第 XIII 部

SCTP および DCCP に関する研究開発

第 13 部
SCTP および DCCP に関する研究開発

第 1 章　はじめに

SCTP ワーキンググループは、SCTP（Stream Control Transmission Protocol）や DCCP（Datagram Congestion Control Protocol）などの次世代トランスポートプロトコルに関する研究を中心に活動を行っている。本年度の主な活動内容は以下の通りである。

- Windows 用 SCTP ドライバの研究と開発
- クロスレイヤアーキテクチャを用いた高速ハンドオーバ・フェイルオーバに関する研究
- 複数パスを同時に利用するトランスポート技術に関する研究
- トランスポート層モビリティ及びマルチホーミングに関する研究
- SCTP の ADD-IP 拡張の改良
- SCTP を用いたビデオストリーム伝送の研究

以下、各章において詳細な内容を報告する。

第 2 章　Windows 用 SCTP ドライバの研究と開発

2.1 研究開発の概要

SCTP ワーキンググループでは、SCTP（Stream Control Transmission Protocol[173]）や DCCP（Datagram Congestion Control Protocol[59]）などの次世代のトランスポートプロトコルに関する研究開発を行っている。本報告書では、株式会社 CO-CONV と共同で行っている FreeBSD 上の SCTP 実装をベースとした Windows 向け SCTP ドライバ（以下、sctpDrv）の研究開発について報告する。

まず、本年度の sctpDrv の研究開発の進捗について簡単に述べる。32 bit バージョンの Windows XP 及び Vista において、SCTP の基本的な機能（アソシエーションの確立、データ通信）を安定して動作させることができるようになったのを区切りに、フィラデルフィアで開催された 71st IETF の期間中に、ベータリリースを行った[1]。その後、活発に開発が続けられている FreeBSD 上の SCTP 実装のコードを定期的に取り込むことで、sctpDrv においても、複数 ASCONF チャンクの同時送信や、UDP トンネリング、NAT 対応などの最新の機能を利用できるようにした。同時に、IETF において策定されている SCTP のソケット API 仕様 [175] への適合も進めた。また、パフォーマンス測定プログラムとしてよく知られている Iperf[78] を SCTP 及び Windows に対応させ、パフォーマンスチューニングの準備を整えた。2008 年末には、sctpDrv の最初の正式リリースを予定している。

このように、本年度の研究開発を通じて、sctpDrv は、最新の機能やソケット API への対応といった側面において、FreeBSD 実装や Linux 実装と比べて遜色のない SCTP 実装となった。さらに、sctpDrv は、FreeBSD の SCTP 実装を効率的に Windows へ移植できるよう工夫を重ねた結果、DCCP などの他のトランスポートプロトコルも容易に移植できるフレームワークに発展しつつある。そこで、以下では Windows の TCP/IP スタックとその拡張方法を概説し、sctpDrv がどのようにして、SCTP や DCCP などの新しいトランスポートプロトコルの BSD 実装の移植を容易にしているのかを述べる。

2.2 Windows の TCP/IP アーキテクチャ

Windows の TCP/IP スタックの構成を（図 2.1）に示す。TCPIP Driver（tcpip.sys）は、ネットワーク層（IPv4/IPv6）、及びトランスポート層（主として、TCP と UDP）の処理を行うカーネルドライバである。Windows における TCP のコネクションの確立や UDP によるデータの送信などの操作は、TCPIP Driver に対するリクエストの送出（I/O コントロールの送信）によって実現される。TCP や UDP を利用しようとするカーネル空間で動作する他のカーネ

