L-029

FreeBSD カーネルから Linux カーネルへの SCTP 実装の移植

Porting of the SCTP Implementation from FreeBSD to Linux kernel

立花 篤男[†] Atsuo Tachibana　横田 侑樹[‡] Yuki Yokota　工藤 千加子[‡] Chikako Kudo　塩澤 暁広[‡] Akihiro Shiozawa

渋谷 恵美[†] Megumi Shibuya　柳谷 尚寿[§] Naohisa Yanagiya　中谷 俊和[§] Toshikazu Nakatani　長谷川 輝之[†] Teruyuki Hasegawa

1. まえがき

近年，Wi-Fi，WiMAX，LTE などの多様なモバイルネットワークが展開されるとともに，複数のネットワークインターフェースを具備したスマートフォン等の端末が普及してきている．このため，通信冗長性の確保やトラヒックの負荷分散を実現できるマルチホーム通信の環境が整いつつある．また，今日では，従来の携帯端末に加えて，スマートメータやウェアラブル端末等の様々な IoT デバイスが登場し，今後，多様なネットワークに接続することで新たなサービスが展開されることが予想されており，マルチホーム通信に対する需要も増加することが期待される．しかしながら，今日のインターネットにおいて，通信の大部分を占める TCP は，マルチホーム通信を標準サポートしておらず，マルチホーム通信の利点を生かした通信は十分に行われていない．

これに対し，マルチホーム通信が可能な新しいトランスポートレイヤプロトコルとして SCTP(Streaming Control Transmission Protocol)[1]が注目されている．SCTP では，TCP と異なり，エンドホストが単一のアソシエーション（コネクション）に対し，複数の IP アドレスをバインドでき，マルチホーム通信が標準サポートされている．特に，近年研究が進んでいる Concurrent Multipath Transfer (CMT)[2]は，マルチホーム通信環境において，同時に複数のパスを介したデータ転送を行うことにより，通信冗長性の確保やトラヒックの負荷分散を実現するとともに，複数パスの通信帯域容量を足し合わせることでデータ転送の高速化を実現できる特徴を持つ．また，SCTP は，TCP が使用する 16 ビットのチェックサムよりも強固な CRC32c チェックサムを使用するため，TCP よりも高信頼な通信が可能になるという特長もある．

このように，SCTP は TCP にはない重要な特徴を持っているが，今日の大半の OS プラットフォームにおいて，SCTP の実装は十分に成熟していない．特に，今日の様々な通信システムや組み込み機器に広く導入されている Linux カーネルでは，2003 年のカーネル 2.4.23 より SCTP スタック[3]が実装されているものの，最新の RFC を十分に追随できておらず，いくつかの重要な機能が利用できない状況となっている．このような Linux カーネルにおける SCTP 実装の不備は，SCTP が広く利用されない原因の一つであると考えられる．そこで，筆者らは，現在の SCTP プラットフォームの実装状況を調査した上で，実装が最も充実している FreeBSD の SCTP 実装を，Linux カーネル(3.2.0)に移植し，最新の SCTP 機能の実現を試みた．本稿では，移植の概要について述べるとともに，実装後の Linux カーネルにおける動作検証結果について述べる．

以下，本稿の構成は以下の通りである．第 2 章では，今日のモバイルインターネット環境において，SCTP を有効利用するための 2 つの必要機能について説明する．第 3 章では，代表的な SCTP 実装の現状と課題について整理する．次に，第 4 章で，Linux カーネルへの SCTP 実装の移植について説明し，第 5 章にて，移植後の SCTP の動作検証結果について報告する．最後に第 6 章にて，結論について述べる．

† （株）KDDI 研究所, KDDI R&D Laboratories Inc.

‡ （株）NTT データ数理システム, NTT DATA Mathematical Systems Inc.

§ KDDI (株), KDDI Inc.

