リアルタイム圧縮によるパケットキャプチャの高速化

清水 奨[†] 風間一洋[††]
廣津登志夫[†††] 後藤滋樹[†]

本論文は，ネットワークの分析やセキュリティ確保のために重要な役割を持つパケットキャプチャシステムを高速化する手法について述べる．本手法の特徴は，パケットを複数まとめてリアルタイムに圧縮する点にある．これにより，パケット保存にともなう処理コストを削減し，キャプチャ性能を向上させる．同時にディスクの使用量を抑制することができる．オペレーティングシステムに手を加える必要はない．本手法は，バッファオーバフロー攻撃パケットのような，冗長データの多いパケットのキャプチャに大きな効果がある．パケットキャプチャの代表的なプログラムである libpcap に本手法を適用すると，このようなパケットを欠落なくキャプチャできる通信速度が従来の 1.2 倍になる．運用中のネットワークにおけるキャプチャでは，従来の圧縮よりも高い負荷まで欠落せずにキャプチャでき，パケットを記録するトレースファイルのサイズは約 1/5 になる．本手法の適用により，安価な汎用 PC を用いるソフトウェアベースのパケットキャプチャシステムを容易に高度化できる．

A New Method for High Speed Packet Capturing with Compression

Susumu Shimizu,[†] Kazuhiro Kazama,[††] Toshio Hirotsu[†††]
and Shigeki Goto[†]

This paper proposes a new method of improving the performance of packet capturing. The method compresses a block of packets while receiving packets. It can reduce the overhead cost of the packet capturing. This method is applied to `libpcap` program which is a well known capturing tool. If the block of packets are almost identical, the method shows 120% faster than the existing capturing tool. The method also reduces the size of packet trace files. We have applied the new method for capturing actual traffic. The result shows that the performance is increased by 20%. And the size of packet trace file is reduced to one fifths of the original size. The method enables efficient packet capturing on software-based capturing systems.

1. はじめに

ネットワークの計測や分析においては，パケットキャプチャシステムが重要な役割を果たす．パケットキャプチャシステムはネットワーク上を流れるパケットを収集して，パケットトレースファイルに記録する．パケットトレースファイルを分析すれば，ネットワーク状況の監視や特性の詳細な把握が可能となる．最近は，ハニーネット[1] のように，ネットワークへの侵入手法を分析するため，多地点にパケットキャプチャシステムを展開し，長時間にわたって多量のパケットを収集する方法が提案されている．正確に分析するためには，パケットを欠落なく保存することが重要である．

パケットキャプチャシステムには高価で専用のハードウェアを用いるものと，汎用の PC で動作するソフトウェアを用いるものがある．前者は高速なキャプチャができるが，一般に高速バッファメモリの容量には限りがあり，長時間のキャプチャには向かない．一方，後者はキャプチャ性能は高くないものの，大容量のディスクを使った長時間のキャプチャが容易に実施できる．このため，ネットワークに対する攻撃の監視にはソフトウェアによるパケットキャプチャシステムが用いられることが多く，代表的なツールに tcpdump [2] や ethereal [3] などがある．

ソフトウェアによるパケットキャプチャシステムのキャプチャ性能は動作させる PC の仕様によって変わる．最近の PC は CPU 能力の向上が目覚ましく，ギ

† 早稲田大学大学院理工学研究科
Graduate School of Science and Engineering, Waseda University
†† NTT 未来ねっと研究所
NTT Network Innovation Laboratories
††† 豊橋技術科学大学情報工学系
Department of Information and Computer Science, Toyohashi University of Technology

ガビットイーサネットなど従来は専用のハードウェアを用いるしかなかった高速なネットワークに対しても，ソフトウェアによるキャプチャシステムが適用できるようになってきた．しかし，トラフィックの負荷が高くなるとCPU能力よりもI/Oの能力がボトルネックになる場合が多く，CPU能力に余裕があるにもかかわらずパケットの欠落が生じるという問題があった．

本論文では，パケットを複数個まとめてリアルタイムで圧縮することにより，入出力のオーバヘッドを削減し，ソフトウェアによるパケットキャプチャシステムのキャプチャ性能を向上させる手法を提案する．オペレーティングシステムに依存する処理を避け，ライブラリの改良のみで実現する．

一般に，データの圧縮による性能向上は，磁気テープ装置やモデムなどの低速なデバイスが対象となる．圧縮には計算コストがかかることから，ネットワークのような高速デバイスが対象となることはなかった．

圧縮によりキャプチャ性能が向上するかどうかは，圧縮対象のデータパターン，圧縮アルゴリズムの圧縮率やスループット，ディスクのスループットなど複数の要因が関係する．そのため，性能要因に対する基本的な特性だけでなく，実際のトラフィックデータに対して性能が向上するかどうかを調べることが重要である．

本論文では，まず最初に広く利用されているパケットキャプチャライブラリの解析と，圧縮アルゴリズムの予備評価を通じて，キャプチャ性能を向上させうる設計について検討する．次に，この設計に基づいた実装に対して，数種類のデータパターンを与え，基本的な特性を調べる．最後に，実際のネットワークから収集したトレースファイルを用いた評価を行い，実環境でキャプチャ性能が向上することを示す．

以下，2章では従来の研究について述べ，さらにパケットキャプチャライブラリにおけるパケット欠落の要因を検討する．3章で性能を向上させうるリアルタイム圧縮機能の設計について述べ，4章で実装したシステムの基本特性を評価する．5章で実トラフィックを用いた性能評価を行い，6章では性能向上のための議論を行う．7章は結びである．

2. 従来のパケットキャプチャ

2.1 圧縮の利用

従来，パケットキャプチャの際に圧縮を用いるのは，パケットトレースファイルの容量削減が主目的であった．特に長時間にわたるキャプチャにおいては，ディスク溢れによるキャプチャ停止を回避するため，圧縮による容量削減はきわめて重要である．これまで，2種類の方法が知られている．

1つはパケットヘッダの圧縮である．たとえば，Peuhkuri[4]によるフローベース圧縮という手法がある．これはヘッダ内の特定領域をハッシュ化してトレースファイルのサイズを小さくする．しかしペイロード部分は圧縮されないので，パケット全体を保存する場合には圧縮効果が小さいという問題がある．セキュリティ確保を目的とするパケットキャプチャにおいてはパケット全体を保存するため，この手法は有効でない．

