TCP トラフィックの能動的な解析

西　田　佳　史 †

TCP の実装の品質を検証するためには、様々なトラフィックパターンを与えて、その挙動を観測する能動的なトラフィック解析が有効である。本稿では、効率的な能動的なトラフィック解析を実現するツールを用いて、複数のオペレーティングシステムに対して、RFC2581 の検証テストを実施した結果を示す。

Analyzing TCP Implementation with Active Traffic Analysis

Yoshifumi Nishida †

To investigate the quality of a TCP implementation, active traffic analysis in which various TCP traffic patterns are tested on a target host to generate responses that can be analyzed has proved to be an effective method. In this paper, we present an analysis tool: tcpprobe developed for active traffic analysis and examine various TCP implementations to test conformance with TCP standards using this tool. Our results show that some TCP implementations have misimplemented congestion control algorithms.

1. はじめに

TCP の輻輳制御機構に関する研究は現在も活発に行われており、様々な改善策が提案され、標準化に向けた議論が行われている[1)2)3)4)5)]。しかし、このような細かな TCP の改善により、TCP の輻輳制御アルゴリズムはより複雑になっている。

このようなことから、新しい機能が実際に有効に機能していることや、TCP の実装が正しく標準に準拠していることを検証する技術の重要性が高まっている。TCP の実装の挙動を解析するためには、TCP を用いて実際に通信を行い、そのトラフィックを解析する手法が有効である。このような背景から、本研究ではトラフィック解析により TCP の挙動を詳細に解析するツールの開発を行った。開発したツールは、能動的なトラフィック解析を行うことにより、一般的に利用される受動的なトラフィック解析よりも効率的に解析作業を行うことができる。本研究では、ツールの有効性を検証するために、RFC2581 の適合テストを実施し、複数のオペレーティングシステムにおける RFC2581 の実装状況の解析を行った。

† ソニーコンピュータサイエンス研究所
Sony Computer Science Laboratories, Inc

2. トラフィック解析による TCP の挙動解析

トラフィック解析によって TCP の挙動を解析を行うには、まず TCP が検証対象となる機能を実行する状況になる必要がある。

例えば RFC2581 で規定されている TCP の高速再送、高速リカバリアルゴリズム[7)] の機能を検証するためには、データ送信側の TCP が 3 つ以上の重複確認応答を受信した後の挙動を解析しなければならない。TCP がこのような状況になるためには、少なくとも TCP のウインドウサイズが 4 セグメント以上の状況で、ウインドウ中に 1 パケット以上のパケットの喪失が生じる必要がある。(この条件を満たす場合でも、必ずしも 3 つ以上の重複確認は生成されない)

RFC2582 で規定される NewReno と呼ばれる TCP の新しい輻輳制御アルゴリズムの検証にはさらに複雑な状況が必要となる。NewReno の機能を検証するためには、データ送信側の TCP は、3 つ以上の重複確認応答を受信し、その直後に部分確認応答 (Partial Acknowledgement) を受信する必要がある。このような状況が生じるためには、少なくとも TCP のウインドウサイズが 5 セグメント以上の状況で、ウインドウ中に 2 パケット以上のパケットの喪失が生じる必要がある。(この条件を満たす場合でも、必ずしも 3 つ以上の重複確認と部分確認応答は生成されない。)

このような TCP のトラフィックの解析の手法には、受動的な手法と能動的な手法がある。以下に２つの方法の特徴について説明する。

2.1 受動的なトラフィック解析

受動的なトラフィック解析では、まずネットワーク上の適当な箇所にトラフィック解析ポイントを設ける。そして、トラフィック解析ポイント上で tcpdump[8] や etherreal[9] の様なツールを利用して、ネットワークを通過するトラフィックを定期的に収集する。その後ツールを用いて得られたトラフィックログを分析することにより、TCP の挙動を解析する。この様な受動的な解析を行うツールには、tcpanaly[13] などがある。

この手法は、解析者の期待するトラフィックパターンが現れた際に、TCP の挙動を解析を行う。このような解析手法の問題点は、解析者の期待するトラフィックパターンが発生している点をトラフィックログの中から検出する必要があることである。トラフィックログが大量にある場合、この作業は容易ではない。また、トラフィックログ中に期待するトラフィックパターンが存在しない場合もある。このような場合トラフィック解析は失敗する。