2. 移植する SCTP 実装

今日のモバイルインターネット環境において，SCTP の特徴を最大限に活用するためには，以下の 2 つの機能を実現する必要がある．

(1) CMT

現在の標準 SCTP のマルチホーム通信では，利用可能な複数のパスの中から 1 本のプライマリパスが選択され，当該パスを介してデータ転送が行われる一方，残りの利用可能なパスは，バックアップ用として使用される．すなわち，標準 SCTP は，同時に複数パスを介してデータ転送を行う CMT をサポートしていない．しかし，マルチホーム接続の特徴を最大限に活用し，高速なデータ転送や負荷分散を実現するために，CMT は重要な機能である．

CMT は，文献[2]によりその基本設計が提案され，その後，様々な改良手法が報告されてきた(e.g., [4-7])．例えば，文献[4]では，複数パスを利用することにより発生するパケットの到着順序逆転が不要なデータ再送を誘発し，アソシエーション全体のスループットを低下させる課題を明らかにし，効率的なデータ再送のためのパス選択手法を提案した．また，文献[5]では，CMT において使用する 2 本のパスのうちの一方でパケットロスが頻発すると，他方のパスの受信バッファ処理がブロックされ，アソシエーション全体のスループットが低下する課題を指摘し，その解決方法を提案した．本稿では，CMT の実装として，文献[4-7]にて提案された改良手法を含むものとし，実装を行った．

(2) SCTP over UDP

現在の標準 SCTP では，同一アソシエーションに属する通信パスでは同一のポート番号を利用しなければならないという制約がある．このため，通信経路上に ブロードバンドルータ等の NAT 装置が存在する場合，通信経路上で送

信時に指定したポート番号が変更されてしまい，通信できなくなる課題がある．この課題に対し，いくつかの解決手法が検討されているものの [8-9]，現在普及している大半の NAT 装置に，その解決手法は実装されていない．そこで，ネイティブな SCTP 通信の代わりに， SCTP パケットを UDP ヘッダでカプセル化して通信を行う SCTP over UDP を利用し，今日のモバイルインターネット環境における通信を実現する．

3. SCTP 実装の現状

代表的な SCTP 実装の現状について調査した結果を以下に示す．また，表 1 に，SCTP 機能の比較結果を示す．

(1) lksctp

lksctp[3]は，Linux カーネルにおける SCTP の標準実装である． lksctp は，Linux カーネルに組み込まれているという点で，十分な処理性能を期待できる．しかし，現在の lksctp は， 最新の RFC を追随できておらず，CMT と SCTP over UDP の両方の機能が実装されていない．

(2) FreeBSD カーネル

FreeBSD カーネルは，現在，SCTP 実装が最も充実している OS プラットフォームであり，SCTP over UDP と CMT の両方の機能を実装している．カーネル実装であることから処理性能面も問題ないと思われるが，当然ながら，今日のインターネットにおいて広く利用されている Linux カーネルでは動作しない．また，CMT と SCTP over UDP を同時利用する際，正常に通信できない問題が発生する課題もある(詳細は 4-(3)節で説明する).

(3) libusrsctp

libusrsctp[10]は，Penoff らが，FreeBSD カーネルの SCTP 実装をユーザ空間で動作するように移植したソフトウェアであり，Linux 等の Unix 環境で動作する．libusrsctp は，CMT と SCTP over UDP の両方を実装しており，機能面での不足はない．一方で，ユーザ空間で動作するという設計のため，カーネル空間で動作する実装と比較すると，処理性能面で劣る可能性がある．特に，IoT デバイス等の処理スペックの低い小型デバイスに搭載する場合，処理性能の問題が顕在化することが予想される．

表 1　SCTP 実装の比較

	CMT	SCTP over UDP	CMT と SCTP over UDP の同時利用	期待される処理性能
lksctp	×	×	×	○
FreeBSD	○	○	×	○
libusrsctp	○	○	×	△

4. SCTP 実装の移植

SCTP 実装の現状を踏まえ，筆者らは，まず，FreeBSD カーネルに実装されている CMT と SCTP over UDP の実装をそれぞれ Linux カーネルに移植した．次に，CMT と SCTP over UDP を同時利用する際に発生する問題に対し，追加実装による解決を試みた．