1　http://www.ietf.org/mail-archive/web/tsvwg/current/msg08229.html を参照。

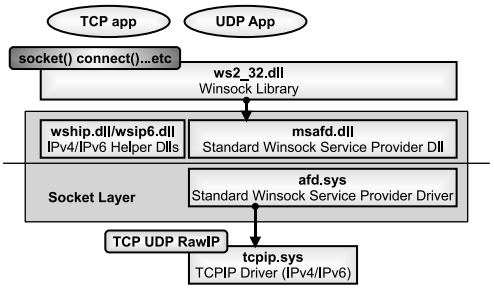

図 2.1.　TCP/IP Architecture on Windows Vista

ルドライバは、TCPIP Driver に対して直接リクエストを送出し、通信を行う。

他方、ユーザ空間で動作するアプリケーションは、カーネル空間で動作する TCPIP Driver に直接アクセスできないため、Winsock の仕組みを通じて、TCPIP Driver に対して間接的にリクエストを送出する。例えば、アプリケーションが TCP の接続を確立しようとする場合、次の順序で TCPIP Driver に対してリクエストが送信される。(1) アプリケーションは、ws2_32.dll に実装された connect() を呼び出す。(2) connect() は、TCP のサービスプロバイダとして登録された msafd.dll のルーチンを呼び出す。(3) msafd.dll のルーチンは、afd.sys に対して、TCPIP Driver へのリクエスト送信を指示する。(4) afd.sys は、他のカーネルドライバと同じように、TCPIP Driver に対して I/O コントロールを送出する。

したがって、Windows に SCTP や DCCP のような新しいトランスポートプロトコルを導入し、ユーザ空間で動作するアプリケーションから利用できるようにするためには、TCPIP Driver に相当するカーネルドライバを作成し、その上でアプリケーションがドライバに対して、Winsock を通じて、リクエストを送出することができるようにしなければならない。

新規に導入したドライバに対して、アプリケーションが Winsock を通じてリクエストを出せるようにするには、TCP や UDP が利用している標準のサービスプロバイダを経由する方法と、新しいサービスプロバイダを作成し、Winsock に組み込む方法の 2 種類がある。前者では、新規に導入するドライバに TCPIP Driver と同じインターフェース（TDI）を実装した上で、標準のサービスプロバイダにドライバを登録する。後者では、BSD ソケット API のエントリポイントを持つ ws2_32.dll に対して、新しく作成したサービスプロバイダを登録する。

前者の手法は、後者の手法と比較して、少ない作業量でトランスポートプロトコルの追加を行い得るが、新しいソケット API の追加や既存の API のセマンティクスの変更が困難という欠点がある。他方、後者の手法では、ソケット API を自由に設計できるが、既存の標準サービスプロバイダ上に実装されたソケット層の機能を一切利用できず、ソケット層の機能を一から作成しなければならないという欠点があった。

2.3 BSD 互換レイヤーの作成

sctpDrv では後者の手法を採用しつつ、FreeBSD のソケット層を Windows に移植することで、少ない作業量で Winsock を通じて SCTP をアプリケーションから利用できるようにした。（図 2.2）に sctpDrv の構成を示す。

SCTP のソケット API は、TCP や UDP で用いられてきた API をベースとして策定されているが、いくつかの API でセマンティクスが変更され、また、sctp_peeloff() のようなソケットを生成するための新しい API も導入されている。前者の手法では、API のセマンティクスを TCP や UDP と異なったものとすることや、ソケットの生成が行われる API を追加することは困難であるため、SCTP のソケット API 仕様への適合ができない。そこで、本研究開発では Winsock への組み込み手法を、前者の手法から後者の手法に変更したが、後者の手法を採用するにあたって、ソケット層の機能を一から作り直す代わりに、FreeBSD のソケット層のコードをドライバ（sctp.sys）内部に取り込み、また、FreeBSD のソケット層の機能にアクセスするサービスプロバイダ（sctpsp.dll）を作成した。この手法により、ソケット API の実装の柔軟性を少ない作業量で大幅に高め、SCTP のソケット API 仕様が要求する API セマン

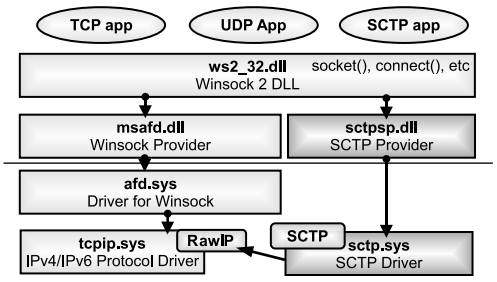

図 2.2.　sctpDrv Architecture on Windows Vista

ティクスの変更や、新しい API の実装を容易に行うことができた。

また、FreeBSD のソケット層の利用は、SCTP のソケット API 仕様への適合を容易にするだけではなく、同期処理を基本とする FreeBSD 上の SCTP 実装の移植の簡素化にも寄与している。すなわち、BSD 系 OS で動作するトランスポートプロトコル実装は、同期処理を原則としており、例えば、アプリケーションが TCP のソケットに対して connect() を呼びだすと、通常、TCP 実装の内部において、コネクションの確立あるいはタイムアウトまでスリープする。他方、TCPIP Driver のような Windows のトランスポートプロトコル実装は、すべての処理を非同期に行うようになっており、アプリケーションが同期処理を要求する場合でも、サービスプロバイダと TCPIP Driver の間では非同期処理が利用される。このため、前者の手法を用いて FreeBSD の SCTP 実装の移植を試みる場合には、アソシエーションの確立などの機能を非同期で呼び出したうえで、アソシエーション確立時に呼び出されるフックを設けるといった変更が必要となっていた。