もう1つの方法は，zlib[5]などの汎用の圧縮ライブラリを使って，キャプチャ操作をしながら圧縮する方法である．DAG[6]やCoralReef[7]などのツールにすでに実装されている．しかし従来のシステムでは，圧縮を用いるとキャプチャ性能が悪化するという問題がある．このため，圧縮の利用はネットワーク負荷が低い条件でのキャプチャに限られていた．

2.2 パケットキャプチャ処理の概略

ここではパケットキャプチャソフトウェアのプラットフォームとして広く使われているlibpcap[2]の，パケットキャプチャ処理の概略を述べ，欠落が発生しうる要因について検討する．

図1に，汎用のPC上でlibpcapを用いてパケットをキャプチャする際のデータの流れを示す．ネットワークデバイスに到着したパケットは，まずカーネル空間内の受信バッファに転送される．BSDパケットフィルタ（BPF）[8]は，受信バッファからパケットを取り出してBPFバッファに保存する．libpcapはBPFバッファをつねに監視しており，BPFバッファにパケットがあればユーザ空間にコピーする．さらに，各パケットにタイムスタンプとデータ長の情報を付加して，ストリーム入出力関数を介してディスクに出力する．

ネットワーク負荷が高まると，この一連の処理のうち，BPFバッファと受信バッファの2カ所でパケットの欠落が発生する恐れがある．BPFバッファにおける欠落は，libpcapのパケット処理速度がBPFに

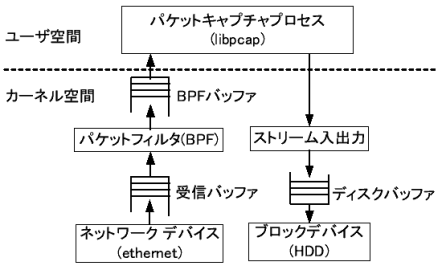

図1 パケットキャプチャ処理の概略
Fig. 1 Packet capturing process.

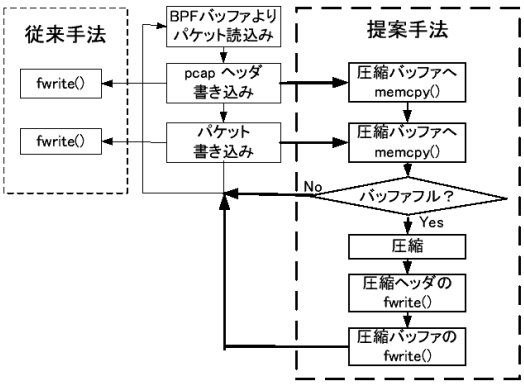

図 2 リアルタイムパケット圧縮の実装
Fig. 2 Implementation of the real-time packet compression.

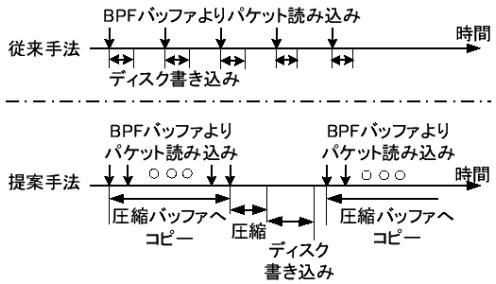

図 3 圧縮のタイミング
Fig. 3 Timing chart of the packet compression.

よるパケット取り込み速度より遅い場合に発生する．また受信バッファにおける欠落は，BPF のパケット取り込み速度がパケットの到着間隔より遅い場合に発生する．BPF はカーネル空間で動作しており，ユーザ空間で動作する libpcap より優先されるため，欠落はまず BPF バッファで生じる．さらにネットワーク負荷が高まると，受信バッファでも欠落が生じる．

受信バッファで生じるパケット欠落をソフトウェア的な手法だけで解消するのは困難である．しかし，BPF バッファで生じるパケット欠落については改善の余地がある．たとえば，BPF バッファサイズを大きくするチューニング法が知られている[9]．キャプチャ性能を高速化することで，ソフトウェアによるパケットキャプチャシステムで対応可能な領域を拡大できる．

3. リアルタイム圧縮機能の設計

本章では，キャプチャ性能を向上させうるリアルタイム圧縮機能の設計について述べる．

3.1 処理の概略

図 2 は，パケットキャプチャ処理のブロック図である．太枠で囲んだ部分が新たに追加した処理である．左側に示した従来の libpcap では，パケットごとに 2 回ずつ，fwrite() 関数を呼び出す．それぞれ，タイムスタンプと長さ情報からなる pcap ヘッダ（16 バイト）と，キャプチャしたパケットをディスクに出力する．

提案手法では，パケットごとに処理をせず，圧縮バッファという固定長のバッファを使用する．これには 2 つの面でオーバヘッドを削減する狙いがある．まず圧縮ルーチンの呼び出し回数を減らすことにより，圧縮計算にともなうオーバヘッドを削減する．次に，fwrite() 関数の呼び出し回数を減らすことで，ソフトウェアのコンテキストスイッチにともなうオーバヘッドを削減する．たとえば，32 K バイトのバッファを使う場合に，64 バイトのイーサネットフレーム☆であれば，409 個（32 K バイト/(16+64) バイト）のフレームをバッファリングして一度に処理する．これにより，パケットごとに処理を行う場合に比べて，圧縮ルーチンと fwrite() 関数の呼び出し回数が 1/409 に削減される．

この結果，新しい処理のタイミングは **図 3** の下段のようになる．図 3 の上段は従来手法のタイミングである．圧縮の計算中，およびディスクへの書き込み中に到着するパケットは，BPF バッファに蓄積される．BPF バッファにおけるパケット欠落を避けるためには，BPF バッファが溢れる前に圧縮と書き込みを終了しなくてはならない．

3.2 性能向上の条件

ここで，圧縮によりキャプチャ性能を向上させるための条件を検討する．図 3 に示したタイミングを考慮すると，圧縮とディスク書き込みの合計処理時間が，圧縮せずにディスク書き込みを行う処理時間よりも短くなれば，性能が向上すると考えられる．

