

iSCSI 解析システムの構築と高遅延環境におけるシーケンシャルアクセスの性能向上に関する考察

山口 実靖[†] 小口 正人^{††} 喜連川 優[†]

iSCSI Analysis System and Performance Improvement of Sequential Access in a Long-Latency Environment

Saneyasu YAMAGUCHI[†], Masato OGUCHI^{††}, and Masaru KITSUREGAWA[†]

あらまし

2003年2月に iSCSI プロトコルが IETF により承認され、データセンターなどで iSCSI を用いたストレージアウトソーシング、遠隔バックアップが提供されつつある。iSCSI プロトコルを用いて遠隔ストレージへアクセスするには、ネットワーク遅延による性能の低下を最小限に抑えることが重要である。iSCSI の性能は SCSI プロトコル、iSCSI プロトコル、ネットワーク、ネットワークデバイスなどの影響を受け、各層を統合した強力な解析システムが必要である。本稿では開発した iSCSI 解析システムの機能を紹介する。既存の iSCSI 実装ではネットワーク環境に対し最適化を行わない場合、遅延の増加に伴い性能が大幅に劣化している。当該 iSCSI 解析システムを用いた結果、性能低下要因が明らかになり、ネットワーク基盤の限界性能とほぼ同等の性能が得られた。

キーワード SAN, iSCSI プロトコル, 解析システム, 高遅延環境, シーケンシャルアクセス

1. はじめに

近年、計算機システムが処理するデータ量は飛躍的に増大し、それを保存するストレージの管理コストの増大が計算機システムの大きな問題の一つとなっている。ストレージシステムの管理にはバックアップ、リストア、キャパシティープランニングなどの専門技術を持つ技術者が必要となり、管理コストは非常に高い。これを解決するために、データセンターなどストレージサービスのアウトソーシングや SAN(Storage Area Network) などの新しい技術が登場してきた。

ストレージのアウトソーシングでは、ユーザが遠隔から接続して利用し、ストレージ管理は専門の管理者が行う。遠隔ストレージ接続の共通規格としては、SCSI プロトコルを TCP/IP プロトコルの中にカプセル化し、IP アドレスを与えられたストレージデバイスに SCSI アクセスを実現する iSCSI [1]~[4] プロトコルが 2003 年 2 月 11 日に IETF [2] により承認さ

れた。

SAN は主にストレージの管理コストを削減するために導入され、SAN を用いてストレージシステムを集約することにより、その管理コストは DAS (Direct Attached Storage) のみを用いて構築されたストレージシステムと比べ大幅に削減される。その効果は高く評価されており、すでに多くの企業が導入している。しかし、FC (Fibre Channel) を用いる現世代の SAN (“FC-SAN”) は、ハードウェアコストの高さ、接続距離の限界、相互接続性、FC 管理技術者の少なさ、などの問題点も明らかになってきており、これらの問題点を解決する次世代 SAN (“IP-SAN”) としての iSCSI が期待されている。

通常の SCSI では、計算機とストレージデバイスが SCSI ケーブルを介して SCSI プロトコルに基づく通信を行うのに対し、iSCSI では TCP/IP ネットワークを介して SCSI プロトコルに基づく通信を行う (図 1 参照)。iSCSI の利用により、アプリケーションからはローカルストレージと同様に遠隔ストレージをシームレスに使用することができる。遠隔ストレージへの

[†]

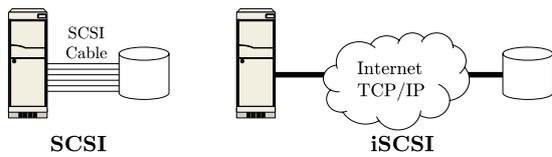


図1 SCSI プロトコルと iSCSI プロトコル
Fig. 1 SCSI protocol and iSCSI protocol

アクセスは、トランザクション処理や遠隔バックアップなどアプリケーションにより性質が異なるが、遠隔ストレージに対するシーケンシャルアクセスにおいては、iSCSI 層および下位層の動作により決まるスループットが重要な性能評価基準となり、この点に関する解析と考察は十分になされてはいない。

iSCSI ストレージアクセスの性能は、下位基盤の TCP/IP 層が提供できる限界性能を超えることはできず、限界性能に対する iSCSI 使用による劣化を最小限に抑えることが重要である。iSCSI によるストレージアクセスは SCSI プロトコル、iSCSI プロトコル、TCP/IP プロトコル、ネットワークシステムなどが関連して動作しており、強力な解析システムが必要である。そこで我々は、これらの各層の解析が可能である“iSCSI ストレージアクセスの解析システム”を構築し、iSCSI 使用環境に適用した。

本論文の構成は以下の通りである。まず第 2. 章で関連研究を紹介し、第 3. 章において構築した iSCSI 解析システムを紹介する。本解析システムは通信内容のプロトコル翻訳 (TCP/IP や SCSI, iSCSI プロトコルに対応)、パケットの時間遷移の可視化、TCP フロー制御の監視、損失パケットの検出、簡易版イニシエータを用いた iSCSI ストレージアクセスの生成などが可能である。次に第 4. 章において、ニューハンプシャー大学の InteroperabilityLab により配布されている iSCSI 実装を用い最適化が行われていない初期環境での高遅延ネットワークの iSCSI シーケンシャルリード性能を測定し、その性能が遅延時間の増加に伴い著しく低下すること、ネットワーク遅延が大きい場合に iSCSI プロトコルを用いて得られる性能はネットワーク基盤が本来提供可能である限界性能に対して大きく劣ることを示す。第 5. 章では本解析システムを用いて同測定の解析を行い、性能低下原因を明らかにする。これを解決することにより iSCSI 性能は大きく向上し、ネットワーク基盤の提供できる限界性能と同程度となった。最後に、第 6. 章において本稿のまとめ

を行う。

2. 関連研究

文献 [5] において、Ng らは独自の SCSI over IP 実装を用いて IP ストレージの性能に関する詳細な測定と考察を行っている。同文献では 8KB のブロックサイズにおけるシーケンシャルアクセスの性能を測定し、そのスループットが遅延時間にほぼ反比例することが示されている。また SCSI over IP を用いるにあたってネットワークの手前におけるキャッシュの適用や、アプリケーションによるプリフェッチが効果的であると指摘している。アプリケーションレベルの応用性能測定も行っており、性能に関してファイルシステムも考慮した考察がなされている。本論文では、アプリケーション層ではなく、iSCSI 層ならびにその下位層を詳細に解析することにより、iSCSI の適用が強く期待されている遠隔バックアップ等に必要なシーケンシャルアクセスの高速化を目指しており、研究の方向性が異なる。

文献 [6] において、Sarkar らは低遅延環境におけるブロックサイズと iSCSI スループットの関係を紹介している。低遅延環境においては CPU による処理がスループットを制限するため、さらなる高性能を得るためにはハードウェアによる TCP/IP 処理と iSCSI 処理が重要であると主張している。しかし iSCSI を用いた遠隔バックアップで直面する高遅延環境についての考察はなされていない。

3. iSCSI 解析システム

3.1 システム概要

我々が開発した iSCSI 解析システムは、iSCSI プロトコルを利用してネットワーク経由のストレージアクセスを行う環境に適用することにより、その内部状態を把握し性能決定の要因を探る機能を持つ。

iSCSI は TCP/IP ネットワークの上に階層構造を成す複雑な構成を取る。図 2 にそのプロトコルスタックの模式図を示す。

iSCSI では、SCSI プロトコルを TCP/IP プロトコルの中にカプセル化するため、プロトコルスタックは SCSI over iSCSI over TCP/IP となり、多くの場合 SCSI over iSCSI over TCP/IP over Ethernet となる。SCSI 層がイニシエータの OS およびターゲットのストレージデバイスに対して SCSI インターフェイスを用いたストレージアクセスを提供し、その下

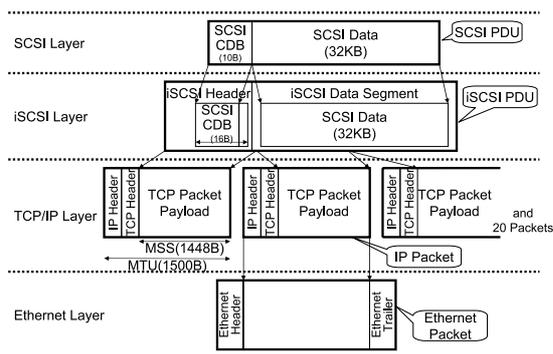


図 2 iSCSI プロトコルスタックと各プロトコルの振る舞い

Fig. 2 iSCSI Protocol Stack

層の iSCSI ドライバが SCSI インターフェイスによるストレージアクセスの送受信を iSCSI プロトコル内にカプセル化し TCP/IP ネットワークに転送する。同図はイニシエータからの SCSI Read コマンドに対しターゲットが “iSCSI Data-in” PDU (Protocol Data Unit) を作成する例である。同図内における各フィールドのサイズは 32KB の Read コマンドに対する Data-in PDU の例であり、TCP/IP ヘッダ等のサイズは我々の実験環境における例である^(注1)。