これに対して、sctpDrv では、SCTP 実装の機能は、同期処理に対応した FreeBSD のソケットレイヤーを経由して呼び出されるため、上述のような複雑な変更は不要となった。また、アプリケーションによるソケット API 呼び出しの処理の流れが簡素化されたため、不具合の対処が容易になるという副次効果も得られた。なお、Winsock では、WSASend() などの拡張 API において、Overlapped I/O と呼ばれる非同期処理を利用できる。sctpDrv では、FreeBSD 上の非同期処理である aio（asynchronous I/O）機構をサポートするためのコードを利用することで、Overlapped I/O も実行できるようになっている。

さらに、sctpDrv では、ソケット層だけでなく、mbuf(9)、callout(9)、sysctl(9) も実装しており、これらを用いて SCTP 実装の移植を行っている。

2.4 今後の予定

本研究開発では、FreeBSD のソケット層のコードを移植し、またバッファ、タイマーなどの互換実装を設けることで、最小限の作業量で Windows 上で動作する SCTP 実装を作成することに成功した。今後は、SCTP だけでなく、DCCP などの他のトランスポートプロトコルも移植できるよう、BSD 互換レイヤーの完成度を高めていく予定である。また、現在の sctpDrv は、FreeBSD の実装と比較するとパフォーマンスに劣っているため、ボトルネックの洗い出しなどを通じて、実装の完成度をより高めていきたい。

第 3 章　クロスレイヤアーキテクチャを用いた高速ハンドオーバ・フェイルオーバに関する研究

3.1 クロスレイヤアーキテクチャにもとづく SCTP の高速ハンドオーバに関する研究

SCTPmx: an SCTP Fast Handover Mechanism Using a Single Interface Based on a Cross-Layer Architecture

Recently, SCTP attracts attention to support mobility in the Internet because it does not require additional equipment such as the Home Agent of Mobile IP. This research focuses on an SCTP fast handover mechanism using a single interface because it is assumed that small mobile devices have a single interface per communication media such as IEEE802.11b due to hardware limitation. The proposed mechanism called SCTPmx (SCTP Mobility support based on Cross-layer architecture) employs a cross layer control information exchange system called LIES to predict handover. LIES was originally designed to achieve network layer fast handover and then it was extended by adding the network layer primitives for efficient interaction among the link layer, the network layer, and the transport layer. Prior to handover, SCTPmx can generate a new address that will be used after handover and can execute duplicate address detection of IPv6. SCTPmx can suppress the delay caused by channel scanning at the link layer by employing selective background scanning mechanism which allows to continue data communication during channel scanning. In addition, SCTPmx can notify the correspondent node of the new address before handover. SCTPmx was implemented on FreeBSD. Our measurement

results showed that SCTPmx achieved less than 1/15 handover latency (100 msec) and more than 4 times throughput in comparison with previous proposals.

3.2 クロスレイヤアーキテクチャを用いた SCTP のフェイルオーバの高速化の研究

SCTPfx: A Fast Failover Mechanism Based on Cross-Layer Architecture in SCTP Multihoming

Small computers nowadays are equipped with multiple network interfaces such as an Ethernet interface and a WiFi interface. This means that even small computers have the capability of multi-homed communication. In daily use, however, such a computer uses only a single interface for communication although it has multiple network interfaces. Suppose that a host with two network interfaces is communicating with another host by using a single network interface and that the path currently used fails. TCP or UDP flows cannot be continued even if it is possible to switch the network interface to another one because the IP address of the end point changes. It would be convenient if TCP and UDP flows could continue even after network interface switching, for example, from an Ethernet interface to a WiFi interface.

