NAT越えを考慮したベストエフォート型TCPの実装と評価

中野宣昭[†1] 桝田秀夫[†2]

近年，インターネットの普及により，拠点間接続の需要が高まってきている．拠点間接続を行う仕組みの中でも，従来の専用線を使う方法に比べ，安価に拠点間接続を実現できる VPN (Virtual Private Network) が注目されている．また，IPv4 (Internet Protocol version 4) のグローバルアドレスの在庫がほぼ無くなり，新規割り当てが困難になっていることもあり，様々な環境で NAT (Network Address Traslation) や，NAPT (Network Address Port Translation) が利用されている．しかしながら，NAT/NAPT には，TCP (Transmission Control Protocl)/UDP (User Datagram Protocol) を利用したホスト間通信を行う際に，ホスト間に NAT 機器があるとプライベート IP アドレスを持つホストとの間でパケットの書き換えが生じるため，直接通信を行うことを前提としたプロトコルでは NAT 機器を越えることが難しいという問題がある．さらに，TCP 通信を利用した VPN トンネル上で TCP 通信を行うと，再送制御などが二重に行われる TCP over TCP と呼ばれる通信状態になり通信性能が悪化してしまう問題が知られている．

本稿では，特に VPN トンネルに適用することを考慮し，NAT 越えを想定したコネクション指向型で，かつ，データやりとりはベストエフォート型であるベストエフォート型 TCP を提案し，そのプロトコルスタックの実装を行う．また，その通信性能を測定した結果，特にバースト的なパケット損失を起こす伝送路上で定常状態に戻るまでの復帰時間が，およそ 60%程度まで改善できた．

Implementation and Evaluation of Best-effort TCP for overcomming NAT

Noriaki Nakano[†1] and Hideo Masuda[†2]

In the few years, demand for connection between remote sites has been growing. In the structure of connection between remote sites, VPN is payed attention becase it can construct private network cheaper than traditional leased line. Also, many small networks use NAT/-NAPT so that IPv4 address exhaustion is unavoidable issue. But, the NAT/NAPT problem arises when two peers behind distinct NAT try to communicate with TCP/UDP. Moreover, VPN connection over TCP has "TCP over TCP" problem. It causes serious performance degradation. In this paper, we provide the best-effort TCP for overcoming NAT in particular to consider use of VPN tunnel and develop that protocol stacks. As a result of performance measurement, in particular burst packet loss environment, the steady-state recovery time improves almost 60% than original TCP.

1. はじめに

近年，インターネットの普及により，拠点間接続の需要が高まってきている．拠点間接続を行う仕組みの中でも，従来使われていた専用線を利用する方法に比べ，IP ネットワーク上に仮想ネットワークを構築することで，安価に拠点間接続を実現できる VPN (Virtual Private Network)[1] が注目されている．VPN は，インターネットなどの公衆通信インフラの中に仮想ネットワークを構築する技術であり，PPTP (Point-to-Point Tunneling Protocol)[2] や IPsec[3),4)] のトンネルモードを用いたもの，SSL/TLS (Secure Socket Layer/-Transport Layer Security)[5] を利用してセキュリティの確保されたコネクションを提供する OpenVPN[6] や，IP ネットワーク上に仮想イーサネットを提供する SoftEther[7] などが利用されている．また，IPv4 (Internet Protocol version 4)[8] のグローバルアドレスの在庫がほぼ無くなり，新規割り当てが困難になっているため，様々な環境で NAT (Network Address Translation)[9] や，NAPT (Network Address Port Translation)[10] が利用されている．NAT/NAPT (以下，単に NAT と呼ぶ) は，IP アドレスやポートを異なる IP アドレスやポートに変換する技術である．この技術を利用して，プライベート IP アドレスをグローバル IP アドレスに変換することにより，グローバル IP アドレスのアドレス空間の消費を抑えることができる．

しかし，PPTP や IPsec を用いた VPN を NAT を

†1 京都大学 大学院情報学研究科
Graduate School of Informatics, Kyoto University
†2 京都工芸繊維大学 情報科学センター
Center for Information Science, Kyoto Institute of Technology