このように iSCSI は複雑な階層構成をとるため、解析を行う際に、一箇所をモニターするだけで全ての振舞いを把握することは難しい。iSCSI や TCP/IP のソースコードにモニタを組み込み、さらに iSCSI アクセス実行時のパケット転送を監視して、複合的に解析を進めなければならない。

本解析システムは以下の機能を提供する。

- (1) 通信内容のプロトコル翻訳 (SCSI, iSCSI, TCP/IP 各層に対応)
- (2) パケット転送の時間軸上における可視化
- (3) TCP フロー制御の監視
- (4) 損失パケットの検出
- (5) 簡易版イニシエータを用いた iSCSI ストレージアクセスの生成とその解析

本解析システム概念図を図 3 に示す。以下、本解析システムの適用例を含めてシステムの詳細を紹介する。また、解析システムに起因するオーバーヘッドについては付録 3. において述べる。

(注1): 同図では TCP ヘッダにおいて Timestamp オプションが付加されている

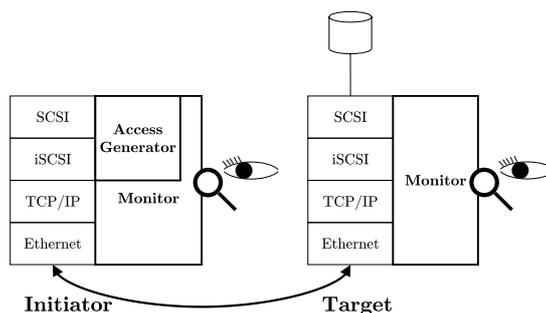


図 3 iSCSI 解析システム概要

Fig. 3 iSCSI Analysis System Overview

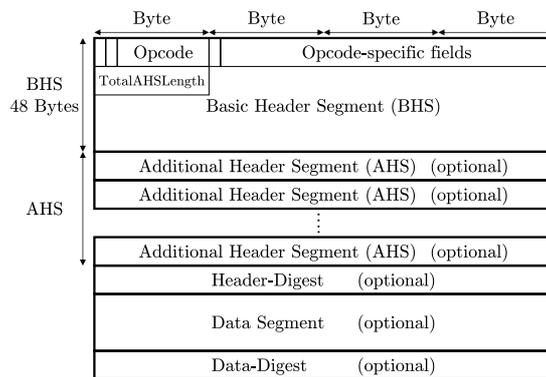


図 4 iSCSI PDU フォーマット

Fig. 4 iSCSI PDU format

3.2 iSCSI 層における解析システムを用いたプロトコル翻訳

本解析システムのプロトコル翻訳機能では iSCSI プロトコルに基づいて行われた通信内容を記録しそれを翻訳する。iSCSI プロトコルの PDU フォーマットおよび iSCSI Read コマンドのプロトコル翻訳例を図 4 と図 5 に示す。SCSI CDB (Command Descriptor Block) は可変長であり、本稿の実装では 10 バイトの例が使用されている。10 バイト時の SCSI CDB は図 5 の “SCSI Command Descriptor Block” に示されるフォーマットである。Logical Block Address および Data Length は 512 バイトのチャンクサイズ単位である。また、正確には最初の 1 バイト (図中の “28 Read” に相当) に SCSI CDB の長さを決定するフィールドが含まれている。

iSCSI PDU は図 4 のフォーマットをしており、“SCSI Command Read” の場合は図 5 の様にヘッダのみのフォーマットとなる。提案解析システムでは、これら全てのフィールドを翻訳し各値を確認できる。

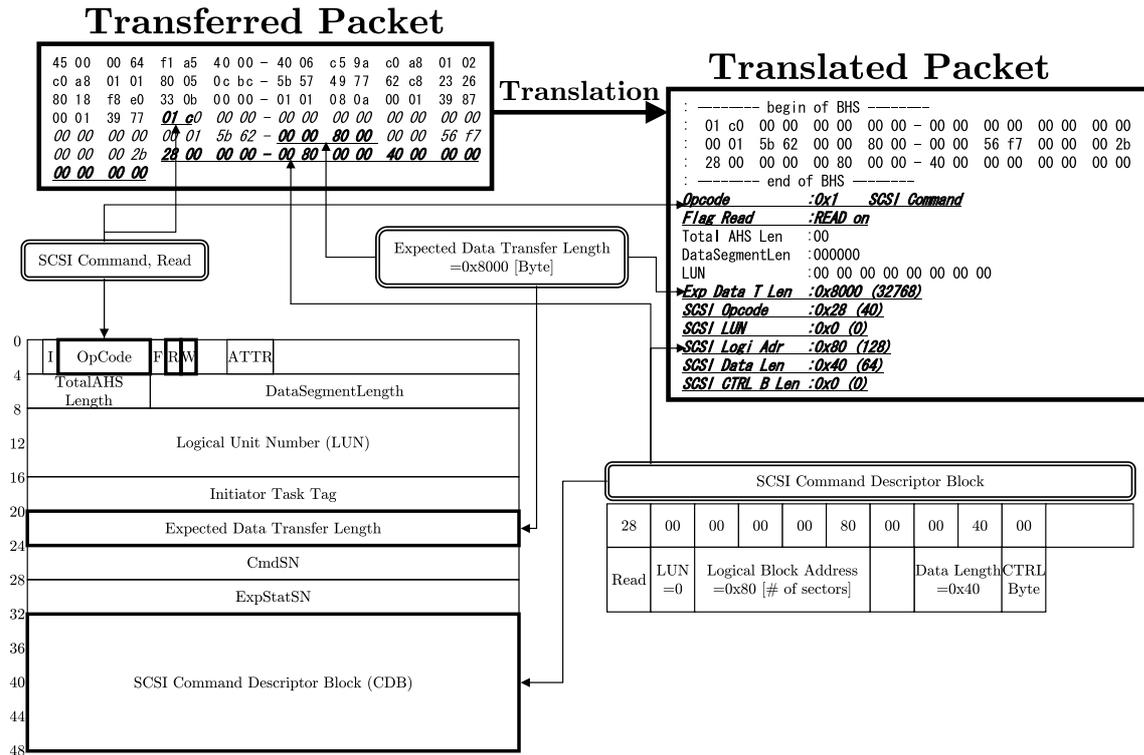


図5 iSCSI Read コマンドのプロトコル翻訳例
Fig.5 iSCSI Read command as an example of protocol translation

図5の左は実際にネットワーク上で転送された100バイトのEthernetパケットの例であるが、小さい文字で記された1バイト目から52バイト目がIPヘッダおよびTCPヘッダであり、斜体で示された53バイト目から100バイト目の計48バイトがiSCSI PDUである。解析システムでは、このiSCSI PDUの全フィールドの内容およびTCP/IPヘッダの内容を翻訳する^(注2)。この例では、対象のパケットが“SCSI Read Command”でありデータ長が32KBであることが確認できる。

また図5は“Opcode”(最初の1バイト)が“SCSI Command”の例である。図4のようにiSCSI PDUはiSCSIヘッダとデータセグメントから成る。“SCSI Command Read”や“SCSI Response”のiSCSI PDUではiSCSI PDUはiSCSIヘッダのみで構成され、“SCSI Data-in”などのiSCSI PDUはiSCSIヘッダとデータセグメントから構成される。同図の

(注2): 同図では、iSCSI PDU内のフィールドの一部と、TCP/IPヘッダが省略して描かれている

様に“SCSI Command”のiSCSI PDUの場合、33バイト目から48バイト目の16バイトがSCSI CDBに割り当てられており、この例ではSCSI CDBが10バイトであるため43バイト目から48バイト目はパディングデータが入る。また“SCSI CDB”内の数値は512バイトのチャンクサイズを単位としているが、iSCSIヘッダ内の“DataSegmentLength”や“Expected Data Transfer Length”はバイト単位で表現されている。図5の例では“Expected Data Transfer Length”=0x8000[バイト]とSCSI CDB内の“Data Length”=0x40[512バイト]は、ともに32KBのReadコマンドを意味する。

以上のように本解析システムはiSCSIのプロトコルを翻訳し、容易に理解可能とする。

3.3 パケット転送の時間軸上における可視化

本システムにおけるパケット転送の可視化機能では、実際にネットワークに流れるパケットを時間軸上で視覚的に確認できる。iSCSI層の下でデータの転送を司るTCP/IP層では、主に“IPパケットの作成”と“フ