すなわち，圧縮ルーチンのスループットを T_c，ディスクの書き込みスループットを T_d とし，圧縮ルーチンの平均圧縮率が r の場合に，圧縮バッファのサイズを S としたとき，式 (1) を満たせばよい．

$$\frac{S}{T_d} > \frac{S}{T_c} + \frac{rS}{T_d}. \tag{1}$$

式 (1) の左辺は非圧縮の場合のディスク書き込み時間であり，右辺は圧縮の計算時間と圧縮結果の書き込み時間の和である．これを変形すると式 (2) となる．

$$T_c > \frac{T_d}{1-r}, (0 < r < 1). \tag{2}$$

たとえば，書き込みスループットが 30 MB/秒のディ

☆ 本論文ではフレームとパケットを区別していない．扱っているデータにはつねにイーサネットヘッダが含まれるため，厳密にはフレームと呼んで統一すべきであるが，パケットキャプチャシステムという呼称を重視し，パケットという語も使用する．

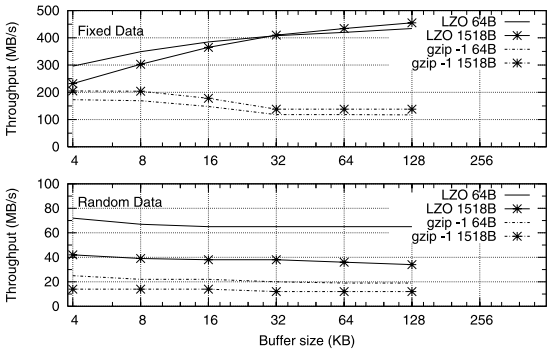

図 4 圧縮バッファサイズとスループット
Fig. 4 Compression buffer size and throughput.

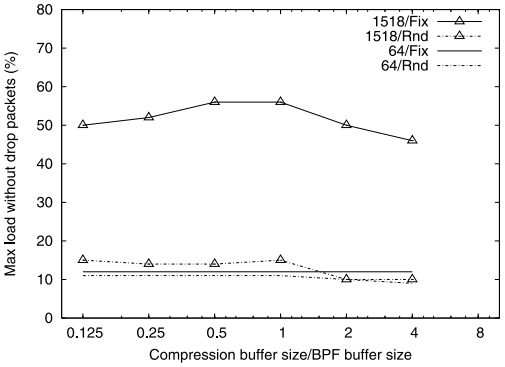

図 5 圧縮バッファサイズと無欠落最大負荷
Fig. 5 Max load (without drop) and the size of compression buffer.

表 1 トレースファイルの平均圧縮率
Table 1 Average compression ratio of packet trace files.

データパターン	フレーム長	LZO (lzo1x-1)	zlib (level 1)
固定	64 バイト	0.068	0.034
固定	1,518 バイト	0.020	0.017
ランダム	64 バイト	0.602	0.534
ランダム	1,518 バイト	0.985	0.973

スク装置を使う場合，平均圧縮率が 0.5 の圧縮ルーチンには 60 MB/秒以上のスループットが要求される．

圧縮ルーチンから得られる平均圧縮率とスループットは，アルゴリズムの実装やデータのパターンによって大きく変わる．次節以降では，複数の実装に対し 2 種類のデータパターンを与えて予備実験を行った結果を示し，圧縮アルゴリズムの選択と圧縮バッファサイズの設定について述べる．

3.3 アルゴリズムの選択

高速な圧縮ライブラリである LZO [10] と，広く使われている zlib を比較し，前節で述べた性能向上の条件を満たすかどうかを調べる．LZO は LZ77 アルゴリズム[11] の実装の 1 つで，圧縮率よりもスループットを重視している．

圧縮バッファの大きさを 4 KB から 128 KB まで変化させながら，スループットを計測した☆．データパターンは固定とランダムの 2 種類であり，フレーム長はイーサネットの最小値（64 バイト）と最大値（1518 バイト）に設定した．計測結果を**図 4** に示す．上段は固定データの場合，下段はランダムデータの場合である．また**表 1** には各パターンのパケットトレースファイルを対象に計測した平均圧縮率を示す．

図 4 からは，LZO の方が高速であることが分かる．そこで我々は LZO を圧縮アルゴリズムとして採用することにした．この場合，固定データでは明らかに大幅な性能向上が見込まれる．一方ランダムデータでは，図 4 下段に示されるスループットの値と，表 1 に示される平均圧縮率の値を式 (2) に適用して性能を見積もる必要がある．たとえば，LZO の平均スループットを 60 MB/s，平均圧縮率を 0.8 と見積もると，平均スループットが 12 MB/s 以下のディスクを持つシステムでは性能が向上するが，これより高速なディスクを持つシステムでは性能が劣化する可能性がある．実際の評価結果については 4 章で述べる．

3.4 バッファサイズの決定

図 4 からは，圧縮アルゴリズムによってバッファサイズがスループットに与える影響が異なることが分かる．LZO と zlib の比較では，前者は圧縮バッファを大きくした方が，後者は圧縮バッファを小さくした方が，スループットが良い．我々は LZO を選択したので，バッファサイズは大きいほうが良いと思われる．

しかし，最適な圧縮バッファのサイズは BPF バッファのサイズにも関係する．**図 5** は，BPF バッファのサイズに対する圧縮バッファのサイズの比と，欠落なくキャプチャできる最大負荷の関係を示す実測結果である．負荷の値はギガビットイーサネットの最大負荷を 100%としたときの比率で表示している．BPF バッファのサイズは 32 KB（libpcap の標準設定）で，圧縮バッファのサイズを 4 KB から 128 KB まで（BPF バッファの 0.125 倍から 4 倍まで）変化させた．圧縮アルゴリズムは LZO を用いている．2 種類のフレーム長を使い，データは固定値とランダム値の 2 種類である．

図 5 によると，フレーム長が 64 バイトの場合には固定データ（Fix と表示），ランダムデータ（Rnd と表示）ともに大きな差異は認められない．フレーム長

☆ 使用した機材は表 2 の DUT である．3.4 節も同じ．

が 1,518 バイトの場合は，固定データとランダムデータでキャプチャ性能が大きく異なる．固定データでは，圧縮バッファが BPF バッファの半分，もしくは同一の場合が最も良い．圧縮バッファが小さすぎても大きすぎても性能が劣化する．ランダムデータでは，圧縮バッファが小さい場合はほぼ同一の性能だが，BPF バッファより大きくなると性能が劣化する．

以上の実測結果より，計測に使用した PC では圧縮バッファを BPF バッファより大きくすると性能が悪化すること，圧縮効果の得られにくいパケットが連続する場合には圧縮バッファを小さく保つ方がペナルティが小さく，性能が良いことが分かる．ただし，前節で述べたように圧縮効果の得やすいデータパターンの場合には圧縮バッファを大きくした方が性能が向上する．このトレードオフのため，ペイロードのデータパターンを勘案して，圧縮バッファのサイズを BPF バッファサイズ以下の範囲に設定する．