2.2 能動的なトラフィック解析

能動的なトラフィック解析は、解析者が解析用のトラフィックパターンを生成し、トラフィックに対する TCP の挙動を観測する手法である。この手法は解析アプリケーションが、実際には到着しているパケットに対する確認応答を返送することなどで実現することができる。能動的なトラフィック解析の利用例には以下の様なものがある。

NMAP[11] は、オペレーティングシステムの識別を行うために能動的なトラフィック解析を行うツールである。NMAP には、ターゲットとなるホストに対し解析用のパケットを転送し、ターゲットホストから返送されたパケットを解析することにより、ターゲットホストのオペレーティングシステムを識別する。このような機能を実現するために、NMAP はターゲットノードから転送された TCP パケットの初期シーケンス番号や、告知ウインドウサイズや最大セグメント長の情報を解析する。

Savage は既存の実装に変更を加えた TCP daytona という実装を利用して、TCP 実装の欠陥を検査した[15]。TCP daytona は、受信していないパケットに対する確認応答の転送 (Optimistic ACKing) や、ねつ造した重複確認応答パケットの転送 (DupACK spoofing) などを行い、ターゲットから返送されたパケットから TCP の実装上の問題を検出する。

Padhye はインターネット上の Web サーバにおける新しい実装や TCP オプションのサポートの比率を計測するために、TBit[12] というツールを開発した。TBit ではターゲットホストの TCP の実装の選択確認応答オプションの対応状況や輻輳制御アルゴリズムのバージョンの推測を行うことができる。

この様に、能動的なトラフィック解析を行うツールは幾つか存在する。しかしながら、これらのツールを用いて本研究の目的である TCP の実装の詳細な解析へ応用することには困難がある。

TCP Daytona は、オペレーティングシステムのカーネル内に変更を加えている。従って、複数の種類のトラフィック解析を行う際に、多くの場合複数のカーネルを用意する必要が生じる。また、解析パケットの転送パターンに変更を加えることは非常に負荷が大きい。

これに対して、TBit や NMAP はユーザレベルのアプリケーションとして実現されている。ユーザアプリケーションの場合、プログラム起動時のオプションなどを利用することにより、複数の解析を１つのツールで行うことを容易に実現できる。しかしながら、これらのツールの検証用のパケット転送のアルゴリズムは、ソースコード内にハードコードされている。このため、解析パケットの転送パターンの変更は柔軟にはできない。

また、tbit が HTTP サーバに対してリクエストを転送し、サーバから返送されたデータ (Web ページの内容) を解析する手法を採用している。この手法の問題点は、サーバが返送するデータの量は、tbit が指定した URL に存在するデータ量に依存してしまうことである。もし、指定した URL に十分なデータ量が存在しない場合、tbit による検査は失敗する。さらに、tbit のこのような手法は、検査対象となるホストで、予め HTTP サーバを起動しておく必要がある。このような制約は、PDA のようなサーバアプリケーションを起動することを想定していないプラットフォームに適していない。

このようなことから本研究では、より柔軟かつ簡易に解析パケットの転送パターンを設定できるツール: tcpprobe の開発を行った。tcpprobe はユーザレベルアプリケーションとして実装されており、様々なプラットフォーム上に移植可能に設計されている。また tcpprobe の解析パケットの転送パターンの設定は TCL 言語を用いて記述することができる。このためある条件に達するまで一定のパターンでパケットを転送するなどの複雑なパケット転送のシナリオを柔軟に設定することができる。

表1に、tcpprobeによるトラフィック解析アルゴリズムの記述の一例を示す。この例の一行目のコマンドは、シーケンス番号が 1000-5000 の範囲のデータを含むパケットの受信を 200msec 間待機する。$filter_match$ は、直前の expect コマンドの成否を示す変数であり、expect コマンドが成功した場合は、1 の値がセットされる。従ってこの例の場合、期待するパケットを受信できた場合は 3,4 行目のコマンドが、受信できなかった場合は、6 行目のコマンドが実行される。この例では、期待するパケットを受信した場合には、これまでに受信したシーケンス番号の最大値を確認応答として返送し、受信しなかった場合は reset パケットを転送し強制的に解析を終了する処理を行う。