(1) 想定ネットワーク構成

図 1 に，本稿で想定するネットワーク構成を示す．図 1 において，ホスト A ，B はともに 2 つのインターフェースを持っており，それぞれ，IP アドレス{C1, C2}と{S1, S2}が付与されている．ホスト A, B 間の 2 本のパス上には，NAT 装置が存在しており，ホスト A から送出されたパケットは，送信元 IP アドレス{C1, C2}が通信経路上でそれぞれ{N1, N2}に変換され，ホスト B に到着する．このとき，パケットの送信元ポート番号は各 NAT 装置によって，任意のポート番号に変更される可能性がある．このため，ホスト A, B 間では，2-(2)節に示した通り，ネイティブな SCTP による CMT は行えない．

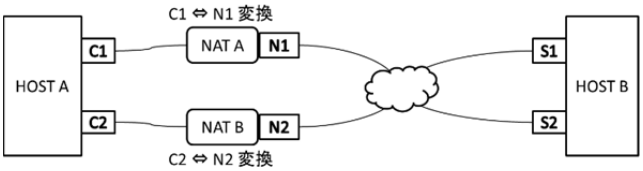

図 1　想定するネットワーク構成

(2) FreeBSD カーネルから Linux カーネルへの SCTP 実装の移植

本研究では，まず，FreeBSD カーネルの SCTP 実装を Linux カーネル空間で動作するよう改修した．FreeBSD カーネルのソースコードから，SCTP 実装部分を抽出し，Linux カーネルのカーネルモジュールとしてコンパイルし，Linux カーネルで読み込むことで動作可能にした．FreeBSD カーネルと Linux カーネルは，ともに，C 言語で記述されており，同様の機能を実装しているが，それぞれの内部構造は大きく異なっている．内部構造の差異は，機能の名前が異なるといった些細なものから，根本的に構造が異なるものまで多様なものが存在する．本稿では，ソースコード実装の詳細な説明は省略するが，以下に，差異が大きな実装の代表的な例として，パケット受信処理とその際に必要な排他制御の構造について説明する．

FreeBSD カーネルでは，パケットの受信のために，Interrupt Thread という，パケット受信専用のスレッドを動作させる．Interrupt Thread は，カーネルによって起動される点を除けば，通常のユーザレベルスレッドと同等のものであり，排他制御に一般的な同期機構を用いることが可能である．また，パケット受信専用のスレッドを動作させるという構造のため，排他制御の必要な箇所について考慮する箇所が少ない．一方で，あらかじめ起動した Interrupt Thread でしかパケットの受信を行わないため，マルチコア CPU を活用してスループットを向上させることは単純ではない．

一方 Linux カーネルには，Interrupt Thread のような仕組みが存在しない．代わりに，パケットを受信した瞬間において，CPU に割り込みがかけられ，パケットの受信処理が実行される．割り込みの一部として実行されるため，排他制御に通常のユーザレベルスレッドの仕組みを使用することは出来ず，spinlock などのカーネル空間専用の仕組みを使用しなければならない．また，どの CPU も，割り込みを受け取ればパケット受信処理を行う可能性があるため，それらの排他制御には，考慮すべき点が多くなる．一方で，割り込みによって自然に全ての CPU を活用することが出来るため，マルチコア CPU の利用には有利である．

今回の移植においては，この構造の違いが大きな問題になる．FreeBSD カーネルの SCTP 実装は，FreeBSD の Interrupt Thread を前提として書かれているため，そのまま Linux カーネルで動作させると，排他制御に失敗してデッ

ドロックを引き起こしてしまう．そのため，本研究では，Linux カーネルの workqueue という機能を用いて，FreeBSD カーネルの Interrupt Thread と同様の構造を実装し，その上で FreeBSD カーネルのパケット受信処理を実行する方法で，移植を実現した．

(3) CMT と SCTP over UDP の同時利用の実装

FreeBSD カーネルに実装されている SCTP over UDP と CMT のそれぞれの機能を Linux カーネルに移植したとしても，これら 2 つの機能を組み合わせて利用する場合，正常に通信を行えない課題が発生する．以下，図 1 において，ホスト A からホスト B に対してマルチホーム接続し，CMT によるデータ転送を行う場合を想定して，課題とその解決方法について説明する．