圧縮バッファと BPF バッファのサイズ比については，キャプチャに使用する PC の環境（CPU 能力やデバイスドライバの性能など）によって最適値が変わる可能性がある．実際の設計にあたっては，本節で述べたようにあらかじめ実測して性能の傾向を調べることが重要である．

4. 基本特性の評価

本章では，前章の設計に基づいて実装したパケットキャプチャシステムの基本特性を評価する．ここではパケットキャプチャと圧縮というそれぞれの観点から，2 種類のパケット長と 2 種類のデータパターンを組み合わせてベンチマークを行った．

4.1 評価機材の仕様

評価に使用した機材の仕様を**表 2**☆に示す．DUT（Device Under Test）は実装したソフトウェアを動作させる PC であり，TG はトラフィックジェネレータの略である．人工的な負荷を生成する際には専用の測定器（TG1）を用い，実トラフィックの生成には PC（TG2，TG3）を用いた．

DUT のディスク書き込み性能を測定したところ，10 MB のファイルを連続書き込みする場合で 28 MB/s，100 MB のファイルを連続書き込みする場合で 45 MB/s となった．測定値は iozone [12] の Fwrite テストの結果であり，fwrite() 関数を介した書き込みスループットを示している．ディスクの書き込みスループットを 45 MB/s とし，圧縮率を 0.2 と見積もると，式 (2) から，圧縮スループットが 56.25 MB/s 以上であれば性能が向上することが分かる．

表 2 機材の仕様
Table 2 Specification of equipment.

名称	仕様
DUT	Intel Pentium 4/3.2 GHz (Intel E7201 chipset) FreeBSD 4.11R 主記憶 1 GB, SATA（aac ドライバ） 1000 Base/T（Intel Pro/1000, em ドライバ）
TG1	IXIA 400T, LM1000T（ギガビットイーサネット）
TG2	Intel Pentium 3/1.13 GHz x2, FreeBSD 4.11R 1000 Base/T（Intel Pro/1000, em ドライバ）
TG3	AMD Opteron 244/1.8 GHz x2, FreeBSD 4.11R 1000 Base/T（Intel Pro/1000, em ドライバ）

なお DUT はエントリレベルの 1U ラックマウントサーバである．使用されているチップセット（E7201）は SATA コントローラを内蔵し，ギガビットイーサネット（GbE）コントローラとの間は高速なチップ間リンク（Hublink 1.5）により接続されている．このリンクの帯域は 266 MB/s であるため，GbE インタフェースに 100%の負荷（約 123 MB/s）が加わっても，ディスク書き込み性能に対するシステムアーキテクチャ上の制約はないと考えられる．

評価にあたっては，前章で述べたトレードオフをふまえて，圧縮バッファのサイズを BPF バッファのサイズと同じ値に設定した．圧縮しにくいデータパターンと圧縮しやすいデータパターンの性能差が大きく現れるため，ベンチマークによる特性の把握が容易になる．また，オペレーティングシステムの標準設定における性能を評価するため，BPF バッファのサイズは標準（32 KB）のまま変更していない．BPF バッファサイズを変更した場合の性能向上については 6 章で議論する．

4.2 固定データからなるパケット

はじめに，データパターンを固定して性能を調べる．この場合は圧縮スループットが高くなるため，提案手法の効果が大きく現れると予想できる．条件として以下を設定する．

- 宛先・送信元 MAC アドレスを固定する．
- Ethertype は IP として固定する．
- 宛先・送信元 IP アドレスを固定する．
- IP のペイロードを固定（すべて 0xFF）とする．

測定は，フレーム長が 64 バイトの場合と 1,518 バ

☆ 表 2 において，em というイーサネットドライバは interrupt moderation を行う高性能なドライバである．moderation のレベルは hw.em.int_throttle_ceil というカーネル変数で変更でき，値を増やすと性能が向上する．本論文ではカーネル変数のチューニングではなく，圧縮による性能向上を議論するため，標準値（8000）のまま変更せず評価を行う．この値を変更すると CPU 利用率に影響が出るので注意が必要である．

イトの場合について，ネットワーク負荷を変化させ，キャプチャレート（保存パケット数/入力パケット数）の変化を見る．同時に，100%でキャプチャ可能な最大の負荷を記録する．比較は以下の 5 通りで行った．

NC nobuf 非圧縮，パケットのバッファリングなし．これは標準の tcpdump に相当する．

NC buf 非圧縮，パケットを BPF バッファと同じサイズ（32 KB）だけバッファリングして書き込む．バッファリング単独での効果を調べる．

zlib -1 zlib の置き換え関数を用いて標準のファイルストリームを圧縮ストリームに置き換えたもの．

lzo nobuf LZO 圧縮，パケットのバッファリングなし．パケットごとに LZO の圧縮ルーチンを呼び出す実装である．

lzo buf LZO 圧縮，圧縮バッファを BPF バッファと同じサイズ（32 KB）に設定した実装で，本論文で提案する手法である☆．

組合せとしては，これらに加えて zlib に対しても圧縮バッファを実装することが考えられる．しかし，3.3 節で検討したように，zlib に別途圧縮バッファを追加するとスループットが悪化することが分かっている．このため，この組合せについては評価しない．

また，受信バッファでの欠落を示すため，NC nobuf, lzo buf, zlib のそれぞれにおける BPF 入力レート（BPF 受信パケット数/入力パケット数）を鎖線で表示した（図では BPF NC, BPF lzo, BPF zlib と表示）．

図 6 は，フレーム長が 64 バイトの場合の結果である．縦軸にキャプチャレートを，横軸に負荷トラフィックをともにパーセント表示した．負荷の 100%とはギガビットイーサネットにおける最大負荷を示す．**表 3** は，パケットの欠落を起こさずにキャプチャできる最大負荷率を示す．

表 3 から，パケット長の短い条件では，性能の向上率は大きくないことが分かる．図 6 からも，提案手法（lzo buf）のキャプチャレートが NC nobuf に比べて，低負荷域で 10%ほど向上するが，高負荷域では 1%から 2%の向上にとどまることが読み取れる．BPF 入力レートに示されるとおり，高負荷域では BPF バッファではなく受信バッファで多くのパケットが失われるため，提案手法による向上率が小さくなる．また，NC nobuf と NC buf の性能差も小さく，グラフではほとんど重なっている．ただし lzo nobuf の性能は全域で NC nobuf を下回る結果を示した．zlib を用いた場合の性能も，同様に全域で非圧縮を下回ったが，BPF 入力レートは非圧縮よりも向上している．理由については 6 章で議論する．

