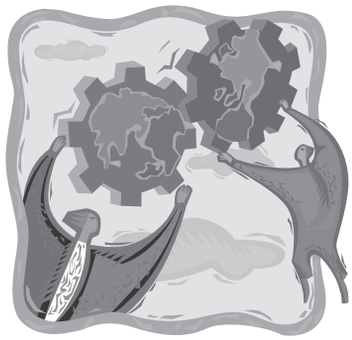
解　説

SoftEtherの内部構造

Internal Structure of SoftEther



登　大遊　筑波大学第三学群情報学類
yagi@coins.tsukuba.ac.jp

SoftEtherの特徴

SoftEtherはVPN（Virtual Private Network）を構築するためのソフトウェアである[5]．特徴として，Layer-2（データリンク層）を仮想化していることが挙げられる．そのために，ソフトウェアによってLANカードおよびスイッチングHUBを仮想化している．これらを**仮想LANカード**および**仮想HUB**と呼ぶ．SoftEtherはクライアント／サーバ型の通信ソフトウェアである．SoftEtherを使用してVPNを構築する場合は，1台のサーバコンピュータ上に仮想HUBをインストールし，仮想LANカードをインストールした複数台のクライアントコンピュータから同時に仮想HUBに接続する．すると，クライアントコンピュータ間はデータリンク層で自由に通信を行うことができるようになる（**図-1**）．

その他の特徴として，仮想LANカードが仮想HUBにTCP/IPを用いたプロトコルによって接続するという点が挙げられる．従来のPPTPやL2TPなどのVPNプロトコルは，GREやIPsecなどのプロトコルを用いてコンピュータ間を接続する．しかし，現在LAN内とインターネットとの間にNATやプロキシサーバ，ファイアウォールなどの装置が設置されていることが多く，これらのプロトコルを通すことができない場合はPPTPやL2TPによるVPNを構成することができなかった．SoftEtherプロトコルは，NATやプロキシサーバを通過して通信することができ，ネットワーク管理者の許可があれば，従来のVPNでは特殊なパケットを通過させる必要があったファイアウォールの特別な設定変更を行うことなく通信することもできるので，従来のVPNプロトコルと比べてより幅広い環境で使用することが可能である．

SoftEtherは平成15年度IPA未踏ソフトウェア創造事業未踏ユース部門の支援を受けて開発を行った．2003年12月にWebサイト（http://www.softether.com/）上でフリーソフトウェアとして公開され，平成16年9月時点でダウンロード数は170万を超えている．

SoftEtherを構成するモジュール

○SoftEtherのコンポーネント

SoftEtherはいくつかのコンポーネントに分けて実装されている．現在，仮想HUBはWindowsおよびLinux上で動作している，仮想LANカードは現在Windows上でのみ動作する．**表-1** は，Windows上のSoftEther実装に含まれているコンポーネントである．