• 標準 SCTP におけるマルチホーム接続手順

まず，RFC 6951 に基づき，多対多のマルチホーム を実現する標準 SCTP の処理手順を示す．

アソシエーションの確立

最初に，ホスト A は，自身の エンドポイント を一つから(図 2 中の C1)ホスト B の 1 つの IP アドレスに対して INIT チャンクを送信し，アソシエーションの確立を試みる．このとき，ホスト A は，IP アドレスの情報(C1)を INIT チャンクに含めず，INIT チャンクを受け取ったホスト B は，IP ヘッダに含まれている IP アドレス情報(N1)を送信元のエンドポイント として認識する．ホスト B は，受信した INIT チャンクに対して INIT-ACK を N1 に対して返信し，さらに NAT 装置 A は当該チャンクをホスト A に転送する．この後，同様の手順で COOKIE-ECHO チャンクと COOKIE-ACK チャンクを交換し， SCTP の handshake が完了する．

エンドポイントの追加

次に，ホスト A は確立したアソシエーションにおいて，RFC 5061 で規定された ASCONF (ASCONF-ADD)チャンクを送信することにより，もう一つのエンドポイントである C2 の追加を試みる．このとき，NAT B で変換されたアドレスをエンドポイントのアドレスとして追加するため，ASCONF チャンクには，C2 のアドレスではなく，wildcard アドレス(IPv4 では 0.0.0.0)を使用する．ホスト B は，wildcard アドレスを含む ASCONF-ADD チャンクを受け取ると，その ASCONF チャンクを含む IP パケットヘッダの送信元アドレス(N2)を ASCONF-ADD の対象として使用する．ホスト B は，受信した ASCONF に対し，NAT 装置 B を経由して ASCONF-ACK チャンクを返信し，エンドポイントの追加が成功する．

追加したエンドポイントの疎通確認

ASCONF-ADD によりエンドポイントの追加が成功した後，CMT で通信を行うためには，新たなエンドポイントが使用可能であることを HEARTBEAT チャンクにより確認する必要がある．そのため，ASCONF-ACK を受け取ったホスト A は，C2 からホスト B へと HEARTBEAT チャンクを送り，ホスト B から HEARTBEAT-ACK を受け取って疎通確認を行う．疎通確認が成功すれば，ホスト A の C1 とホスト B の S1，ホスト A の C2 とホスト B の S2 の二つのパスが使用可能と判断でき，CMT によるデータ転送が実行される．

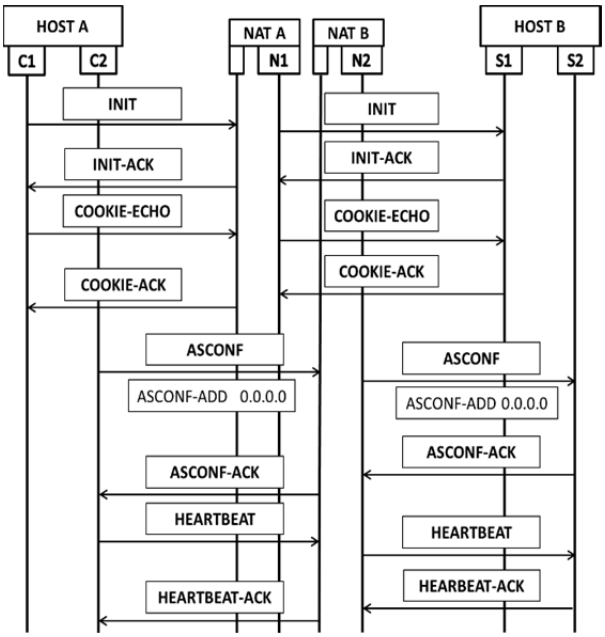

図 2　正常な SCTP 処理フロー

• Linux カーネル実装における課題

次に，Linux カーネル実装において発生する課題と，実施した対処方法について示す．

[課題1]　FreeBSD の実装では，INIT チャンクや ASCONF チャンクの送受信について，受信ホストは，受信した UDP パケット(SCTP over UDP パケット)の送信元ポート番号に対してではなく，SCTP over UDP 専用のポート番号(9899)に対して返信を行う．このため，図 3 に示すように，ホスト A から送信された UDP パケットの送信元ポート番号が NAT 装置により変換される状況であっても，ホスト B から返信される UDP パケットではポート番号 9899 が利用され，この結果，当該パケットは NAT 装置により破棄されてしまう（図 3）．