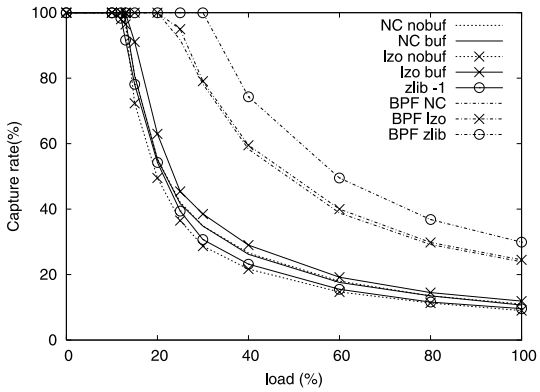

図 6 キャプチャ性能と負荷の関係：64 バイト，固定データ
Fig. 6 Capture rate and traffic load (fixed payload): 64 byte frame.

表 3 パケット欠落を生じない最大負荷率（64 バイト固定データ）
Table 3 Max load without packet drop, 64 Byte fixed.

NC nobuf	NC buf	LZO nobuf	LZO buf	zlib -1
12%	12%	11%	13%	10%

図 7 ならびに**表 4** は，フレーム長が 1518 バイトの場合の結果である．この条件では提案手法と非圧縮との差が最も大きく現れる．全体的には，非圧縮時のバッファリングの有無による差はほとんどない．提案手法では 80%負荷時に NC nobuf の 3.6 倍，100%の負荷時で 3.4 倍のキャプチャレートを示した．また，欠落せずにキャプチャ可能なネットワークの負荷が NC nobuf では 6% 程度であるのに対し，バッファリングを行った NC buf では 20%，さらに圧縮を行う提案手法では 56% となった．この結果，圧縮しない場合の BPF 受信性能を上回った．zlib を用いた場合も，非圧縮を上回る性能を示している．なお圧縮バッファを用いない lzo nobuf は lzo buf よりも 2%から 6%程度，性能が低下した．

以上より，高い圧縮スループットが得られるデータパターンでは提案手法がキャプチャ性能を大きく向上させることが分かる．

4.3 ランダムデータからなるパケット

次に，ランダムなデータパターンを設定して，圧縮スループットが低くなる条件下での性能を調べる．条件として以下を設定する．

☆ zlib の圧縮レベルは Z_BEST_SPEED（level 1）を使用している．標準の Z_DEFAULT_COMPRESSION（level 6）を使用すると，圧縮率は上がるがスループットが悪化するため，キャプチャ性能はここであげた値より劣化する．

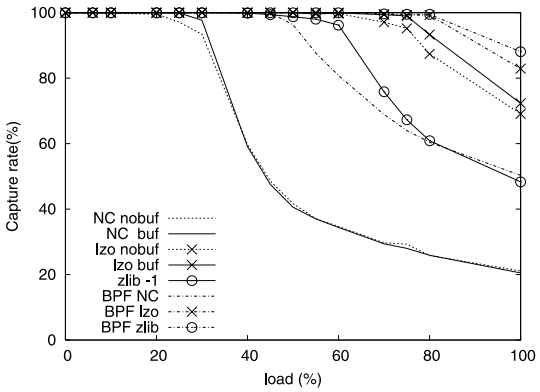

図 7 キャプチャ性能と負荷の関係：1,518 バイト，固定データ
Fig. 7 Capture rate and traffic load (fixed payload): 1,518 byte frame.

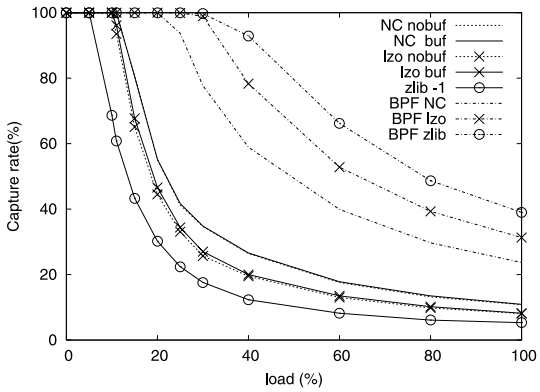

図 8 キャプチャ性能と負荷の関係：64 バイト，ランダムデータ
Fig. 8 Capture rate and traffic load (random payload): 64 byte frame.

表 4 パケット欠落を生じない最大負荷率（1,518 バイト固定データ）
Table 4 Max load without packet drop, 1,518 Byte fixed.

NC nobuf	NC buf	LZO nobuf	LZO buf	zlib -1
6%	20%	52%	56%	31%

表 5 パケット欠落を生じない最大負荷率（64 バイト，ランダムデータ）
Table 5 Max load without packet drop, 64 Byte random.

NC nobuf	NC buf	LZO nobuf	LZO buf	zlib -1
11%	11%	10%	11%	5%

- 宛先・送信元 MAC アドレスを固定する☆.
- Ethertype は IP として固定する.
- 宛先・送信元 IP アドレスはランダムに生成する.
- IP のペイロードはランダムデータとする.

イーサネットのフレーム長は 64 バイトと 1,518 バイトとして，前節と同じく 5 通りの実装を比較する．また BPF 入力レートを鎖線で示す．

図 8 ならびに**表 5** は，フレーム長が 64 バイトの場合の結果である．非圧縮でバッファリングを行う NC buf と NC nonbuf の性能差は小さく，グラフ上で両者はほとんど重なっている．

3 章で予測したとおり，提案手法のキャプチャレートは非圧縮の場合より劣化する．ただし劣化の幅は小さく，1%から 12%の範囲に収まった．一方，zlib を使った従来の実装では 6%から 37%と，大きく劣化する．また，圧縮バッファを用いない lzo nobuf では提案手法である lzo buf よりも 0.1%から 3%程度性能が低下した．これはパケット長が短く呼び出し回数が増えるため，圧縮ルーチン内のオーバヘッドが無視できなくなるためと考えられる．

