスレッドコンテキストスイッチが発生するが、それでも OS の提供するパイプを用いる場合より高速である。

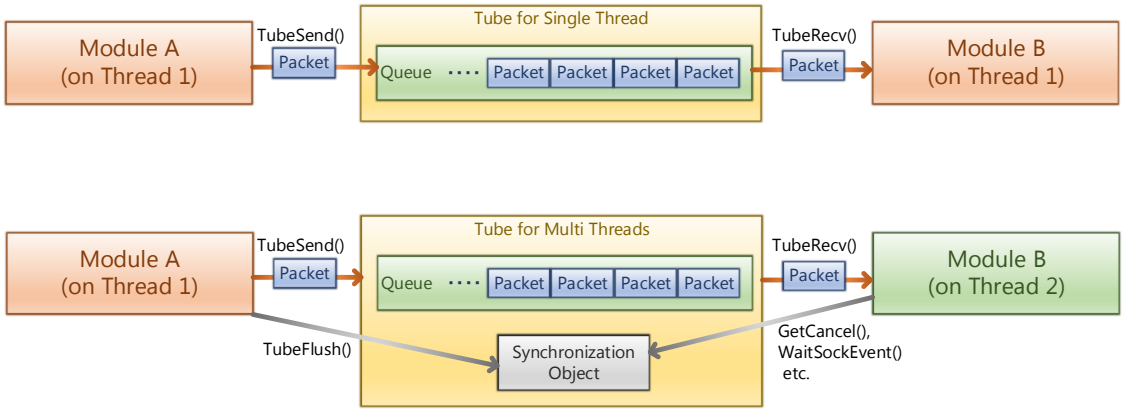


図 28. シングルスレッドおよびマルチスレッドにおける Tube の利用

双方向 Tube および TubePair

Tube は 1 個のキューを持っているため、2 つのモジュール間で一方向のデータの受け渡しが可能である。双方向でのデータの受け渡しのためには、双方向の Tube を作成する必要がある。

双方向の Tube を同時に作成したとき、「TubePair」という共有オブジェクトを両方の Tube に設定することができる。双方向の Tube は一端作成した後は分離して全く別のコードによって利用されるが、両方の Tube が参照する TubePair データのみは共通である。

TubePair は主に双方向の Tube のうちいずれか一方が切断された場合に、もう一方を自動的に切断するために使用される。TCP ソケットには shutdown などでソケットを切断すると、自分側でも相手方でもそれ以降は直ちに send と recv ができなくなるという性質がある。そのような性質を双方向の Tube で実現する。

InProc Socket

双方向 Tube の振る舞いはソケットに似ているが、Socket API を用いた送受信はできない。たとえばモジュールの一方が以前より TCP ソケットを用いて遠隔地のコンピュータとの間でデータを送受信するようなプログラムになっていたとする。このとき、遠隔地のコンピュータの代わりにプロセス内の別のモジュールを通信相手とするようなモジュール間リンクをしてパケットの受け渡しを行いたい場合、元のコードを大幅に書き換える必要が生じる。

これを避けるために、新たに InProc Socket (In-Process Socket) を実装する。InProc Socket は VPN Server のプロセス内でのみ使用することができるソケットの派生物であり、これまで使用していたソケットに対する send, recv、エンドポイント情報の取得などの各種の操作を、コードを変更せずに呼び出すことができる。同期ソケット、非同期ソケットの 2 種類のモードを切替えることができるため、元のコードが select や poll のようなソケットを待機する関数で新しいデータが届くまで待機状態とするような実装になっている場合でも、そのコードを変更することなく、そのコードの接続先をプロセス内の別のモジュールとすることができる。

5.6. 実装上の最適化

今回の実装では、通信速度を高速化し遅延を最小限にするため、以下のような最適化行う。

メモリコピーの回数の最小化

モジュール化、サブモジュール化を行う場合、モジュール間結合の箇所が多くなる。今回の実装では、たとえば L2TP/IPsec クライアントが仮想 HUB に対してパケットを送信するときは、UDP リスナモジュール、IPsec サブモジュール、L2TP サブモジュール、PPP サブモジュール、レイヤ変換器モジュール、VPN セッションモジュール、仮想 HUB モジュールのように 7 個のモジュール間でパケットが受け渡しされる。

受け渡しされるパケットが 1,500 バイトの場合、モジュール間でパケットの受け渡しが発生する度にメモリコピーを発生させれば 9,000 バイトものコピーが発生する。この他にも、暗号化・復号化のときには必ずメモリコピー以上のオーバーヘッドが発生する。暗号化・復号化の処理はやむを得ないとして、それ以外の場合はできる限りメモリコピーの回数を減らしたほうが良い。メモリコピーの回数を減らすことは、メモリの確保回数を減らすことにもつながる。

安易なメモリコピーは、モジュール間の境界をまたぐ直前または直後に発生することが多い。たとえば、パケットをより下位のレイヤのプロトコルでカプセル化する場合は、大抵の場合はカプセル化対象データの前にヘッダを挿入する。この際、熟考せずにプログラムを書くとメモリコピーが発生する。

メモリコピーの回数を削減するための方法として、BSD の mbuf、Linux の skbuff および Windows の NET_BUFFER_LIST, NET_BUFFER データ構造におけるコピー回数削減手法がある。これらの手法では、パケットをカプセル化する際に挿入すべきデータを別のメモリブロックとして確保し、ペイロードデータの直前にリンクする方法と、予め余分なメモリ領域を先頭に確保しておき、ペイロードデータを書き込むときにはその余分なメモリ領域の分だけオフセットを取って書き込む方法との 2 種類が組み合わされている。

前者の手法はたとえば TCP/IP スタックを記述するときのように、パケットデータのセグメンテーションや再送、再利用などが多く発生する場合に有効であるが、プログラムが複雑になる。パケットデータのセグメンテーションや再送、再利用などがあまり発生しない本実装では、より実装が単純な後者の方法を用いてメモリコピーの回数を

節約した。

逆に、下位のレイヤのプロトコルからカプセル化解除を行い中身のパケットを抜き出す場合も、メモリコピーや再確保を発生させないためには、抽出されるべきパケットのメモリブロックの先頭ポインタをそのまま上位レイヤに渡す。

スレッド間同期回数の最小化

モジュール間でパケットを受け渡す場合において、両方が同一スレッドに属する場合は、前述の Tube を使用することにより、どれだけ大量のパケットを受け渡しする場合でもシステムコール呼び出し回数は 0 回とすることができる。

一方、両モジュールをやむを得ず別々のスレッドで動作させる場合は、前述の Tube を使用する場合であっても、必ず Tube に対してデータが書き込まれた (読み出し可能になった) ことを通知するためのスレッド間同期メッセージを、スレッドライブラリを経由して他方のスレッドに送信する必要がある (待ち受け側スレッドでポーリングを使用すれば必要ないが、ポーリングの使用は論外である)。また、Tube にデータを書き込む際や読み出す際のキューはロックしなければならない。これらのマルチスレッド関係の処理はスレッドライブラリを呼び出すためオーバーヘッドがある。

このオーバーヘッドを避けるための手段の 1 つとして、パケットを 1 個ずつ受け渡す度に到着を通知するのではなく、ある程度の個数のパケットが溜まった時点で一気に受け渡すという手法を採ることができる。たいていの場合、VPN クライアントからは連続して多数の個数のパケットが届く。複数のパケットは全く同時に届く訳ではなく、実際には少しずつ時間差を置いて届けられる。しかしその時間差は、ドライバが物理的な LAN カードからパケットを汲み上げる処理を行う際に回すループより短い場合が多い。この場合、各パケットは同時にユーザモードのプロセスから汲み上げることができる。あるサブモジュールにとって、すでに複数の処理すべきパケットが到着していることが判明しているときに、先頭のパケットから順に処理して 1 個ずつ Tube に書き込むとその回数だけスレッドライブラリを呼び出すことにより、オーバーヘッドが大きくなる。そこで、できる限り多数個のパケットをまとめて Tube に書き込み、最後に 1 回だけスレッドライブラリを呼び出すことにより、オーバーヘッドを削減することができる。

AES 暗号化・復号化におけるハードウェアアクセラレーションの使用

VPN クライアントは、VPN Server がサポートしている任意のプロトコルから 1 つプロトコルを指定してそれを用いて VPN 通信をすることを要求できる。ほとんどの VPN クライアントは AES-128bit を要求してくる (Windows XP やそれ以前の L2TP/IPsec クライアントは 3DES を要求してくる)。AES の処理は重いが、最近の Intel のプロセッサは AES-NI (Advanced Encryption Standard New Instructions)[25] 命令が実装されているため、利用可能な場合は AES-NI を用いて暗号化・復号化を行う。

5.7. プログラミング

SoftEther VPN Server 3.0 に対して新たに上述のモジュールを追加することにより実装を行った。実装は C 言語で行った。実装したソースコードの一覧はのとおりである。

表 4. 本研究で実装したソースコードの一覧

モジュール	ソースコード名	行数
レイヤ変換器モジュール	IPsec_IPC.c IPsec_IPC.h	2,117 行
IPsec モジュール	IPsec.c IPsec.h IPsec_IKE.c IPsec_IKE.h IPsec_IkePacket.c IPsec_IkePacket.h	10,742 行
L2TP モジュール	IPsec_L2TP.c IPsec_L2TP.h	2,695 行
PPP モジュール	IPsec_PPP.c IPsec_PPP.h	2,848 行
EtherIP モジュール	IPsec_EtherIP.c IPsec_EtherIP.h	541 行
SSTP モジュール	Interop_SSTP.c Interop_SSTP.h	1,281 行
OpenVPN モジュール	Interop_OpenVPN.c Interop_OpenVPN.h	3,078 行
Windows Vista / 7 / 8 用 カーネルモードフィルタドライバ	IPsec_Win7.c IPsec_Win7.h IPsec_Win7Inner.h Wfp.c Wfp.h	1,847 行
合計		25,149 行

第 6 章 評価

第 5 章では VPN サーバプログラムの実装について述べた。第 6 章では実装した VPN サーバプログラムが第 3 章で述べた従来手法における問題点を解決することができたかを検証する。

6.1. 従来の問題点が解決されたことの検証

複数の VPN プロトコルの VPN クライアントからの通信を 1 台の VPN サーバコンピュータで受付ける場合、複数の VPN サーバプログラムを同時に 1 台のコンピュータで組み合わせて動作させる手法と、本研究の提案するように 1 個の VPN サーバプログラムにまとめて実装する手法とがある。

第 5 章で実装したプログラムによって、第 3 章で述べた従来手法の各問題点が解決されているかどうかを実験によって確かめる必要がある。検証結果は以下のとおりである。

機能の検証 (手元環境)

実装した VPN サーバプログラムに対し、下表のような様々な VPN クライアントソフトウェアまたはデバイスを用いて多様な VPN プロトコルによる接続と通信が行えるかどうかを検証した結果、のとおりとなった。

表 5. 各 VPN プロトコルおよび VPN 位案デバイスからの接続検証結果

VPN プロトコル	接続元 VPN クライアント	結果
L2TP/IPsec	iPhone (iOS 4.x, 5.x, 6.x)	○
	iPad (iOS 4.x, 5.x, 6.x)	○
	Android (2.x, 3.x, 4.x)	○
	Windows XP, Vista, 7, 8, RT	○
	Mac OS X (10.6, 10.7, 10.8)	○
SSTP	Windows Vista, 7, 8, RT	○
OpenVPN (L3)	Windows 版 OpenVPN Linux 版 OpenVPN	○
L2TPv3/IPsec	Cisco 892J	○
	Cisco 1812J	○
EtherIP/IPsec	NEC IX2015	○
OpenVPN (L2)	Windows 版 OpenVPN 2.2 Linux 版 OpenVPN 2.2	○

機能の検証 (多くのユーザを募集して検証)

実装した VPN サーバプログラムの各機能にはバグがある可能性がある。深刻なバグが存在していないかどうかを検証する (バグが発見された場合は修正する) ためには、インターネットを利用してできるだけ多くのユーザを募集して動作テストをしてもらうことが効率的である。

そこで、2012 年 9 月 1 日頃から Twitter や Facebook を利用してインターネット上のユーザに対して SoftEther VPN Server 4.0 のテスト版を試しに使ってもらうよう依頼した。2012 年 11 月 26 日からは Web サイトでも広報を行った。その結果のユーザ数 (VPN Server のインストール台数) は図 29 のとおりである。

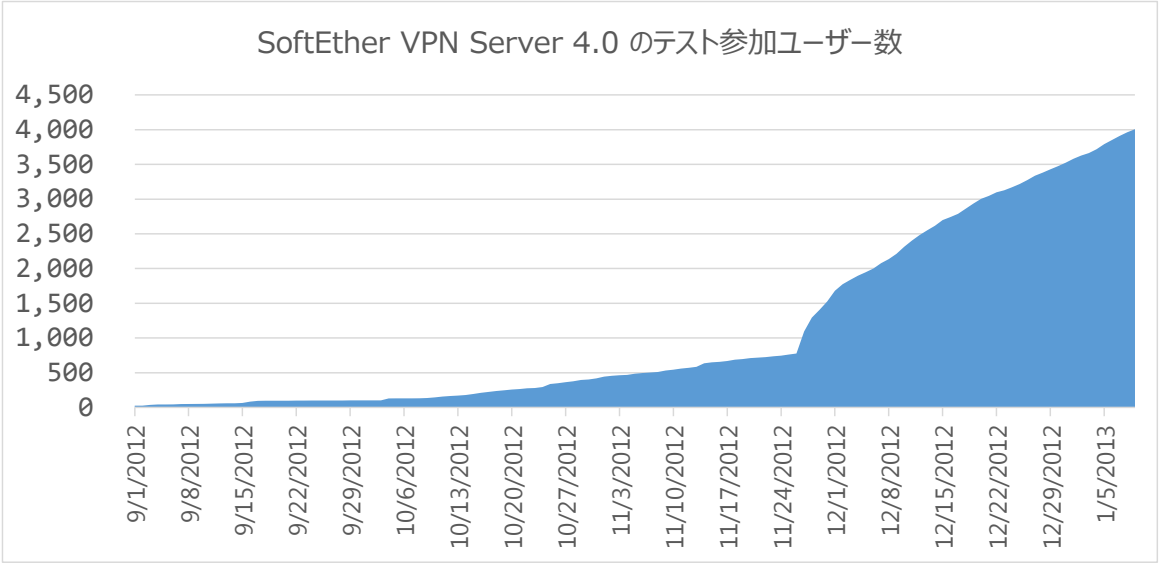


図 29. 本研究で実装したプログラムのテスト版のユーザ数の推移

2013 年 1 月 9 日時点におけるテスト版のインストール台数は 4,007 台である。テスト版のユーザを国別で見ると、図 30 とおり、日本 77.5%、中国 12.4%、米国 2.5%、韓国 1.8%となる (国の判定はダウンロード元 IP アドレスの割当組織の国籍で行ったため、必ずしも正確ではない)。

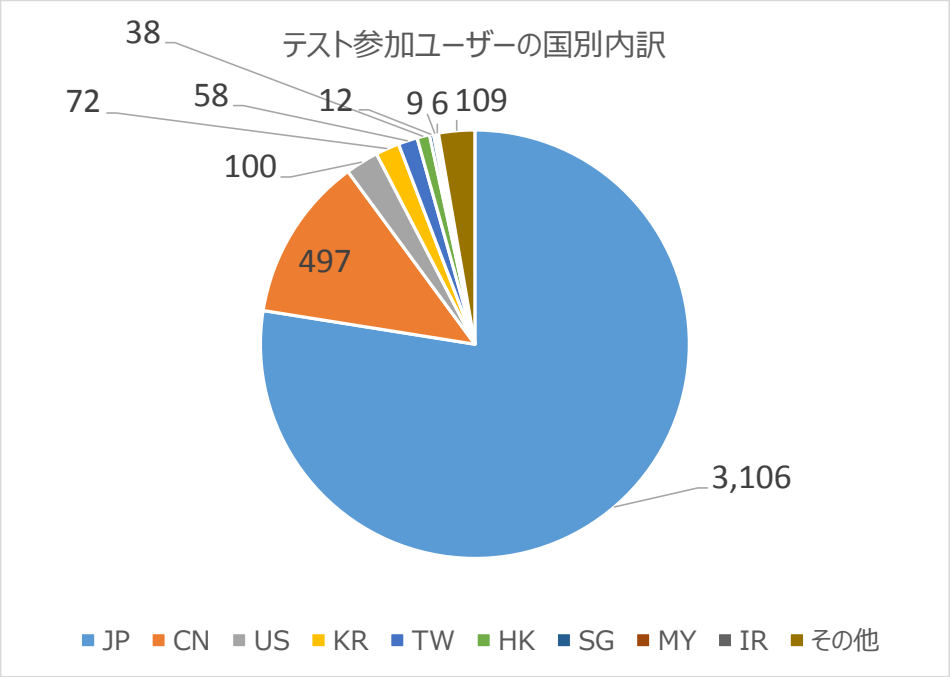


図 30. テスト参加者の国別内訳

上記のようなテストを行った結果、今回実装した VPN プロトコルモジュールに関する軽微なバグがいくつか発見されたのでプログラムを修正した。2013 年 1 月 9 日時点では深刻なバグは発見されていない。このように、本研究で実装した機能は安定して動作している。

これらの実験結果から、1 個の VPN サーバプログラムで複数の VPN プロトコルをサポートすることが実現されたことが確認できた。

第 3 章における各種問題点は、従来手法では解決不可能なものであったが、本実験におけるプログラムの実

装とそのテストの結果、以下の問題点が解決されたこととなる。

- VPN 通信グループ間の分離が不能
- ユーザ認証、パケットフィルタ設定、ポリシー設定の複雑化
- ログの形式、パケットログ機能の有無の相違
- VPN クライアント用 IP アドレスの管理の複雑化および使用効率の低下

3.2 節における既存の VPN サーバプログラムの VPN プロトコルに対する比較表 (表 2) に今回実装したプログラムの検証結果を追記すると、表 6 のようになる。

表 6. 既存 VPN サーバと本研究で実装した SoftEther VPN Server 4.0 との対応プロトコルの比較

	L2TP/IPsec	SSTP	PPTP	OpenVPN	L2TPv3/IPsec	EtherIP/IPsec	SoftEther VPN
Microsoft Routing and Remote Access Service (Windows Server に付属)	○	○	○	×	×	×	×
Mac OS X Server	○	×	○	×	×	×	×
OpenVPN	×	×	×	○	×	×	×
Cisco IOS	○	×	○	×	○	×	×
NEC IX Router OS	×	×	×	×	×	○	×
IIJ SEIL Router OS	○	×	○	×	○	×	×
PacketiX VPN	×	×	×	×	×	×	○
UT-VPN	×	×	×	×	×	×	○
SoftEther VPN Server 4.0 (本研究で実装)	**○**	**○**	**×**	**○**	**○**	**○**	**○**

なお、表 6 のように本研究では PPTP を実装していない。PPTP は通信の際に TCP のほかに GRE (Generic Routing Encapsulation) という UDP とよく似た IP プロトコルを使用する必要があるが、GRE に対応していない NAT やファイアウォールが存在し、PPTP が利用できない環境がある (一方、L2TP は UDP が通過できる環境であればほとんどの環境で利用できる)。また、PPTP に対応している VPN クライアントは大半が L2TP にも対応している。そのため、PPTP を実装することで得られる利点は少なく、労力がかかるため実装を省略したが、ユーザの利便性を下げることにはつながらない。

6.2. 速度測定実験の概要

6.1 節では従来手法では実現することができなかった機能を本研究の手法で実現することができるようになったことを示した。しかし、3.4 節で述べた従来手法における通信のオーバーヘッドについて本研究の手法でどの程度の改善が実現したかは 6.1 節では明確ではない。