This proposes a fast failover mechanism called SCTPfx (SCTP fast failover based on a cross-layer architecture). In SCTPfx, a failure event occurred in the link layer or the network layer is reported to SCTPfx by a cross-layer architecture called CEAL (Control information Exchange between Arbitrary Layers)[177]. CEAL originally defined the link layer primitives for fast handover in the network layer. This paper[58] extends CEAL by defining the network layer primitives for collaboration among the link layer, the network layer, and the transport layer. By employing the link layer and network layer primitives in CEAL, SCTPfx largely reduces the failover latency. This method can also be used for fast handover in mobility support by using SCTP multi-homing. SCTPfx was implemented on FreeBSD and its performance was measured on a test network.

第 4 章　複数パスを同時に利用するトランスポート技術に関する研究

現在、ネットワークインタフェース接続の多様化によりホストが複数のネットワークインタフェースを持つマルチホーム環境において、同時に複数の path へデータを送信し帯域集約・負荷分散を実現する手法が脚光を浴びている。そこで本研究では既存研究の path 選択の問題点を解決する為に、SCTP を利用した帯域集約機構に path 選択機能を付加した。本提案方式では、Association 確立時にお互いのインタフェース情報を交換し、それをもとに最適な path を選択する機能を実現した。以上の提案を FreeBSD6.2-Release 上に実装し、性能を評価した。100 Mbps と 10 Mbps のインタフェースをそれぞれ 1 つずつもつホスト同士の通信に提案方式を使用したところ、同一性能のインタフェース間で path を形成することが確認できた。また、異なる性能のインタフェース間で path を形成した場合に比べ、約 6.42 倍のスループットが得られ、提案方式の有効性を示した。本年度は本研究のまとめとして論文発表 [201] を行った。

第 5 章　トランスポート層モビリティ及びマルチホーミングに関する研究

昨年に引き続き、モビリティ機能拡張の強化に関する研究を行った。具体的には、移動後に新たなネットワークに接続した際、輻輳制御アルゴリズムに起因して通信復旧までに長い時間を要する問題を解決した。本年度は本研究のまとめとして論文発表 [68] を行った。実装は FreeBSD 7.0 以降に取り込まれている。また、本年度はこの技術を PR-SCTP（Partial Reliable SCTP）にも応用し、PR-SCTP における実時間転送の性能を向上させた。研究の途中経過として、学会のポスターセッションにて発表を行った [69]。本研究は来年度以降も継続して行う。

第 6 章　SCTP の ADD-IP 拡張の改良

我々は、アソシエーション確立後の動的なアドレスの変更を行うための SCTP の ADD-IP 拡張に対して、連続したアドレスの増減によってアソシエーションが切断されうる問題を解決するために、以前に送信した未処理リクエストが存在する場合においても、新しいリクエストの送信を可能にする改良を提案してきており、我々の提案を含んだ形で、ADD-IP 拡張の提案が発行されている [174]。そこで本年度は、我々の改良提案の詳細を論文 [227] として公表すると共に、FreeBSD や MacOS X、Windows で動作する SCTP 実装の ADD-IP 拡張の改良を行った。改良された ADD-IP 拡張の実装は、MacOS X、Windows 用の SCTP 実装だけでなく、FreeBSD 本家のコードにも取り込まれており、近日中のリリースが予定されている FreeBSD 7.1 RELEASE では、カーネルの修正を行うことなく利用できる。

第 7 章　SCTP を用いたビデオストリーム伝送の研究

インターネットの上のビデオストリーム伝送には、これまで一般に UDP が用いられてきたが、UDP の代わりに、マルチホーム機能と部分信頼性機能（PR-SCTP）を有する SCTP を用いることで、複数回線の効率的な利用や、音声などの特定データの優先伝送を実現することができる。そこで、本研究では、既存の UDP を用いたビデオストリーム伝送を SCTP による伝送に変換するための手法を検討した [226]。今後は、ビデオストリーム伝送に適したマルチホームや部分信頼性の利用法を確立するよう研究を進めていく予定である。

第 8 章　まとめ

本年度は、昨年度から行っている Windows 用の SCTP ドライバの開発やトランスポート層モビリティ及びマルチホーミングに関する研究を発展させた。また、SCTP を用いて新たなビデオストリーム伝送技術の研究にも着手した。来年度は、これまでの研究を継続するだけでなく、SCTP が多くのプラットフォームで利用可能になり始めたことから、その利用方法を模索する。また、SCTP に限らずトランスポート技術全般に関する研究を行い、活動の幅を広げる予定である。