トラフィック解析のためのアルゴリズムは、この例に示した tcpprobe の基本コマンドを利用することにより、ほとんどのケースを記述することができる。また、作成した解析アルゴリズムの記述を tcl のパッケージとして配布することによってツールの機能をさらに拡張していくことができる。

```
expect data 1000-5000 wait 200 msec
if {$filter_match == 1} {
   set ackno [expr $max_seq +1]
   send ack $ackno
} else {
   send reset
}
```

図 1 トラフィック解析アルゴリズムの記述例

また、tcpprobe は HTTP サーバの様に動作し、検査対象からの HTTP リクエストを受信するとデータを返送する。このデータには、検査用のデータと、指定されたアドレスに POST method で検査用データを送り返すリクエストが記述されている。この手法により、tcpprobe が検査データの量を指定することができ、検査に必要なデータ量を検査対象から送信させることが可能になる。また、この手法では、検査対象となるホストで必要となるアプリケーションは HTTP ブラウザのみであり、特別なアプリケーションを一切必要としない。

3. RFC2581 適合テスト

3.1 高速再送、高速リカバリアルゴリズム

本研究ではこのツールを利用して RFC2581 の機能のうち高速再送、高速リカバリアルゴリズムに着目し、この機能の適合テストを実施した。高速再送、高速アルゴリズムの概要は以下の手順で機能する。

(1) 3つ目の重複確認応答を受信すると、スロースタート閾値 (ssthresh) を減少させる。
(2) 喪失セグメントを再送する。輻輳ウインドウ (cwnd) の値を ssthresh + 送信側 TCP の最大セグメント長 (SMSS) ×3 に設定する。
(3) 重複確認を受信する度に cwnd を SMSS 分増加させる。
(4) 新しい cwnd の値に応じてパケットを転送する。
(5) 新しいデータの受信を確認する確認応答パケットを受信すると、cwnd の値を ssthresh に設定する。

これらの手順のうち 手順 1 と手順 5 については簡易に実装可能であるが、手順 2,3,4 の実装はやや複雑で注意が必要となる。手順 2,3,4 は見落とされがちなアルゴリズムであるが、パケットロスが生じた後に TCP のパイプサイズを、パケットロスが生じる前の 50%に減少させるという重要な役割を果たしている[14)]。手順 2,3,4 の実装が不正確でも、手順 5 を実行すれば影響は少ないと考えるのは正しくない。手順 2,3,4 を実行する期間に転送されたパケットにより、受信側の TCP は確認応答を生成する。この確認応答は手順 5 以降の輻輳ウインドウの値に影響を与える。手順 2,3,4 の段階で規定より多くのパケットを転送するアルゴリズムを実装している場合、以後の輻輳ウインドウは標準を超えて増加していく。手順 2,3,4 の段階で送出するパケット数が十分に多くなると、輻輳回避アルゴリズムが機能しないことがある。この様な実装が普及すると輻輳崩壊の危険性がある。

3.2 検 証 方 法

図 2 に標準的な高速再送、高速リカバリアルゴリズムの例を示す。

図 2 において、左の垂線は送信側の TCP の挙動を示し、右の垂線は受信側の TCP の挙動を示している。2つの垂線は時間軸を表しており、上から下に向かって時間が経過している。左から右に向かう矢印は、送信側の TCP から受信側の TCP に送られたデータパケットであり、右から左に向かうパケットは受信側の TCP から返送される確認応答パケットである。図 2 は、輻輳ウインドウの値が 8 セグメントの際に 1 つのパケットが喪失した状況を示している。この場合、A の期間において TCP は 8 つのパケットを送出する。そして次の B の期間において、3 つの重複確認応答を受信後にパケットの再送を行い、その後さらに到着