クライアント側プログラムの実装

○ユーザモードとカーネルモードに分割された実装

従来のVPNソフトウェア，たとえばWindowsに標準搭載されているPPTPやL2TPなどのクライアントプログラムは，すべての処理がカーネルモードで実装されて

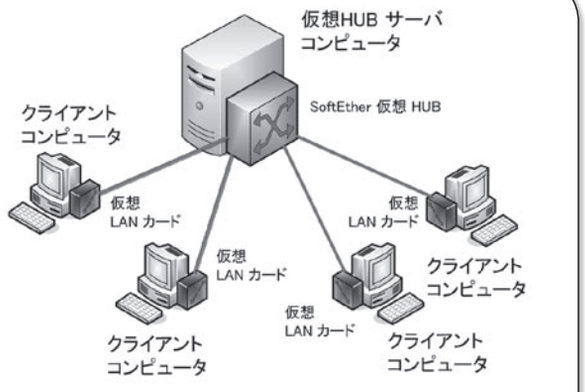


図-1　SoftEtherの仮想HUBと仮想LANカードの関係

コンポーネント名		機能と目的	動作モード
クライアント側	仮想 LAN カードドライバ	LAN カードとして OS やアプリケーションから認識される．OS のプロトコルスタックと仮想 LAN カードユーザモードプログラムとの間で動作する．	カーネルモード
	仮想 LAN カード ユーザモード プログラム	仮想 LAN カードドライバと遠隔地にある仮想 HUB の間を中継するプロセス．パケットのカプセル化（暗号化・電子署名）／カプセル化解除を行う．	ユーザモード
	接続マネージャ プログラム	仮想 LAN カードユーザモードプロセスをユーザが操作するための GUI を提供する．	ユーザモード
サーバ側	仮想 HUB プログラム	仮想 LAN カードをインストールした複数のクライアントコンピュータからの接続を受け付けるサーバプロセス．仮想的なスイッチング HUB として動作し，複数のクライアント間のパケット転送処理を行う．	ユーザモード

表 -1　SoftEther を構成するモジュールの一覧

いる．これに比べて，SoftEtherのクライアントプログラムはユーザモードとカーネルモードの2つに分けて実装しており，動作中はその間で絶えず通信を行っている．この実装方法ではユーザモード／カーネルモード間の切り替え処理によるオーバーヘッドが存在するので，すべてカーネルモードで実装するという方法も考えられるが，以下のような理由でユーザモードとカーネルモードに分離して実装することにした．

1．移植性を高めるため

SoftEtherの開発における目標の1つとして，できる限りOSに依存しないコードを記述するというものがある．現在，Win32ユーザモードとLinux/UNIXにおけるユーザモードの両方で同等に動作するプログラムを記述することは容易であるが，Win32 カーネルモードと他のOSのカーネルモードで同等に動作するプログラムを書くことは難しい．

そこで，ユーザモードで動作するプロセスとカーネルモードで動作するドライバの2つに分割し，OSに依存するコードはドライバ側のみで記述し，ユーザモードプログラムではOSに依存しないコードを記述した．それにより，各OSに対応したドライバのコードを開発することによって容易に別のOSに移植することができるようにした．

2．SSL通信を使用するため

SoftEtherはセキュアな暗号化通信を実現するためにSSLを使用している．SSLによる暗号化を行うためにはOpenSSLなどのライブラリが必要である．しかし，カーネルモードで動作するSSLの実装がまだ存在していない．仮にOpenSSLを独自にカーネルモードに移植する作業を行うことにした場合は膨大なコストが発生する．

3．OSが持つTCP/IPスタックを利用するため

SoftEtherではトンネリング通信にはTCPを使用している．もし，すべての処理をカーネルモードで実装しようとすると，仮想LANカードのプログラムは独自にTCPパケットを生成し，すでにコンピュータに装着されている別のLANカードに対して直接MACフレームを送信したり，また受信したMACフレームからTCPストリームを組み立てたりする必要がある．しかしながら，独自にTCPスタックやIPスタックを制限の多いカーネルモード内で実装することはやはり困難であり，仮に実装できたとしてもそれは既存のTCPスタックの性能には遥かに及ばないものになる可能性もある．

そこでもう1つの方法として，OSがあらかじめ持っているTCPスタックにカーネル内で直接データを投げることも検討した．その場合，Win32 カーネルモード内に存在するTDI（Transport Driver Interface）を呼び出すことによりカーネルモード内のみでTCP/IP通信を行う必要がある．しかしながら，TDIはWin32 カーネル内のみの実装である上に，そもそもTDIに関するドキュメントがほとんど存在しないため，プログラミングが困難であることが分かった．

○仮想LANカード

SoftEtherの仮想LANカードは，OSやアプリケーションから見ると1枚の物理的なLANカードと同等に認識される．しかし，実際には仮想デバイスであり，実際の通信は既存の物理的なLANカードが行うことになる．

図-2にWindows 2000以降が動作しているコンピュータ上にインストールされた，仮想LANカードドライバ，および仮想LANカードユーザモードプログラム，接続マネージャプログラムの間の関係とパケットの流れを示す．

Windowsにおける一般的なLANカードドライバは，NDIS（Network Driver Interface Specification）と呼ばれる規格に従った層構造の最下層で動作する．一般的なLANカードのデバイスドライバは**NDISミニポートドライバ**と呼ばれるコンポーネントとしてWindowsに認識される[4]．