そこで、従来手法と本研究の手法との両方において通信速度を計測する実験を行い、同一のハードウェアやネットワーク構成の上で比較することにより、本研究の手法が第 3 章における問題点を解決しているかどうかを検証した。

速度測定実験の目的

本研究で実装した SoftEther VPN Server プログラムは、1 個のプロセスで複数の VPN プロトコルをサポートし、異なるプロトコルの VPN クライアント間の通信を実現する。

従来の方式である、異なる VPN サーバプログラムを複数立ち上げて各ソフトウェア間のパケット交換を IP ルーティングまたは Ethernet ブリッジにより行う方法と比較して、本研究で実装した VPN Server ソフトウェアは 1 個のプログラムで異なる VPN プロトコルを処理することができる。これにより、プログラム間のプロセス通信のオーバーヘッドが減少することにより、異なる VPN プロトコルのクライアントが 1 台ずつ接続されている環境でそれらのクライアント間のスループットが向上することが見込まれる。

そこで、実際に従来方式と本研究で実装した SoftEther VPN Server を用いる方式とでどのようにスループットが異なるかを調べるために測定実験を実施した。

また、SoftEther VPN Server は同一の VPN プロトコルのクライアント同士の通信も行うことができる。そこで、同一の VPN プロトコルのクライアント 2 台が接続された VPN サーバコンピュータで SoftEther VPN Server を用いた場合と従来の VPN サーバプログラムを用いた場合それぞれのスループットを測定し、性能を比較した。

実験環境

同一の以下のスペックの 3 台のコンピュータ (k1, k2, k3) を用いた。その概要を表 7 に示す。実験環境の詳細は付録 1 のとおりである。

表 7. 実験に用いたコンピュータの概要

型番	Fujitsu PRIMERGY TX100 S3
CPU	Intel Xeon E3-1230 3.2GHz 8M
RAM	16GB (Kingston 4GB 1333MHz DDR3 ECC CL9 DIMM x 4)
チップセット	Intel C202
NIC #1, #2	Intel 10 Gigabit CX4 Dual Port Server Adapter
OS	Windows Server 2008 R2 x64 版 Windows Server 2003 R2 x64 版 (OS 抽象化レイヤの性能比較時のみ) Linux 2.6.32 x64 版 (OS 抽象化レイヤの性能比較時のみ)

実験方法

以下のプロトコルについて実験を行った。

- SoftEther VPN Protocol
- L2TP/IPsec
- Microsoft SSTP
- OpenVPN Protocol (L3 ルーティングモード)
- OpenVPN Protocol (L2 ブリッジモード)

上記のプロトコルについて、従来方式の VPN サーバの実装と本研究での VPN サーバの実装との性能を比較した。従来方式の VPN サーバとしては、L2TP/IPsec および Microsoft SSTP サーバとして Windows Server 2008 R2 に付属の RRAS プログラムを、OpenVPN サーバとして OpenVPN 2.2.2 を用いた。実験方法の詳細は付録 2 のとおりである。

6.3. 速度測定実験 (従来手法と本研究の手法との比較)

速度測定実験は、以下の手順で実行した。

実験 6.3.1. VPN を用いない物理的な通信速度の測定

今回の実験では 10 Gigabit Ethernet (10GbE) を用いるため、実験に使用するコンピュータや LAN カードが十分な性能を有しているかどうか最初に検証する必要がある。コンピュータや LAN カードのパフォーマンスを確認するため、単純にコンピュータ間の通信速度および RTT を測定した (図 31)。

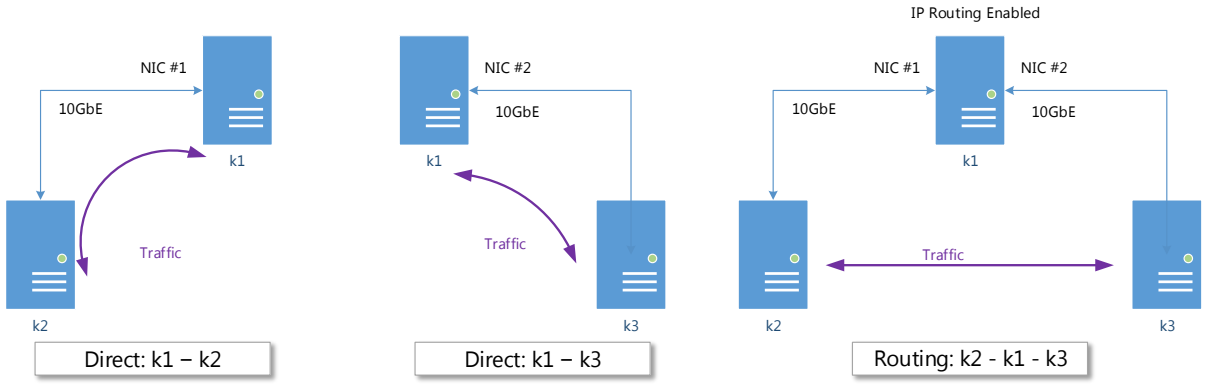


図 31. 10GbE ネットワーク環境の基本的性能試験方法

実験 6.3.2. 単一の VPN プロトコルの性能測定

SoftEther VPN Protocol, L2TP/IPsec, SSTP, OpenVPN (L3), OpenVPN (L2) の 5 種類のプロトコルに関して、従来方式の VPN サーバプログラムと、本方式の VPN サーバプログラムとの間で性能測定を行った。このとき、VPN サーバプログラムを入れ替える以外のその他の条件 (ハードウェア、ネットワーク構成、VPN クライアント側のソフトウェア) は同一とした。なお、SoftEther VPN Protocol については我々の実装以外の実装がないため、従来方式の VPN サーバプログラムにおける実験は行っていない。

この実験においては、2 台の VPN クライアントを 1 台の VPN サーバに接続して VPN クライアント間で通信をする方式 (以下、「PC to PC 接続」という。) と、1 台の VPN クライアントを 1 台の VPN サーバに接続してその VPN サーバコンピュータが物理的に接続されている LAN 上にある他のコンピュータとの間で通信をする方式 (以下、「PC to LAN 接続」という。) の 2 種類の実験を行った。たとえば、SSTP の VPN サーバの実装として、Microsoft 社による Windows Server 2008 R2 に搭載されている SSTP サーバ機能と、本研究による実装における SSTP サーバ機能との比較を、「PC to PC 接続」については図 32 のように、「PC to LAN 接続」については図 33 のように行った。

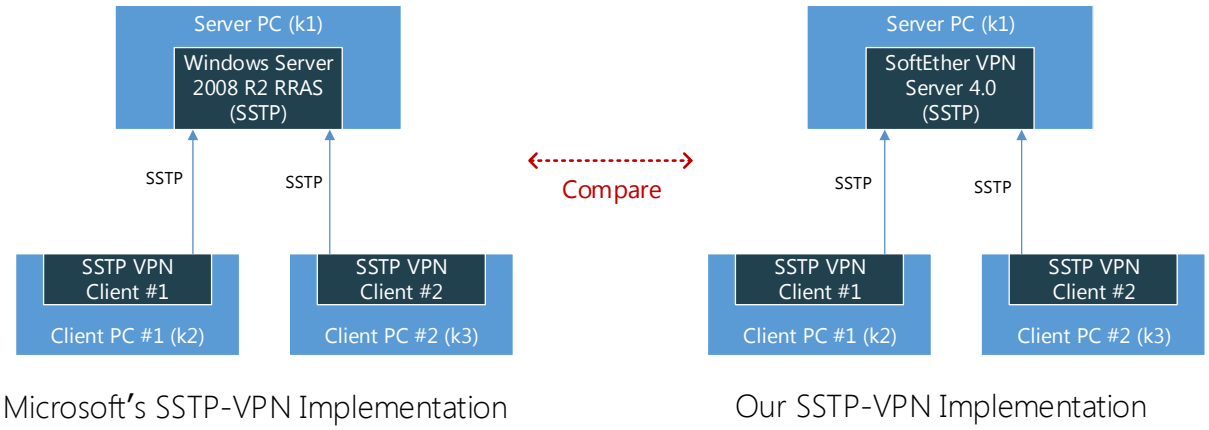


図 32. PC to PC 接続の実験例

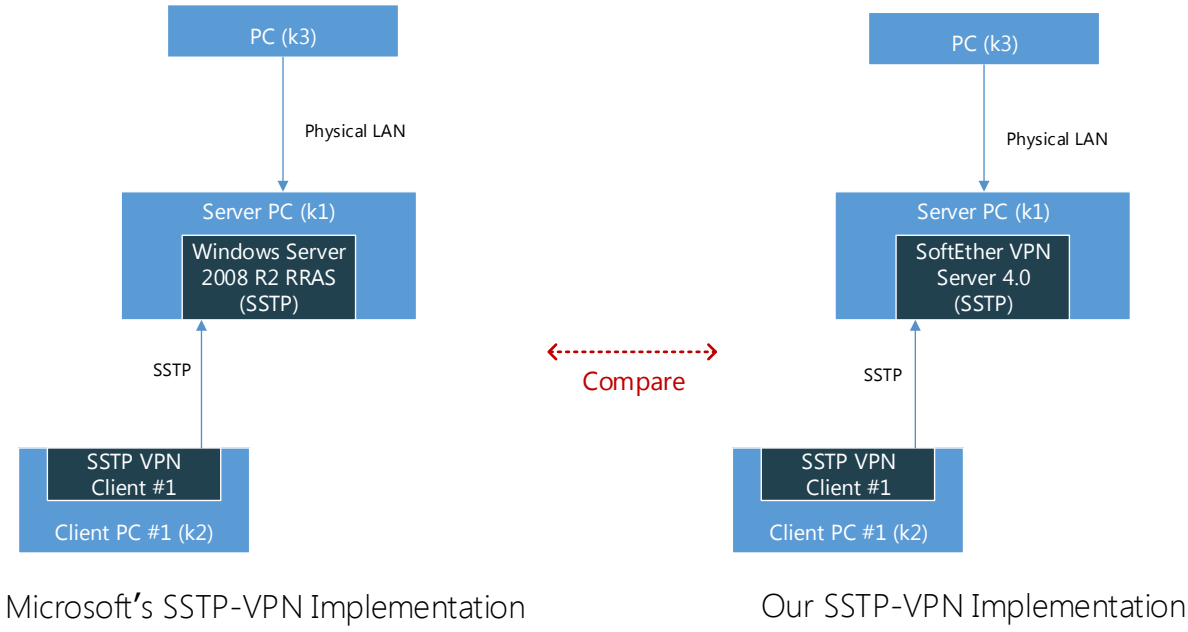


図 33. PC to LAN 接続の実験例

実験 6.3.3. 従来の 2 個の VPN サーバプログラムの組み合わせる方法と本研究の実装の VPN サーバを用いる方法との性能比較

2 種類の VPN プロトコルの VPN クライアント間の VPN 通信を 1 台の VPN サーバコンピュータで処理する場合、従来手法 (2 個の VPN サーバプログラムを組み合わせる手法) と本研究による手法 (本研究で実装した 1 個の VPN サーバプログラムである SoftEther VPN Server 4.0 を用いる手法) とで通信性能が異なるかどうかを検証した。

2 個の VPN プロトコルの VPN クライアント 2 台を 1 台の VPN サーバに接続させる実験を行うための VPN プロトコルの組み合わせは、のとおりとした。

表 7. 2 個の VPN プロトコルの組み合わせ一覧表

番号	プロトコル 1	プロトコル 2	結合方式
1	SEVPN**	L2TP/IPsec	IP ルーティング
2	SEVPN	SSTP	IP ルーティング
3	SEVPN	OpenVPN_L3	IP ルーティング
4	SEVPN	OpenVPN_L2	Ethernet ブリッジ
5	L2TP/IPsec	SSTP	IP ルーティング
6	L2TP/IPsec	OpenVPN_L3	IP ルーティング
7	L2TP/IPsec	OpenVPN_L2	IP ルーティング
8	SSTP	OpenVPN_L3	IP ルーティング
9	SSTP	OpenVPN_L2	IP ルーティング
10	OpenVPN_L3	OpenVPN_L2	IP ルーティング

たとえば、上表の 8 番目の組み合わせでは、「SSTP」と「OpenVPN (L3)」とを組み合わせることになっている。この場合、図 34 のように、従来手法として Windows Server 2008 R2 の SSTP サーバ機能と OpenVPN 2.2.2 の OpenVPN サーバ機能とを組み合わせる方法 (左側) と、本研究における手法として第 5 章で実装した SoftEther VPN Server 4.0 を用いる方法 (右側) との 2 種類を比較した。

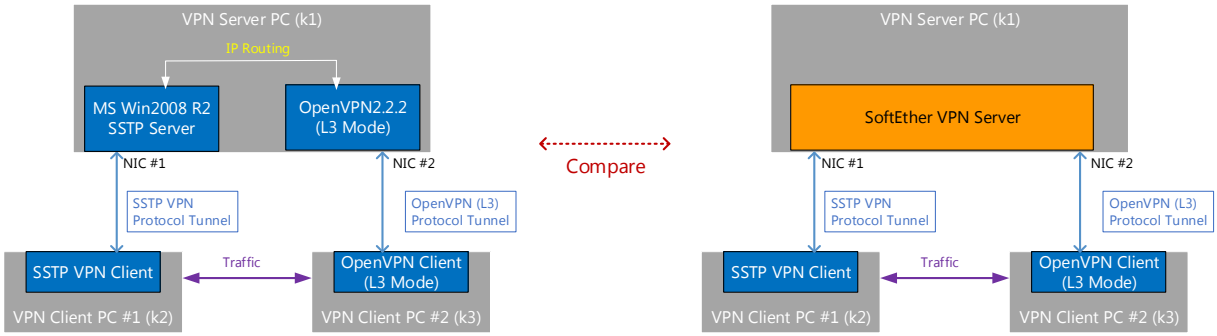


図 34. SSTP+OpenVPN(L3) の組み合わせにおける比較例

6.4. 実験結果

今回実施したすべての実験結果は分量が多いため付録 3 に記載した。以下では、結果の要旨を示して考察する。

実験で用いたハードウェアの性能の結果 (実験 6.3.1)

実験で用いたハードウェアで VPN を用いない物理的な 10GbE 通信を行った結果、各ハードウェア間でアップロード方向、ダウンロード方向ともに約 9.8Gbps 程度の通信を行うことができた。したがって、ハードウェアの性能は適切であると考える。

単一の VPN プロトコルの性能測定 (従来の VPN サーバ 対 本研究で実装した VPN サーバの比較) (実

** SoftEther VPN プロトコルの略。

験 6.3.2)

SoftEther VPN Protocol, L2TP/IPsec, SSTP, OpenVPN (L3), OpenVPN (L2) の合計 5 個の VPN プロトコルについて、2 台の VPN クライアント間で通信を行った (PC to PC 接続)。2 台の VPN クライアントは同一の種類の VPN プロトコルとした。つまり、1 個の VPN サーバプログラムに 2 台の VPN クライアントが接続して通信した。

VPN サーバプログラムとしては、従来の VPN ソフトウェアとして、L2TP および SSTP については Windows Server 2008 R2 RRAS を、OpenVPN (L3) および OpenVPN (L2) については OpenVPN 2.2.2 を用いた。これらの従来の VPN ソフトウェアと、今回実装した SoftEther VPN Server 4.0 との性能を比較した。その結果は、図 35 のようになった (送信方向のテストと受信方向のテストの結果の平均値。以下同じ)。

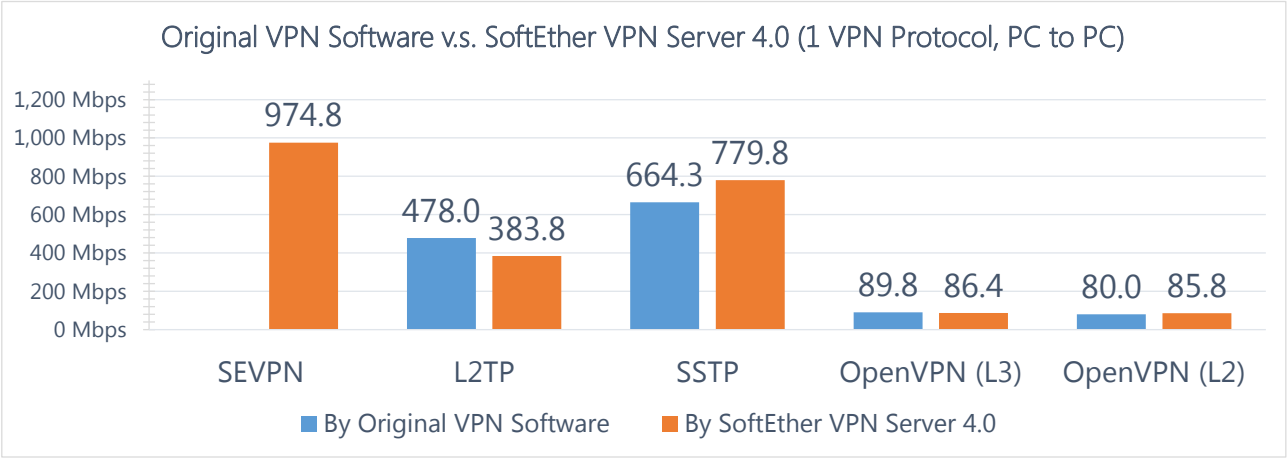


図 35. 本研究で実装した VPN サーバと従来との単一 VPN プロトコルの性能比較 (PC to PC)

次に、1 台の VPN クライアントと、1 台の VPN サーバコンピュータに物理的に接続されているコンピュータとの間で通信を行った (PC to LAN 接続)。使用したプログラムは、PC to PC 接続の場合と同様である。その結果は、図 36 のようになった。

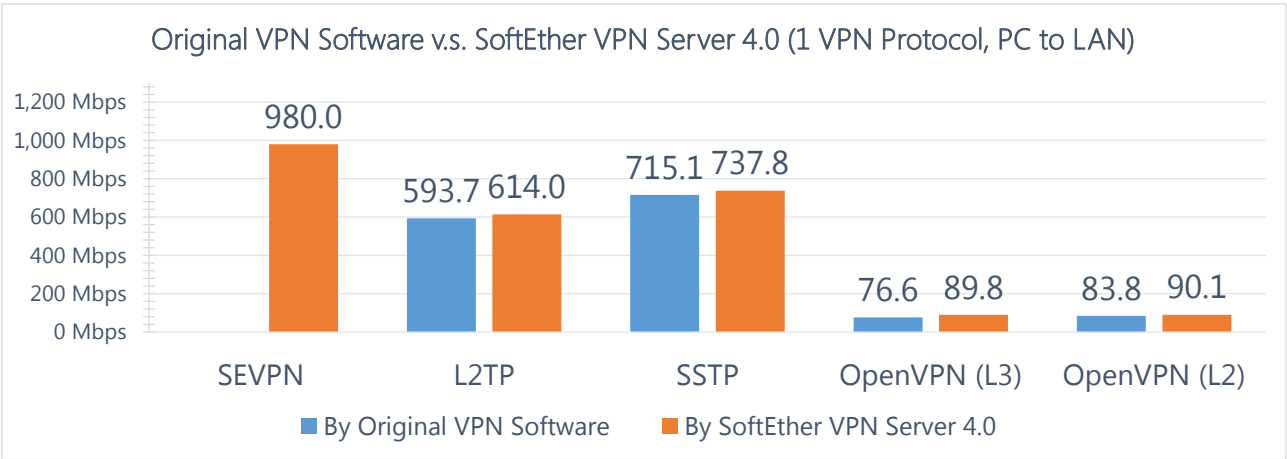


図 36. 本研究で実装した VPN サーバと従来との単一 VPN プロトコルの性能比較 (PC to LAN)

これらの結果を比較すると、PC to PC 接続について、L2TP では従来の実装 (Microsoft 社の実装) が本研究による実装よりも高速となっている。しかし、SSTP では本研究の実装のほうが高速となった。OpenVPN については L3、L2 のいずれも、本研究の実装のほうが高速となった。