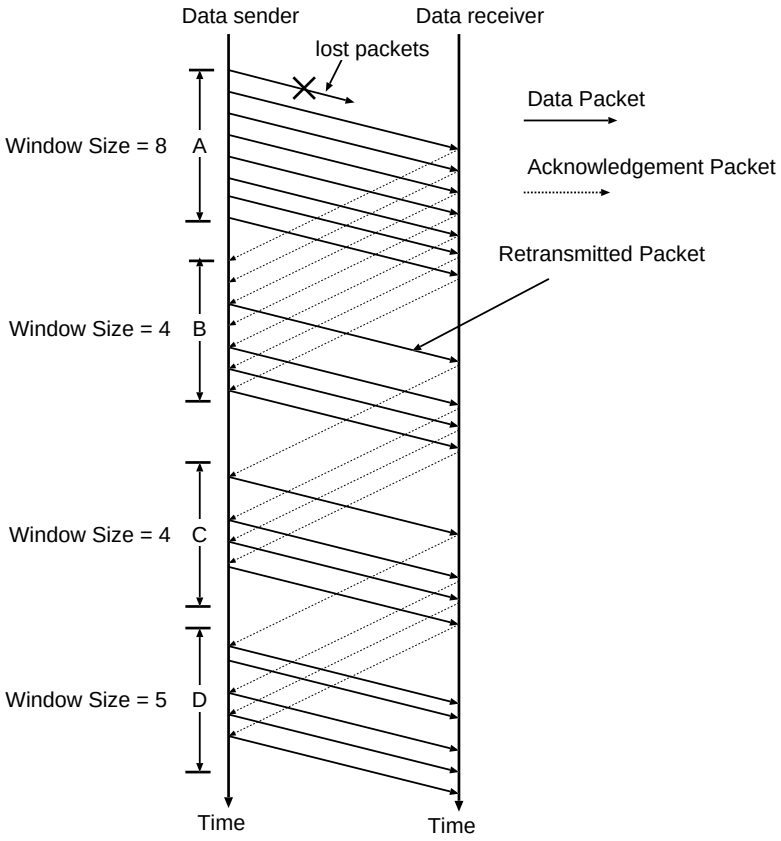

図 2 standard rfc2581

した重複確認応答から新たに 3 つのパケットを転送する。従って、B の期間では合計で 4 つのパケットが転送される。C の期間では、高速再送アルゴリズムの成功により、TCP は新しいデータの受信を確認する確認応答パケットを受信する。最初の確認応答の受信により、TCP のウインドウサイズはパケット喪失を検出する前の値の 50%である 4 セグメントに設定され、新しいセグメントを 1 つ送出する。この後、TCP は輻輳回避段階になり 3 つの確認応答を受信する。そしてこの結果、3 つのセグメントが送出される。従って C の期間でも合計で 4 セグメントが転送される。D の期間では、確認応答により $MSS/cwnd$ づつ輻輳ウインドウの値が増加されるため、合計で 5 セグメントが生成される。

このような TCP 挙動のうち、高速再送、高速リカバリのアルゴリズムとして主要なポイントは、以下のものである。

(1) 3 つ目の重複確認応答を受信した時点でパケット再送を行っている
(2) 期間 B で期間 A の 50%のパケットを転送している
(3) 期間 C で転送するセグメント数は、期間 B と同一である
(4) 期間 D で転送するセグメント数は、期間 C より 1 セグメント多い

1 のポイントは、高速再送アルゴリズムの起動のトリガが正しく実装されていることを検証できる。2 のポイントは、TCP が高速リカバリアルゴリズムにより、適切にウインドウサイズを減少させていることを検証できる。3,4 のポイントは高速リカバリアルゴリズムの後に TCP が輻輳回避段階に入り、正しくウインドウサイズを増加させていることを検証できる。

3.3 検 証 結 果

上記の様な検証を行うために、tcpprobe と検証対象となる実装の間でデータ転送を行い、TCP 実装のデータ転送のパターンを解析した。tcpprobe の転送パターンには以下の転送アルゴリズムが設定されている。また、今回の検証では tcpprobe から転送される確認応答は遅延確認応答アルゴリズムを用いずに、データセグメントの到着毎に転送するように設定した。

(1) ウインドウサイズが一定値に達するまで通常のTCP と同様に確認応答を生成する
(2) ウインドウサイズが一定値に達した後は、重複確認応答を (ウインドウサイズ - 1) セグメントだけ生成する。
(3) 重複確認応答に対する再送パケットを受信した後は、通常の TCP と同様に確認応答を生成する。