一般的なLANカードは，対応したデバイスドライバが

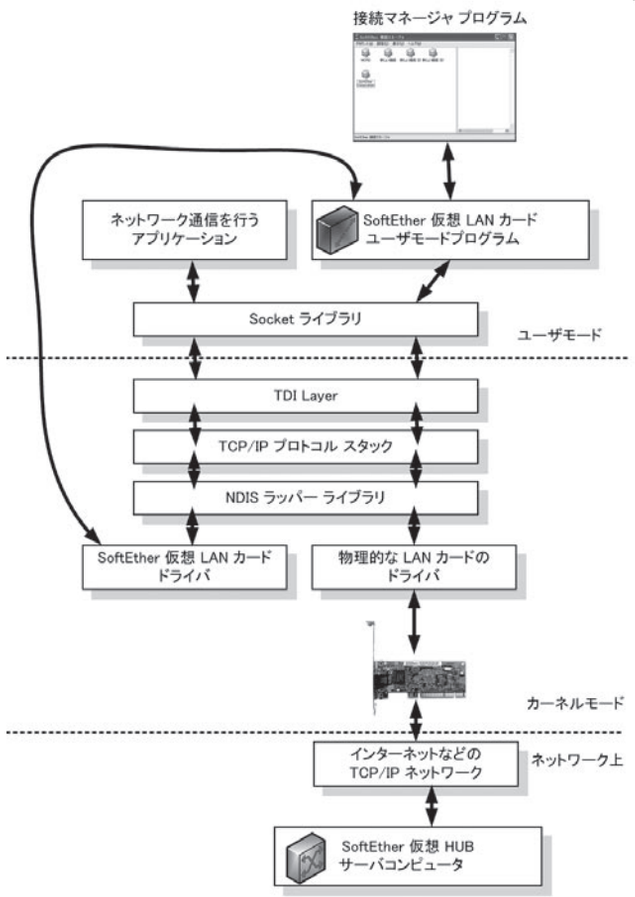


図 -2　SoftEther クライアントにおけるパケットデータの流れ

存在し，OSがデバイスドライバをロードすることによって稼働する．Windowsでは，LANカードのデバイスドライバはNDISミニポートドライバとして実装されるのが一般的である．ネットワーク通信を行うアプリケーションがSocketライブラリなどを使用してデータを送信しようとすると，OSが持つTCP/IPスタックがIPパケットを生成し，MACフレームが付加されたEthernetパケット（MACフレーム）のかたちでNDISミニポートドライバに引き渡される．

同様に，SoftEtherの仮想LANカードドライバもNDISミニポートドライバとして実装されている．そのため，Windowsやアプリケーションからは本物のLANカードのように見える．しかし，通常のLANカードのドライバとは異なり，仮想LANカードドライバは受け取ったEthernetパケットを直接ハードウェアに転送することはない．代わりに，Ethernetパケットを別のプログラムとして動作している仮想LANカードユーザモードプログラムに渡す．ユーザモードプログラムは渡されたEthernetパケットを順番にカプセル化し，TCPストリームとして接続先の仮想HUBに送信する．逆に，仮想HUBから受信したデータのカプセル化を解除し，仮想LANカードドライバに渡す．

○TCP/IPスタックの通過

仮想LANカードユーザモードプログラムによってカプセル化されたデータは既存の物理的なLANカードを経由してTCP/IPネットワーク（インターネットなど）に流れていく．仮想LANカードユーザモードプログラムが直接，物理的なLANカードにデータを渡すのではなく，OSのTCP/IPプロトコルスタックの助けを借りている．送信するべきEthernetパケットを受け取った仮想LANカードユーザモードプログラムが，OSの持つSocketライブラリの関数を呼び出すことにより，カプセル化されたデータを遠隔地の仮想HUBに対して行っている．仮想HUBからのデータの受信についても同様にSocket関数を呼び出している．この点において，仮想LANカードユーザモードプログラムは，一般的なユーザモードで動作するネットワークアプリケーションと等価である．SoftEtherのクライアント側では，ネットワークアプリケーションのデータは2回，OS内のプロトコルスタックやNDIS中間層を通過することになる．