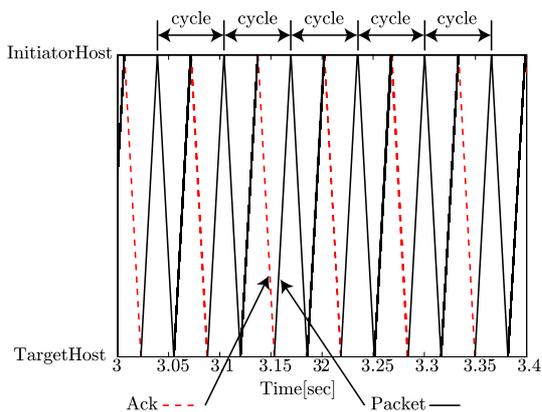


図 6 パケット転送 A
Fig. 6 Packet Transfer A

ロー制御”の2機能が実装されている。前者は iSCSI 層から転送されたデータを MSS (Maximum Segment Size) 毎に分割し、それぞれに TCP ヘッダ, IP ヘッダを付加し、それを下位層の Ethernet のデバイスドライバに転送する。後者は上位層からデータ転送を要求された場合に、受信者のバッファ空き容量とネットワークの状況を考慮してデータの転送を制限する機能であり、スループットを制限するため iSCSI などの上位層の性能に直接影響を与える。

TCP の Window には広告 Window と輻輳 Window の2種類が存在し、両者の小さい方が有効な Window サイズとなる。送信側は受信側アプリケーションが TCP のバッファからデータを受け取ったことを検出できないが、TCP ヘッダには Ack フィールドと 広告 Window フィールドが割り当てられており、受信者が毎回広告 Window 情報を送信者に通知し、送信者は送信から 1 往復時間後に受信者の受信バッファの情報を得る。送信者が Ack を受け取ることなしに非同期的に送信できるデータ量は広告 Window に制限され、広告 Window サイズが十分でない場合は、送信が停止しネットワークを十分に使うことができない。送信者がデータを送信し次の Window 更新を受け取るのが 1 往復時間後であるため、十分な広告 Window サイズは「往復時間×スループット」となる。

実装の詳細は各 OS 付随の TCP 実装に依存しており、Linux における輻輳回避フェイズの実装の詳細は後述する。また TCP 実装が輻輳と判断する基準にはパケットロスなどによるネットワーク輻輳の判断があり、これも後述する。

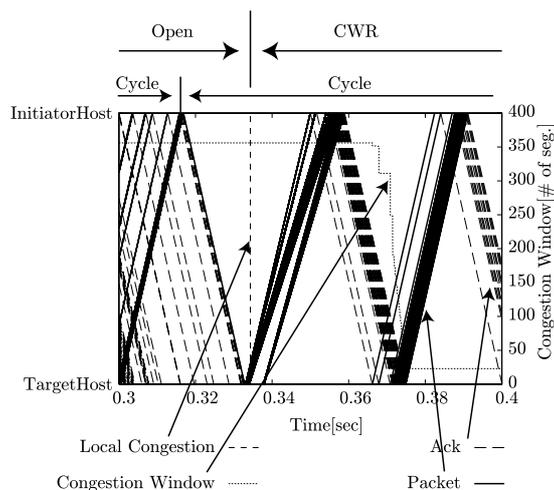


図 7 パケット転送 B
Fig. 7 Packet Transfer B

図 6, 7 に、本解析システムを用いて時間軸上で可視化されたパケット転送の例を示す。図 6 は、イニシエータとターゲットの間の片道遅延時間が 16ms でありブロックサイズ 32KB (ブロックサイズの意味については第 4 節で後述する)におけるパケット転送のみを表現しており、Read コマンドがイニシエータから発行され、それに対しターゲットがデータを送信する様子が 5 サイクル表現されている(以下、この iSCSI シーケンシャルリードのサイクルを“iSCSI Seq. Read サイクル”と呼ぶ)。このケースでは、ネットワークの待ち時間が多く含まれておりネットワークの使用率が非常に低い。

図 7 は、イニシエータとターゲットの間の片道遅延時間が 16ms、ブロックサイズ 4MB におけるパケット転送、Linux 状態機械の遷移(後述)、輻輳 Window の値等のイベントを解析システムに表示させた例であり、“iSCSI Read Command”がターゲットに到着しターゲットがデータを転送開始した部分の図である。同図はある“データ受信”が終了し、次の“Read 発行およびそれによるデータ受信”が開始する瞬間である。同図より、ターゲットは Read コマンドを受信した瞬間に大量のデータパケットの送信を試みローカル輻輳(具体的にはローカルに装備されているネットワークインターフェイスカードの輻輳)が発生している。

3.4 TCP フロー制御の監視

本解析システムにおける TCP フロー制御の監視機能はターゲット機、イニシエータ機の TCP 実装が行っ

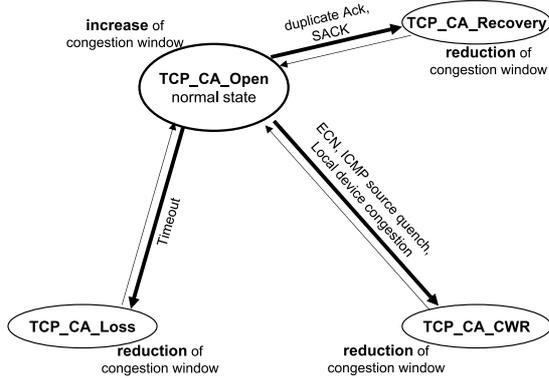


図 8 Linux TCP 実装 : 状態機械
Fig. 8 Linux TCP implementation : State Machine

ているフロー制御の状態を監視する。TCP フロー制御は各 TCP 実装（多くの場合は OS に付随しカーネル空間において実行される）の内部において管理されるため、解析システムの提供機能群のうち“TCP フロー制御監視”のみシステム実装依存となっており、本システムでは Linux TCP 実装を用いて構築されている。同機能では、輻輳 Window の推移、TCP 実装内におけるイベント発生（確認応答のタイムアウトや Sack の受信など）をカーネル外部から確認することが可能となる。

Linux OS の TCP は図 8 の様な状態機械として実装されている。状態 TCP_CA_Open は輻輳発生と認識されていない正常の状態であり、この状態においては輻輳 Window は単調に増加をつづける。パケットロス等の検出により TCP 実装が輻輳と判断した場合、状態は TCP_CA_Recovery, TCP_CA_CWR, TCP_CA_Loss に移行し、輻輳 Window は大きく減少して iSCSI 等上位層の性能も大きく減少する。状態 TCP_CA_Recovery は図 8 のように送信者が重複 Ack や Sack（選択 Ack）を受信者から送られたときに移行する。重複 Ack や Sack は受信者が途中の欠損した不連続のパケットを受信したときに、受信者が送信者に対してそのパケットの損失を通知するために用いられる。状態 TCP_CA_CWR は、送信者がネットワークからの ECN（Explicit Congestion Notification）や Local Device Congestion の通知を受けたときに移行する。状態 TCP_CA_Loss は、確認応答のタイムアウト（タイムアウト時間までに期待した Ack を受信しなかった場合）などにより送信者がパケットの損失を検出したときに移行する。Linux における TCP 輻輳制御の実装の詳細は付録に述べる。

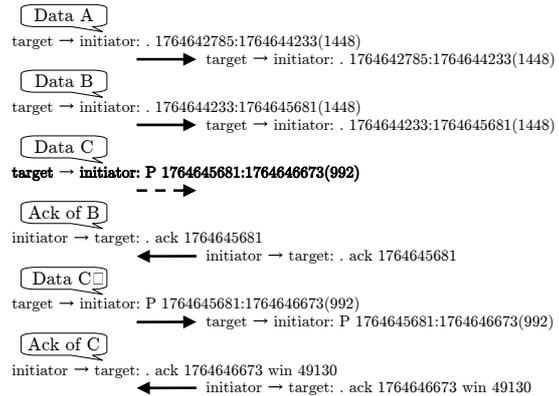


図 9 損失パケットと TCP 実装の振る舞い : Timeout
Fig. 9 Packet loss and TCP implementation behavior : Timeout

本解析システムでは、ソースコードにモニタを組み込み、実行時における制御の推移を把握することによって、TCP フロー制御の監視を行う。図 7 に示された Linux 状態機械の推移は、解析システムの当該機能を用いて表示したものである。同図のように、解析システムを用いて TCP フロー制御を iSCSI パケット転送の時間軸上での表現に重ね合わせることで、iSCSI の内部状態を視覚的に理解できる。

3.5 損失パケットの検出と iSCSI ストレージアクセスの生成

損失パケットの検出機能では、ネットワークで失われたパケット（送信側で送信記録されているが、受信側で受信記録されていないパケット）を特定する。前述の TCP フロー制御監視システムと併用することにより、ネットワークで発生したパケットロスイベントと TCP 実装の対処の様子が可視化される。