行う機器の配下で使用する場合，NAT 機器ではアドレスやポート番号の付け替えのためパケットの中身を変更してしまうことから，そのままでは使用できない場合が多い．最近の NAT 機器では，VPN パススルーと呼ばれる機能を持つことが多く，特定の VPN 通信について例外扱いすることで使用可能としている場合があるが，複数の VPN トンネルを同時に扱えなかったりするといった制約も多い．また，UDP (User Datagram Protocol)[11] では UDP ホールパンチング[12] などの方法が使われるが，多くの UDP の NAT 機器越えの技術はグローバル IP アドレスをもつ外部サーバが必要となる．

従って，VPN 接続を NAT 機器の VPN パススルーのような特別な機能を仮定せずに実施するためには，TCP 通信を利用した接続にすることが最も有効と考えられる．しかしながら，TCP 通信を利用した VPN トンネル上で TCP 通信を行うと，再送制御などが二重に行われる TCP over TCP と呼ばれる通信状態になり，パケット損失の多い通信路では，通信性能が非常に低下することが知られている．

文献[13] では，TCP over TCP における性能評価が行われており，通信性能を劣化させないためには通信環境や状態に適した TCP のオプションを設定する必要があると述べられている．しかし，このような TCP のオプションを通信環境や状態に合わせて設定する手法は，未知の伝送路に対して適用することは難しい．

本稿では，特に VPN トンネルに適用することを考慮し，NAT 越えを想定したコネクション指向型で，かつ，データの通信はベストエフォート型である，ベストエフォート型 TCP を提案し，そのプロトコルスタックの実装を行う．ベストエフォート型通信とは UDP に代表されるパケットの損失を前提とした通信方式である．これにより TCP over TCP の状態を回避しつつ，コネクション指向の NAT 越え技術が利用することができると考えられる．

2. 要　　求

本論文でいうベストエフォート型とは，UDP のように，やりとりされる通信自体は (送信 IP アドレス，送信ポート，受信 IP アドレス，受信ポート) の 4 項組で規定されるが，パケット自体の再送を全く行わない通信プロトコルのことを言う．つまり，ベストエフォート型 TCP(以下，BE-TCP と呼ぶ) とは，再送処理や輻輳制御を省いた TCP と言える．なお，BE-TCP を利用するアプリケーションからは，通信路の信頼性はないため，実質的には UDP 通信と同様の扱いをする必要が生じることになる．

本論文では，BE-TCP の適用先として，一般的な NAT 機器に対しては，通常の TCP 通信が行われているように見え，TCP 通信として処理されることを考慮し，BE-TCP を実装する方法について，要求を以下の 3 つに整理する．

[R1] セッション管理が可能であること
通信開始時にスリーウェイハンドシェイクを行い，通信終了まで IP アドレスとポート番号を保持し，セッションを管理する必要がある．

[R2] 扱い易いインターフェイスがあること
既存のプロトコルに類似したインターフェースを提供する必要がある．これにより，既存のアプリケーションに組み込む際に変更点を少なくできると考えられる．

[R3] ルータから，既存の TCP と同様に見えること
NAT 機器などのネットワーク通信経路上のルータに対して BE-TCP は，既存の TCP として扱われなければならない．

3. 設　　計

2 章で定義した要求を満たすプロトコルスタックを設計する．

3.1 BE-TCP のプロトコル概要

BE-TCP は，外形は TCP に見せるため、パケットフォーマットは通常の TCP をそのまま踏襲する。また、コネクション設定 (SYN, SYN/ACK) および コネクション切断 (FIN, RST) におけるパケットのやりとりや状態遷移も、通常の TCP をそのまま踏襲する。

通常の TCP と異なる点は、以下の 2 点である。

- データ送信側は、コネクション設定時の初期シーケンス番号に対して、データを送るたびに、単純に送ったバイト数分だけ進めたシーケンス番号をパケットに埋め込む。なお、ウィンドウサイズは、BE-TCP としては利用しないので、65535 に固定しておく。
- データ受信側は、パケットを受信した時に、受信したパケットに含まれるシーケンス番号を得て、それをそのまま応答確認番号としてパケットに埋め込み、ACK フラグを立てて送信する。なお、順序制御をしないので、受信したデータは即座に上位層に渡す。つまり、PSH フラグが立っている場合の意味合いで動作させる。