○カーネルモードとユーザモード間のデータの受け渡し

仮想LANカードドライバが動作するカーネルモードと，仮想LANカードユーザモードプログラムが動作するユーザモードとの間で短いレイテンシでデータを受け渡す必要がある．ユーザモードとカーネルモードの両側で動作する2つのプログラム間のデータの受け渡しを行うためには，POSIX標準のread / write / ioctl命令を用いる方法が一般的である．

SoftEtherの仮想LANカードユーザモードプログラムが仮想LANカードドライバにデータ（Etherパケット）を渡す際の処理は，write命令を使用している（Win32の実装では，WriteFile APIを使用する）．遠隔地の仮想HUBからカプセル化されたEthernetパケットが届くたびにこの処理を行うだけでよいので，処理内容は単純である．

困難だった部分は，逆に仮想LANカードドライバが仮想LANカードユーザモードプログラムにEthernetパケットを渡すという処理を行うことであった．SoftEtherの仮想LANカードが処理しなければならないEthernetパケットは最大毎秒数十万フレームであるため，仮想LANカードドライバがOSから送信すべきEthernetパケットを受け取ったとき，そのことを仮想LANカードユーザモードプログラムに短いレイテンシで通知する必要がある．また，仮想LANカードドライバがEthernetパケットを受け取っていないときは，仮想LANカードユーザモードプログラムのスレッドはCPU時間を消費しないようにするため，待機状態となっている必要がある．したがって，ポーリング処理を使用することはできない．

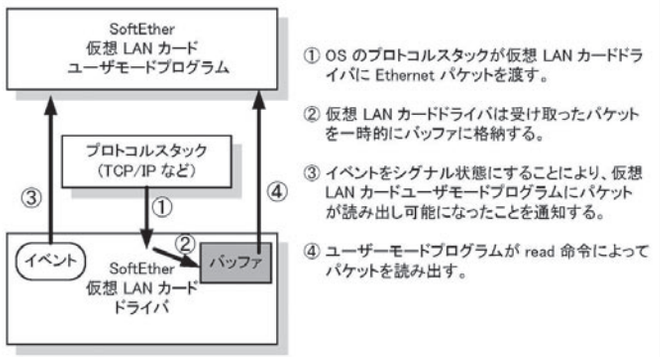


図 -3　仮想 LAN カードドライバから
パケットデータを読み出す仕組み

また，Windowsのデバイスドライバでは内部でブロッキング処理を記述する場合はIRPキューの処理によるI/O完了ポート機能を使用する必要がありプログラミングが難しい．そこで，OSが提供する同期処理のためのAPIを使用した．

Win32においては，同期処理のためのいくつかの方法があるが，イベントオブジェクトによる同期処理が最も高速でありオーバーヘッドも小さい[2]ので，イベントオブジェクトを用いた．Win32にはカーネルモード空間のオブジェクトに名前を付けてユーザモードのプロセスからそのオブジェクトのハンドルを取得できる**Win32シンボリックリンク**という仕組みがある．仮想LANカードドライバがWin32シンボリックリンクを作成しておき，仮想LANカードユーザモードプログラムはそのWin32シンボリックリンクを指定して取得したイベントオブジェクトを待機しておく．

仮想LANカードドライバがOSのプロトコルスタックから送信すべきEthernetパケットを受け取ると，まずそのパケットをカーネルモード空間内に確保してあるバッファに一時的に保存し，次にイベントオブジェクトをシグナル状態にする．仮想LANカードユーザモードプログラムはイベントがシグナル状態になったらドライバに対してread命令（Win32の実装ではReadFile APIを使用する）を呼び出すことにより，一時的なバッファから仮想LANカードユーザモードプログラムが保有するバッファへパケットを読み出すように実装した（**図-3**）．この実装の結果，パケット読み出し処理は0.05～0.1ミリ秒程度の遅延しかないことが確認できた．