筆者らは，この問題を回避するため， INIT チャンクや ASCONF チャンクを受け取った場合，そのチャンクの送信元の UDP ポート番号を記録し，当該 UDP ポート番号を用いて通信を行うように改修した．

[課題2]　移植した FreeBSD カーネルの SCTP 実装は，ASCONF チャンクを送受信する機能や，wildcard アドレスを含んだ ASCONF チャンクを処理する機能は持っているものの，特定のエンドポイントから wildcard アドレスを使用して ASCONF を送る実装を持っていない問題があった．つまり，図 2 において C2 から送出すべき ASCONF チャンクを，誤って C1 から送出する場合があった．

本実装においては， wildcard アドレスを含んだ ASCONF チャンクを送る際にも，送出するエンドポイントを明示的に指定できるように改修した．

[課題3]　移植した FreeBSD カーネルの SCTP 実装では，ASCONF チャンクにより追加されたエンドポイントについて，直ちに HEARTBEAT チャンクを送出せず，一定時間の経過後に HEARTBEAT による疎通確認を行う仕様となっていた．このため，エンドポイントの追加を行っても，追加したエンドポイントの疎通

確認に時間を要してしまい，迅速な CMT によるデータ転送が行えない課題があった．

本実装では，ASCONF チャンクによりエンドポイントが追加された場合は，当該のエンドポイントに関するパスについて，直ちに HEARTBEAT チャンクによる疎通確認を行うように改修し，より積極的な CMT の利用を可能にした．

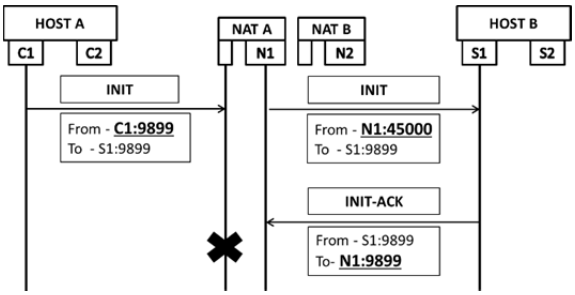

図 3 FreeBSD カーネルの SCTP 実装での UDP ポート番号

5. 動作確認

Linuxカーネルに移植したSCTP実装の正常性を確認するため，プロトタイプ実装による簡易な動作確認試験を実施した．図4に実験ネットワークの構成を示す．

ホストAは，USB経由で2台のスマートフォン（Sony Xperia SOL22）に接続し，それぞれ，LTEとWi-Fiを介してインターネットに接続している．一方，ホストBは，光ファイバ回線(200Mbps)を介してISPネットワークに接続されており，2つのネットワークインターフェースに，それぞれ異なるグローバルIPアドレス（IP_aとIP_b）が付与されている．ホストA，B間のLTE側の通信経路には，モバイルキャリア網内にキャリアグレードNATが存在しており，一方，Wi-Fi側の通信経路には，NAT装置として，市販のブロードバンドルータが配置されている．ここで，エンドホストには，小型Linuxボックス(OS: Debian GNU/Linux 7.1，Linux カーネル - 3.2.0-4-686-pae，CPU: Dual Core Marvell ARMADA XP 1.33GHz，C コンパイラ: GCC 4.7.2，Make - GNU Make 3.81)を利用した．また，CMT-SCTPの設定パラメータとして，Delayed Ack for CMT (DAC) [2], Receiver and Sender Buffer Splitting [7], Non-Renegable Selective Acknowledgments (NR-SACK) [6]を有効にした．

これらの環境において，ホストAからBに対してデータ転送を実行し，動作確認を行った．この結果，期待通り，SCTP over UDPを利用するCMT通信に成功することを確認した．また，CMTによる通信状況を把握するため，ホストBの各NICにおいて，tcpdumpによるパケットキャプチャを実施した結果を図5に示す．図5より，LTEとWi-Fiを経由する2本のパスでCMTによるデータ転送に成功していることが確認できた．