本プロトコルの性質上、パケットの処理順序を気にしないため、URG フラグの立ったパケットは作成する必要がない。

また、他のフィールドについては、通常の TCP と同様にデータを埋め込む。

3.2 ソケット API

BE-TCP は，コネクションの設定と破棄については TCP と同様の性質を持たせた上で，メッセージの伝送については UDP と同様に，信頼性を提供しないエンド-エンド間通信を行うため，UDP 通信の置き換えを想定して UDP のソケット API との共通化を考

える．

ただし，コネクションの設定と破棄の機能も提供するため，二層構造とし，まず，TCP としてのインターフェース (システム側 API) として設計した上で，UDP のようにして使えるインターフェース (ユーザ側 API) をラッパーとする構成とした．

3.3 システム側の BE-TCP の API

システム側での BE-TCP は TCP のソケット API の扱いと同様である．API の一覧を，表 1 に挙げる．

表 1 BETCP のシステム API 一覧

API 名	be_sys_init(void)
引数	無し
動作	デバイス情報などの初期化を行う．
API 名	be_sys_socket(void)
引数	無し
動作	RAW ソケットを作成し，ソケット情報やフィルタの初期化を行う．
API 名	be_sys_bind(int sock, const struct sockaddr *addr)
引数	sock：ソケットのファイルディスクリプタ，addr：アドレス情報
動作	ソケットとアドレス情報を関連付ける．
API 名	be_sys_listen(int sock)
引数	sock：ソケットのファイルディスクリプタ
動作	sock が参照するソケットを接続待ち状態にし，接続待ち用のフィルタを作成する．
API 名	be_sys_accept(int sock, struct sockaddr_in *sin)
引数	sockfd：ソケットのファイルディスクリプタ，sin ： 接続相手のアドレス情報
動作	ソケットへの接続要求を受付け，そのソケットを参照する新しいファイルディクリプタを返す．また接続相手とのセッション用のフィルタを作成し，スリーウェイハンドシェイクを行う．
API 名	be_sys_connect(int sock, struct sockaddr_in *sin)
引数	sock：ソケットのファイルディスクリプタ，sin ： 接続相手のアドレス情報
動作	接続要求を待っている相手に対してコネクションを行う．既にセッションが張られていた場合，この関数は失敗する．
API 名	be_sys_send(int sock, const void *buffer, size_t length, int flags)
引数	sockfd：ソケットのファイルディスクリプタ，buffer ： 送信用バッファ，length ： バッファの長さ，flags ： フラグ
動作	セッションが確立されたアドレスに対して buffer のメッセージを送信する．セッション未確立の場合 be_bind で指定されたアドレスにコネクションを試みる
API 名	be_sys_sendto(int sock, const void *buffer, size_t length, int flags, const struct sockaddr *dest_addr, socklen_t addrlen))
引数	sockfd：ソケットのファイルディスクリプタ，buffer ： 送信用バッファ，length ： バッファの長さ，flags ： フラグ，dest_addr ： 送信先アドレス，addrlen ： アドレスのサイズ
動作	セッションが確立された dest_addr に対して buffer のメッセージを送信する．セッション未確立の場合 dest_addr で指定されたアドレスにコネクションを試みる
API 名	be_sys_recv(int sock, const void *buffer, size_t length, int flags)
引数	sock：ソケットのファイルディスクリプタ，buffer ： 受信用バッファ，length ： バッファの長さ，flags ： フラグ
動作	セッションが確立されたアドレスから buffer にメッセージを受信する．セッション未確立の場合 be_bind で指定されたアドレスのコネクションを待つ
API 名	be_sys_recvfrom(int sock, const void *buffer, size_t length, int flags, struct sockaddr *dest_addr, socklen_t addrlen)
引数	sock：ソケットのファイルディスクリプタ，buffer ： 受信用バッファ，length ： バッファの長さ，flags ： フラグ，dest_addr ： 受信先アドレス，addrlen ： アドレスのサイズ
動作	セッションが確立された dest_addr から buffer にメッセージを受信する．セッション未確立の場合 dest_addr で指定されたアドレスのコネクションを待つ
API 名	be_sys_select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)
引数	nfds ： 三つのファルディスクリプタのうち最大値に 1 足したもの，readfds ： 読み込み用ファイルディクリプタ，writefds ： 書き込み用ファイルディクリプタ，exceptfds ： 例外用ファイルディクリプタ，timeout ： タイムアウトするまでの時間
動作	多重化された I/O の同期をとる．
API 名	be_sys_close(int sock)
引数	sock ： ソケットのファイルディスクリプタ
動作	セッションを終了し，フィルタの削除を行う．セッションが張られていない場合，EBADF を返し終了する．