○ユーザモード仮想メモリ空間へのカーネルモードからのアクセス

カーネルモードで動作するドライバとユーザモードのプロセスとの間でデータを受け渡しする際には，通常はOSによってバッファリングされる．実際には，ユーザモードプロセスの保有する仮想メモリ空間に対して，ドライバ側から直接アクセスすることは可能である．しかし，一般的なハードウェアのデバイスドライバは複数のプロセスから同時に利用される可能性があるため，ドライバから直接呼び出し元のプロセスのメモリ空間にアクセスすることは行わず，OSが提供する一時的なバッファを介してデータを受け渡しするのが一般的となっている．

Win32では，プロセスがドライバに対してread / write命令を呼び出すたびに自動的にOSが用意するバッファのことを**System Buffer**と呼び，System Bufferを介したデータのI/O処理のことを**Buffered I/O**と呼ぶ．これに対して，プロセスがドライバに対してプロセスの保有する仮想メモリ上のアドレスを渡し，ドライバがそのアドレスに対して直接アクセスすることによって行うデータのI/O処理のことを**Direct I/O**と呼ぶ．Direct I/Oを使用することにより，ユーザモードとカーネルモードの間でデータのI/Oを行う際のメモリコピーの回数が1回減る．

SoftEtherの仮想LANカードドライバに対するread / write命令は，必ず仮想LANカードユーザモードプログラムによって呼び出される．仮想LANカードユーザモードプログラムのプロセスは1つしか存在しないため，それ以外のプロセスが仮想LANカードドライバに直接アクセスすることはあり得ない．そのため，仮想LANカードドライバと仮想LANカードユーザモードプログラム間のI/OにはDirect I/Oを使用することで高速化することができた．

サーバ側プログラムの実装

○仮想HUBプログラム

SoftEtherの仮想HUBには複数のクライアント（SoftEtherの仮想LANカードが動作しているコンピュータ）が同時に接続することができ，同一のHUBに接続されているクライアント同士は仮想ネットワーク内でEthernetパケットを交換することができる．仮想HUBに接続しているクライアントからの接続を管理するオブジェクトを**セッション**と呼ぶことにする．仮想HUBはセッションごとにユーザ名や権限（セキュリティオプション），MACアドレステーブルやIPアドレステーブルを管理している．以下に，仮想HUBサービスの内部構造について述べる．

仮想HUBプログラムはWin32またはLinux上で動作するバックグラウンドプロセスである．Win32版ではシステムに統合されたシステムサービス（UNIXにおけるデーモン）として動作し，OSの起動時に稼働する．Linux版では現在では単独のプログラムであるので，

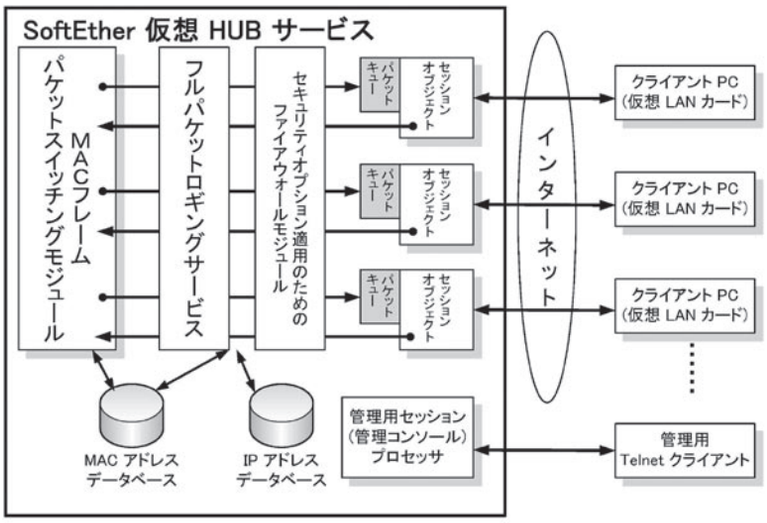


図 -4　SoftEther 仮想 HUB サービスの内部構造

nohupやscreenなどで常駐させて動作させることができるほか，起動スクリプトに記述することによってLinuxの起動時にバックグラウンドで起動させることもできる．Windows版とLinux版は同一のソースツリーを元にしてビルドされたものであり，コンフィグレーションファイル（SoftHUB.cfg）のフォーマットや管理用Telnetコンソールの操作方法などは完全に同一となっている．