PC to LAN 接続について、L2TP, SSTP, OpenVPN (L3), OpenVPN (L2) のいずれも、従来の実装 (Microsoft 社または OpenVPN 社の実装) よりも本研究の実装のほうが高速となった。

なお、OpenVPN (L3, L2) について、SoftEther VPN Protocol, L2TP, SSTP と比較すると 1 桁低い性能が出て

いる。これは、VPN サーバプログラムとして OpenVPN 社の開発した OpenVPN 2.2.2 を用いた場合でも、今回の実装のプログラムを用いた場合でもほぼ同様である。OpenVPN のスループットが低い理由については不明であるが、Web サイト等で検索しても、通常の設定で 100Mbps を大きく超えるような高いスループットが出たという実験結果が見つからないことから、これは OpenVPN クライアント側のプログラムの特性あるいは OpenVPN プロトコルの性質によるものであると考える。

本研究では VPN プロトコル間のパフォーマンスの比較を対象としていないため、OpenVPN に関する結果が他の VPN プロトコルと比べて低いことは特に問題ではないと考える。

2 個の異なる VPN プロトコルのクライアント間の性能比較（従来の 2 個の VPN サーバプログラムの組み合わせる方法 対 本研究の実装の VPN サーバ) (実験 6.3.3)

SoftEther VPN Protocol, L2TP/IPsec, SSTP, OpenVPN (L3), OpenVPN (L2) の合計 5 個の VPN プロトコルについて、10 通りの組み合わせを行い、2 台の異なるプロトコルの VPN クライアントを 1 台の VPN サーバに対して接続して通信速度を測定した (PC to PC 接続)。その結果は、図 37 のようになった。

VPN サーバプログラムとしては、従来方式の場合は 2 個の異なる VPN サーバプログラムを組み合わせた。本研究の方式の場合は本研究で実装した SoftEther VPN Server 4.0 をスタンドアロンで用いた。

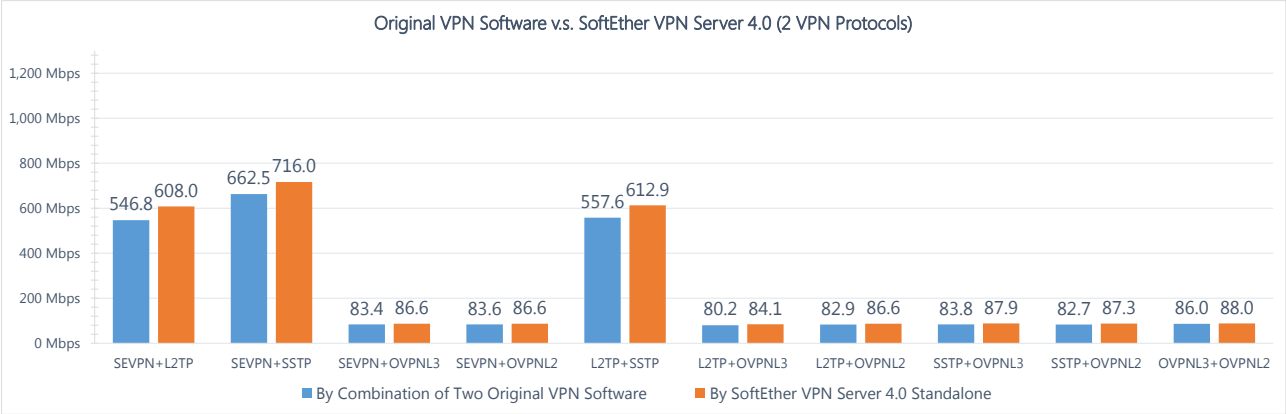


図 37. 従来の VPN サーバの組み合わせと本研究で実装した VPN サーバとの性能比較

結果を検討すると、いずれのプロトコル同士の組み合わせであっても、従来手法 (2 個の VPN サーバプログラムを組み合わせる方式) よりも本研究の方式 (1 個の VPN サーバプログラムで 2 種類のプロトコルを処理する) のほうが高速となった。従来方式と比較して本研究の実装が高速になった度合いを計算すると、図 38 のようになる。

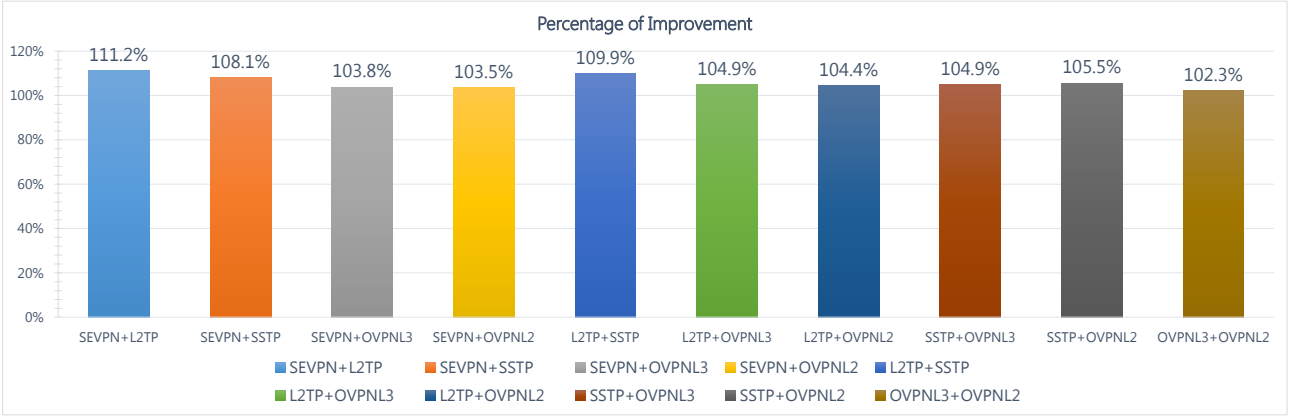


図 38. 本研究の実装の高速化の割合

本研究の方式によって従来方式におけるオーバーヘッドは減少したかどうか

3.4 節で述べた従来方式におけるオーバーヘッドは、本研究の方式によって減少したかどうかを検討する。

一見、図 37 および図 38 のみを見ると、本研究の方式のほうが従来方式よりも高速でありオーバーヘッドが少ないことが明らかなように思える。しかし、本研究の方式の測定のために用いた VPN サーバプログラムは本研究で新たに実装した SoftEther VPN Server 4.0 を用いており、従来方式のために用いた VPN サーバプログラムである Microsoft 社や OpenVPN 社の既存の VPN サーバプログラムを拡張した訳ではない。そのため、単に Microsoft 社や OpenVPN 社の既存の VPN サーバプログラムの性能が低く、今回実装した SoftEther VPN Server 4.0 の性能が比較的高いことが要因となって図 37 および図 38 のような結果が出ているのであって、必ずしも 3.4 節で述べたような通信オーバーヘッドを本研究の手法によって解決できた訳ではないのではないかという疑問が生じる。

しかし、図 36 の L2TP の結果を見ると、Microsoft 社の既存の L2TP サーバプログラムの実装のほうが本研究で実装した SoftEther VPN Server 4.0 よりも高いスループットを実現していることがわかる。したがって、今回実装したプログラムは、同一種類の VPN プロトコル 2 個を同時に処理する場合について、既存の VPN サーバプログラムと比較して性能が高い訳ではないということが言える。そして、図 37 において L2TP が含まれる結果 (SEVPN+L2TP, L2TP+SSTP, L2TP+OVPNL3, L2TP+OVPNL2) ではすべて本研究で実装した SoftEther VPN Server 4.0 のほうが Microsoft 社の既存の L2TP サーバプログラムの実装よりも高いスループットを実現している。

同様に、図 36 の OpenVPN (L3) の結果は、OpenVPN 社の既存の OpenVPN サーバプログラムの実装のほうが本研究で実装した SoftEther VPN Server 4.0 よりも高いスループットを実現しているが、図 37 において OpenVPN (L3) が含まれる結果 (SEVPN+OVPNL3, L2TP+OVPNL3, SSTP+OVPNL3) ではすべて本研究で実装した SoftEther VPN Server 4.0 のほうが Microsoft 社の既存の L2TP サーバプログラムの実装よりも高いスループットを実現している。

これらのことから、図 37 および図 38 のような良好な結果を得ることができた要因は本研究の実装によって 3.4 節で述べたオーバーヘッドが削減されたことによるものであると言うことができる。

上述の実験結果から、2 種類の異なる VPN プロトコルの VPN クライアントからの接続を処理する VPN サーバコンピュータにおいては、それぞれのプロトコル用の VPN クライアントプログラムを同時に動作させる方法 (従来手法) よりも、1 個の VPN サーバプログラムでそれぞれのプロトコルの VPN サーバ機能を実装してその 1 個の VPN サーバプログラムのみを動作させる方法のほうが、オーバーヘッドが少なく、スループットが高速となることが分かった。

OS 抽象化レイヤの性能の評価 (実験 6.3.4)

5.3 節で述べた OS 抽象化レイヤの性能を評価するため、Windows Server 2003 R2、Windows Server 2008 R2 および Linux 2.6.32 の 3 種類の OS で同等の環境で速度測定を行った。実験の方法および結果の詳細は付録 4.1.1 から付録 4.1.6 に記載する。図 39 から図 42 は主要な結果を抜粋したグラフである。

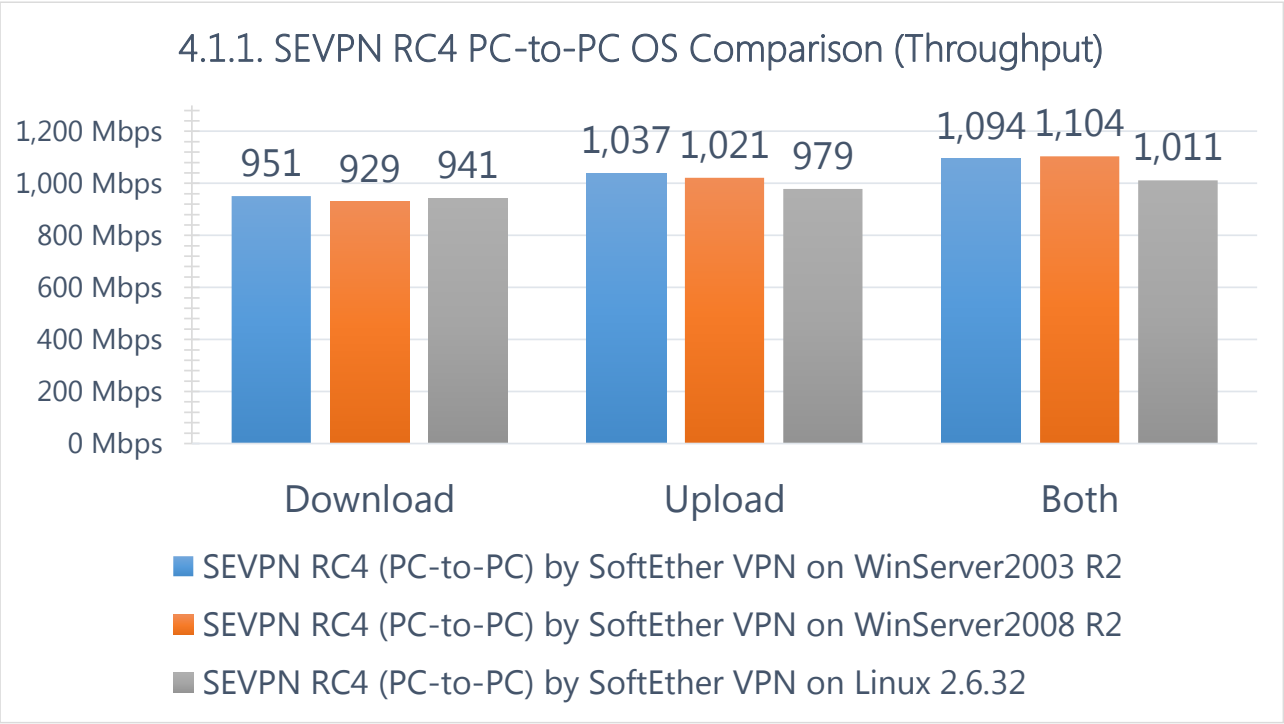


図 39. SoftEther VPN プロトコルにおける PC to PC 通信の通信速度の 3 種類の OS 間の比較結果

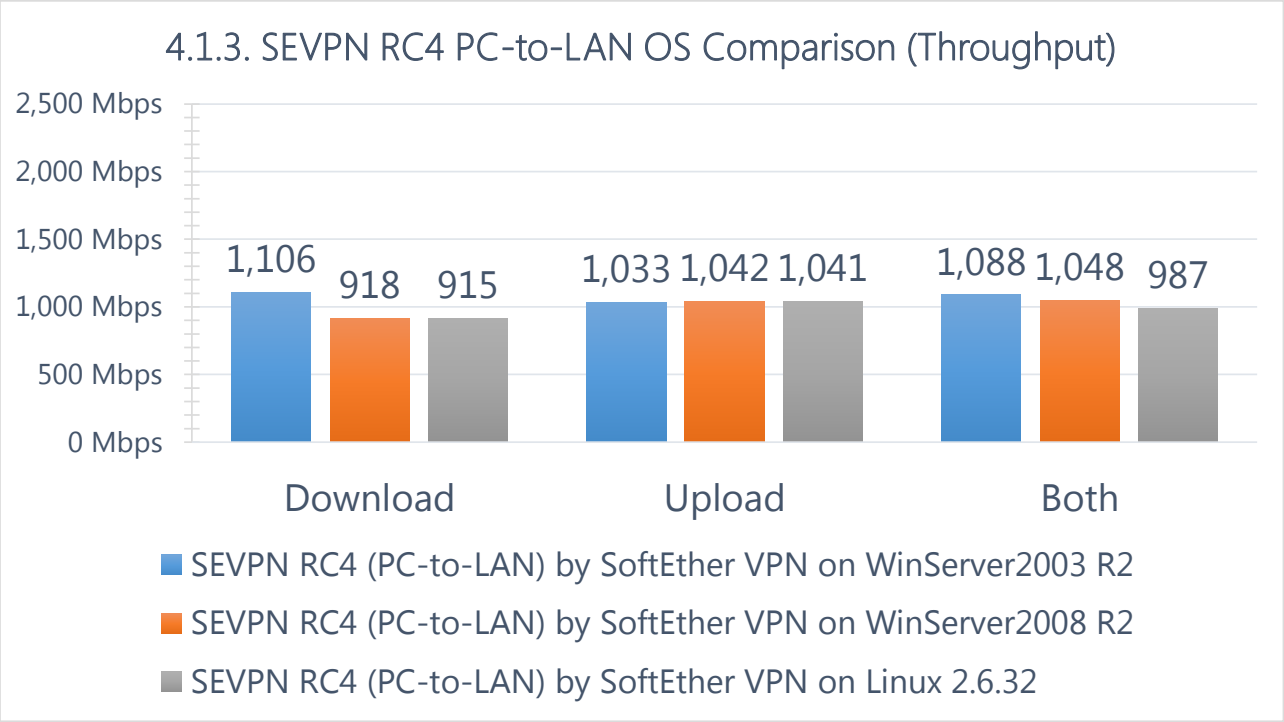


図 40. SoftEther VPN プロトコルにおける PC to LAN 通信の通信速度の 3 種類の OS 間の比較結果

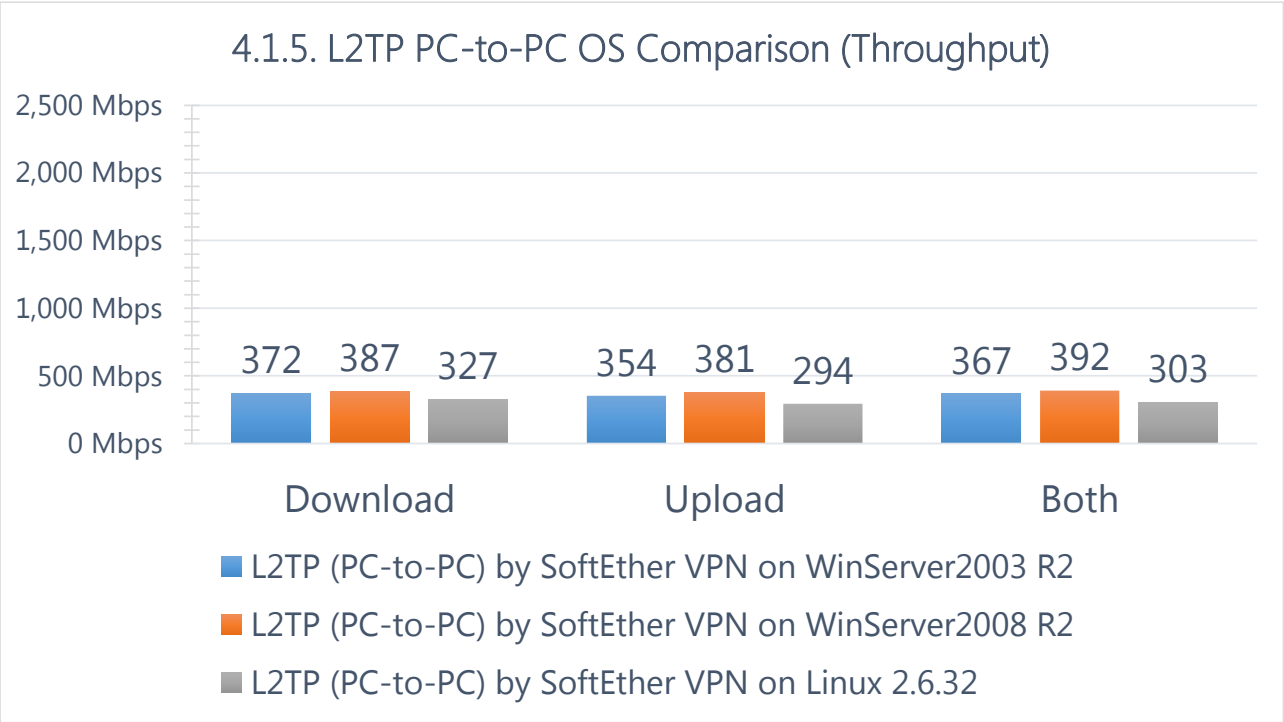


図 41. L2TP/IPsec プロトコルにおける PC to PC 通信の通信速度の 3 種類の OS 間の比較結果

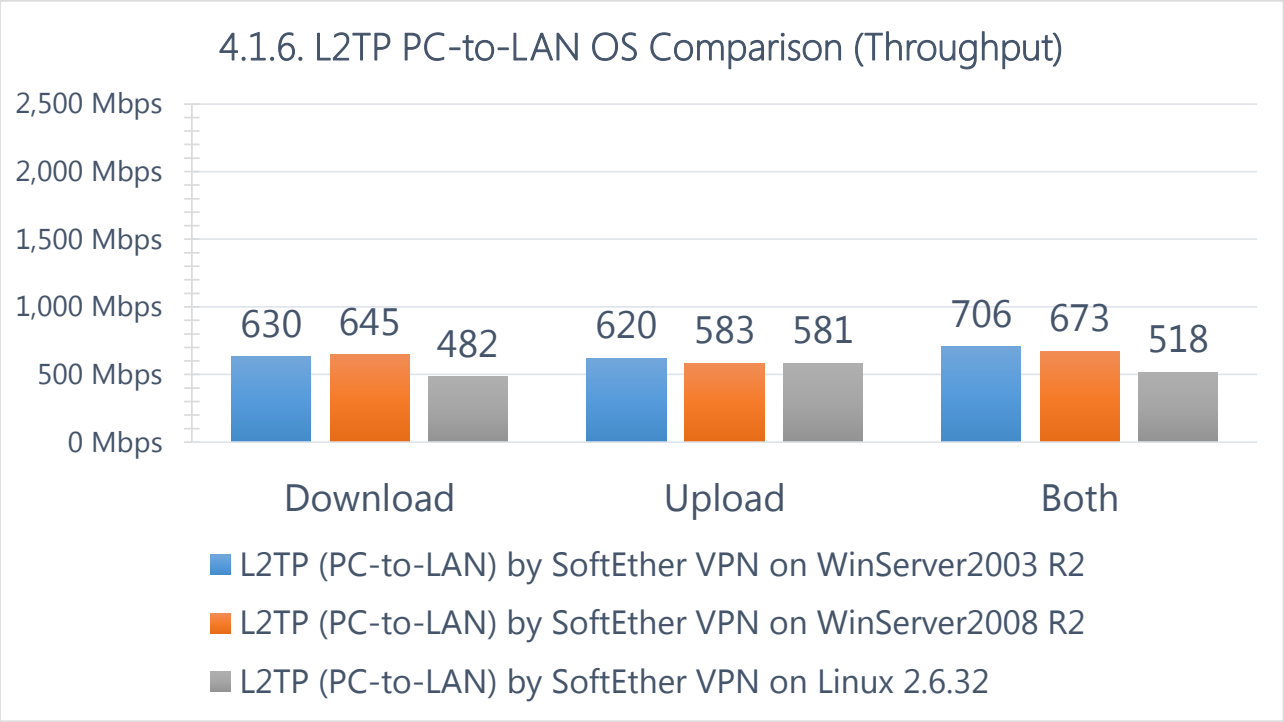


図 42. L2TP/IPsec プロトコルにおける PC to LAN 通信の通信速度の 3 種類の OS 間の比較結果

結果として、Windows Server 2003 R2、Windows Server 2008 R2 上と比較して Linux 2.6.32 上で動作させた場合、プロトコルが SoftEther VPN のときの結果は同等となったが、L2TP/IPsec の場合は若干低いスループット(測定毎にばらつきがあるが、12%から 23%程度低速) となった。このことは、本研究で実装した OS 抽象化レイヤの性能が Linux において若干悪いことを示している。その原因として、UDP 処理の性能の違いや 5.3 節で述べた Intel AES-NI を呼び出すライブラリの実装が異なることなどが考えられる。これらについてより詳細な要因を調査し解決することは将来の課題としたい。