3.4 ユーザ側の BE-TCP の API

ユーザ側での BE-TCP は，UDP のソケット API の扱いと同様であるが，一部例外がある．パケットの送受信 API (be_sendto，be_recvfrom，be_send，be_recv) は，システム側の送受信 API (be_sys_send，be_sys_recv) とほぼ対応させるのみで良いが，下層でのコネクションの設定 API (be_sys_connect，be_sys_listen，be_sys_accept) は，UDP のソケット API では存在しないため，どこかに埋め込む必要がある．本設計では，be_socket の呼び出し後，be_setsockopt を用いて，コネクションイニシエイト側かどうかを指定してもらうことで，be_sys_connect を呼び出すか，be_sys_listen と be_sys_accept を呼び出すかを決めるものとする．

API の一覧を，表 2 に挙げる．

表 2 BETCP のユーザ API 一覧

API 名	be_init(void)
引数	無し
動作	デバイス情報などの初期化を行う
API 名	be_socket(void)
引数	無し
動作	RAW ソケットを作成し，ソケット情報の初期化を行う
API 名	be_setsockopt(int sock, int optname, void optval)
引数	sock：ソケットのファイルディスクリプタ，optname ： オプション名，optval ： オプションの値
動作	コネクションイニシエイト側かどうかやキープアライブの使用を指定する．
API 名	be_bind(int sock, const struct sockaddr *addr)
引数	sock：ソケットのファイルディスクリプタ，addr：アドレス情報
動作	ソケットとアドレス情報を関連付ける．
API 名	be_send(int sock, const void *buffer, size_t length, int flags)
引数	sockfd：ソケットのファイルディスクリプタ，buffer ： 送信用バッファ，length ： バッファの長さ，flags ： フラグ
動作	セッションが確立されたアドレスに対して buffer のメッセージを送信する．セッション未確立の場合 be_bind で指定されたアドレスにコネクションを試みる
API 名	be_sendto(int sock, const void *buffer, size_t length, int flags, const struct sockaddr *dest_addr, socklen_t addrlen))
引数	sockfd：ソケットのファイルディスクリプタ，buffer ： 送信用バッファ，length ： バッファの長さ，flags ： フラグ，dest_addr ： 送信先アドレス，addrlen ： アドレスのサイズ
動作	セッションが確立された dest_addr に対して buffer のメッセージを送信する．セッション未確立の場合 dest_addr で指定されたアドレスにコネクションを試みる
API 名	be_recv(int sock, const void *buffer, size_t length, int flags)
引数	sock：ソケットのファイルディスクリプタ，buffer：受信用バッファ，length ：バッファの長さ，flags ： フラグ
動作	セッションが確立されたアドレスから buffer にメッセージを受信する．セッション未確立の場合 be_bind で指定されたアドレスのコネクションを待つ
API 名	be_recvfrom(int sock, const void *buffer, size_t length, int flags, struct sockaddr *dest_addr, socklen_t addrlen)
引数	sock：ソケットのファイルディスクリプタ，buffer：受信用バッファ，length ：バッファの長さ，flags：フラグ，dest_addr：受信先アドレス，addrlen ：アドレスのサイズ
動作	セッションが確立された dest_addr から buffer にメッセージを受信する．セッション未確立の場合 dest_addr で指定されたアドレスのコネクションを待つ
API 名	be_select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)
引数	nfds ： 三つのファルディスクリプタのうち最大値に 1 足したもの，readfds ：読み込み用ファイルディクリプタ，writefds ： 書き込み用ファイルディクリプタ，exceptfds ： 例外用ファイルディクリプタ，timeout ： タイムアウトするまでの時間
動作	多重化された I/O の同期をとる．
API 名	be_close(int sock)
引数	sock ： ソケットのファイルディスクリプタ
動作	セッションを終了する．セッションが張られていない場合 EBADF を返し終了する．