図 9 ならびに**表 6** は，フレーム長が 1,518 バイトの場合の結果である．同様に，NC buf と NC nonbuf の差は小さく，0.4%から 2%程度 NC buf の方が良くなるものの，グラフ上ではほとんど重なっている．提案手法である lzo buf では，全域でキャプチャ性能が NC nonbuf より劣化するが，両者の差は 3%から 14%にとどまる．zlib を使った場合には大幅に性能が劣化した．また，圧縮バッファを用いない lzo nonbuf は lzo buf よりも性能が良く，欠落しない最大負荷に関しては NC buf とほぼ同等であった．圧縮しにくいデータについてはバッファが小さい方がスループットが向上するためと考えられる．

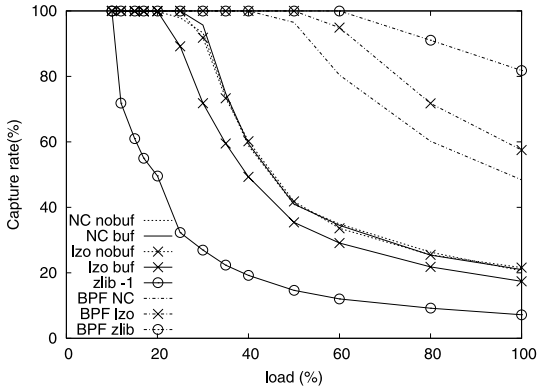

図 9 キャプチャ性能と負荷の関係：1,518 バイト，ランダムデータ
Fig. 9 Capture rate and traffic load (random payload): 1,518 byte frame.

表 6 パケット欠落を生じない最大負荷率（1,518 バイト，ランダムデータ）
Table 6 Max load without packet drop, 1,518 Byte random.

NC nobuf	NC buf	LZO nobuf	LZO buf	zlib -1
16%	25%	24%	15%	8%

☆ イーサネットスイッチを介して接続しているため，固定となる．

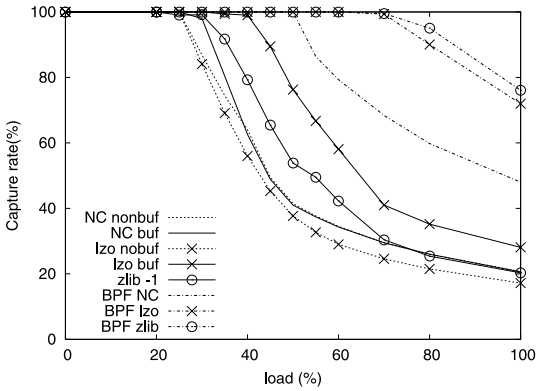

図 10 キャプチャ性能と負荷の関係：SQL 攻撃パケット
Fig. 10 Capture rate and traffic load: MS-SQL attack.

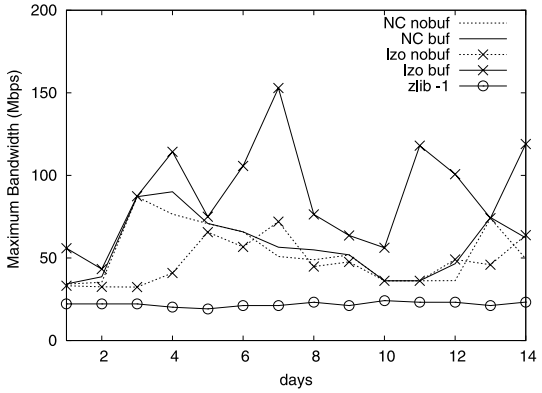

図 11 実環境におけるキャプチャ性能の変化：スキャンパケット
Fig. 11 Non-drop capture performance and traffic load: network scanning.

表 7 パケット欠落を生じない最大負荷率（SQL 攻撃パケット）
Table 7 Max load without packet drop, SQL injection.

NC nobuf	NC buf	LZO nobuf	LZO buf	zlib -1
25%	25%	25%	30%	21%

5. 実トラフィックに対する評価

前章に引き続き，本章では実環境でよく観測されるパケットを用いてキャプチャ性能を測定する．さらに，運用中のネットワークで収集したパケットトレースファイルを用いて負荷を与え，実運用環境におけるキャプチャ性能を調べる．

5.1 Slammer パケット

実際のセキュリティホール攻撃に使われるパケットの列を生成し，性能を調べる．実際の攻撃を模擬するため，送信元 IP アドレスはクラス A ネットワーク 1 つ分の領域でランダムに生成し，宛先 IP アドレスはクラス C ネットワーク 1 つ分の領域でランダムに生成した．

ここで用いたパケットは Slammer ワームが用いる UDP ポート 1434（ms-sql）への攻撃で，フレーム長は 418 バイトである．この攻撃手法は，長いパケットにより，脆弱性のある MS SQL サーバにバッファオーバフロー攻撃を試みるもので，UDP ヘッダに続いて 96 バイトにわたり同じデータが埋められている．すなわち，全体の 1/4 程度にわたって同じデータが連続する．

結果を**図 10** と**表 7** に示す．30%程度の負荷領域で，NC buf は NC nonbuf に比べ最大 12%の改善を見せたが，40%以上の負荷ではほとんど変わらない性能であった．

提案手法では圧縮の効果により，負荷率 50%時に非圧縮の 1.8 倍のキャプチャレートを達成した．また欠落なくキャプチャできる負荷は非圧縮では 25% 程度であるのに対して，提案手法では 30% まで，1.2 倍に伸びた．この例では zlib を使った場合にも性能が向上しているが，向上幅は 1%から 17%にとどまっている．一方，圧縮バッファを用いない lzo nonbuf は非圧縮よりも 3%から 8%，性能が劣化した．

5.2 実環境への適用

最後に，実際のトラフィックを生成して実環境におけるキャプチャ性能の評価を試みる．

図 11 に，運用中の定点観測装置（Class-C 相当のネットワークテレスコープ[13]）で収集したトレースファイルを用いた結果を示す．収集期間は 2005 年 5 月上旬の 2 週間である．最も多い攻撃は前節でも取り上げた MS-SQL 攻撃パケットで，全体の 2 割から 4 割を占めている．

負荷生成ツールとして tcpreplay [14] を用い，トレースファイルを PC（TG2）からレートを変化させながら送信する．個々の実装に対して，それぞれパケット欠落が生じない最大負荷を記録した．図 11 の横軸はトレースファイルの収集日（1 から 14）を，縦軸は負荷（Mbps）を示している．