検証は複数のオペレーティングシステムを対象に行った。データ転送は各実装のデフォルトの最大セグメント長を使用した。表 1 に実装を検証したオペレーティングシステムとそのバージョン番号および、各実装のデフォルトの最大セグメント長を示す。RFC1122 では、デフォルトの最大セグメント長として 536 バイトの利用が要求されている。

表 3.3 に RFC2581 検証テストの結果を示す。これは TCP のウインドウサイズの値が 4,8,16 の値をとる場合における検証結果を示している。それぞれのテーブルにおいて、dupack threshold は高速再送アルゴリズムによってセグメントの再送が行われるまでに、連続して受信した重複確認応答セグメント数を示している。RFC2581 では、３つの連続した重複確認応答を受信した後に、セグメントの再送を行うことが規定されている。

期間 B, 期間 C, 期間 D は、3.2 節で述べた TCP の高速リカバリアルゴリズムから TCP が輻輳回避段階へと移行する期間を期間を示し、表中の数値はこれらの期間の間に転送されたセグメント数や受信した確認応答セグメント数を示している。表中の数値は、'転送セグメント数 (再送セグメント数) / 受信確認応答セグメント数' のフォーマットで示されている。この数値を検証することにより、高速リカバリアルゴリズムの挙動を検証することができる。

3.3.1 windows 98 の検証結果

検証結果より、Windows98 の TCP の実装にはいくつかの問題があることがわかる。Windows98 の TCP 実装では、２つ目の重複確認応答を受信した時点で、高速再送アルゴリズムによりセグメント再送が行われている。これは RFC2581 で規定された値よりも小さな値である。高速再送アルゴリズムのトリガとなる重複確認応答セグメント数に小さな値を設定すると、パケットの送達順序の違いやパケットの複製などによってアルゴリズムが誤作動する可能性が高くなる。

また Windows98 では重複確認応答を受信する毎に、必ずセグメントの転送を行っている。検証結果からは、Windows98 の実装は重複確認応答を受信すると、無条件に輻輳ウインドウを１セグメント増加させるアルゴリズムを採用しているものと推測される。このため期間 B における転送パケット数が受信した重複確認セグメント数と一致している。RFC2581 では期間 B で転送されるセグメント数は、パケット喪失が起こる前から 50%減少させることが要求されている。これに対し Windows98 の場合、期間 B において転送されるセグメント数は、セグメント喪失以前より１セグメント分減少しているだけである。これは、輻輳検出時のデータ転送速度を減少させる高速リカバリアルゴリズムが正しく実装されていないことを示している。このような実装は輻輳崩壊の原因となる可能性がある。

さらに、Windows98 では高速再送アルゴリズムにより転送されるセグメントの大きさが、デフォルトの 536 バイトより小さい 524 バイトとなっている。このため、例え高速再送アルゴリズムによるセグメントの再送が成功しても、12 バイトのデータはさらなる再送が必要となっている。この結果として、受信側の TCP から重複確認応答がさらに生成され、輻輳ウインドウの増加と共に期間 C で冗長な再送が行われ、期間 D でのウインドウサイズがセグメント喪失前の２倍近くなっている。

3.3.2 Windows 2000 の検証結果

Windows2000 の TCP の実装は、Windows98 の実装といくつかの類似点を持っている。Windows2000 の実装でも、２つ目の重複確認応答を受信した時点で、高速再送アルゴリズムによりセグメント再送が行われている。また 重複確認応答を受信する毎に、必ずセグメントの転送を行っている。検証結果からは、Windows2000 の実装は Winsdows98 の実装と同様、重複確認応答を受信すると、無条件に輻輳ウインドウを１セグメント増加させるアルゴリズムを採用しているものと推測される。

Windosw2000 と Windows98 の実装との違いは、高速再送アルゴリズムによって転送されるセグメントサイズが、再送前のセグメントサイズと同じものになっている点である。このため、Windows98 に見られる期間 C の冗長な再送が Windows2000 では見られない。したがって高速再送アルゴリズムによるセグメント再送の成功後は、受信側の TCP からは重複確認応答は生成されない。このことが期間 C, 期間 D の送出セグメント数の減少に関与しているものと思われる。Windows2000 の期間 C, 期間 D の送出セグメント数は、ほぼ RFC2581 に準拠している。