図 9, 10 に、パケット損失を、解析システムを用いて検出し可視化したものを示す。図 9 はパケット“Data C”がネットワーク内で損失し、ターゲットは送信したがイニシエータは受信していない。その結果、イニシエータは“B への Ack”のみをターゲットに送信するが“C への Ack”を送信せず、ターゲットにはタイムアウト時間までに“C への Ack”が届かずタイムアウトしている。そして、タイムアウト時間経過後ターゲットは“Data C”のパケットを再送している。

図 10 はパケット“Data C, D, E, F”がネットワーク上で損失し、同様にターゲットでは送信されているがイニシエータに受信されていない。同例ではパケット“Data G, H”は正しくイニシエータに受信されて

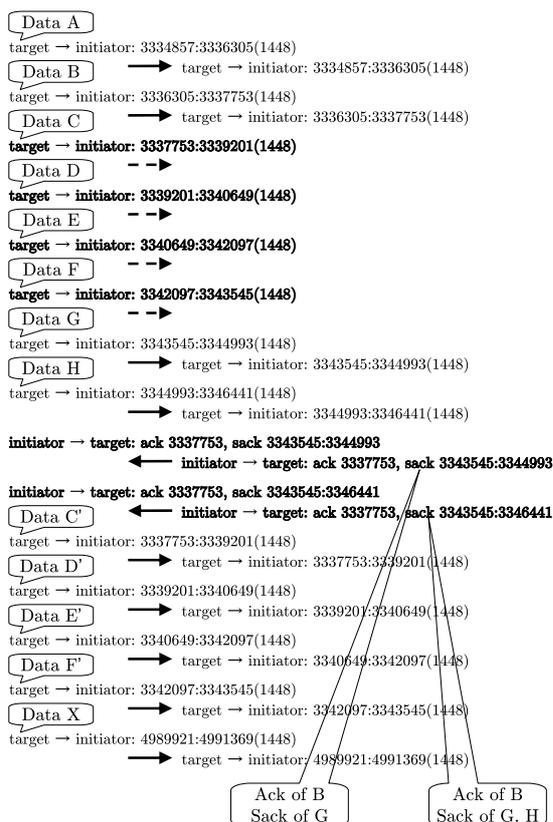


図 10 損失パケットと TCP 実装の振る舞い : Sack
 Fig. 10 Packet loss and TCP implementation behavior : Sack

いるため、イニシエータが途中の “Data C, D, E, F” が欠落している異常を検出し、“Data B” までに対する Ack, および “Data G, H” に対する Sack を含む確認応答をターゲットに対して送っている。この確認応答は “Data C, D, E, F” の損失通知を意味しており、受信したターゲットが “Data C, D, E, F” をイニシエータに対し再送している。

また本システムは iSCSI プロトコルの純粋な性能を評価するための手段として、簡易版イニシエータを用いた iSCSI ストレージアクセスの生成機能を提供している。簡易版イニシエータでは、iSCSI 規格に基づいた iSCSI Read コマンド等を任意に作成し、ソケット API を用いて任意の iSCSI ターゲットに対して iSCSI ストレージアクセスを生成することが可能である。この iSCSI アクセスの解析には、上記のすべての解析機能を適用することができる。本機能を用いて、特定の OS のドライバやストレージデバイスに起

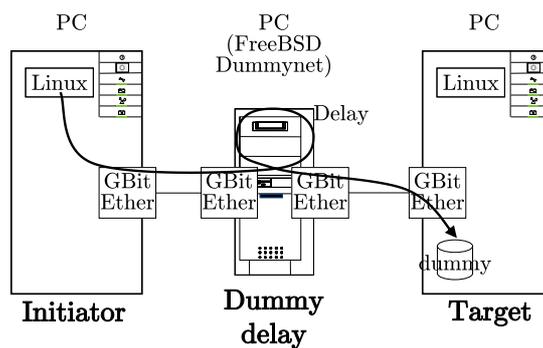


図 11 実験環境
 Fig. 11 Experiment Environment

因する制限により検証が不可能となる場合も、その制限を回避することが可能となる。本機能は以下の実験において iSCSI 環境をカスタマイズする際に利用している。

4. 高遅延環境における iSCSI 基本性能評価

本節では、高遅延ネットワーク環境における iSCSI によるストレージアクセスの基本性能を評価する。純粋な iSCSI プロトコルのみの影響を調べるため、iSCSI ターゲットはメモリモードで動作させた。以下の測定は物理的なディスクアクセスを伴わず iSCSI プロトコルに基づいてシーケンシャルリードアクセスの手続きを行った場合の性能である。すなわち本結果は iSCSI プロトコルによるネットワークストレージを使用する際の性能の上限を示す。

4.1 実験環境

性能評価実験は以下の環境で行った。

図 11 のように、iSCSI イニシエータ (サーバ) と iSCSI ターゲット (ストレージ) を Gigabit Ethernet で接続して TCP/IP 接続を確立する。Ethernet の接続は、途中に人工的な遅延装置として FreeBSD Dummynet [7] を挟みクロススケールで接続した。

iSCSI イニシエータおよび ターゲットの実装は、ニューハンプシャー大学 InterOperability Lab [8] が提供する reference implementation を用いた (以下、この iSCSI の実装を UNH と呼ぶ)。iSCSI の性能評価実験環境を表 1 に記す。イニシエータ、ターゲット、遅延装置はすべて PC 上に構築し、イニシエータとターゲットには Linux を、遅延装置には FreeBSD をインストールした。イニシエータ、ターゲット PC

表 1 性能評価実験環境 1

Table 1 Environment for Performance Evaluation 1

iSCSI Initiator, Target	UNH IOL Draft 18 reference implementation ver. 3
iSCSI	
MaxRecvDataSegmentLength	16777215 Byte
iSCSI MaxBurstLength	16777215 Byte
iSCSI FirstBurstLength	16777215 Byte
ベンチマーク	Single Thread

表 2 性能評価実験環境 2 : 使用計算機

Table 2 Environment for Performance Evaluation 2
: PC Specification

CPU	Pentium4 2.80GHz
Main Memory	1GB
OS	Linux 2.4.18-3
Network Interface	Gigabit Ethernet Card Intel PRO/1000 XT Server Adapter

表 3 性能評価実験環境 3 : 使用計算機

Table 3 Environment for Performance Evaluation 3
: PC Specification

CPU	Pentium4 1.5GHz
Main Memory	128MB
OS	FreeBSD 4.5-RELEASE
Network Interface	Gigabit Ethernet Card Intel PRO/1000 XT Server Adapter × 2

の詳細は表 2 の通りであり、遅延装置の PC の詳細は表 3 の通りである。

実験は、イニシエータ計算機からターゲット計算機に iSCSI で接続を行い、イニシエータ計算機の OS(Linux) 上から raw デバイスに対してシーケンシャルリードを行った。iSCSI ターゲットはメモリモードで動作をさせており、ディスクへのアクセスは伴っていない。これは無限に高速なストレージデバイスと見なせる。また、TCP 広告 Window サイズは 2MB である。

4.2 実験結果

上記の測定実験を行い図 12 の結果を得た。以下、この実験を“実験 A”，その結果を“実験結果 A”と呼ぶ。横軸 (One Way Delay) は、イニシエータ計算機とターゲット計算機間の片道遅延である。“0ms”，“1ms”，“2ms”，“4ms”，“8ms”，“16ms”は遅延装置経由でイニシエータ計算機とターゲット計算機を接続しそれぞれの値の片道遅延を人工的に作成した。“0ms”は遅延装置を経由するが人工的には遅延を作成しなかった場

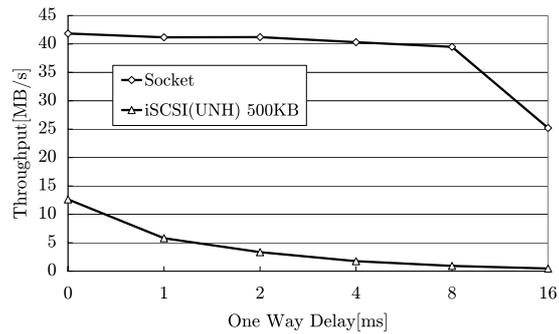


図 12 実験結果 A : iSCSI シーケンシャルリードスループット

Fig.12 Experiment Result A : iSCSI Sequential Read Throughput

合である。“0ms”の遅延は片道 140 μ s 程度である。

図中の“iSCSI(UNH)”が、上記の実験条件における iSCSI を用いてシーケンシャルリードを行ったときのスループットである。図中の“Socket”は、“iSCSI(UNH)”と同条件 (片道遅延, TCP Window Size が等しい) において単純なソケット通信を行ったときのスループットであり、これが iSCSI の下位層が提供できる限界の通信速度である (以下、この単純なソケット通信を“素のソケット通信”と呼ぶ)。シーケンシャルリードのブロックサイズは 500KB^(注3)である。ブロックサイズとは OS に対して raw デバイスの read() システムコールを行う際にアプリケーションが指定したサイズである。このブロックサイズの意味は次節で議論する。“Socket”のスループットの上限が約 40[MB/sec]となっているのは、遅延装置の限界である。