第 7 章 結論

第 7 章では、本研究のまとめを行う。

7.1. まとめ

本研究の目的は、現存する VPN プロトコルのうち SoftEther VPN、L2TP over IPsec、SSTP、OpenVPN (L2 モード)、OpenVPN (L3 モード)、EtherIP over IPsec、L2TPv3 over IPsec の合計 7 種類の VPN プロトコルをすべてサポートする単一の VPN サーバプログラムを設計、実装することであった。これにより、従来これらの VPN プロトコルをそれぞれ個別にサポートしていた VPN サーバプログラムを組み合わせて利用する場合と比べて、ユーザ管理などのセキュリティ設定の共通化、IP アドレスの管理の簡素化、VPN プロトコル間のレイヤの差異の吸収を得ることができる。また、VPN サーバプログラムのコードをマルチプラットフォーム対応させ、複数の OS 上で同等に動作することを目標とした。

そのため、まずは既存の VPN、LAN、リモートアクセスプロトコルについて調査を行った。LAN 内では Ethernet が使われているが、WAN では PPP が使われることが多い。PPP をインターネット上で利用できるプロトコルとして L2TP および SSTP がある。また、WAN 上で Ethernet フレームを送受信するためのプロトコルとして L2TPv3 および EtherIP がある。L2TP, L2TPv3 および EtherIP は IPsec を用いて伝送され、SSTP は SSL を用いて伝送される。これらの VPN プロトコルのほか、OpenVPN および SoftEther VPN プロトコルも存在する。

前掲の多数の VPN プロトコルにはそれぞれ特徴があるが、互いに非互換である。たとえば、SSTP はファイアウォールを通過し易いが対応 VPN クライアントが限定され、L2TP は多くの VPN クライアントが対応しているがファイアウォールを通過しにくい。そのため、多くの VPN クライアントをサポートするにはこれらすべてに対応した VPN サーバプログラムが必要となるが、従来はそのような VPN サーバプログラムは存在せず、複数の VPN サーバプログラムを組み合わせて使用する必要があった。しかし VPN サーバプログラムを組み合わせると、オーバーヘッドの発生により通信速度が低下したり、管理が複雑化したり、VPN プロトコルを跨いだセキュリティ機能の実現ができなかったりするという問題があった。

これらの問題を解決するためには、1 個のプロセスで前掲の多数の VPN プロトコルすべてを受付することができる新しい VPN サーバプログラムを設計する必要があった。本設計では VPN プロトコルの種類やカプセル化対象のレイヤの差異を吸収するため、VPN プロトコルで伝送されてきたパケットに適切な処理を行い Ethernet フレームにレイヤ変換して扱う手法を用いた。また、通信グループを仮想 HUB 単位で分離し、仮想 HUB ごとにユーザ認証などのセキュリティ設定を行うことができるような設計とした。

このような設計の VPN サーバプログラムは、SoftEther VPN Server 3.0 に拡張を加えることで実装した。SoftEther VPN Server 3.0 はモジュール化された内部構造を持っているため、新たに複数の VPN プロトコルのサポートを追加するためのモジュールを実装した。またモジュール間のパケットデータの受渡しを高速化することと、モジュール間の独立性を保つことを両立させるためのデータ構造も利用した。

実装した VPN サーバプログラムに対して実験を行った。まず iPhone、iPad、Android、Cisco ルータ、NEC ルータ、Windows、Linux などに付属の VPN クライアント機能から VPN 接続が行えること、従来手法では実現できなかった通信やセキュリティが実現できたことを確認した。また、インターネットを用いて約 4,000 ユーザにテストを依頼し、各ユーザの環境で安定して動作することも確認した。そして、高速な 10GbE の実験環境を用意し、複数 VPN プロトコル混在環境において従来手法と本研究の手法とを比較する通信速度測定実験を行った。その結果、結果、従来手法と比較して本手法のほうがプロトコルの組み合わせによるが 2.3%から 11.2%程度高速であることを確認した。VPN サーバプログラムを各種 OS 上で動作させるための OS 抽象化レイヤの性能を測定したところ、いずれも同等程度の性能を得ることができた。ただし、L2TP/IPsec プロトコルにおいては Windows に比べて Linux が 12%から 23%程度低速な結果となった。

これらの結果から、本研究で解決することを目的としていた以下の問題をすべて解決した。

- 通信のオーバーヘッドの発生

- VPN 通信グループ間の分離が不能
- ユーザ認証、パケットフィルタ設定、ポリシー設定の複雑化
- ログの形式、パケットログ機能の有無の相違
- VPN クライアント用 IP アドレスの管理の複雑化および使用効率の低下

7.2. 今後の課題

今後は、今回実装した SoftEther VPN Server 4.0 が対応する VPN プロトコルの種類を増やしていくことを目標としたい。現在まだ実装を行っていない VPN プロトコルとして主要なものには、IKEv2[26]、PPTP (Point to Point Tunneling Protocol) [27]、IPsec トンネルモードなどがある。

また、SoftEther VPN Server 4.0 は今後オープンソースプロジェクトとして GPL ライセンスで公開し、誰でも利用することができるようにして利用者を増やすほか、実装上の改良点を発見した人からのソースコードの改良も受付け、すべての VPN プロトコルおよび VPN プロトコル同士の組み合わせによって現存する VPN サーバプログラムの中で最も高速な VPN サーバプログラムとすることを目指す。

謝辞

本研究を行うにあたり、筑波大学大学院システム情報工学研究科ソフトウェア研究室の新城靖先生 (指導教員)、板野肯三先生、佐藤聡先生およびオペレーティングシステム研究室の加藤和彦先生には、多くのご助言およびご指導をいただきました。ここに深く感謝申し上げます。特に新城先生には、本研究期間のみではなく、筑波大学に入学してから現在までの約 10 年間に渡り頻繁に技術的なご助言をいただきましたが、本研究にはそれらを基礎に考えたアイデアによるものが多数含まれています。また、本研究で実装した VPN プロトコルモジュールのプログラムの一部 (IPsec モジュール) には、加藤先生の BitVisor プロジェクトにおいて研究を行った際の知見を活用いたしました。

また、著者が所属しているソフトウェア研究室 (www.softlab.cs.tsukuba.ac.jp) の皆様、実験環境を活用させていただいた筑波大学学術情報メディアセンターおよび産学リエゾン共同研究センターの関係者の皆様には大変お世話になりました。

さらに、本研究で実装した SoftEther VPN Server 4.0 のテスト版はインターネット上でテスト利用者を募集したところ、約 4,000 ユーザの皆様にテストいただくことができました。

上記を含めた多くの方々のご協力により、本研究を円滑に実施し、まとめることができたと思います。誠にありがとうございました。

参考文献

[1] Gurdeep Singh Pall, Bill Palter, Allan Rubens, W. Mark Townsley, Andrew J. Valencia and Glen Zorn: "Layer Two Tunneling Protocol 'L2TP"', http://tools.ietf.org/html/rfc2661, August 1999.

[2] Microsoft Corporation: "[MS-SSTP]: Secure Socket Tunneling Protocol (SSTP)", http://msdn.microsoft.com/en-us/library/cc247338.aspx, October 2012.

[3] 登 大遊: "SoftEther の内部構造", 情報処理 Vol.45, No.10, pp.1057-1062, Oct 2004.

[4] OpenVPN Technologies, Inc.: "OpenVPN", http://openvpn.net/index.php/open-source.html, December 2012.

[5] Jed Lau, W. Mark Townsley and Ignacio Goyret: "Layer Two Tunneling Protocol - Version 3 (L2TPv3)", http://tools.ietf.org/html/rfc3931, March 2005.

[6] Russell Housley and Scott Hollenbeck: "EtherIP: Tunneling Ethernet Frames in IP Datagrams", http://tools.ietf.org/html/rfc3378, September 2002.

[7] Digital Equipment Corporation, Intel Corporation and Xerox Corporation: "The ethernet: a local area network: data link layer and physical layer specifications", ACM SIGCOMM Computer Communication Review Homepagearchive Volume 11 Issue 3, July 1981 Pages 20 - 66, September 1980.

[8] William Allen Simpson: "The Point-to-Point Protocol (PPP)", http://tools.ietf.org/html/rfc1661, July 1994.

[9] Smoot Carl-Mitchell and John S. Quarterman: "Using ARP to Implement Transparent Subnet Gateways", http://tools.ietf.org/html/rfc1027, October 1987.

[10] Stephen Kent and Karen Seo: "Security Architecture for the Internet Protocol", http://tools.ietf.org/html/rfc4301, December 2005.

[11] Dan Harkins and Dave Carrel: "The Internet Key Exchange (IKE)", http://tools.ietf.org/html/rfc2409, November 1998.

[12] Stephen Kent: "IP Encapsulating Security Payload (ESP)", http://tools.ietf.org/html/rfc4303, December 2005.

[13] Alan O. Freier, Philip Karlton and Paul C. Kocher: "The Secure Sockets Layer (SSL) Protocol Version 3.0", http://tools.ietf.org/html/rfc6101, August 2011.

[14] Robert Morris, Eddie Kohler, John Jannotti and M. Frans Kaashoek: "The Click modular router", ACM SIGOPS Operating Systems Review Volume 33, Issue 5, pp 217-231, December 1999.

[15] The Apache Software Foundation: "Apache Portable Runtime Project", http://apr.apache.org/, December 2012.

[16] Ramachandran, Vivek & Nandi and Sukumar: "Detecting ARP Spoofing: An Active Technique", Information systems security: first international conference, ICISS 2005, Kolkata, India, December 2005.

[17] Cisco Systems: "DHCP Snooping", Catalyst 6500 Release 12.2SX Software Configuration Guide, http://www.cisco.com/en/US/docs/switches/lan/catalyst6500/ios/12.2SX/configuration/guide/snoodhcp.html, March 2009.

[18] Tero Kivinen, Ari Huttunen, Brian Swander and Victor Volpe: "Negotiation of NAT-Traversal in the IKE", http://tools.ietf.org/html/rfc3947, January 2005.

[19] Tero Kivinen, Ari Huttunen, Brian Swander and Victor Volpe: "Negotiation of NAT-Traversal in the IKE", http://tools.ietf.org/html/draft-ietf-ipsec-nat-t-ike-08, February 2004.

[20] Geoffrey Huang, Stephane Beaulieu and Dany Rochefort: "A Traffic-Based Method of Detecting Dead Internet Key Exchange (IKE) Peers", http://tools.ietf.org/html/rfc3706, February 2004.

[21] Brian Lloyd and William Allen Simpson: "PPP Authentication Protocols", http://tools.ietf.org/html/rfc1334, October 1992.

[22] Glen Zorn: "Microsoft PPP CHAP Extensions, Version 2", http://tools.ietf.org/html/rfc2759, January 2000.

[23] Microsoft Corporation: "LsaLogonUser function (Windows)", MSDN Library, http://msdn.microsoft.com/en-us/library/windows/desktop/aa378292(v=vs.85).aspx, December 2012.

[24] Steve Cobb: "PPP Internet Protocol Control Protocol Extensions for Name Server Addresses", http://tools.ietf.org/html/rfc1877, December 1995.

[25] Shay Gueron: "Intel Advanced Encryption Standard (AES) Instructions Set - Rev 3.01", http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set/, August 2012.

[26] Charlie Kaufman, Paul Hoffman, Yoav Nir and Pasi Eronen: "Internet Key Exchange Protocol Version 2 (IKEv2)", http://tools.ietf.org/html/rfc5996, September 2010.

[27] Kory Hamzeh, Gurdeep Singh Pall, William Verthein, Jeff Taarud, W. Andrew Little and Glen Zorn: "Point-to-Point Tunneling Protocol (PPTP)", http://tools.ietf.org/html/rfc2637, July 1999.

[28] O. Honda, H. Ohsaki, M. Imase, M. Ishizuka, and J. Murayama: "Understanding TCP over TCP: Effects of TCP tunneling on end-to-end throughput and latency," Proceedings of OpticsEast/ITCom2005, Jan. 2005.

[29] Samuel J. Leffler and Marshall Kirk McKusick: "The Design and Implementation of the 4.3 Bsd Unix Operating System", ISBN 0201546299, March 1991.

[30] Michael Beck, Robert Magnus and Ulrich Kunitz : "Linux Kernel Internals with Cdrom (3rd Edition)", ISBN 0201719754, http://dl.acm.org/citation.cfm?id=560490, September 1, 2002.

[31] Microsoft Corporation: "NET_BUFFER Architecture (Windows Drivers)", http://msdn.microsoft.com/en-us/library/windows/hardware/ff568377(v=vs.85).aspx, November 2012.

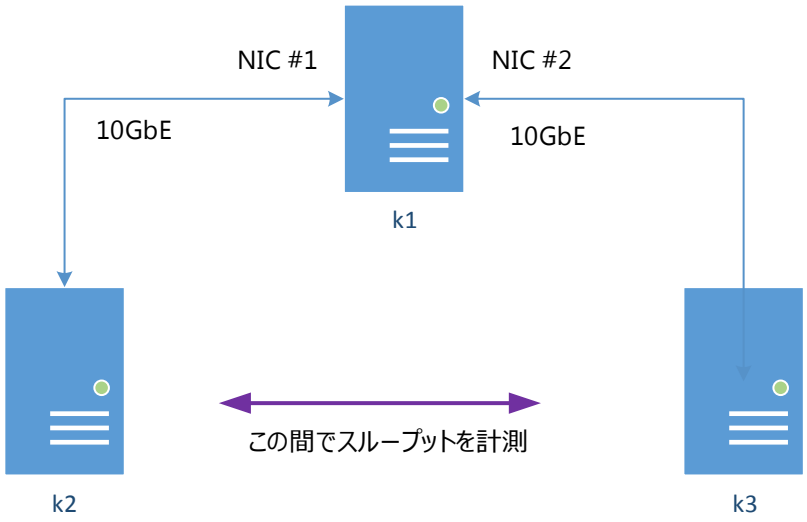
付録 1. 実験環境

同一のスペックの 3 台のコンピュータ k1, k2 および k3 を用意した。スペックは以下のとおりである。

型番	Fujitsu PRIMERGY TX100 S3
CPU	Intel Xeon E3-1230 3.2GHz 8M
RAM	16GB (Kingston 4GB 1333MHz DDR3 ECC CL9 DIMM x 4)
チップセット	Intel C202
NIC #1, #2	Intel 10 Gigabit CX4 Dual Port Server Adapter
OS	Windows Server 2008 R2 (x64 版)

なお、付録 4.1.1 から 4.1.6 では OS 間の性能結果を得るため、上記 OS に加え Windows Server 2003 R2 および Linux 2.6.32 も用いた。

実験では主に k1 を VPN サーバ用、k2 および k3 を VPN クライアント用として使用した。コンピュータ間の配線は 10GBASE-CX4 ケーブルを用いて直接接続 (AUTO-MDI/MDIX によるクロス接続) を行った。Ethernet スイッチは用いていない。



付録 2. 実験方法

付録 2.1. 実験対象プロトコル

実験では、以下の VPN プロトコルを対象とした。L2TPv3/IPsec および EtherIP/IPsec については、VPN クライアント側として高速なハードウェアが用意できなかったため、今回の実験では省略した。

名称	略称	トンネリングの対象レイヤ
SoftEther VPN Protocol	SEVPN	L2 (Ethernet)
L2TP/IPsec	L2TP	L3 (IPv4)
Microsoft SSTP	SSTP	L3 (IPv4)
OpenVPN Protocol (L3 ルーティングモード)	OVPNL3	L3 (Ethernet)
OpenVPN Protocol (L2 ブリッジモード)	OVPNL2	L2 (Ethernet)

付録 2.2. 比較対象の VPN サーバプログラム

SEVPN 以外の VPN プロトコルについて、従来方式における通信実験にはそれぞれ以下の VPN サーバプログラムの実装を使用した。本研究の方式における通信実験には、第 5 章で実装した SoftEther VPN Server 4.0 を使用した。

VPN プロトコル	従来方式の VPN サーバ実装	本研究での VPN サーバ実装
L2TP	Windows Server 2008 R2 に付属の「Routing and Remote Access Service (RRAS)」という VPN サーバプログラム (Microsoft Corporation が開発)	SoftEther VPN Server 4.0 (今回の研究で実装)
SSTP	Windows Server 2008 R2 に付属の「Routing and Remote Access Service (RRAS)」という VPN サーバプログラム (Microsoft Corporation が開発)	SoftEther VPN Server 4.0 (今回の研究で実装)
OVPNL3	OpenVPN 2.2.2 (OpenVPN, Inc. 開発)	SoftEther VPN Server 4.0 (今回の研究で実装)
OVPNL2	OpenVPN 2.2.2 (OpenVPN, Inc. 開発)	SoftEther VPN Server 4.0 (今回の研究で実装)

付録 2.3. 暗号化・デジタル署名アルゴリズム

各 VPN プロトコルでは、以下の暗号化・デジタル署名アルゴリズムを用いた。各 VPN プロトコルの仕様や制約により、全く同一のアルゴリズムを選択することができなかった。本実験は VPN プロトコル同士の通信スループット等を比較するものではないため、問題ないと考える。

VPN プロトコル	アルゴリズム
SEVPN	RC4-SHA1
L2TP	AES128-CBC-SHA1
SSTP	RC4-SHA1
OVPNL3	AES128-CBC-SHA1
OVPNL2	AES128-CBC-SHA1

付録 2.4. VPN クライアントプログラム

VPN クライアント側となるコンピュータでは、以下の VPN クライアントプログラムの実装を使用した。

VPN プロトコル	VPN クライアント実装
SEVPN	PacketiX VPN Client 4.0 Build 8676
L2TP	Windows Server 2008 R2 に付属の L2TP/IPsec VPN クライアント
SSTP	Windows Server 2008 R2 に付属の SSTP VPN クライアント
OVPNL3	OpenVPN 2.2.2
OVPNL2	OpenVPN 2.2.2