3.5 内部動作

be_sendto は，セッションが確立されていなければ be_sys_connect を行い，コネクション要求を送る．セッションが確立されていれば，データを送信する．be_recvfrom は，セッションが確立されていなければ be_sys_listen，be_sys_accept を行い，コネクションを待つ．セッションが確立されていれば，データを受信する．セッションの切断は be_close により行われる．これにより，UDP のソケット API と共通化を図り

ながら，セション管理を実現できる．これらは，要求 [R1]，[R2] を満たすと考えられる．

NAT 機器などの中継器から TCP として扱われるには，IP ヘッダのプロトコルフィールドを TCP にする，ウィンドウサイズに応じて ACK パケットを送信する，シーケンス番号と確認番号を TCP と同じ手順で更新する，などが必要になる．BE-TCP で使われる IP ヘッダのプロトコルフィールドには，上位のプロトコルが TCP であることを示す値を挿入する．IP ヘッダの IP アイデンティファフィールドや，BE-TCP ヘッダのシーケンス番号フィールド，確認番号フィールドは，従来の TCP と同様のアルゴリズムで値を挿入する．また，受信したパケットの合計サイズがウィンドウサイズを越えた場合，ACK パケットを送信する．これらのことから通信している機器以外に，パケットが従来の TCP として扱われることが期待される．これは要求 [R3] を満たすと考えられる．

4. 実　　装

本プロトコルスタックを実装するにあたって，Linux カーネル 2.6 上で運用することを前提とし，パケットソケット上での RAW ソケットを利用する．RAW ソケットを利用することで，OSI レイヤ 2 レベルで生のパケットを扱うことができるため，新しいプロトコルスタックをユーザ空間に実装できる．RAW ソケットによるパケットの受信では適切に動作しなかったため，受信のために libpcap[14] を利用する．libpcap はパケットキャプチャ用のライブラリである．今回の実装に用いた libpcap のバージョンは 1.1.1 である．

パケットの送信は，BE-TCP のヘッダを生成した上で，RAW ソケットに送出すると，ユーザから渡されたデータに RAW ソケット側で適切な値を挿入した IP ヘッダが付けられ sendto (2) で送信される．パケットの受信は，IP アドレスとポート番号によるフィルタリングを libpcap で行う．フィルタを通過したパケットの IP ヘッダと BE-TCP ヘッダを外し，データをユーザに渡す．カーネルに BE-TCP のデータが渡った場合，RST が送信されるため，ファイアウォールで RST の送信を抑制する．

BE-TCP の通信は必ずコネクション設定が行われ，セッションを確立してから開始する．セッションが確立されていないホストからの通信は，コネクション要求を除き全て破棄される．セッションを確立した IP アドレスとポート番号をテーブルに保存し，それを元にセッション管理を行う．セッションが確立された後の通信に対して，BE-TCP は再送制御やフロー制御などの機能を提供しない．そのため，シーケンス番号は送信側では送信バイト数をそのまま設定し，パケットを受信した際には，送信されてきたシーケンス番号をそのまま ACK 番号として返送する．また，送信したパケットが通信相手に届く保証はない．コネクション終了要求を受け取ると，セッションを管理しているテーブルから通信相手の IP アドレスとポート番号を削除し，セッションを終了する．

4.1 コード規模

既存のプログラム内で使われているプロトコルの置き換え易さと実行速度を考え，実装には C 言語を使用した．実装したコードの規模は約 2000 行となった．

4.2 API の使用方法

ユーザは必ず最初に be_init を呼び出さなければならない．また，コネクション設定を開始する側かどうかを，be_setsockopt で設定する必要がある．これらの設定後は，UDP のソケット API をマクロ的に置き換えることで，BE-TCP 通信を利用できる．

4.3 BE-TCP の API の実装

be_sys_socket を用いてソケットを作成し，be_sys_bind，be_sys_connect，be_sys_listen，be_sys_accept でセッションを確立する．送受信は be_sys_send または be_sys_recv で行われるが，信頼性は保証しない．