測定結果より、iSCSI プロトコルを用いてシーケンシャルリードを行うときのスループットはイニシエータとターゲット間の遅延の増加に伴い、大きく低下することが分かった。また、図中の“iSCSI(UNH)”を“Socket”と比較すると、iSCSI のスループットは下位層の提供できるスループットと比較して著しく低下している。

5. 解析システムを用いた性能低下要因の発見と解決

第 4 節の実験結果より、前述の実験環境の以下の 2 点が確認された

(注3): 500KB は 500×1024B

(1) iSCSI プロトコルを適用すると、下位層が提供可能なスループットの限界性能を十分に利用できない。

(2) iSCSI プロトコルにおける性能は遅延時間の増加により劣化する。

本節では上記の実験環境に解析システムを適用することにより、問題点が容易に見え性能向上が可能であることを示す。

5.1 iSCSI アクセスの解析

本節において 4.2 節の実験 A の解析結果を示す。この実験の本解析システムによるプロトコル翻訳結果は図 5 に示されているものであり、また時間軸上で可視化したパケット転送は図 6 である。

まず先述のように図 5 の iSCSI PDU 翻訳結果より、発行されている SCSI コマンドは 32KB の Read コマンドであることがわかる。実装に依存するが、一般にアプリケーションが read() を発行するとファイルシステム、ブロックデバイスやキャラクタデバイス、SCSI ドライバ、iSCSI ドライバを経由し、TCP/IP 実装に渡されネットワークで転送される。これらファイルシステム等を経由した結果、アプリケーションが発行したシステムコールのブロックサイズ(これを“システムコールブロックサイズ”と呼ぶ)と実際に発行される iSCSI PDU 内のブロックサイズ(これを“PDU ブロックサイズ”と呼ぶ)が必ずしも一致しない。本実験の例では、アプリケーションは 500KB の read() を発行し、これが複数の 32KB Read iSCSI PDU に分割されターゲットに送られていることが解析により確認できる。次に図 6 を見ると、サイクル中において消費されている時間の多くがネットワーク I/O の待ち時間であり、同サイクル内において実際に通信が行われている時間は極めて少ない。

5.2 性能低下要因

iSCSI シーケンシャルリードのスループットは

$$\frac{PDU \text{ ブロックサイズ}}{4 \times \text{片道遅延} + \frac{PDU \text{ ブロックサイズ}}{\text{下位層スループット}}}$$

の形にモデル化することができる(このモデル詳細については付録 2. 参照)。ここで、各サイクル内において実際にネットワークが使用されデータが転送されている時間を“非アイドル時間”と呼び、通信相手からのデータの受信を待っている等でネットワークを使用していない時間を“アイドル時間”と呼ぶ。付録 2. より、非アイドル時間率は

$$PDU \text{ ブロックサイズ} / \text{下位層スループット}$$

$$4 \times \text{片道遅延時間} + PDU \text{ ブロックサイズ} / \text{下位層スループット}$$

となり、PDU ブロックサイズが 32KB の場合、アイドル時間比率は片道遅延時間が 1ms において 84%、2ms において 91%、4ms において 95%、8ms において 97%、16ms において 98% となっている。5.1 節の実験結果から、ネットワーク利用率が低いことが分かった。付録 2. の解析より、ネットワーク利用率が低下する原因はブロックの細分化である。したがって、性能低下原因はブロックサイズの細分化と予想される。

5.3 低下要因回避と性能向上

前節では性能低下要因が“ブロックサイズの細分化によるアイドル状態”であることを確認した。本節では、この低下要因の回避と性能向上について述べる。

前節で述べたように、これは SCSI ドライバや iSCSI ドライバ等を経由した結果の細分化であり、iSCSI の本質的な制限ではない。実験環境ではアプリケーションから大きなブロックサイズの iSCSI Read PDU を発行できず、iSCSI 本来の性能を計測することができないため、提案実装の簡易版イニシエータを用いて大きなブロックサイズの iSCSI Read コマンドを発行し測定を行った。実験環境は、第 4.1 節と同様である。

実験は (1) 素のソケット通信、(2) “UNH” の iSCSI アクセス、(3) 簡易版イニシエータによるアクセスの 3 測定を行い、図中ではそれぞれ (1) “Socket”, (2) “iSCSI(UNH)”, (3) “iSCSI(KI)” と記す。その際に用いたブロックサイズも併せて記す。これは、“iSCSI(UNH)” においてシステムコールブロックサイズであり、“iSCSI(KI)” において PDU ブロックサイズである。(1) はイニシエータ計算機とターゲット計算機間のソケット通信のスループットである。(2) はイニシエータ計算機からターゲット計算機に iSCSI 接続を行いその raw デバイスに対してシーケンシャルリードを行う。イニシエータ側、ターゲット側とも UNH の iSCSI 実装を用いており、ターゲットはメモリモードである。OS の提供する SCSI ドライバや iSCSI イニシエータドライバ等を経由する。(3) はイニシエータ計算機から簡易版イニシエータを用いて、iSCSI ターゲットに対して iSCSI PDU を送りシーケンシャルリードを行う。ターゲット側は UNH の iSCSI ターゲット実装であり、同様にメモリモードを用いた。簡易版イニシエータは、ソケットに対して直接 iSCSI PDU を送信するため OS の提供する SCSI ドライバや、iSCSI ドライバは経由しない。本実験で

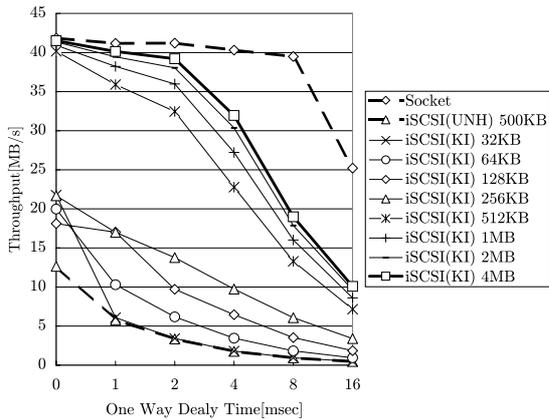


図 13 実験結果 B : Large PDU ブロックサイズ
Fig. 13 Experiment Result B : Large PDU Block Size

は (2) と (3) の違いとしてブロックサイズの他にドライバを經由するか否かがあるが、ネットワーク遅延と比べてこの差は十分に小さく、以下の実験結果においても遅延のある環境では iSCSI(UNH) と iSCSI(KI) 32KB はほぼ同等の性能となっている。

実験結果を図 13 に示す(本実験を“実験 B”と呼ぶ)。同図より iSCSI(KI) のスループットは iSCSI(UNH) の性能を大きく上回り、ブロックサイズを拡大することによって iSCSI 性能が大きく改善された。特に高遅延環境において大きな向上がみられ、片道遅延時間 16ms において PDU ブロックサイズが 4MB (图中的 iSCSI(KI) 4MB) の場合、スループットは 10.1[MB/s] となり、iSCSI(UNH) の 0.47[MB/s] に対して 20 倍以上向上した。しかし iSCSI(KI) は素のソケット通信のスループット 25.2[MB/s] の半分以下の性能となっており、依然 iSCSI プロトコルを用いることによる実験環境が提供できる限界性能(素のソケット通信)に対する劣化は残されている。

5.4 ブロック細分化回避後の解析

ブロック細分化の問題を解決した場合の iSCSI(KI) 4MB のスループットの時間推移および TCP 実装の輻輳 Window と状態遷移は図 14 となり、同環境における素のソケット通信のスループット時間推移、輻輳 Window、状態遷移は図 15 となる。また TCP フロー制御監視機能で監視した TCP 実装内のローカル輻輳の検出および TCP 実装が状態 TCP_CA_CWR に遷移するイベントは同図内に“Local Congestion”と表示されている(注4)。

(注4): イベントは縦線で表現されているが、イベントは“発生した時

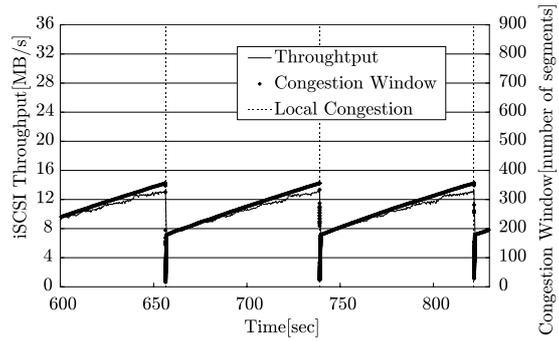


図 14 輻輳 Window の推移 (iSCSI)
Fig. 14 Congestion Window (iSCSI)