付録 2.5. 速度測定の方法

2 台のコンピュータ間の速度の測定には、UT-VPN に付属の TrafficServer / TrafficClient コマンドを用いた。

期間	30 秒
通信方向	・ アップロードのみ (Upload) ・ ダウンロードのみ (Download) ・ 双方向 (Full)
指定パラメータ	/TYPE:DOWN|UP|FULL /SPAN:30
計算方法	TrafficServer で TCP ポート 9821 を待受状態とし、TrafficClient から 32 本の TCP コネクションを接続する。 32 本すべての TCP コネクションが接続されたら、一斉にデータの送受信を開始する。 (双方向モードの場合は、16 本がアップロード方向、残り 16 本がダウンロード方向となる。) データの送受信が開始された時点から 30 秒間が経過したらその瞬間までに受信側に届いたデータのバイト数を集計し、bps を計算する。

付録 2.6. 遅延測定の方法

ICMP Echo Request / Echo Response (Ping パケット) を用いて 2 台のコンピュータ間のパケットの往復時間 (RTT) を測定するための .NET Framework 2.0 用の C#のプログラムを作成した。

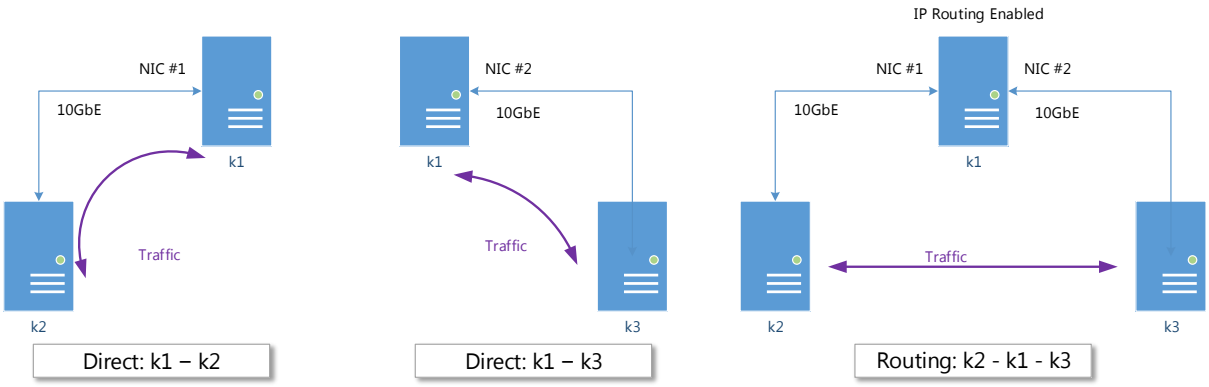
Ping パケットの送信には、System.Net.NetworkInformation.Ping クラスを用いた。RTT の測定には、System.Diagnostics.Stopwatch クラスを用いて精度の高い経過時間測定を実施した。

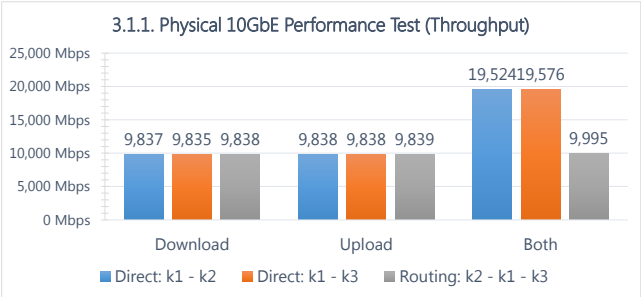
Ping パケットを送信し、受信が確認されてから次の Ping パケットを送信する実装となっている。Ping パケットの送信から受信までの間の所要時間を積算し、送受信したパケット数で割ることで RTT の平均値を算出した。パケット数は最低 10,000 パケットとした。

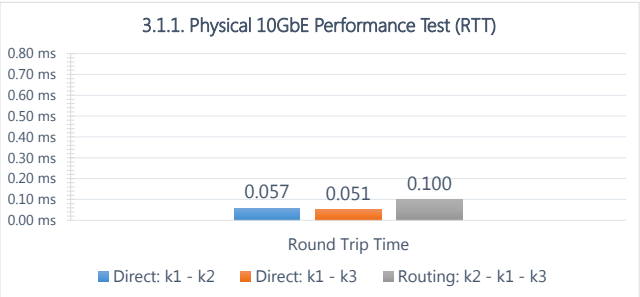
付録 3. 実験結果データ

付録 3.1. VPN を用いない物理的な通信速度の測定

最初にコンピュータや LAN カードのパフォーマンスを確認するため、単純にコンピュータ間の通信速度および RTT を測定した。

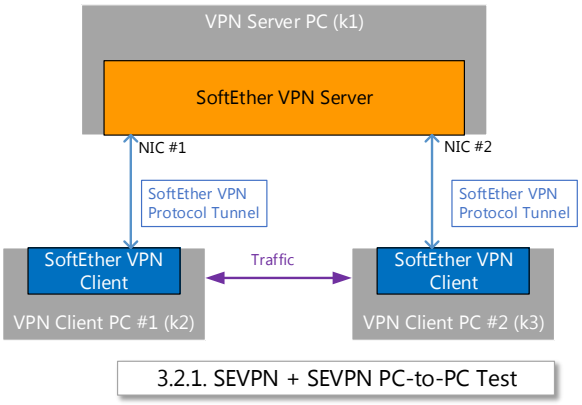

NIC #1
10GbE
k1
Traffic
k2
Direct: k1 – k2
NIC #2
10GbE
k1
Traffic
k3
Direct: k1 – k3
IP Routing Enabled
NIC #1
NIC #2
10GbE
10GbE
k1
Traffic
k2
k3
Routing: k2 - k1 - k3

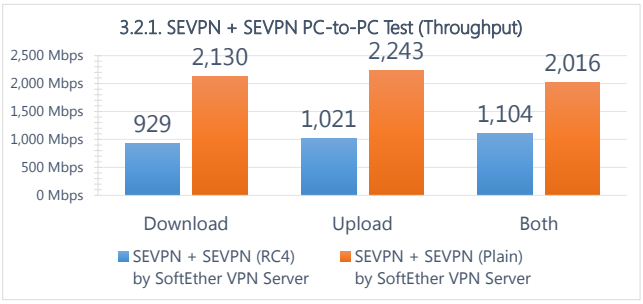

3.1.1. Physical 10GbE Performance Test (Throughput)
25,000 Mbps
20,000 Mbps
15,000 Mbps
10,000 Mbps
5,000 Mbps
0 Mbps
9,837
9,835
9,838
9,838
9,838
9,839
19,524
19,576
9,995
Download
Upload
Both
Direct: k1 - k2
Direct: k1 - k3
Routing: k2 - k1 - k3

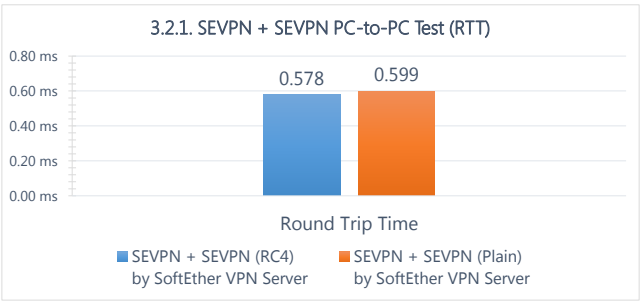

3.1.1. Physical 10GbE Performance Test (RTT)
0.80 ms
0.70 ms
0.60 ms
0.50 ms
0.40 ms
0.30 ms
0.20 ms
0.10 ms
0.00 ms
0.057
0.051
0.100
Round Trip Time
Direct: k1 - k2
Direct: k1 - k3
Routing: k2 - k1 - k3

付録 3.2. 単一の VPN プロトコルの性能測定

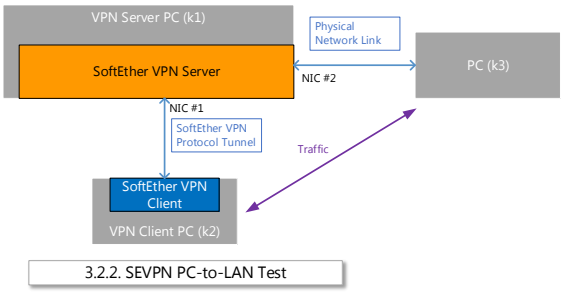
付録 3.2.1. 本研究実装における 2 台の SEVPN クライアント間の通信性能測定（PC to PC 接続）

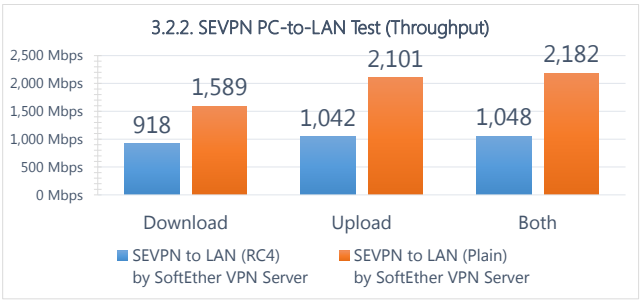


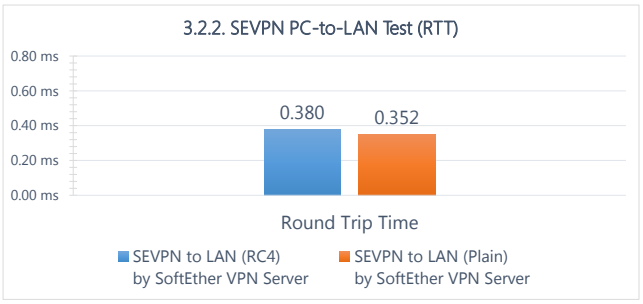




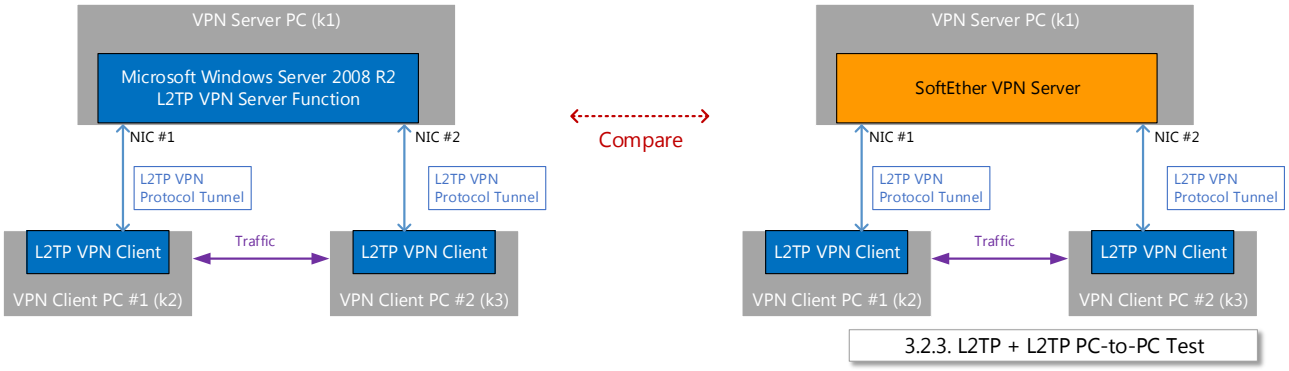
付録 3.2.2. 本研究実装における 1 台の SEVPN クライアントと LAN との間の通信性能測定（PC to LAN 接続）

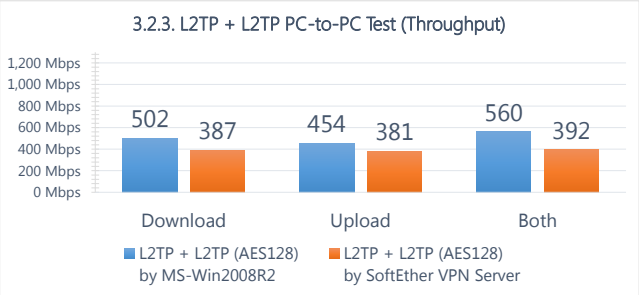


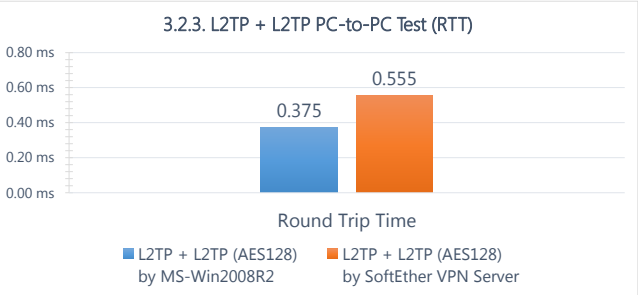




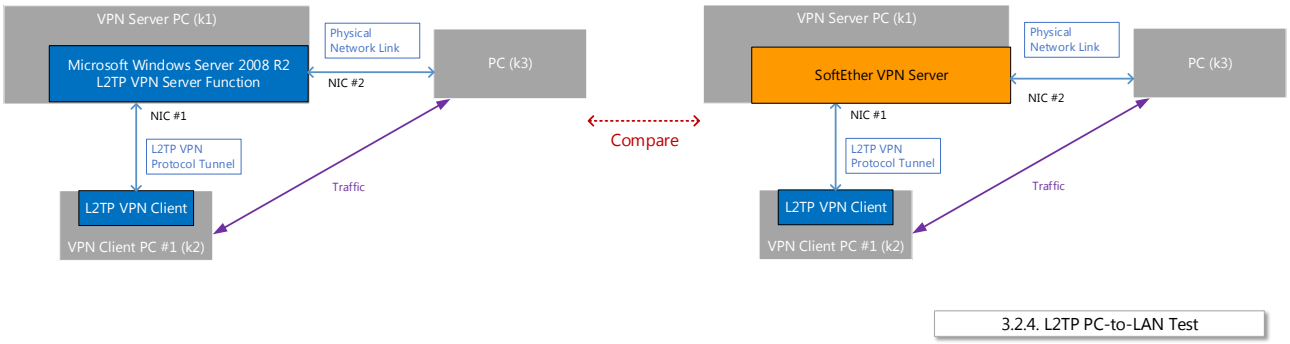
付録 3.2.3. Microsoft 社実装と本研究実装との L2TP 性能比較（PC to PC 接続）

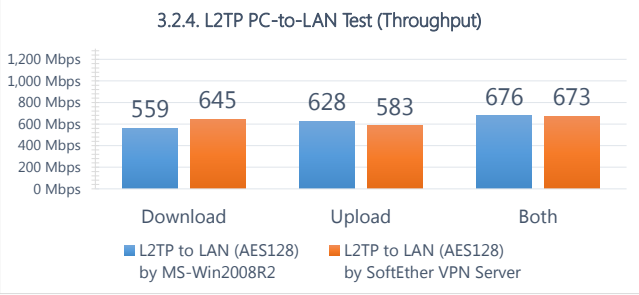


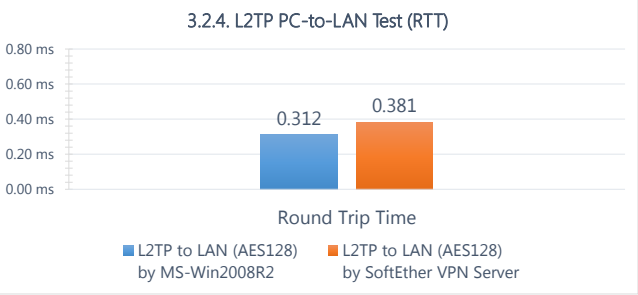




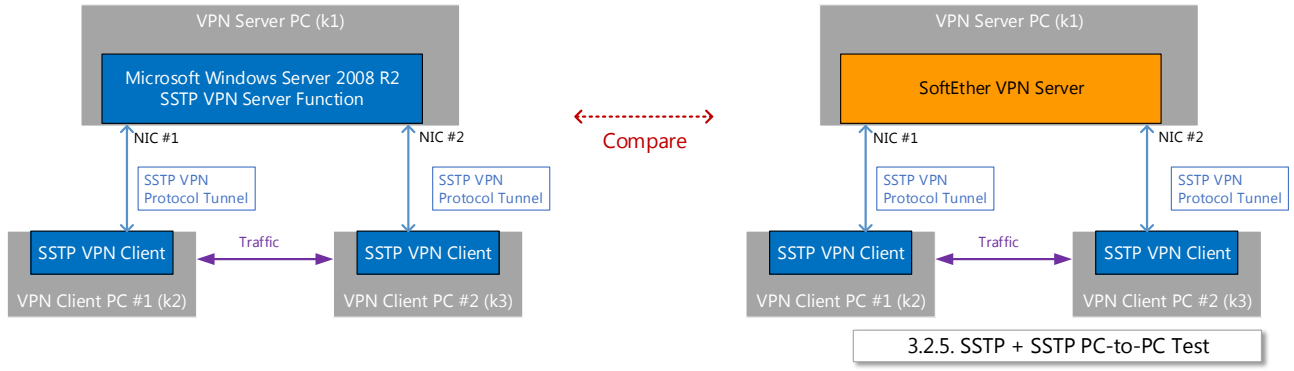
付録 3.2.4. Microsoft 社実装と本研究実装との L2TP 性能比較（PC to LAN 接続）

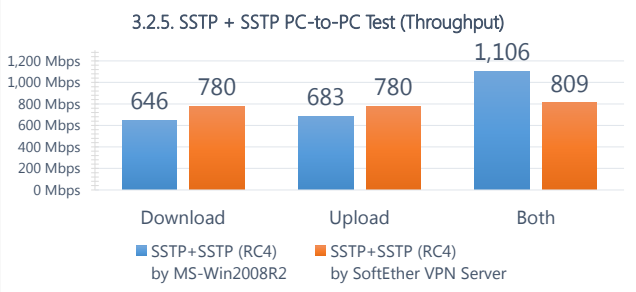


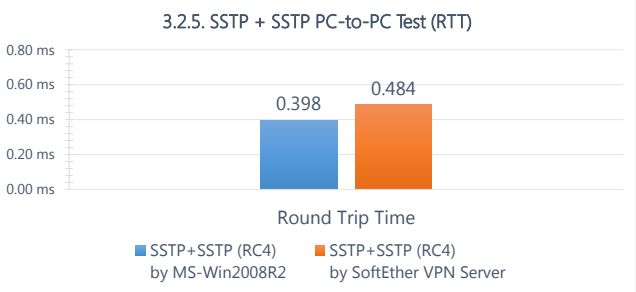




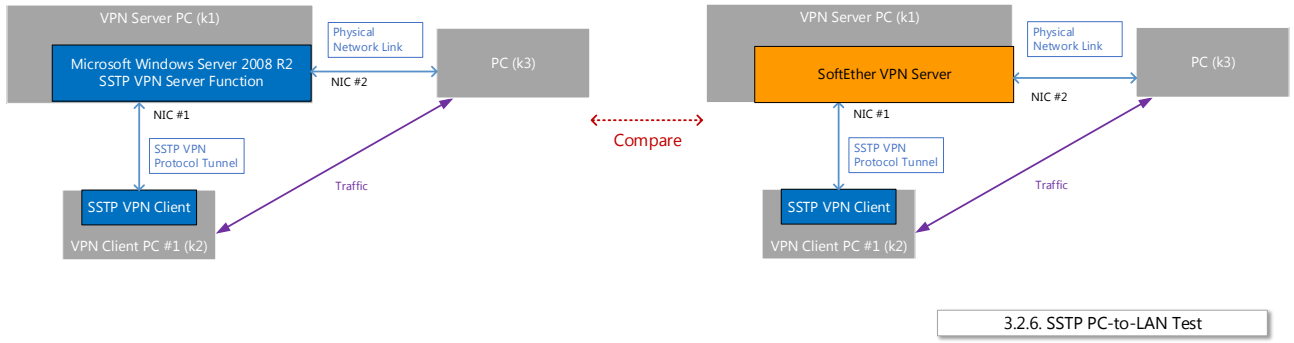
付録 3.2.5. Microsoft 社実装と本研究実装との SSTP 性能比較（PC to PC 接続）