4.4 BE-TCP を UDP として扱える API の実装

be_socket を用いてソケットを作成し，be_bind でアドレスとソケットを関連付け，be_sendto または be_recvfrom で送受信を行う．ほぼマクロの置き換え程度で，UDP socket を使うソースコードと互換性を持つ．以下に簡易コードの例を示す．

```
/* 初期化*/
be_init();

/* ソケット作成*/
sock = be_socket();
if (sock < 0)
        exit(1);

/* アドレスをバインド*/
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = htonl(
    INADDR_ANY);
server_addr.sin_port = htons(
    LOCAL_SERVER_PORT);
ret = be_bind(sock, (struct sockaddr *)&
    server_addr, sizeof(server_addr));
if (ret < 0)
        exit(1);

/* データを受信*/
ret = be_recvfrom(sock, msg, MAX_MSG, flags
    , (struct sockaddr *)&client_addr,
    sizeof(client_addr));
if (ret < 0)
        exit(1);

/* データを受信*/
be_sendto(sock, msg, ret, flags, (struct
    sockaddr *)&client_addr, sizeof(
    client_addr));
```

5. 評　　価

本章では本プロトコルスタックが NAT 機器などの中継器において既存の TCP の NAT 越え技術が利用

可能なのかの評価を行う．また，通信性能面の評価として，それぞれのプロトコルで構成した VPN トンネルに TCP 通信を行った際に，通信回線の不良が発生した状況からの復帰時間の評価を行う．

5.1 BE-TCP による NAT 機器越えの性能評価

5.1.1 評価方法

測定のために BE-TCP を利用したエコーサーバとクライアントを実装した．評価に使用した機器を表 3 に示す．エコーサーバとクライアントの間に NAT 機器またはファイアウォールを設置し，両端のホストと実行可能であれば NAT 機器で tcpdump[15] を実行して IP アドレスとポート番号を監視し，通信できているかを確認した．(図 1)．

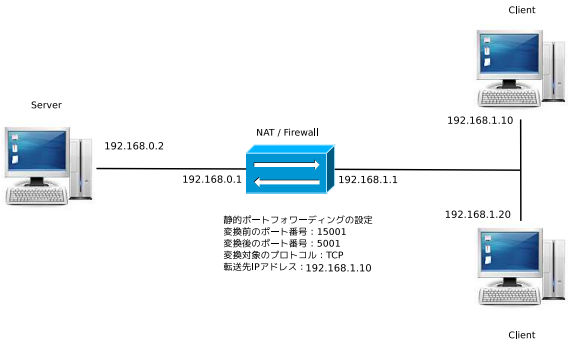

図 1 NAT 機器越え性能評価環境

表 3 評価に使用した NAT 機器とファイアウォール

機種名	備考
NTT Web Caster V110	
NEC Aterm WBR75h	
NetBSD 5.1	iptables + ipnat
FreeBSD 8.1	ipfw
OpenBSD 4.8	pf

また，WAN 側ポートから LAN 側ポートへの接続は静的ポートフォワーディングを使用した．図 1 では 192.168.0.2 から 192.168.0.1:15001 宛の TCP 通信を 192.168.1.10:5001 にポートフォワーディングするように設定している．従って，UDP 通信と NAT 機器が判断すればパケットは転送されず，通信できない．

5.1.2 評価結果

表 3 に示す全ての機器でスリーウェイハンドシェイクに成功し，エコーサーバとクライアント間で NAT 機器とファイアウォールを越えて通信ができた．また，静的ポートフォワーディングを使用した場合も，問題なく通信できた．

5.2 BE-TCP の通信性能評価

通信性能を評価するために OpenVPN に BE-TCP を組み込んだ．OpenVPN は P2P モードで動作させ，DummyNet[16] を NAT 機器上で実行した．DummyNet は伝送路の帯域や遅延時間，パケット損失率の制御ができるツールである．VPN クライアント，VPN サーバ，測定器には Arch Linux[17]，Linux kernel 2.6.32 を利用した．DummyNet の実行及び NAT 機器には FreeBSD 8.1[18] を利用した．また，通信環境は 100BASE-TX[19] で統一した．

5.2.1 測定方法

VPN サーバと VPN クライアントの間に DummyNet を実行する NAT 機器を設置し，TCP，UDP，BE-TCP のそれぞれで VPN トンネルを構成した．測定のために測定器 HostA と測定器 HostB をそれぞれ VPN サーバ，VPN クライアントに接続した．(図 2)．