表 1 検証したオペレーティングシステム

オペレーティングシステム	バージョン番号	デフォルト最大セグメント長
FreeBSD	4.2	512
Linux	2.2.16 (Redhat7.1)	536
Windows 98	Second Edition 4.10.22222 A	536
Windows 2000	5.00.2195	536

表 2 検証結果

オペレーティングシステム	dupack threshold	期間 B	期間 C	期間 D
RFC2581	3	2(1)/3	2(0)/2	3(0)/2
Windows 98	2	3(1)/3	3(0)/3	6(0)/3
Windows 2000	2	3(1)/3	2(0)/3	3(0)/2
Linux	3	1(1)/3	5(0)/1	5(0)/5
FreeBSD	3	1(1)/3	3(0)/1	3(0)/3

ウインドウサイズ=4

オペレーティングシステム	dupack threshold	期間 B	期間 C	期間 D
RFC2581	3	4(1)/7	4	5
Windows 98	2	7(1)/7	7	12
Windows 2000	2	7(1)/7	5(0)/7	6(0)/5
Linux	3	3(1)/7	5(0)/3	6(0)/5
FreeBSD	3	4(1)/7	5(0)/4	6(0)/5

ウインドウサイズ=8

オペレーティングシステム	dupack threshold	期間 B	期間 C	期間 D
RFC2581	3	8	8	9
Windows 98	2	15	16	30
Windows 2000	2	15	9	10
Linux	3	7	9	10
FreeBSD	3	8	9	10

ウインドウサイズ=16

3.3.3 Linux の検証結果

Linux の実装は概ね RFC2581 に準拠しているが、いくつかの問題点を含んでいる。Linux の実装では、ウインドウサイズが 4 セグメントの場合、期間 C, 期間 D において 5 セグメントを転送している。これは、linux の高速リカバリアルゴリズムの輻輳ウインドウの制御に問題があることに起因している。RFC2581 では、高再送アルゴリズムによりセグメントの再送を行う際に、一時的にウインドウサイズを増加させる。そして、新しいセグメントの到達を確認する確認応答セグメントを受信した際に、一時的に増加させたウインドウサイズを減少させる。Linux の実装では、この増加させたウインドウサイズを減少させるために、clear_fast_retransmit() という関数を呼び出す。しかしながら、clear_fast_retransmit() では、重複確認応答数が 4 セグメント以上という条件を満たす場合のみ、ウインドウサイズが減少させられる。ウインドウサイズが 4 の場合は、TCP が受信する重複確認応答数は 3 セグメントになる。このため、Linux の実装ではウインドウサイズが 4 セグメントの場合、ウインドウサイズの減少が機能していない。

また Linux の実装では、RFC2581 と比較して期間 B における転送セグメント数が少ない。これは Linux の実装の FlightSize の計算手法に起因している。FlightSize は TCP が転送したセグメントのうち、受信側への到達が確認されていないデータ量を表している。Linux の実装では、高速再送アルゴリズムにより再送されたセグメントもこの FlightSize に加算されている。そして、セグメントの転送は、ウインドウサイズが FlightSize を超える場合にのみ行われる。このような Linux の転送制御の実装により、期間 B における転送セグメント数は標準より少なくなっている。この FlightSize の計算手法は、一見保守的なアプローチに見えるが、この解釈は正しくない。この計算手法を用いる場合、期間 B における転送セグメント数は減少する。しかし、その後新しいセグメントの到達を確認する確認応答セグメントを受信した時点で、FlightSize は、再送セグメント分の大きさ分減少させられる。この結果と

して、期間 C での転送セグメントが増加することになる。したがって Linux 実装では、期間 B、期間 C の転送セグメント数の合計は、RFC2581 と一致しているものの、期間 C での転送セグメント数が大きくなる。このことは、期間 C における転送のバースト性が高くなることを示している。