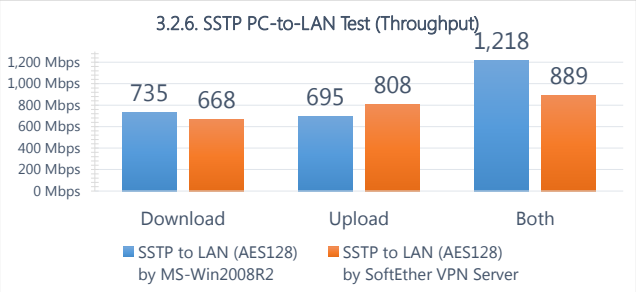


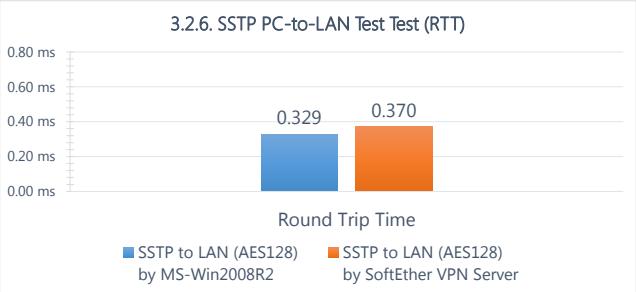




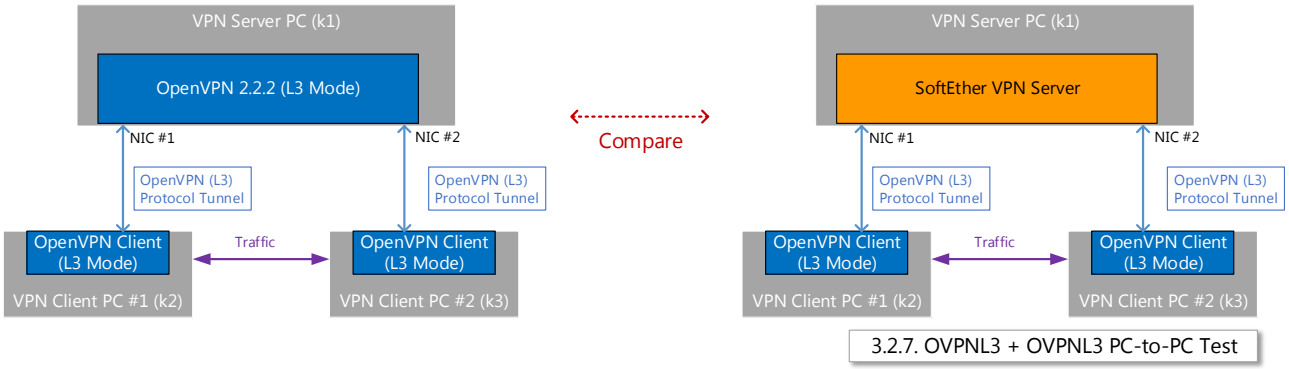
付録 3.2.6. Microsoft 社実装と本研究実装との SSTP 性能比較（PC to LAN 接続）

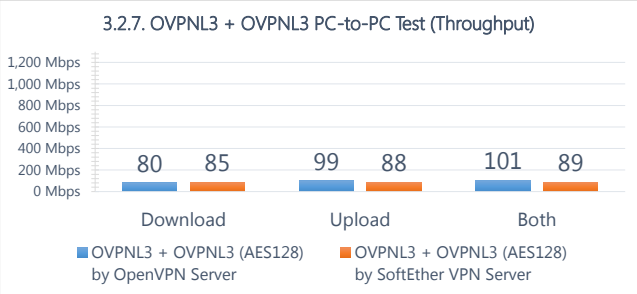


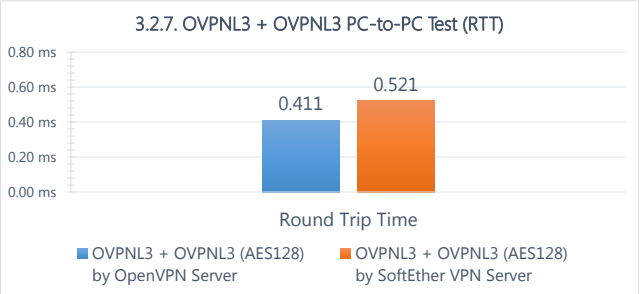




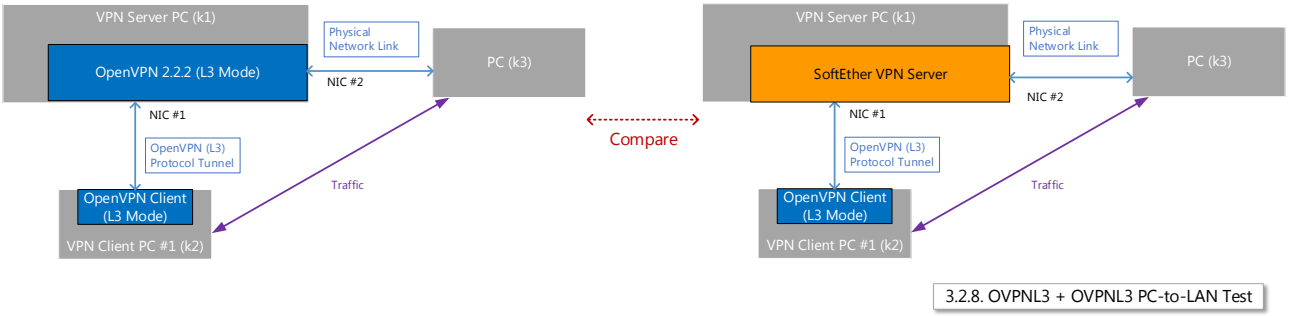
付録 3.2.7. OpenVPN 社実装と本研究実装との OpenVPN (L3) 通信性能比較 (PC to PC 接続)

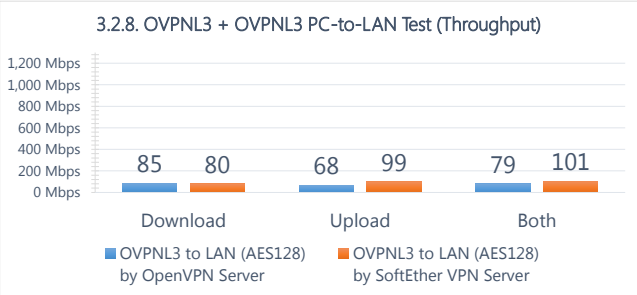


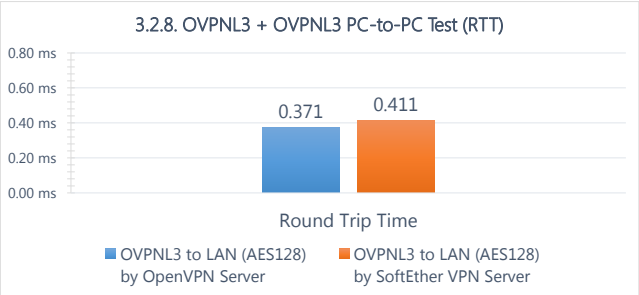




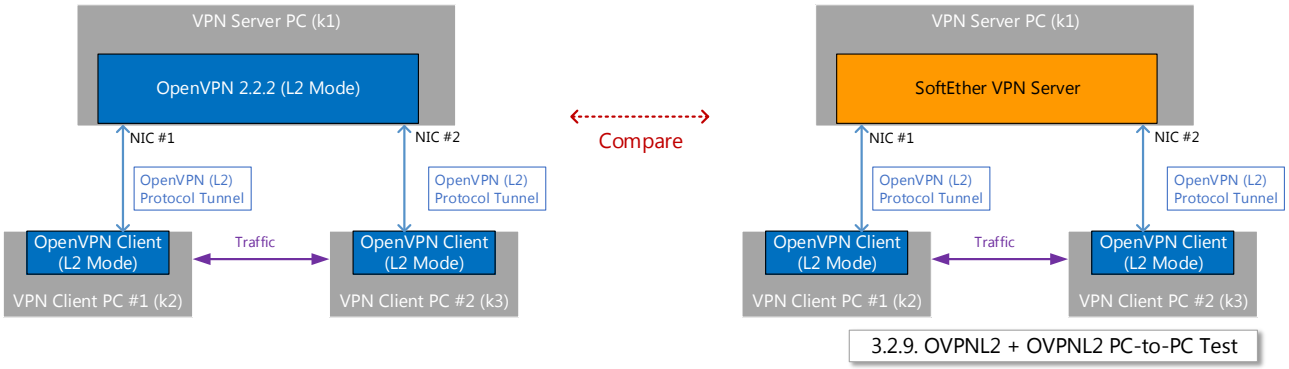
付録 3.2.8. OpenVPN 社実装と本研究実装との OpenVPN (L3) 通信性能比較 (PC to LAN 接続)

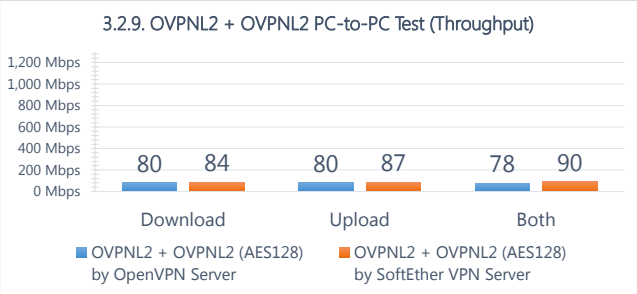


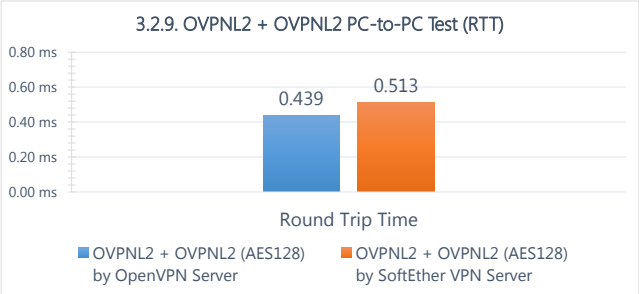




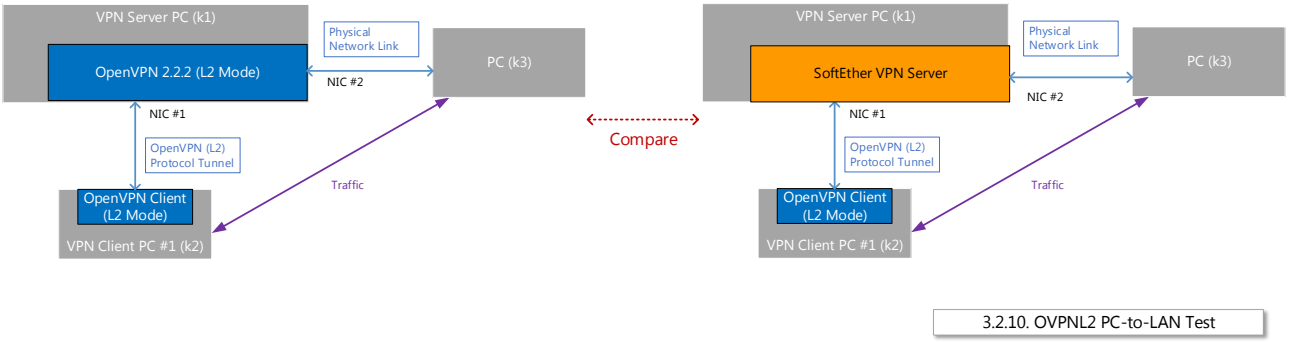
付録 3.2.9. OpenVPN 社実装と本研究実装との OpenVPN (L2) 通信性能比較（PC to PC 接続）

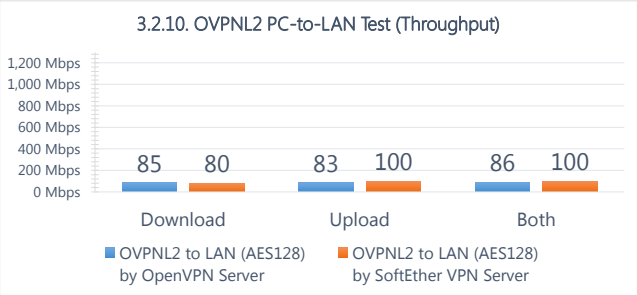


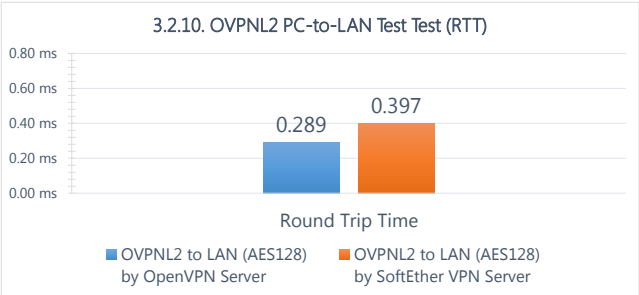




付録 3.2.10. OpenVPN 社実装と本研究実装との OpenVPN (L2) 通信性能比較（PC to LAN 接続）

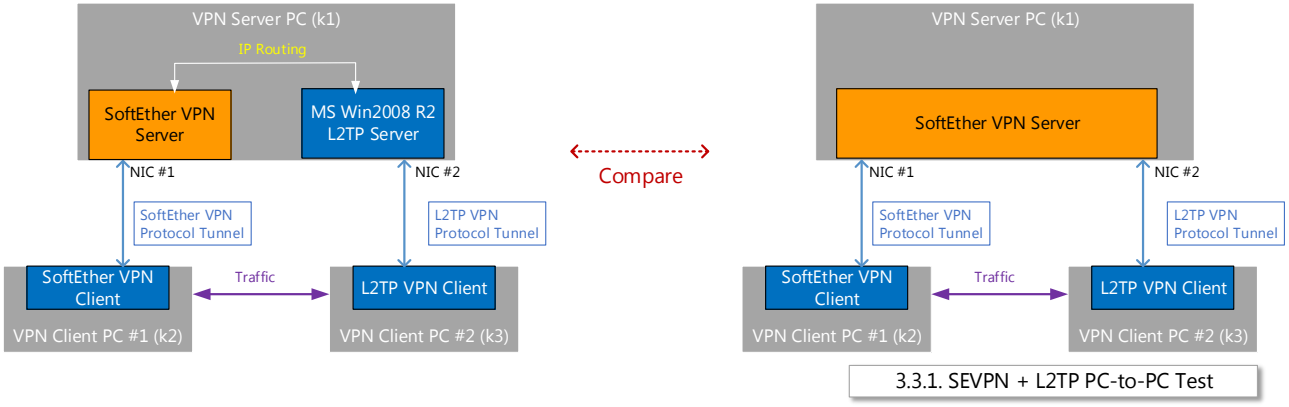


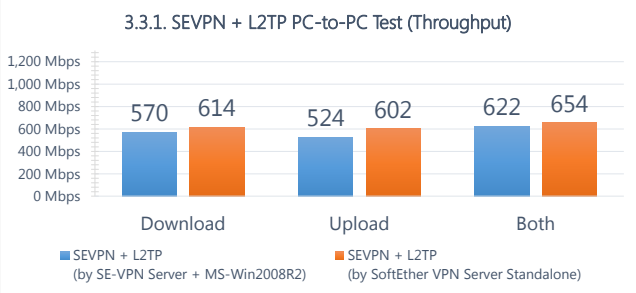


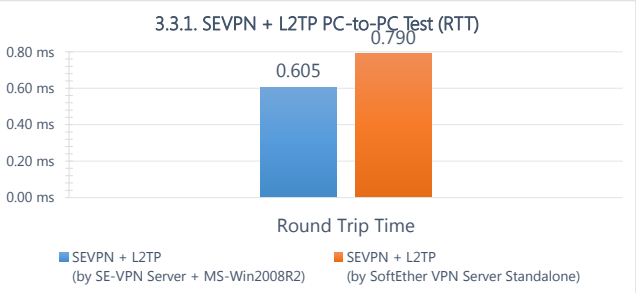


付録 3.3.1. SEVPN クライアント - L2TP クライアント間で通信する場合の従来手法と本研究との通信性

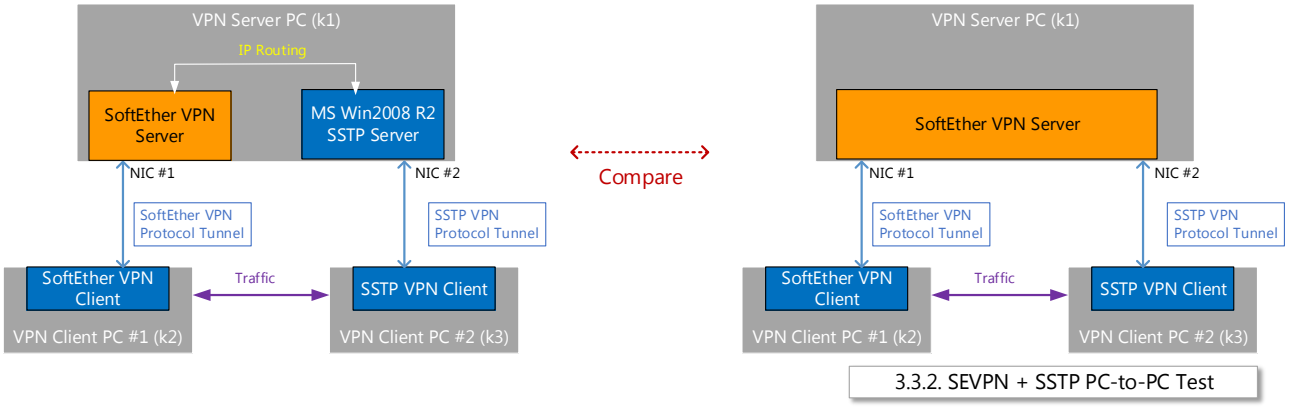
能比較

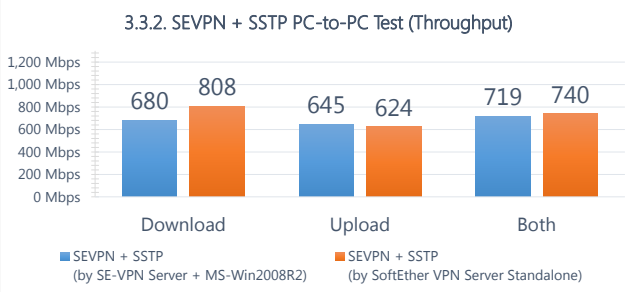


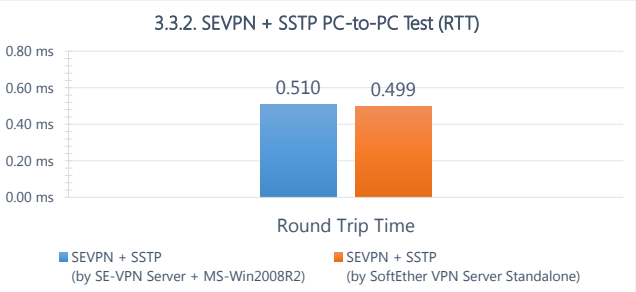




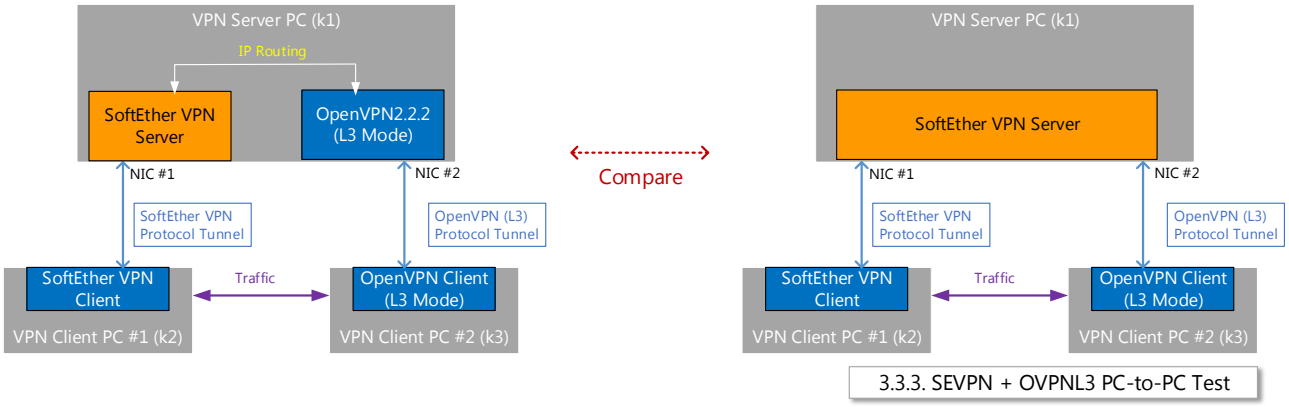
付録 3.3.2. SEVPN クライアント - SSTP クライアント間で通信する場合の従来手法と本研究との通信性能比較