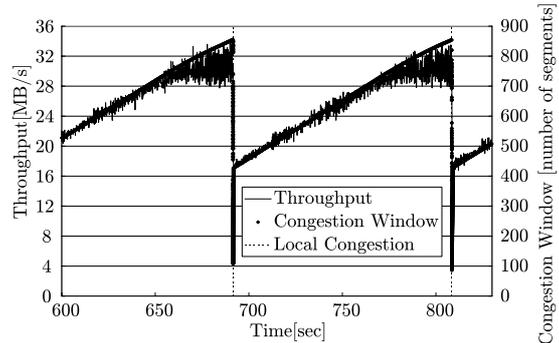


図 15 輻輳 Window の推移 (ソケット)
Fig. 15 Congestion Window (Socket)

PDU ブロックサイズ 4MB, 広告 Window 2MB, 片道遅延システム 16ms

解析システムにより得られたこれらの図より、素のソケット通信では輻輳 Window は十分に大きな値(約 850)まで上昇し、スループットも十分に大きな値となっているのに対して、iSCSI(KI) は輻輳 Window の値が 356 に上昇した時点で必ず TCP 実装がローカル輻輳エラーを検出し輻輳 Window を縮小している。その結果 TCP 実装が出力を制限し、iSCSI のスループットもこれに同期して低下している。素のソケット通信は、TCP が持つセルフクロッキング [9] によりそのバースト性が時間経過とともに減衰しバーストの無い送信を行っているのに対し、iSCSI(KI) においては TCP の上位層である iSCSI 層が TCP 層とは独立に確認応答 (SCSI Response) を行い iSCSI Seq. Read サイクル毎に同期を取るため TCP の持つセルフクロッキングによるバースト性の減衰は機能せず時間経過後もバーストが残る。この結果、輻輳 Window の上昇がローカル輻輳の発生につながりスループットの

刻”のみが意味を持つ

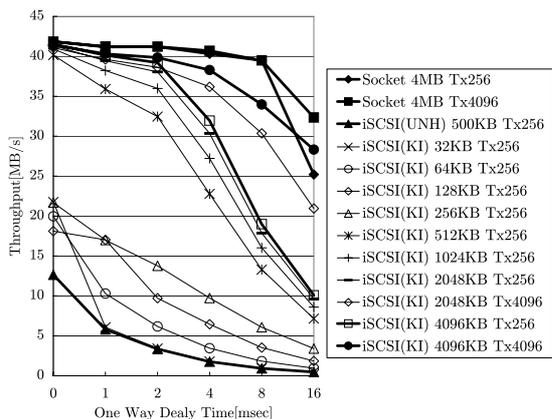


図 16 実験結果 C : Large PDU ブロックサイズ, ローカル輻輳回避

Fig.16 Experiment Result C : Large PDU Block Size, Local Congestion Avoidance

低下を招く。これは SCSI プロトコルを TCP プロトコルの上で転送する iSCSI プロトコルの本質的な問題である。

ローカル輻輳は、デバイスドライバ内におけるパケットの識別子の枯渇が起きている。これを回避することによりさらなる性能の向上が期待される。本実験で用いたネットワークカードはパケット識別子数を 80 以上 4096 以下の範囲で指定することが可能であり、前節の実験は初期値の 256 個を用いた。これを 4096 とし性能を評価した結果を図 16 に示す(本実験を“実験 C”と呼ぶ)。同図における Tx とは NIC ドライバにおける識別子数のことである。同図においてローカル輻輳の回避によるさらなる性能の向上が確認され、ブロックサイズ 4MB の場合、ローカル輻輳回避以前 (Tx 256) と比べ、片道遅延時間 16ms において 2.81 倍の性能向上が確認され、解析による性能劣化原因回避前 (実験 A) に対しては 60.5 倍の性能向上がなされた。またシステムの提供できる限界スループットに対する iSCSI 層が原因となるスループットの劣化は、識別子数増加前は片道遅延時間 16ms において 60% であったのに対し 12% となり、iSCSI プロトコルを使用することによるスループットの劣化の大幅な削減がなされた。

このように提案 iSCSI 解析システムを用いて各層を統合的に解析することにより iSCSI 性能を低下させている要因を発見することが可能となり、それらに適切に処理することにより iSCSI 性能を大きく向上さ

せることができる。

6. おわりに

本論文では、我々の実装した iSCSI ストレージアクセスの解析システムを紹介し、これにより iSCSI ストレージアクセスの性能劣化要因の検出を容易に行うことが可能となり、性能劣化原因の回避により iSCSI 性能を大きく向上 (本論文の実験において最大 60 倍以上の性能向上) できることを示した。

提案解析システムは、iSCSI プロトコル翻訳機能、送受信パケットの時間軸上における可視化、TCP フロー制御の監視、損失パケットの検出、簡易版イニシエータを用いた iSCSI ストレージアクセスの生成などの機能を有しており、本論文では各機能の解説を行った。既存の iSCSI 実装とネットワーク環境を用いて iSCSI プロトコルを用いたストレージアクセスを行った場合、その性能は下位層が提供できる限界性能に比べ著しく劣化してしまう。提案解析システムを用いた結果、性能低下要因の一つがブロックサイズの細分化にあることがわかり、細分化を回避することにより性能が大きく向上した。次にブロックサイズ細分化を回避した状況において再度解析システムを適用し、他の性能劣化要因がローカル輻輳であることが示された。ローカル輻輳を回避することにより性能はさらに改善され、iSCSI 層におけるスループットがシステムが提供できる限界スループットに近づいた。

今後は本解析システムを適用して iSCSI ストレージアクセスのさらなる解析を進め、スループット向上のための下位層も含めた最適化、異なるアクセスパターンにおける iSCSI 内部の性能劣化の回避などを検討する予定である。

謝辞 本研究の一部は、文部科学省科学研究費補助金基盤研究 S 課題番号:13852015 ならびにリーディングプロジェクト e-society による。ここに記して謝意を表します。

文 献

- [1] 喜連川優, “ストレージネットワークング”, オーム社出版局, 2002
- [2] IETF : <http://www.ietf.org/>
- [3] IETF IPS, <http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-20.txt>
<http://www.ietf.org/html.charters/ips-charter.html>
- [4] SNIA : <http://www.snia.org/>
- [5] Wee Teck Ng, Bruce Hilly Elizabeth Shriver, Eran Gabber, Banu Ozden, “Obtaining High Perfor-

- mance for Storage Outsourcing”, *Proc. FAST 2002, USENIX Conference on File and Storage Technologies*, January 28-29, 2002, pp. 145-158
- [6] Prasenjit Sarkar and Kaladhar Voruganti, “IP Storage: The Challenge Ahead”, *Proc. of Tenth NASA Goddard Conference on Mass Storage Systems and Technologies*, April 2002
- [7] L. Rizzo, “dumynet”,
http://info.iet.unipi.it/~luigi/ip_dumynet/
- [8] The University of New Hampshire’s InterOperability Lab
http://www.iol.unh.edu/consortiums/iscsi/iscsi_linux.html
- [9] Van Jacobson, “Congestion Avoidance and Control”, *Proc. of SIGCOMM '88, ACM*, 1988, pp. 314-329.

付 録

1. Linux における TCP 輻輳制御の実装

Linux カーネル 2.4.18 における TCP 輻輳制御の実装を抜粋し、図 A-1, A-2, A-3, A-4 に示す。ソースファイル名およびそのファイル内における行数も併記してあるがこれは Linux 2.4.18 におけるファイル名およびその行数である。コネクションに関する情報は、`struct sock` 型構造体およびそのメンバ変数 `struct tcp_opt` 型構造体に保存されており、フロー制御に用いられる輻輳 Window の値は `struct tcp_opt` 型構造体のメンバ変数 `snd_cwnd` に、スロースタートフェイズと輻輳回避フェイズの閾値は `struct tcp_opt` 型構造体のメンバ変数 `snd_ssthresh` に保存される。また、同実装ソースコード内の各関数内では、対象とするコネクションの情報が格納されている `struct tcp_opt` 型変数へのポインタは、変数名 `struct tcp_opt *tp` を用いて統一的に表現されている。

1.1 Ack 受信と輻輳 Window の増加

まず、状態 `TCP_CA_Open` (注5)において正常に Ack を受信し、輻輳 Window が増加する過程について述べる。図 A-1 は、状態 `TCP_CA_Open` において輻輳 Window を上昇させる実装部分である。図の様に状態 `TCP_CA_Open` では Ack を受信するたびに関数 `tcp_ack()` が呼び出される(図中の“1:”)。関数 `tcp_ack()` 内では、関数 `tcp_ack_is_dubious()` が呼び出され(図中の“2:”) Ack が正常なものであるかを確認し(図中の“3:”)、正常なものである場合(図中の“4:”)は関数 `tcp_cong_avoid()` が呼び出され(図中の“5:”)輻輳 Window が増加する。Linux の TCP 実装においては輻輳 Window の値はバイト単位ではな