提案手法（lzo buf）は非圧縮（NC nobuf）より全体に良い性能を示している．最も効果が大きかったのは 7 日目のトレースで，非圧縮に比べ 3 倍のトラフィックを与えてもパケット欠落を生じなかった．最も効果が小さかったのは 13 日目のトレースで，lzo buf と NC nobuf との差はほとんど生じなかった．また，バッファリングしない lzo nobuf は全体に非圧縮を下回る性能であった．zlib に関してはデータパターンによらず性能がほぼ変わらず，lzo nobuf をさらに下回った．

提案手法により生成されたトレースファイルの容量は，非圧縮の場合に比べて約 1/5 に抑えられた．zlib を用いた場合は約 1/6 であった．キャプチャ性能をほとんど犠牲にすることなく，トレースファイルの容量を削減できるという点は，特に長期間のキャプチャを要する応用に対して重要な貢献である．

6. 議　　論

本論文で提案した手法は，汎用の圧縮アルゴリズムを用いてデータ量を削減し，ディスク書き込みのオーバヘッドを軽減する．これにより，ディスク性能がボトルネックとなる場合に，従来よりも高負荷を与えてもパケットを欠落させずにキャプチャできる．実際に，5 章で述べたように，実環境下においてキャプチャ性能が向上し，トレースファイルのディスク使用量が大幅に削減された．

ここではいっそうの性能向上をはかるための手法について議論する．まず，評価に用いた環境において BPF バッファ容量を変化させた場合のキャプチャ性能を測定する．次に，キャプチャ装置の CPU 時間がどのように割り振られたかを観測する．

6.1 BPF バッファ容量の拡張

パケット欠落を抑えるためには，BPF バッファを増やすことが有効である[9]．本節では 5.2 節と同じ負荷を生成し，キャプチャ性能と BPF バッファ容量の関係を調べる．

前章ではトラフィックジェネレータが 1 台の構成であったが，BPF バッファを増加させることによりキャプチャ性能が向上し，ジェネレータの制御範囲を超えたため，ジェネレータを 2 台用意した（表 2 の TG2, TG3）．それぞれ宛先 MAC アドレスを同じ DUT に設定し，同じトレースファイルを用い，同じレートで送信する．TG2，TG3，DUT はギガビットイーサネットスイッチにより接続されている．測定値として，TG2 と TG3 が発生した帯域の合計値を用いる．

この構成では，イーサネットスイッチによるバッファリングの影響で，トラフィックジェネレータが 1 台の場合よりもパケット到着間隔が狭まる場合がある．このため欠落なくキャプチャできる負荷は，同じ条件でも前章より小さくなる．

図 12 は，BPF バッファサイズを 32 KB，64 KB，128 KB，256 KB と変化させたときのキャプチャ性能を示した図である．プロットは圧縮を用いるものが 5 種類，用いないものが 4 種類の合計 9 種類である．凡例における数字が圧縮バッファサイズを示している．パケット欠落を生じない最大の負荷を測定した．

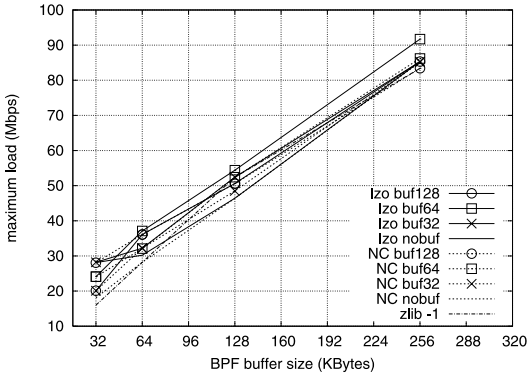

図 12 BPF バッファの変化によるキャプチャ性能の変化
Fig. 12 Capture performance and BPF buffer size.

図 12 を見ると，BPF バッファサイズの増加に応じてキャプチャ性能が向上している．提案手法は，いずれの場合においても非圧縮（NC nobuf）を上回る性能を示した．また，圧縮バッファサイズが 128 KB の場合よりも 64 KB の場合の方が良い性能を示した．これは DUT の環境における LZO のスループット特性（図 4）によるものと考えられる．また，zlib を用いた実装においても BPF バッファを増加させることで性能が向上し，非圧縮に近い性能を示すようになった．

以上より，提案手法に加えて BPF バッファサイズを増加させることにより，さらなる性能向上が見込めることが分かる．最適なバッファサイズはディスク性能や CPU 性能によって異なると考えられるが，環境に応じたチューニング手法を開発することは今後の課題である．

6.2 CPU 処理負荷のバランス

図 13 は，5.1 節で述べた MS-SQL 攻撃パケットのキャプチャの際に，iostat コマンドが報告したモード別の CPU 占有率を，負荷トラフィックに応じて並べた図である．モードは 4 種類あり，それぞれユーザ（user），システム（system），割込み（interrupt），アイドル（idle）である．非圧縮，zlib，提案手法のそれぞれについて，25%，40%，80%の 3 種類の負荷を与えたときの状態を表示した．カッコ内には，各負荷条件におけるキャプチャレートを付記した．また，それぞれの場合におけるディスク負荷を**表 8** に示す．表示の値は，systat コマンドが報告したディスクのビジー率である．

欠落が起こらない程度の負荷（25%）では CPU 時間にアイドル（図 13 の黒い領域）が含まれ，CPU に余裕が残っている．負荷の増加にともない，割込み（interrupt）が大半を占めるようになる．割込みの内訳は，パケットの受信と，ディスクの書き込みである．

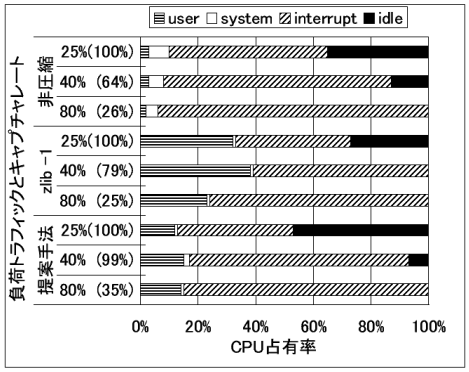

図 13 iostat による CPU 占有率と負荷トラフィックの関係
Fig. 13 CPU time and traffic load.

表 8 負荷トラフィックとディスク負荷の関係
Table 8 Average load of network traffic and disk I/O.