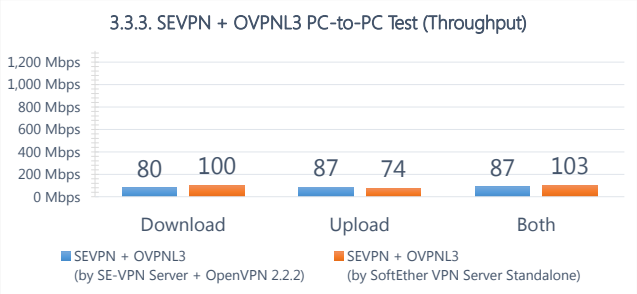


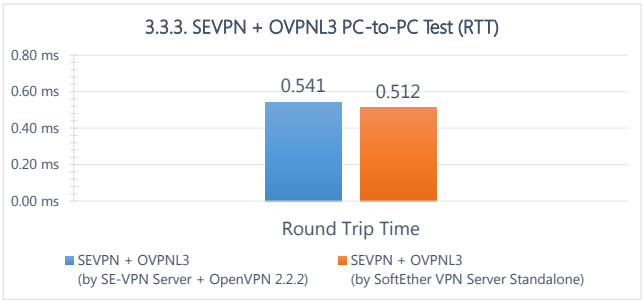




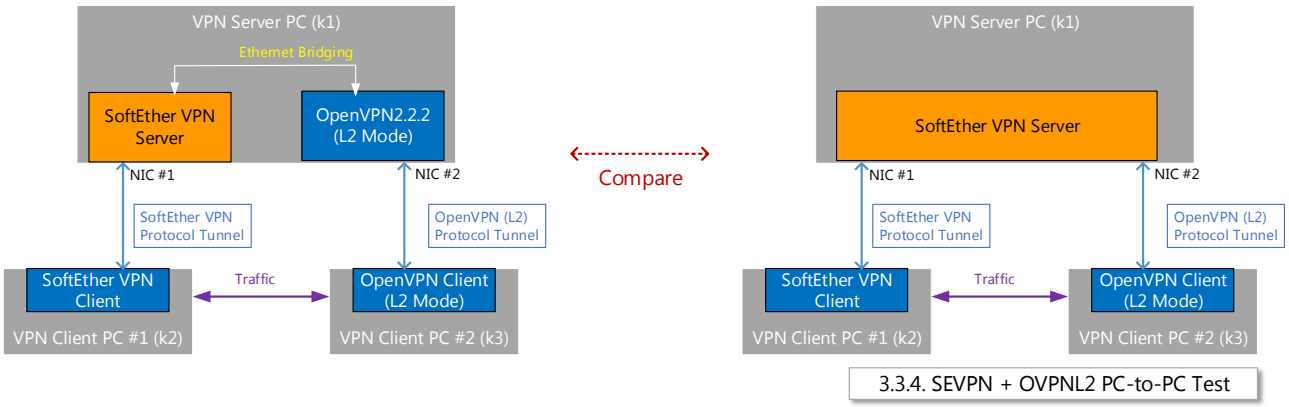
付録 3.3.3. SEVPN クライアント - OpenVPN（L3）クライアント間で通信する場合の従来手法と本研究との通信性能比較

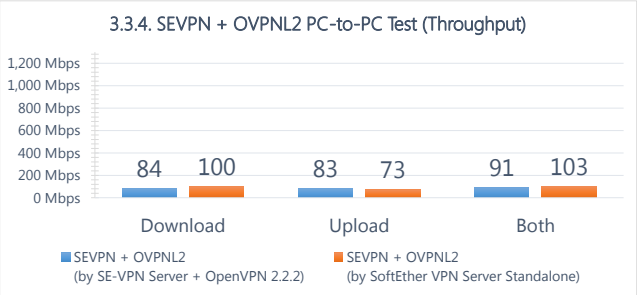


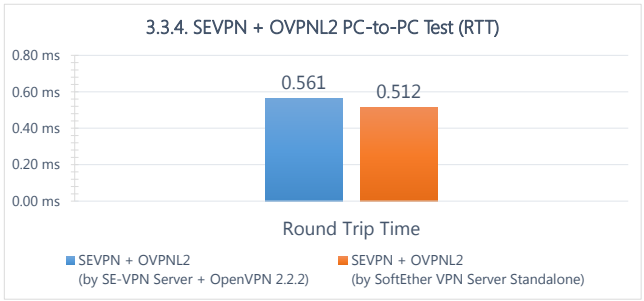




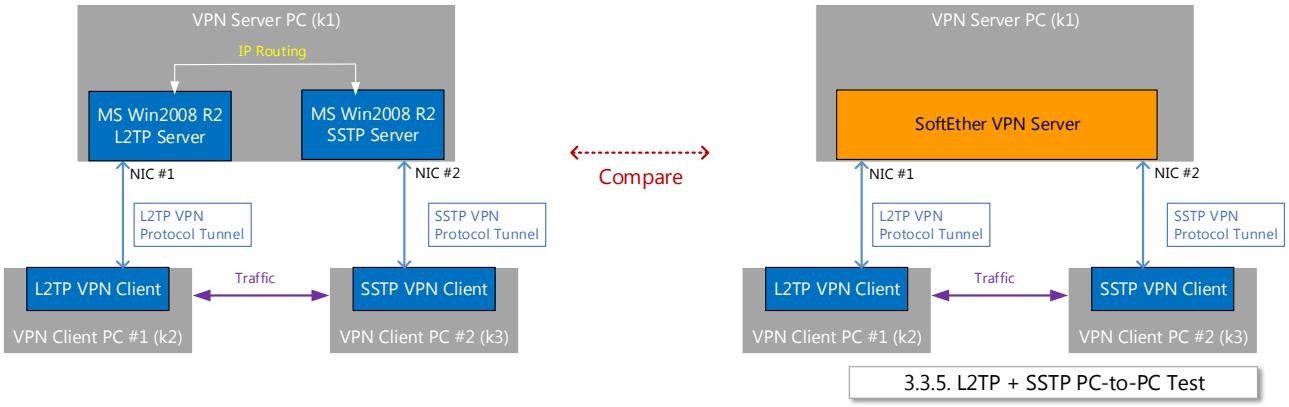
付録 3.3.4. SEVPN クライアント - OpenVPN（L2）クライアント間で通信する場合の従来手法と本研究との通信性能比較

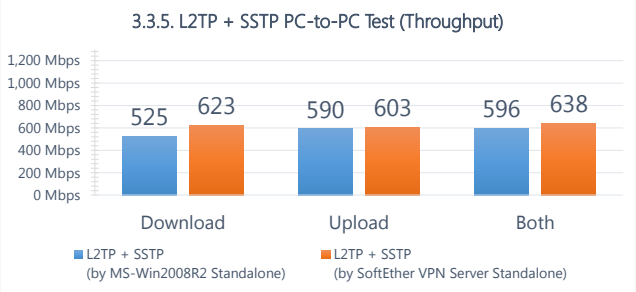


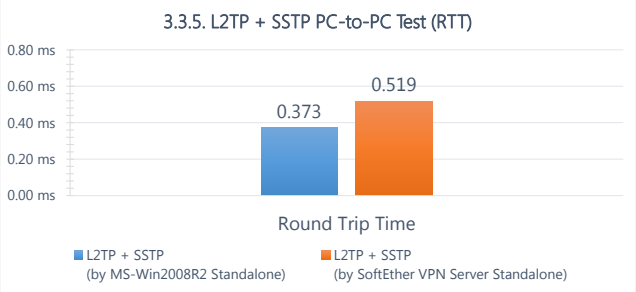




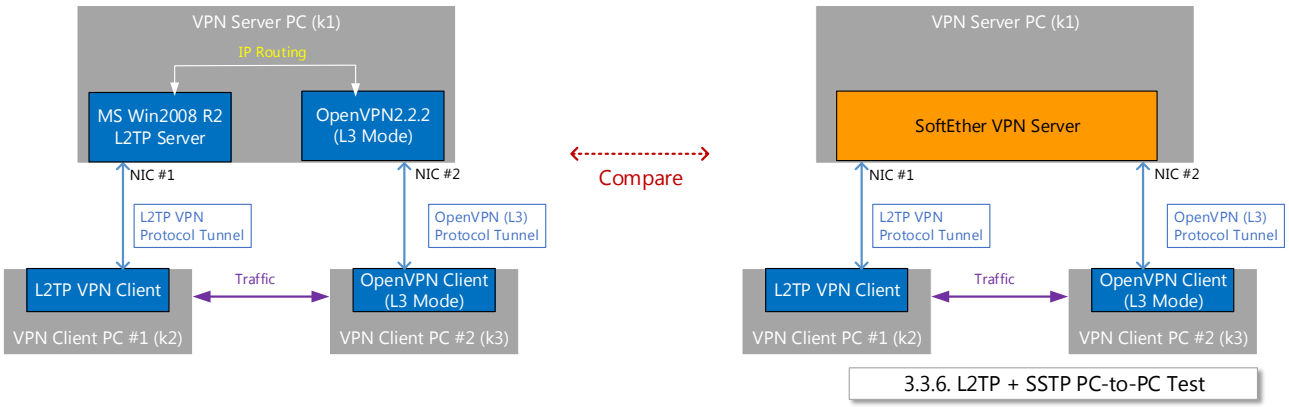
付録 3.3.5. L2TP クライアント - SSTP クライアント間で通信する場合の従来手法と本研究との通信性能比較

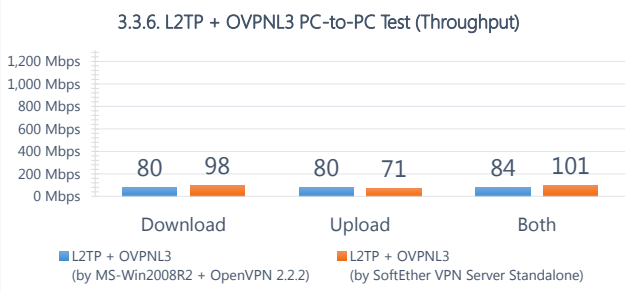


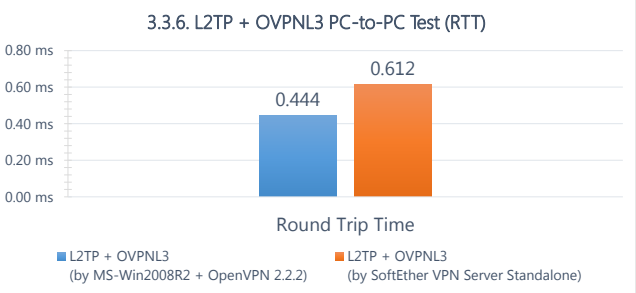




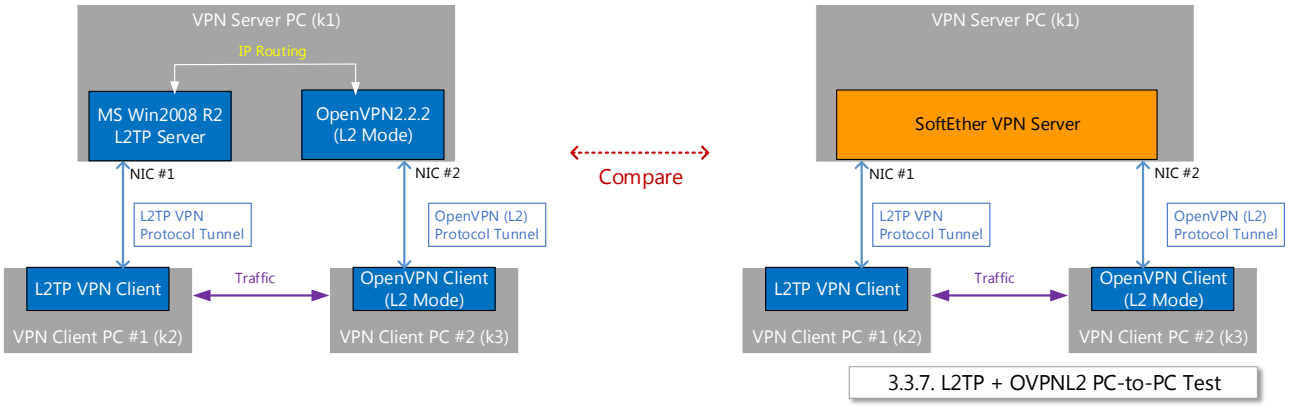
付録 3.3.6. L2TP クライアント - OpenVPN (L3) クライアント間で通信する場合の従来手法と本研究との通信性能比較

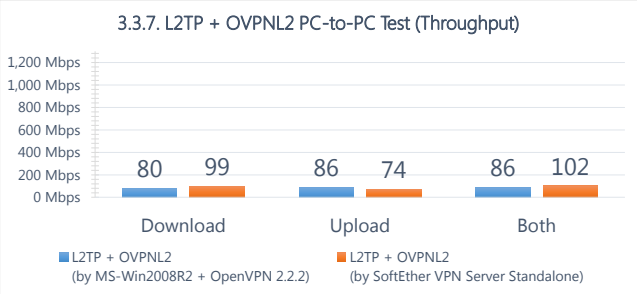


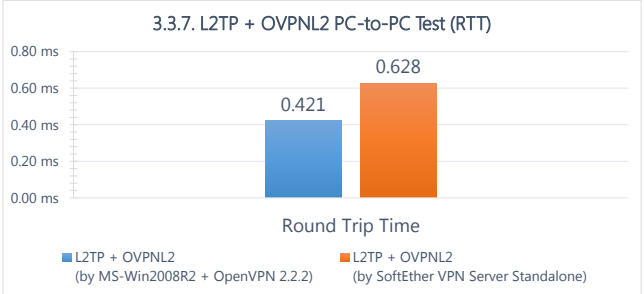




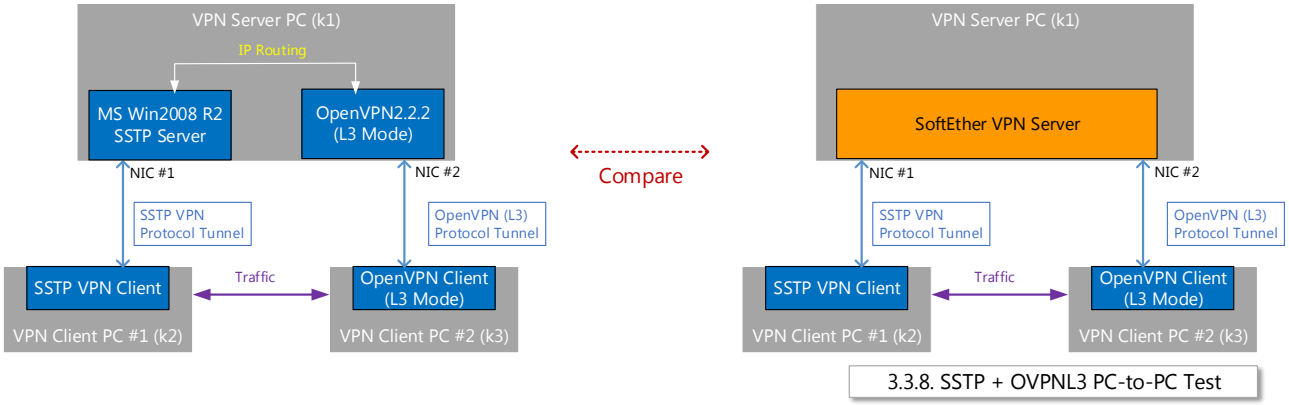
付録 3.3.7. L2TP クライアント - OpenVPN（L2）クライアント間で通信する場合の従来手法と本研究との通信性能比較

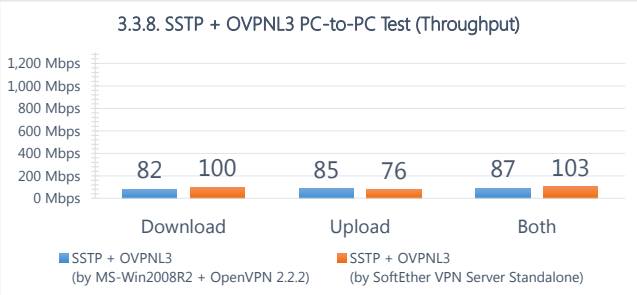


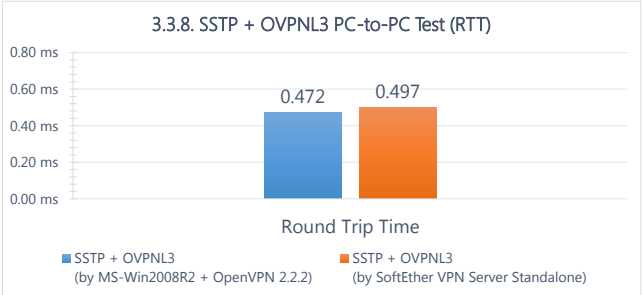




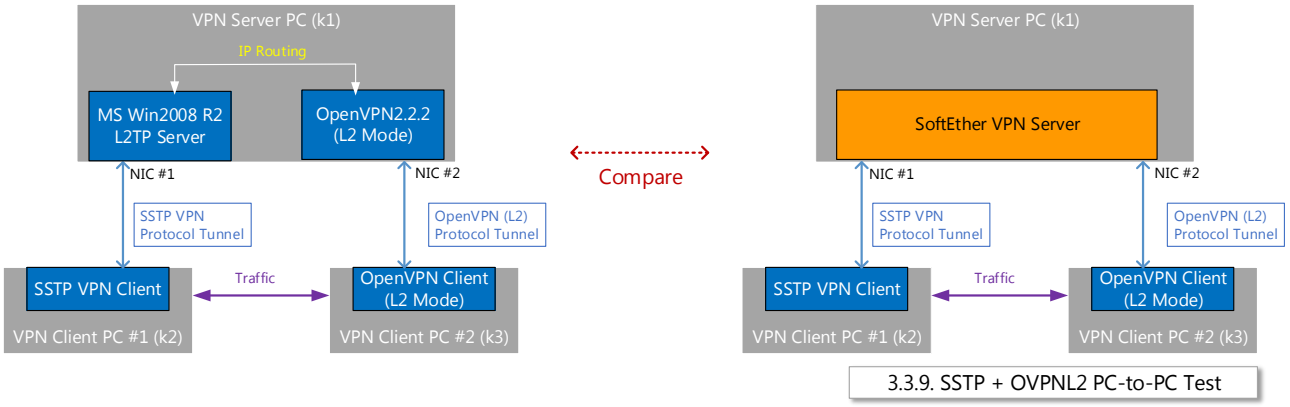
付録 3.3.8. SSTP クライアント - OpenVPN（L3）クライアント間で通信する場合の従来手法と本研究との通信性能比較