(注5): 輻輳が無いと判断されている正常な状態

く MSS の個数単位で管理されており、スロースタートフェイズにおいては Ack 1 個毎に輻輳 Window が 1 個(注6)上昇し(注7)、輻輳回避フェイズにおいては変数 `snd_cwnd_cnt` により受信した Ack 数を記録し、現在の輻輳 Window の値と同じ数(`snd_cwnd` 個)毎に輻輳 Window が 1 個上昇する(注8)。

スロースタートフェイズと輻輳回避フェイズは現在の輻輳 Window の値により分岐し、その閾値として変数 `snd_ssthresh` が用いられている。

1.2 Sack の受信

次に、Sack 受信や重複 Ack 受信によりパケットの損失を検出し輻輳 Window が減少する過程について述べる。図 A-2 は、Sack 受信等により状態 `TCP_CA_Open` から状態 `TCP_CA_Recovery` に遷移し輻輳 Window を減少させる実装部分である。送信者が送信したパケット群の一部がネットワーク内において損失し、受信者に途中が欠損した不連続なパケット群として受信された場合、受信者はパケットの損失を検出しそれを送信者に通知する。送受信者双方に Sack が実装されている場合は Sack が用いられ、送受信者の少なくとも一方に Sack が実装されていない場合は重複 Ack が用いられる。双方に Sack が実装されている場合、受信者は連続して完全に受信したデータに対する通常の Ack および途中が欠損しその後受け取ったデータ部分に対する Sack を送信する。Sack を受信すると Linux の TCP は図 A-2 の振る舞いを示す。すなわち、状態 `TCP_CA_Open` において Sack が付随している Ack パケットを受信し関数 `tcp_ack()` が呼び出される(図中の“1:”)。関数 `tcp_ack_is_dubious()` が呼び出され(図中の“2:”) Ack が正常なものであるかの確認に入る。ここで、Sack 付随の Ack は関数 `tcp_ack_is_dubious()` 内における `(flag & FLAG_CA_ALERT)` が真となり(注9)、関数 `tcp_ack_is_dubious()` から真が返される(図中の“3:”)。これにより関数 `tcp_ack()` 内において `if(tcp_ack_is_dubious())` は成立し(注10)(図中の“4:”)、関数 `tcp_fastretrans_alert()` が呼び出される(図中の“5:”)。関数 `tcp_fastretrans_alert()`

(注6): バイト単位において MSS 分

(注7): 図 A-1 におけるファイル ‘net/ipv4/tcp_input.c’ 1707 行目

(注8): 図 A-1 におけるファイル ‘net/ipv4/tcp_input.c’ 1712 行目から 1717 行目

(注9): 図 A-2 におけるファイル ‘net/ipv4/tcp_input.c’ 1841 行目

(注10): 図 A-2 におけるファイル ‘net/ipv4/tcp_input.c’ 1854 行目

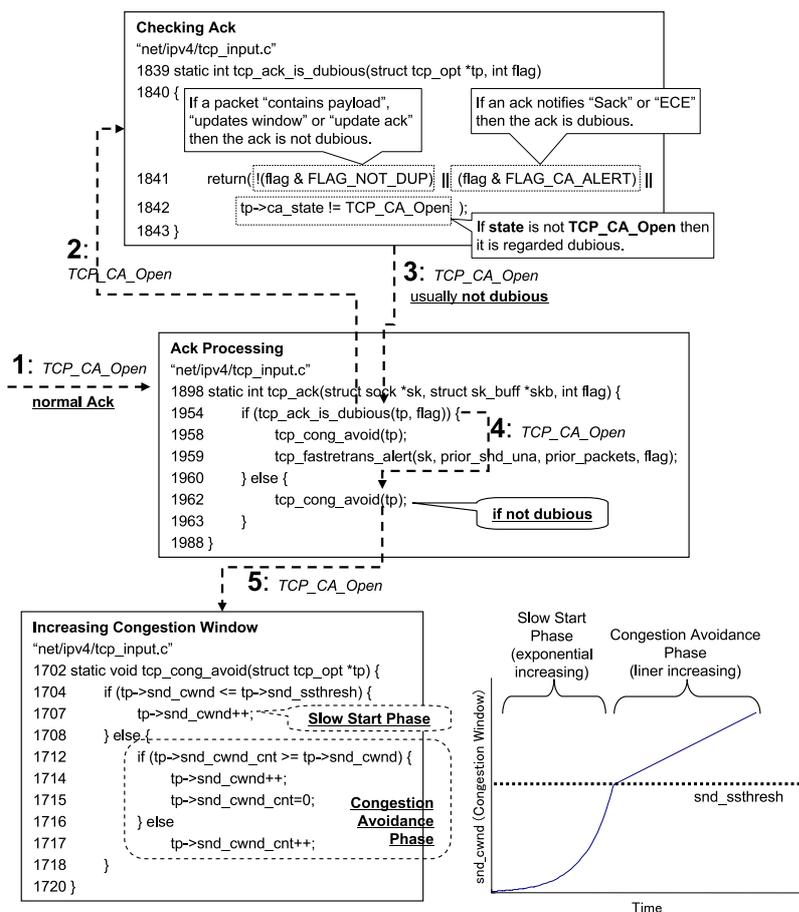


図 A.1 Linux TCP 実装 : 状態 "TCP_CA_Open"
 Fig. A.1 Linux TCP implementation : State "TCP_CA_Open"

において状態 TCP_CA_Open から状態 TCP_CA_Recovery に遷移し、tcp_cwnd_down() 内において輻輳 Window (snd_cwnd) が減少する。状態は TCP_CA_Recovery に遷移したため、今後 Ack を受け取り関数 tcp_ack() が呼び出される(図中の "6:") 毎に関数 tcp_ack_is_dubious() は真となり (tp->ca_state != TCP_CA_Open が真となる)、関数 tcp_fastretrans_alert() が呼び出され関数 tcp_cwnd_down() が呼び出され輻輳 Window は減少していく。

1.3 輻輳通知

次に、輻輳通知により輻輳 Window が減少する過程について述べる。図 A.3 がパケットの送信を試み、ローカル輻輳により送信が失敗した場合の輻輳 Window 減少過程である。TCP の上位層から

TCP 実装に対しデータ送信要求が発生するとすると tcp_transmit_skb() が呼び出され(図中の "1:")、同図内の tp->af_specific->queue_xmit(skb); において下位層のキューへのエンキューが試みられる。エンキューが正常に終了した場合は tcp_transmit_skb() はここで終了し(注11)、ローカル輻輳により正常に終了しなかった場合は関数 tcp_enter_cwr() が呼び出され(図中の "2:") 同関数内で状態 TCP_CA_CWR に遷移する(注12)。状態 TCP_CA_CWR へ遷移したため今後 Ack を受け取り(図中の "3:")、関数 tcp_ack_is_dubious() が呼び出される(図中の "4:") 毎に真が返され(図中の "5:")、関数 tcp_fastretrans_alert() が呼びだ

(注11): 図 A.3 におけるファイル 'net/ipv4/tcp_output.c' 277 から 279 行目

(注12): 図 A.3 におけるファイル 'include/net/tcp.h' 1141 行目

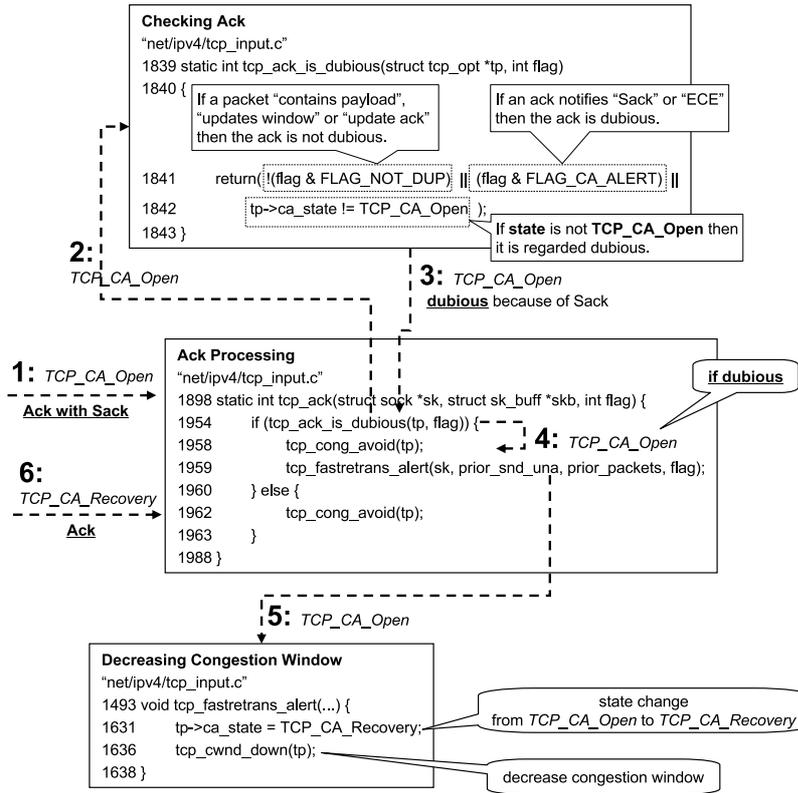


図 A.2 Linux TCP 実装：状態 “TCP_CA_Recovery”
Fig. A.2 Linux TCP implementation：State “TCP_CA_Recovery”