○仮想HUBの内部構造

図-4に，SoftEther仮想HUBプログラムの内部構造を示す．基本的には，クライアントとの接続情報はセッション単位で管理している．クライアントから到着したEthernetパケットはファイアウォールモジュールにおいてセキュリティオプションを適用し，ポリシーに一致しないパケットをフィルタリングする．次に，パケットデータをロギングする設定になっている場合はロギングサービスを通過し，仮想HUBの中核部分であるスイッチングモジュールに到達する．スイッチングモジュールではMACアドレスデータベースを管理しており，宛先MACアドレスをEthernetパケットから取り出して既存のセッションと関連付けられたMACアドレスデータベース内のテーブルを検索して適切な送信先セッションを確定する．そして，そのセッションのパケットキューにパケットデータを追加する．各セッションは自身が持っているパケットキューにパケットデータが追加された段階でカプセル化を開始し，クライアントに対してTCPデータストリームとして伝送する．

TCPのパケットを別の物理的な回線上のTCPによる通信にカプセル化した場合，物理的な回線状態が悪化してパケットロスの率が高まった時に問題が生じ得ることが文献1）で指摘されている．パケットがキューに溜まっており，下位のTCPがエラーによる再送中でデータのやりとりが進まない間に上位のTCPでもタイムアウトが発生すると，上位層が再送を行おうとすることによってさらに大量のパケットが送られてくる．上位のTCPプロトコルスタックの再送タイマは，先にエラーとなった下位のTCPタイマより短い時間でタイムアウトするので，いったんこの状態になると，物理的な回線が処理できる速度より速くキュー内のパケットが増え続け，著しい通信速度の低下を引き起こすというものである．

そこでSoftEther仮想HUBでは，送信先セッションのキューに格納されているパケット数と送信セッションから一度に送信されたパケットの流量を常に監視し，送信元セッションが送信先セッションの回線速度を超えるパケットを送り付けようとした場合にそれを検出して動的にパケット数の制限を行っている．これがSoftEther仮想HUB内における輻輳制御アルゴリズムである．このアルゴリズムによって，回線の品質が悪いPHSや無線LANなどでも，TCP over TCP通信時にあまり通信速度を低下させることなくトンネリング通信を行うことが可能になった．さらに，UDPパケットなどを大量に送信するようなワーム等が仮想ネットワーク内のコンピュータに存在しても，そのコンピュータからのパケットは自動的にある程度遮断されるので，仮想ネットワークの麻痺を防止することができている．

○Ethernetパケットの解析

仮想HUBでは仮想ネットワーク内を流れるすべてのパケットを解析し，IPパケットである場合はIPヘッダを取り出し，さらにTCP / UDP / DHCP / ICMP / ARPなど

のパケットである場合はそれらのパケットのヘッダを参照して情報を抽出して，その内容をパケットデータと関連付けるようにしている．このようにパケットをアプリケーションレイヤまで解析する技術は近年のファイアウォールやIDSなどに多く搭載されているが，SoftEther仮想HUBにもLayer-3以上の層を解釈するファイアウォールに似たフィルタリング機能が実装されている．このフィルタリング機能を制御するためのセキュリティオプションを設定することにより，仮想HUBの管理者はポリシーに一致しない不正なパケットをフィルタリングし，設定のおかしなクライアントによるネットワークの混乱を防止することができるようになっている．