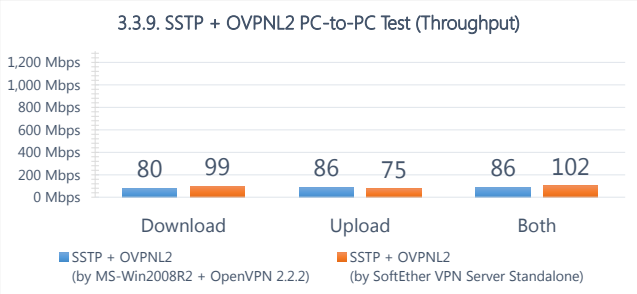


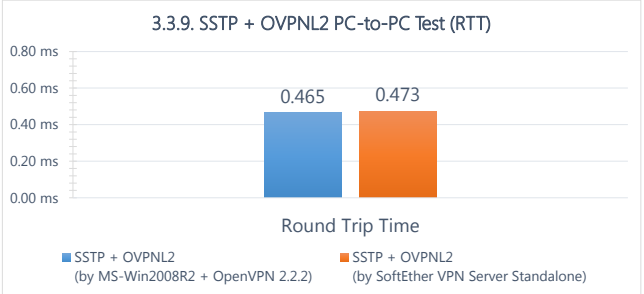




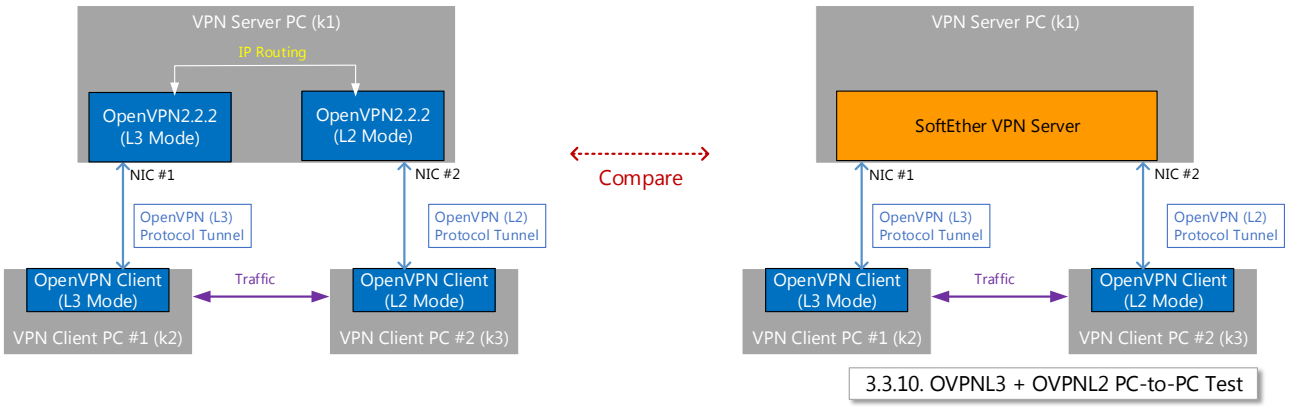
付録 3.3.9. SSTP クライアント - OpenVPN (L2) クライアント間で通信する場合の従来手法と本研究との通信性能比較

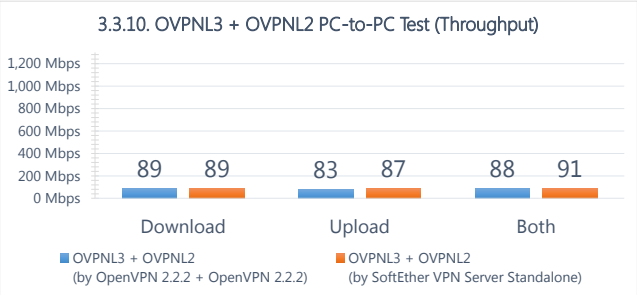


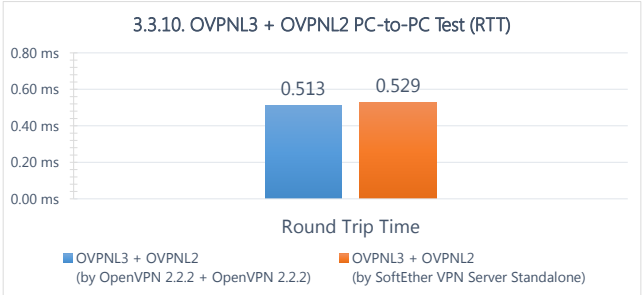




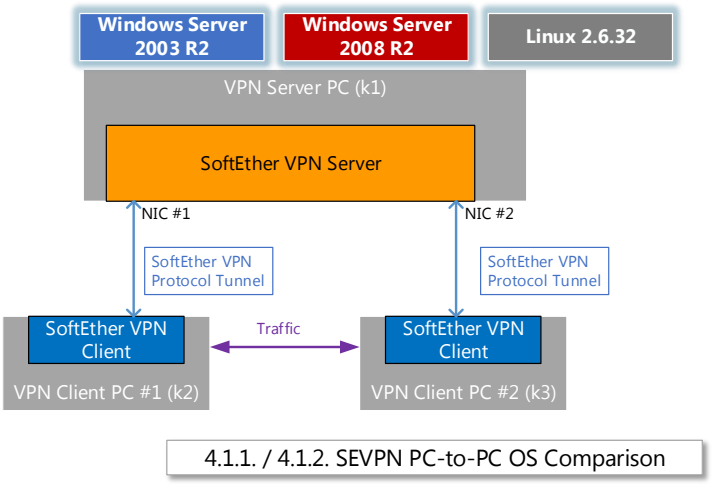
付録 3.3.10. OpenVPN (L3) クライアント - OpenVPN (L2) クライアント間で通信する場合の従来手法と本研究との通信性能比較

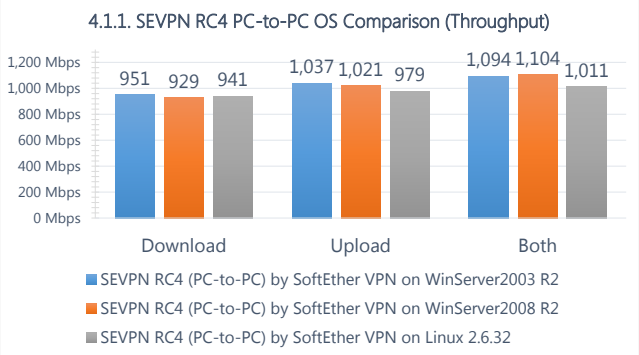


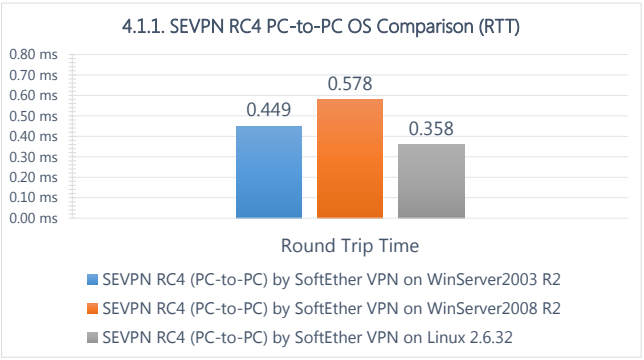




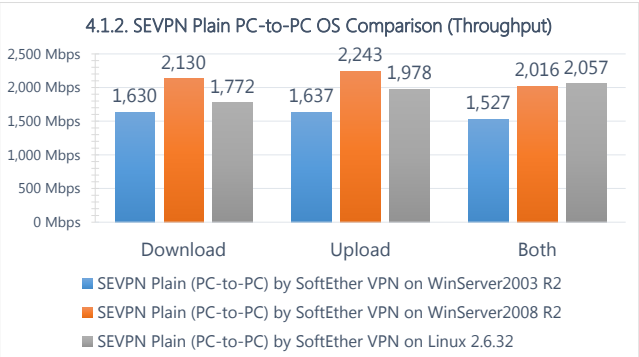
付録 4.1.1. 本研究で実装した VPN Server の PC to PC 接続の OS 間の性能比較（SEVPN, RC4 暗号化）

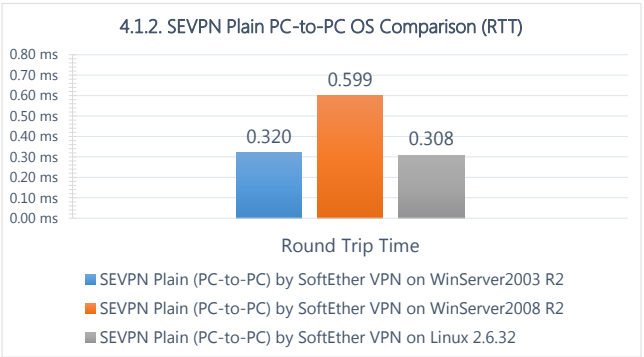






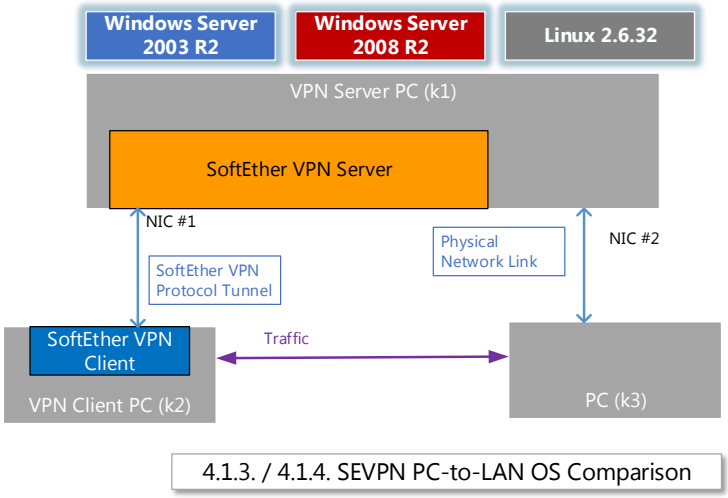
付録 4.1.2. 本研究で実装した VPN Server の PC to PC 接続の OS 間の性能比較（SEVPN, 平文）

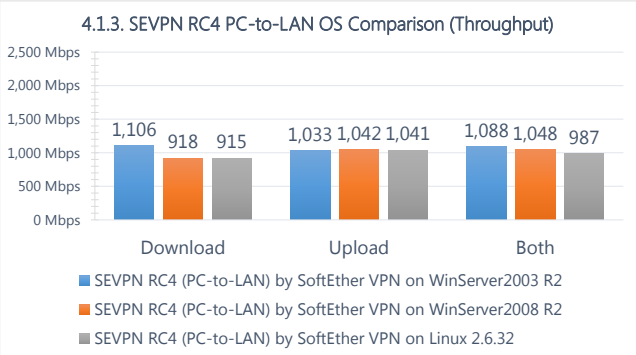


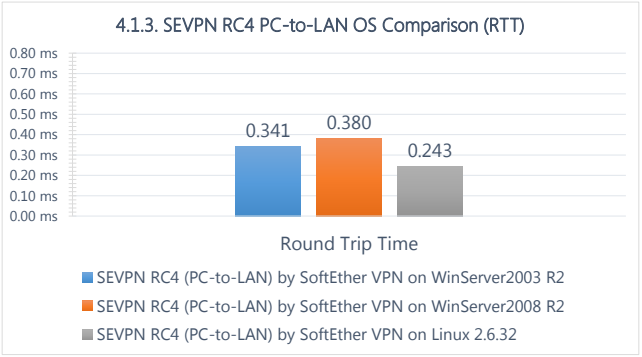


付録 4.1.3. 本研究で実装した VPN Server の PC to LAN 接続の OS 間の性能比較（SEVPN,

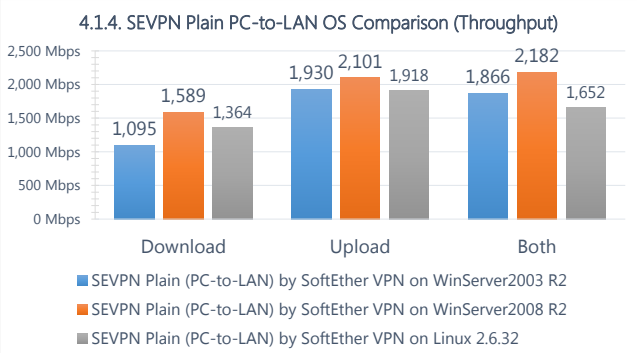
RC4 暗号化)

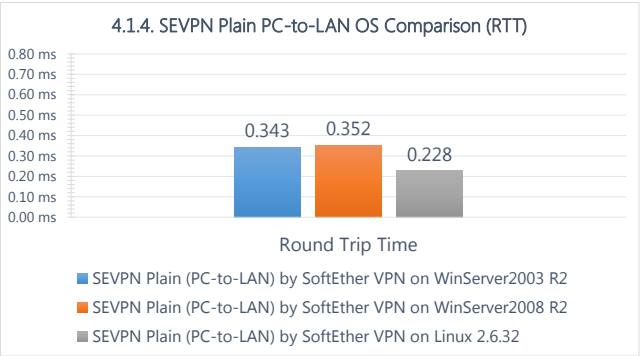




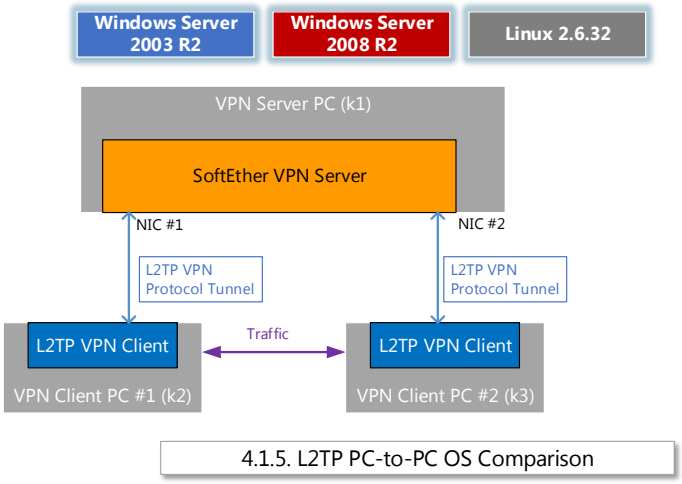


付録 4.1.4. 本研究で実装した VPN Server の PC to LAN 接続の OS 間の性能比較（SEVPN，平文)

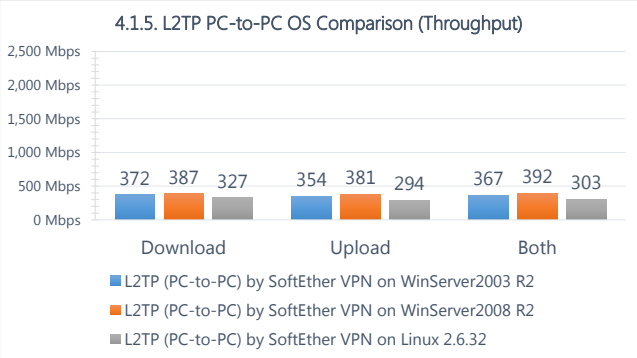


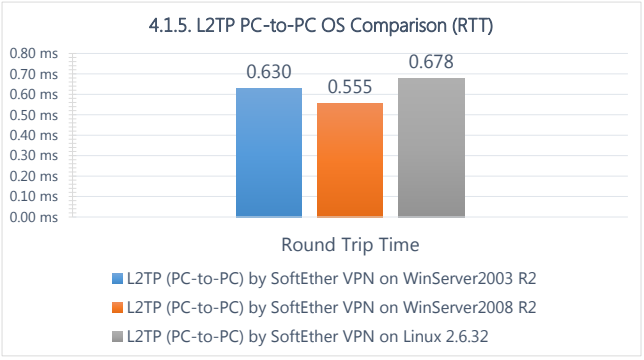


付録 4.1.5. 本研究で実装した VPN Server の PC to PC 接続の OS 間の性能比較（L2TP）

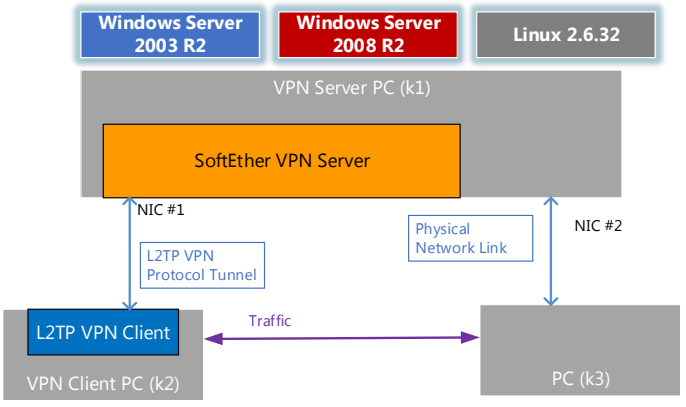


4.1.5. L2TP PC-to-PC OS Comparison

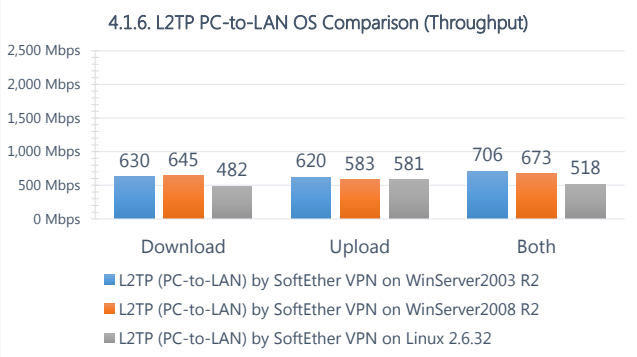


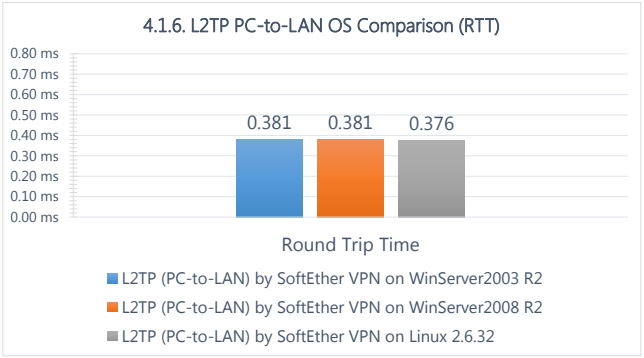


付録 4.1.6. 本研究で実装した VPN Server の PC to LAN 接続の OS 間の性能比較（L2TP）



4.1.6. L2TP PC-to-LAN OS Comparison


4.1.6. L2TP PC-to-LAN OS Comparison (Throughput)
2,500 Mbps
2,000 Mbps
1,500 Mbps
1,000 Mbps
500 Mbps
0 Mbps
630
645
482
620
583
581
706
673
518
Download
Upload
Both
L2TP (PC-to-LAN) by SoftEther VPN on WinServer2003 R2
L2TP (PC-to-LAN) by SoftEther VPN on WinServer2008 R2
L2TP (PC-to-LAN) by SoftEther VPN on Linux 2.6.32


4.1.6. L2TP PC-to-LAN OS Comparison (RTT)
0.80 ms
0.70 ms
0.60 ms
0.50 ms
0.40 ms
0.30 ms
0.20 ms
0.10 ms
0.00 ms
0.381
0.381
0.376
Round Trip Time
L2TP (PC-to-LAN) by SoftEther VPN on WinServer2003 R2
L2TP (PC-to-LAN) by SoftEther VPN on WinServer2008 R2
L2TP (PC-to-LAN) by SoftEther VPN on Linux 2.6.32