負荷トラフィック	ディスク負荷		
	非圧縮	提案手法	zlib -1
25%	100%	6%	4%
40%	100%	11%	6%
80%	100%	9%	5%

負荷が 40%になると，非圧縮では BPF バッファでの欠落が発生する．しかし，CPU には余裕がある．表 8 から分かるように，この場合ボトルネックがディスク性能にある．提案手法は，CPU の余裕の範囲内で圧縮操作を行い，99%のキャプチャレートを得た．ディスク負荷は約 1/10 に削減されている．一方，zlib を用いた場合には CPU の余裕がない．キャプチャレートも 79%にとどまっている．

負荷が 80%の場合には，BPF バッファに加え，受信バッファでの欠落が発生する．提案手法や zlib を使うと，ディスクからの割込み数が減少し，その分の CPU 時間はパケット受信割込みの処理に振り向けられるため，受信バッファでの欠落が減少する．しかし，ユーザモードの占有率は負荷が 40%のときに比べて減少しており，圧縮を行う CPU 時間が少なくなる．このため，受信バッファでの欠落が発生する高負荷領域では，提案手法による性能向上幅は小さくなる．

7. おわりに

本論文では，固定長のバッファを用意し，リアルタイムでパケットを圧縮することで，従来よりパケットキャプチャ性能を向上させる手法を提案した．従来の実装では，圧縮のスループットを考慮していないため，圧縮をするとキャプチャ性能が悪化する．提案手法では圧縮バッファを用いてオーバヘッドを削減し，高速なアルゴリズムを用いてスループットを重視する実装を行った．ワームが攻撃に用いるパケットのキャプチャにおいて，非圧縮の場合に較べ最大 1.8 倍のキャプチャレートを得た．また，種々の攻撃パケットを保存した場合のトレースファイルの容量は圧縮前の約 1/5 となった．

ネットワークにおける攻撃の脅威が増すにつれて，パケットキャプチャシステムの役割はますます重要になっている．攻撃の解析のためには，多地点で，長期間にわたり，欠落なくパケットをキャプチャする必要がある．本論文で述べた方法を適用することで，特別なハードウェアを使うことなく，既存のパケットキャプチャシステムを容易に高度化することができる．キャプチャ性能の高速化に加え，トレースファイル容量が縮小するため，長期間にわたるパケットキャプチャを行う場合に必要なハードディスク容量を削減できる．

さらなる性能向上のためには，圧縮アルゴリズムを改良するアプローチが考えられる．特に，圧縮がしにくい場合でもスループットの低下が小さくなるようなアルゴリズムを選ぶことが重要である．提案手法では一度に圧縮するデータサイズを調整することでキャプチャ性能の低下を抑えたが，圧縮がしにくい場合には非圧縮に自動的に切り替えるなどの改良により，キャプチャ性能を向上させられる可能性が残されている．今後は，この改良技法の検討を進めると同時に，より多くのプラットフォームで実験を行い，CPU 性能やネットワーク性能，ストレージ性能の違いにより圧縮が有効な範囲がどのように変化するかを評価する．

参 考 文 献

1) The Honeynet Project: *Know Your Enemy*, Addison-Wesley (2004).
2) The tcpdump group: tcpdump/libpcap. http://www.tcpdump.org/
3) Combs, Gerald: Ethereal. http://www.ethereal.com/
4) Peuhkuri, M.: A Method to Compress and Anonymize Packet Traces, *Proc. 1st ACM SIGCOMM Workshop on Internet Measurement* (*IMW '01*) (2001).
5) Gailly, J., et al.: zlib. http://www.gzip.org/zlib/
6) Endace measurement systems: DAG Network Monitoring Interface Cards. http://www.endace.com/
7) Keys, K., et al.: The architecture of CoralReef: An Internet traffic monitoring software suite, *Proc. workshop on Passive and Active Measurements* (*PAM'01*) (2001).
8) McCanne, S. and Jacobson, V.: The BSD

packet filter: A new architecture for userlevel packet capture, *Proc. Winter 1993 USENIX Conference* (1993).
9) Agarwal, D., et al.: An Infrastructure for Passive Network Monitoring of Application Data Streams, *Proc. workshop on Passive and Active Measurements* (*PAM'03*) (2003).
10) Oberhumer, M.F.X.J.: LZO compression library (2002). http://www.oberhumer.com/opensouce/lzo/
11) Ziv, J. and Lempel, A.: A Universal Algorithm for Sequential Data Compression, *IEEE Trans. Inf. Theory*, Vol.23, No.3, pp.337–343 (1977).
12) Norcott, W.D.: Iozone Filesystem Benchmark. http://www.iozone.org/
13) Moore, D., et al.: Network Telescopes: Technical Report CAIDA TR-2004-04, Univ. of California, San Diego (2004).
14) Turner, A.: Tcpreplay: Pcap editing and replay tools for *NIX. http://tcpreplay.synfin.net/

(平成 17 年 10 月 3 日受付)
(平成 18 年 2 月 17 日採録)

清水 奨（正会員）

平成 4 年東京大学工学部機械情報工学科卒業．同年日本電信電話（株）入社．高速レイヤ 2 プロトコル，インターネット計測の研究開発に従事．平成 15 年より早稲田大学大学院理工学研究科情報・ネットワーク専攻博士後期課程在学中．ACM，IEEE Communications Society，USENIX 各会員．

風間 一洋（正会員）

昭和 63 年京都大学大学院工学研究科精密工学専攻修士課程修了．同年，日本電信電話（株）入社．現在 NTT 未来ねっと研究所主任研究員．博士（情報学）．分散協調処理，情報検索の研究に従事．日本ソフトウェア科学会，ACM 各会員．

廣津登志夫（正会員）

平成 7 年慶應義塾大学大学院理工学研究科計算機科学専攻博士課程修了．同年日本電信電話（株）入社．平成 16 年より豊橋技術科学大学情報工学系．博士（工学）．分散システム，OS，ネットワーク，ユビキタスシステム等の研究に従事．日本ソフトウェア科学会，ACM，IEEE Computer Society 各会員．

後藤 滋樹（フェロー）

昭和 48 年東京大学大学院理学系研究科修士課程修了．同年日本電信電話公社武蔵野電気通信研究所入所．平成 8 年より，早稲田大学理工学部教授．工学博士．これまでコンピュータアーキテクチャ，自然言語処理，プログラム合成，論理プログラミング，コンピュータネットワークの研究に従事．IEEE，ACM，ISOC，電子情報通信学会，人工知能学会，日本ソフトウェア科学会，応用数理学会各会員．