さる（図中の“6:”）こととなる．よって，Ackを受信するたびに輻輳 Window が減少（図中の“7:”）していくこととなる．紙面の都合上詳細は省略するが，図 A.4 に確認応答のタイムアウトの動作を示す．

2. iSCSI シーケンシャルリードスループットモデリング

iSCSI Seq. Read サイクルで転送されるデータ量は，サイクルの最初にイニシエータが発行する iSCSI Read コマンド PDU で要求されているブロックサイズである．1 サイクルに要する時間は

$$1 \text{ サイクルに要する時間} = 4 \times \text{片道遅延時間} + \text{データ転送時間} \quad (\text{A.1}) \text{TCP/IP による制限}$$

$$\text{データ転送時間} = \frac{\text{要求データサイズ}}{\text{下位層のスループット}} \quad (\text{A.2})$$

となる．“要求データサイズ”は，PDU ブロックサイズである．“下位層のスループット”とは iSCSI 層にとっての下位層の提供するスループットのことであり，

素のソケット通信がこれにあたる．以上より，iSCSI プロトコルによるシーケンシャルリードのスループットは

$$\text{スループット} = \frac{\text{ブロックサイズ}}{4 \times \text{片道遅延} + \frac{\text{ブロックサイズ}}{\text{下位層スループット}}} \quad (\text{A.3})$$

とモデル化することができる．また下位層である，素のソケット通信のスループットは (TCP Window サイズ)/(2×片道遅延時間) 以下に制限される．以上をまとめると，iSCSI でのスループットは以下の制限を受けることになる．

$$\begin{aligned} &\text{素のソケット通信スループット} \\ &\leq \frac{\text{TCPWindowSize}}{2 \times \text{片道遅延時間}} \quad (\text{A.4}) \end{aligned}$$

iSCSI プロトコルによる制限

iSCSI スループット

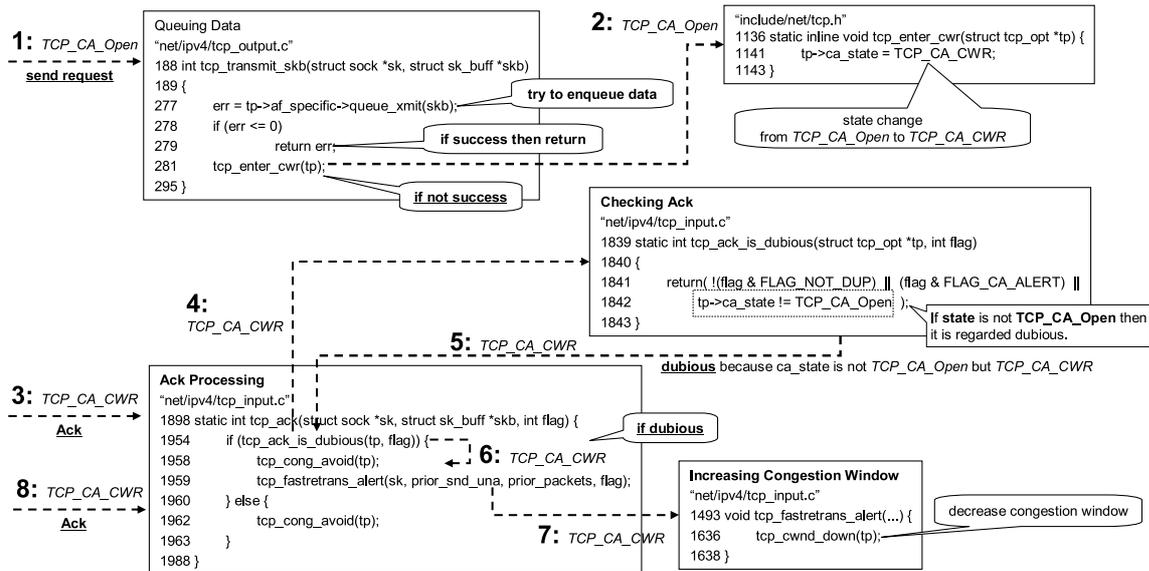


図 A.3 Linux TCP 実装：状態“TCP_CA_CWR”
Fig. A.3 Linux TCP implementation : State“TCP_CA_CWR”

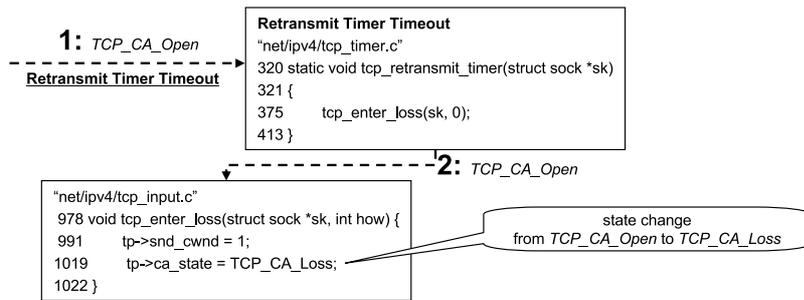


図 A.4 Linux TCP 実装：状態“TCP_CA_Loss”
Fig. A.4 Linux TCP implementation : State“TCP_CA_Loss”

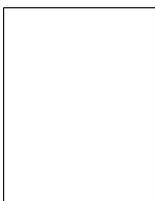
$$= \frac{\text{ブロックサイズ}}{4 \times \text{片道遅延} + \frac{\text{ブロックサイズ}}{\text{素のソケット通信スループット}}} \quad (\text{A.5})$$

3. 解析システムのオーバーヘッド

本解析システムを適用した状態では解析システムに起因するオーバーヘッドが発生し性能が劣化することが予想される。本稿では性能低下要因の考察においては解析システムが適用されている状態での計測を利用し、考察後の性能測定では解析システムが適用されていない状態で計測を行った。従って、測定された性能は解析システムの影響を受けていない。具体的には、図 6, 7, 9, 10, 14, 15 が解析システムを適用した状態においての解析結果であり、解析によるオーバーヘッドが含まれている。図 12, 13, 16 の性能測定は解析システムの動作しない状況での測定結果である。ま

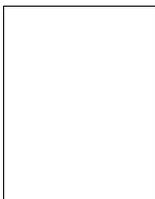
た、本解析システムの TCP フロー制御監視機能を用いることのスループットへの影響は PDU ブロックサイズが 4MB の場合に片道遅延時間が 8ms かつ NIC ドライバ識別指数 256 個, 8ms 4096 個, 16ms 256 個の 3 測定において 1%未満, 16ms 4096 個において 1.3% となった。多くの場合においてその影響は十分に小さいと考える。また、パケットの可視化にはパケットログ採取の際にオーバーヘッドが生じ、片道遅延時間 16ms 識別指数 256 個において 7%程度, 16ms 4096 個において 1%程度の影響となった。

(平成 x 年 xx 月 xx 日受付)



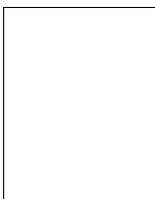
山口 実靖

東京大学生産技術研究所 産学官連携研究員。2002年 東京大学大学院博士課程修了,工学博士。iSCSIを用いたネットワークストレージシステムの性能向上の研究に従事。情報処理学会会員。



小口 正人 (正員)

お茶の水女子大学理学部情報科学科助教授。1995年 東京大学大学院工学系研究科博士課程修了,工学博士。並列・分散処理, 計算機ネットワークに関する研究に従事。IEEE, ACM, 電子情報通信学会, 情報処理学会各会員。



喜連川 優 (正員)

1978年東京大学工学部卒。1983年同大学院工学系研究科情報工学博士課程了。工学博士。同年同大生産技術研究所講師。現在,同教授。2003年より同所戦略情報融合国際研究センター長。データベース工学,並列処理,Webマイニングに関する研究に従事。現在、情報処理学会理事,日本データベース学会理事、平成11-14年 ACM SIGMODJapan Chapter Chair 平成9,10年本学会データ工学研究専門委員会委員長。VLDB Trustee(97-02), IEEE ICDE, PAKDD, WAIM などステアリング委員