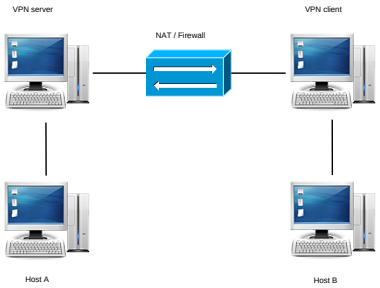

図 2 通信性能評価環境

DummyNet を利用して通信路のパケット損失率が 0%の定常状態とパケット損失率が 100%の異常状態を 30 秒，60 秒，120 秒，180 秒間隔で繰り返し，HostA と HostB の間で mbuffur[20] を実行した．mbuffer はデータストリームのバッファリングを行い I/O の速度を測定するツールで，TCP ネットワークにも対応している．これを改造して 1 秒ごとに I/O の速度を出力するようにした．HostA と HostB の間で行われる mbuffer の通信を監視し，異常状態から定常状態に戻る際に掛かる復帰時間を測定した．

5.2.2 測定結果

測定結果を表 4，5，6，7 に示す．この結果から，BE-TCP が TCP の約 60%程度早く定常状態に復帰できていることが確認できた．TCP が異常状態から定常状態に復帰できなくなった場合「復帰せず」，BE-TCP が異常状態から定常状態に復帰できなくなり，再コネクションを試みた場合「再コネクト」と書いている．

表 4 復帰に掛かった時間 (切替間隔 30 秒)

読み込むバッファのレート	TCP	UDP	BETCP
100kB/s	2 秒	2 秒	2 秒
250kB/s	4 秒	2 秒	2 秒
500kB/s	7 秒	4 秒	4 秒
1000kB/s	復帰せず	6 秒	再コネクト

表 5 復帰に掛かった時間 (切替間隔 60 秒)

読み込むバッファのレート	TCP	UDP	BETCP
100kB/s	4 秒	2 秒	2 秒
250kB/s	6 秒	3 秒	4 秒
500kB/s	12 秒	6 秒	7 秒
1000kB/s	復帰せず	12 秒	15 秒

表 6 復帰に掛かった時間 (切替間隔 120 秒)

読み込むバッファのレート	TCP	UDP	BETCP
100kB/s	5 秒	3 秒	3 秒
250kB/s	12 秒	6 秒	7 秒
500kB/s	23 秒	12 秒	14 秒
1000kB/s	31 秒	24 秒	25 秒

表 7 復帰に掛かった時間 (切替間隔 180 秒)

読み込むバッファのレート	TCP	UDP	BETCP
100kB/s	6 秒	3 秒	4 秒
250kB/s	12 秒	6 秒	7 秒
500kB/s	24 秒	12 秒	14 秒
1000kB/s	56 秒	25 秒	28 秒

6. 考　　察

6.1 NAT 越え性能

表 3 に示す全ての機器で NAT 機器とファイアウォールを越えることができたことから，BE-TCP が既存の TCP と同程度に NAT 機器を越えることや通信が可能であることが言える．しかし，評価対象となった NAT 機器の台数が少ないため，さらに検証する必要があると考えられる．また，NAT が TCP セッションを切断したかどうかを判定するキープアライブの仕組みにも改善点が考えられる．

6.2 通 信 性 能

バースト的なパケット損失を起こす伝送路上で定常状態に戻るまでの復帰時間は，TCP に比べ，BE-TCP が 60%程度早く復帰できていることが表 4，5，6，7 から言える．また，バッファのサイズに比例して TCP，UDP，BE-TCP の全てで，復帰時間が増加している．このことから，高速大容量通信ほど TCP よりも BE-TCP の方が短時間で復帰できると考えられる．

6.3 問題点と解決策

本プロトコルスタックを実装し，評価する過程でいくつかの問題を発見できた．

まず，TCP/UDP とのスループットの差が問題となる (図 3)．BE-TCP はユーザランドで処理されるため，スループットそのものがネットワークカードの性能よりも CPU の性能に依存し，他のプロセスが CPU を占有すると影響を受ける．この通信テスト中に，OpenVPN の CPU 使用率を確認すると常時 100%近く占有していた．一方，TCP/UDP はカーネルで処理されるため，CPU の性能への依存が少なく，ネットワークカードの性能を引き出すことができる．そこで，TCP/UDP と同様に BE-TCP をカーネルモジュール化することが考えられる．これにより，処理がユーザランドからカーネルに移ることで，スループットの向上が期待できる．