パケットの解析結果は，フィルタリングだけではなくロギングのためにも使用されている．仮想HUBではすべてのパケットのログをディスクに保存することができる．場合によってはコンピュータのディスクの速度が低速であるため，ディスクへのログの書き込み処理がボトルネックになってしまい，仮想ネットワークの通信速度が低下してしまう危険性がある．たとえば，仮想ネットワーク内を毎秒10万パケットが流れている状態ですべてのパケットをログに保存すると，毎秒10万行のログファイルを保存することになる．これはOSによるバッファリング（書き込みキャッシュ）では不十分であった．そこで，仮想HUB側で独自にバッファリングしてディスク書き込みの回数を減らし，オーバーヘッドを小さくしている．これによって，数十Mbpsの通信が行われている仮想ネットワークのパケットデータをすべてディスク上のログに保存する場合でも，通信速度の低下を軽減することができた．

○スレッドによる複数セッションの処理

現在，仮想HUBに接続したクライアントのセッションごとにスレッドを作成している．各スレッドはそのセッションに対して送信パケットのカプセル化を行う．各セッションのスレッドはパケットキューが空のときはスリープ状態になっている．パケットスイッチングモジュールはセッションのパケットキューにパケットを挿入すると同時に，そのセッションのスレッドを実行状態にしている．すべてのパケットの処理を終えると，スレッドは再びスリープする．通常，コネクションごとに個別にスレッドを生成して処理することは従来のネットワークプログラミングではあまり行われず，selectによる多重化を用いていたプログラムが多い．SoftEtherを開発する過程で実験を行ったところ，Win32環境においてはselectを用いたI/O処理を行い，1つのスレッドで複数のコネクションを処理する場合のCPU負荷と，コネクションごとに1つずつスレッドを生成した場合のCPU負荷は同等程度であることが分かった．したがって，SoftEtherの複数コネクションの同時処理にはスレッドを用いることにした．

Win32のソケット実装では内部でカーネルモードスレッドのスイッチが行われているので，単一スレッドでselectを使用しても，複数スレッドでそれぞれイベント処理をしても，スレッドコンテキストスイッチは同等程度に発生する[3]．また，Win32ではselect関数で64個を超えるソケットを同時に待機することが正式にはできないという制約もある．Windows版の仮想HUBではすべてスレッドを使用した複数コネクションの実装を行っている．Linux版についても，pthreadを使用してスレッドを複数生成している．そもそもネットワークサーバプログラミングにおいて従来forkの使用が避けられ，selectが用いられてきた最大の理由は，コネクションの生成や切断ごとに重たいプロセスを作成または廃棄する際の処理コストが大きいためであり，HTTPサーバのようなコネクションの確立が頻発するサーバプログラムであれば確かにそのコストは問題となる．SoftEther仮想HUBは一度クライアントが接続したら長時間接続を切らないという特徴があるので，新しいコネクション発生時のコストはそれほど大きくない．SoftEtherのようなクライアント／サーバ型のVPNゲートウェイサーバを実装するにあたって，スレッドの使用と多重化I/Oの使用のどちらがより効率的であるか，今後より正確に性能比較をしたい．

SoftEtherの今後

現在，SoftEtherの次期バージョンとして，SoftEther VPN 2.0の開発を行っている．SoftEther VPN 2.0の目標として，通信プロトコルの簡素化，仮想HUBの処理速度の向上，トンネリングプロトコルにおける通信速度の高速化，ユーザ認証方式の多様化，トンネリングにおけるUDPパケットの使用，およびロードバランシング技術の導入による大量アクセス時の負荷分散などが挙げられる．SoftEther VPN 2.0もSoftEtherと同様にフリーソフトウェアとして配布する予定である．

参考文献

1）Titz, O. : Why TCP Over TCP Is A Bad Idea, http://sites.inka.de/sites/bigred/devel/tcp-tcp.html（Apr. 2001）.

2）Beveridge, J. E. and Wiener, R.: Multithreading Application in Win32, ISBN4-7561-1404-0.

3）Tangentsoft: Winsock Programmer' s FAQ, http://tangentsoft.net/wskfaq/

4）Kernel-Mode Driver Architecture: Windows DDK, Microsoft Corporation, http://msdn.microsoft.com/library/

5）登　大遊：SoftEtherによるEthernetの仮想化とトンネリング通信，情報処理学会第45回プログラミング・シンポジウム論文誌，pp.147-158（2004），http://www.softether.com/jp/overview/paper/

（平成16年9月14日受付）