Linux 実装のこのアルゴリズムを修正するには、tcp_packets_in_flight() 関数に変更を加える必要がある。

3.3.4 FreeBSD の検証結果

FreeBSD の実装はほぼ RFC2581 に準拠しており、大きな問題はない。FreeBSD の実装ではウインドウサイズが 4 セグメントの場合、期間 B での転送セグメントが 1 セグメントとなっており、RFC2581 の値よりも少ない。これは、FreeBSD の実装が高速再送アルゴリズムによるセグメント再送を行った後に、次に確認応答セグメントを受信するまでセグメントの転送を行わない事に起因している。しかしながら、RFC2581 では、高速再送アルゴリズムによる再送を行った後に、ウインドウサイズを 3 セグメント分増加させ、ウインドウサイズが許容する範囲で新しいセグメントを送出することが要求されている。高速再送アルゴリズムによる再送を行った後に、新しいセグメントを送出できる状況は、ウインドウサイズが $cwnd/2+3MSS >= cwnd+1MSS$ という条件を満たす場合、すなわちウインドウサイズが 4 セグメントの場合に生じる。

また、FreeBSD の実装では高速再送アルゴリズムによるセグメント再送後に、新しいセグメントの到達を確認する確認応答セグメントを受信した場合、必ず 2 セグメント以上の転送を行っている。これは、FreeBSD の実装ではこの確認応答セグメントの受信時に、ウインドウサイズの値をパケット喪失が生じる前のウインドウサイズの 50%に設定した後、再度スロースタートアルゴリズムによって 1MSS 分のウインドウサイズの加算が行われる事に起因している。従って FreeBSD の実装では、高速リカバリアルゴリズムが終了した直後のウインドウサイズは、喪失前のウインドウサイズの 50% + 1MSS となる。

4. 結論および今後の課題

本稿では、能動的なトラフィック解析を実現する tcpprobe を紹介した。そしてツールの有効性を検証するために、RFC2581 の適合テストを実施し、複数のオペレーティングシステムにおける RFC2581 の実装状況の解析を行った。そしていくつかのオペレーティングシステム上で、問題となりうる可能性のある実装上の誤りを検出することができた。十分に広まっている実装においても、この様な誤りが存在することは、TCP の実装が複雑であり、かつ実装の検証が困難であることを示している。従って、本研究で示した実装検証技術への需要は高いものがあると思われる。

今後は、SACK や NewReno などの実装が広まりつつある輻輳制御状況の検証シナリオの開発を進めていく。また、インターネット上に実装の検証を行うシステムを構築し、利用者が自由に実装の検証を行う環境を提供していくことを検討する。

参 考 文 献

1) M. Allman, “TCP Congestion Control with Appropriate Byte Counting”, draft-allman-tcp-abc-01.txt, March 2001.
2) M. Allman, H. Balakrishnan, S. Floyd, “Enhancing TCP’s Loss Recovery Using Limited Transmit”, RFC3042, January 2001.
3) K. K. Ramakrishnan, S. Floyd, D. Black, “The Addition of Explicit Congestion Notification (ECN) to IP” draft-ietf-tsvwg-ecn-03.txt, March, 2001.
4) M. Mathis, J. Semke, J. Mahdavi, K. Lahey, “The Rate-Halving Algorithm for TCP Congestion Control”, Draft, June, 1999.
5) M. Mathis, J. Mahdavi, “Forward Acknowledgment: Refining TCP Congestion Control,” Proceedings of SIGCOMM’96, August 1996.
6) S. Floyd, T. Henderson, “The NewReno Modification to TCP’s Fast Recovery Algorithm” RFC2582, April 1999.
7) M. Allman, V. Paxson, W. Stevens, “TCP Congestion Control,” RFC2581, April 1999.
8) http://www.tcpdump.org/
9) http://ethereal.zing.org/
10) http://jarok.cs.ohiou.edu/software/tcptrace/
11) http://www.insecure.org/nmap
12) J. Padhye, S. Floyd, “Identifying the TCP Behavior of Web Servers,” International Computer Science Institute Technical Report 01-002. February 2001. http://www.aciri.org/tbit/
13) V. Paxson, “Automated Packet Trace Analysis of TCP Implementations,” ACM SIGCOMM ’97, September 1997.
14) V. Jacobson, “Modified TCP Congestion Avoidance Algorithm”, end2end-interest Mailing List, Apr. 1990 ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt
15) S. Savage, N. Cardwell, D. Wetherall, T. Anderson, “TCP Congestion Control with a Misbehaving Receiver,” ACM Computer Communications Review, October, 1999.