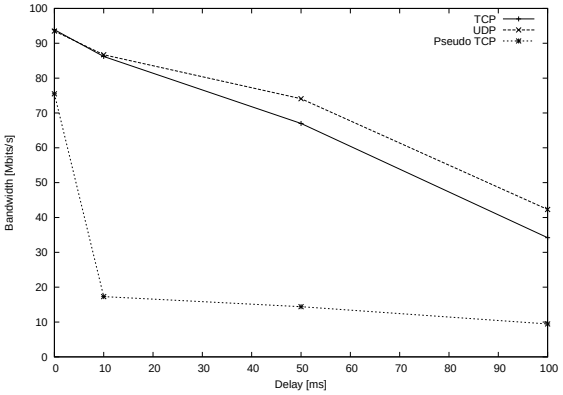

図 3 スループットとディレイの相関図

7. ま と め

本稿では，NAT 越えを考慮した BE-TCP の実装を行った．また，その NAT 越えの性能と通信性能を測定した結果，特にバースト的なパケット損失をおこす伝送路上で定常状態に戻るまでの復帰時間が，およそ 60%まで改善できた．このことから，BE-TCP の有用性を実証できたと考えられる．また，ユーザランドの実装であったため，カーネル空間の実装に改めれば，コンテキストスイッチが大幅に改善され性能向上が期待できる．

今後の課題として，OpenVPN 以外への組み込みのために一部不完全な API の実装と，高性能化のためのカーネルモジュール化，および実環境における継続的な運用での安定性の評価などが挙げられる．

参 考 文 献

1) : A Framework for IP Based Virtual Private Networks. RFC 2764.
2) : Point-to-Point Tunneling Protocol (PPTP. RFC 2637.
3) : IP Authentication Header (AH). RFC 4302.
4) : IP Encapsulating Security Payload (ESP). RFC 4303.
5) : HTTP Over TLS. RFC 2818.
6) OpenVPNTechnologies, i.: OpenVPN - Open Source VPN, ,
available from ⟨http://openvpn.net/⟩
(accessed 2010-08-24).
7) Corporation, S.: ソフトイーサ (株) - 業界標準の広域イーサネットとインターネット VPN を目

指して，，入手先⟨http://www.softether.co.jp/⟩（参照 2010-08-24）.
8) : Internet Protocol. RFC 791.
9) : The IP Network Address Translator (NAT). RFC 1631.
10) : IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663.
11) : User Datagram Protocol. RFC 768.
12) Ford, B., Srisuresh, P. and Kegel, D.: Peer-to-Peer Communication Across Network Address Translators, *USENIX 2005*, Vol.16, No.1, pp. 1–6 (2005).
13) 本田　治，大崎博之，今瀬　誠，石塚美香，村山純一：TCP over TCP の性能評価-TCP トンネルがエンド-エンドのスループットに与える影響-，電子情報通信学会信学技報，Vol.IN2004-121 (2004-11), No.1, pp.79–84 (2004).
14) Tcpdump/Libpcap: TCPDUMP/LIBPCAP public repository, , available from ⟨http://www.tcpdump.org/⟩ (accessed 2010-08-24).
15) Tcpdump/Libpcap: TCPDUMP/LIBPCAP public repository, , available from ⟨http://www.tcpdump.org/⟩ (accessed 2010-08-24).
16) Rizzo, L.: Dummynet home page, , available from ⟨http://info.iet.unipi.it/~luigi/dummynet/⟩ (accessed 2010-08-24).
17) Vinet, J. and Griffin, A.: Arch Linux, , available from ⟨http://www.archlinux.org/⟩ (accessed 2010-08-24).
18) Project, F.: The FreeBSD Project, , available from ⟨http://www.freebsd.org/⟩ (accessed 2010-02-12).
19) Diab, W. W.: IEEE 802.3 ETHERNET, , available from ⟨http://www.ieee802.org/3/⟩ (accessed 2010-08-24).
20) Maier-Komor, T.: mbuffer/ - the home of the measuring buffer -, , available from ⟨http://www.maier-komor.de/mbuffer.html⟩ (accessed 2010-08-24).