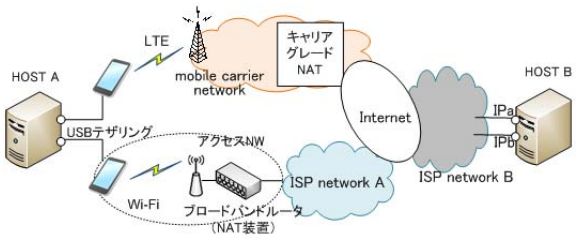

図 4 実験ネットワーク

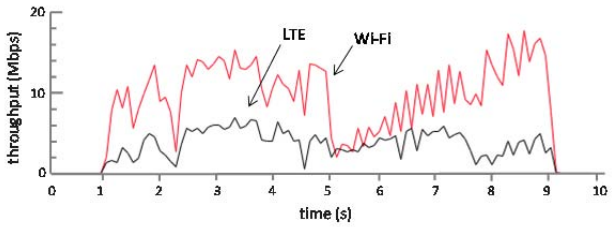

図 5 実験ネットワーク

6. まとめ

本稿では，FreeBSD カーネル における SCTP 実装を Linux カーネルに移植した．特に，今日のインターネット環境において SCTP を利用する上で重要である CMT と SCTP over UDP の 2 つの機能を Linux カーネル上で実現した．移植における課題と対処方法について説明するとともに，プロトタイプ実装による動作確認を行い，NAT 装置を介した通信経路において，CMT によるデータ転送を正常に実行できることを確認した．今後，より詳細な性能評価を実施する予定である．

謝辞

本研究は，総務省の「スマートグリッドの通信ネットワーク技術高度化実証事業」の成果であり，関連各位に深謝する．

参考文献

1) Stewart, R., “Stream Control Transmission Protocol (SCTP)” (September 2007), RFC 4960. Available from: <http://www.ietf.org/rfc/rfc4960.txt>.
2) Iyengar J., P. Amer, and R. Stewart, “Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths,” IEEE/ACM Transactions on Networking, vol. 14, no. 5 (October 2006), pp. 951-964.
3) "Linux Kernel Stream Control Transmission Protocol (lksctp) project," available at http://lksctp.sourceforge.net/.
4) J. Iyengar, P. Amer, and R. Stewart, “Retransmission Policies for Concurrent Multipath Transfer Using SCTP Multihoming,” in Proceedings of ICON 2004, Singapore, November 2004.
5) J. Iyengar, R. Janardhan, Paul D. Amer, and Randall R. Stewart, “Performance Implications of a Bounded Receive Buffer in Concurrent Multipath Transfer,” Computer Communications 30(4) (2007), pp. 818-829.
6) P. Natarajan, N. Ekiz, E. Yilmaz, P. Amer, J. Iyengar, R. Stewart, “Non-Renegable Selective Acknowledgments (NR-SACKs) for SCTP,” in Proceedings of International Conf on Network Protocols (ICNP), Orlando, October 2008.
7) T. Dreibholz, M. Becke, E. P. Rathgeb, and M. Tüxen, “On the Use of Concurrent Multipath Transfer over Asymmetric Paths,” in Proceedings of the IEEE Global Communications Conference (GLOBECOM), Miami, Florida/U.S.A., December. 2010.
8) T. Stegel, J. Sterle, U. Sedlar, J. Bešter, and A. Kos, “SCTP Multihoming Provisioning in Converged IP-based Multimedia Environment,” Computer Communications, Vol. 33, Issue 14 (September 2010), pp. 1725-1735.
9) D. A. Hayes, J. But, and G. Armitage, “Issues with Network Address Translation for SCTP,” ACM SIGCOMM Computer Communication Review, v.39n.1 (January 2009).
10) B. Penoff, A. Wagner, M. Tuexen, I. Ruengeler, “Portable and Performant Userspace SCTP Stack,” Computer Communications and Networks (ICCCN), 2012, pp.1-9. Available from :http://sctp.fhmuenster.de/sctp-user-land-stack.html
11) IETF, “RFC6951”, https://tools.ietf.org/html/rfc6951
12) IETF, “RFC5061”, http://www.ietf.org/rfc/rfc5061.txt