広域ネットワークにおける伝送制御機構の検証に関する研究

東京工業大学大学院
情報理工学研究科
本庄 利守
(96M37240)

1997年度修士論文

指導教官 大野 浩之 講師

1998年 1月 30日

目 次

1 はじめに **5**

2 伝送制御機構の検証 **7**
2.1 伝送制御機構 7
2.2 関連研究 8
2.3 伝送制御機構の検証システム 10

3 インターネットのエミュレーション **13**
3.1 インターネットのモデル化 13
3.1.1 伝送路のモデル化 13
3.1.2 トラフィックのモデル化 15
3.1.3 まとめ 15
3.2 モデルの解析 16
3.3 理論値と実測値の比較検討 18
3.4 まとめ 19

4 PEERシステムの設計と実装 **20**
4.1 PEERシステムの特徴 20
4.2 PEERシステムの構成 20
4.3 動作原理 21
4.4 ネットワーク部 22
4.5 制御部 23
4.6 情報収集部 25
4.7 実装に使用したオペレーティングシステムとツール 27
4.8 PEERシステムの性能評価 27
4.9 実測結果と理論値の比較 30
4.9.1 系内時間の測定 30
4.9.2 実測結果と理論値の比較 31
4.10 考察 32
4.10.1 実トラフィックの影響 33
4.10.2 伝送制御機構の検証結果の比較方法 33
4.10.3 ユーザインターフェイス 33
4.10.4 機能の拡張 33

5 PEERシステムを用いたTCPの検証 35
5.1 PEERシステムによる検証 . . . 35
5.2 TCPの検証 . . . 35
5.3 関連研究 . . . 36
5.4 検証項目 . . . 37
5.5 実験に使用した機材とツール . . . 37
5.6 Retransmission Timeout (RTO) . . . 38
5.6.1 検証内容 . . . 38
5.6.2 検証方法 . . . 38
5.6.3 検証結果 . . . 38
5.7 Retain Above-Sequence Data . . . 40
5.7.1 検証内容 . . . 40
5.7.2 検証方法 . . . 41
5.7.3 検証結果 . . . 41
5.8 Initial Slow Start . . . 41
5.8.1 検証内容 . . . 41
5.8.2 検証方法 . . . 41
5.8.3 検証結果 . . . 41
5.9 Zero Window Probe . . . 42
5.9.1 検証内容 . . . 42
5.9.2 検証方法 . . . 43
5.9.3 検証結果 . . . 44
5.10 Slow Start After Retransmission . . . 44
5.10.1 検証内容 . . . 44
5.10.2 検証方法 . . . 44
5.10.3 検証結果 . . . 44
5.11 Fast Retransmit and Fast Recovery . . . 45
5.11.1 検証内容 . . . 45
5.11.2 検証方法 . . . 46
5.11.3 検証結果 . . . 47
5.12 TCPの検証結果 . . . 47

6 今後の展望 48
6.1 トランスポートやアプリケーションの開発 . . . 48
6.2 衛星通信 . . . 48
6.3 非対称リンクの実験 . . . 48
6.4 マルチメディア実験 . . . 49

7 まとめ 50

A 待ち時間分布の計算 56

B カーネル内部関数の関係図 **59**

C **TCP** における帯域および遅延とスループットの関係 **60**

D **Solaris2.5.1** の **TCP** によるデータ転送の様子 **61**

図の一覧

2.1 伝送制御機構を検証する方法 12

3.1 伝送路のモデル 14
3.2 １ホップのモデル 14
3.3 待ち時間の関係 16
3.4 実験に用いたネットワーク 18
3.5 系内時間の比較 19

4.1 PEER システムの構成 21
4.2 PEER システムの使用方法 22
4.3 PEER システムの動作原理 23
4.4 PEER システムの全体像 24
4.5 パケットの流れ 25
4.6 isysctl の使用例 26
4.7 inetstat コマンドの出力例 27
4.8 ivsual コマンドによる表示例 28
4.9 実験環境 29
4.10 PEER の性能評価 29
4.11 UDP パケットの到着間隔の分布 30
4.12 系内時間の分布 32

5.1 タイマーを用いた再送制御 39
5.2 セグメントの順番制御 40
5.3 スロースタート 42
5.4 ゼロウィンドウプローブ 43
5.5 タイマーを用いた再送後のスロースタート 45
5.6 早期回復 46

B.1 カーネル内の関数の関係 59

C.1 帯域、遅延、スループット 60

D.1 Solaris2.5.1 における再送の様子 61

第1章

はじめに

インターネットに代表されるコンピュータネットワークではデータ通信が行なわれている。特にリンク共有型のパケット交換型ネットワークであるインターネットの最大の特徴はベストエフォート型の通信を提供していることである。すなわち、インターネットにおいてはパケットの到達保証やパケットの到達時間の保証が一切ない。したがって、パケットの喪失、順序逆転、遅延の変動などが生じる。このようなインターネットを使用して確実かつ効率良くデータ転送を行なうためには、輻輳回避制御、再送制御などの伝送制御機構が必要である。本論文における伝送制御機構とは、ネットワークを介してデータ通信を行なう両端のアプリケーションやトランスポートで行なわれている伝送に関する制御のことを指す。さらに、インターネットはリンク共有型のネットワークであるため、各通信の公平性にも配慮しなければならない。この伝送制御機構の性能の差は、データ転送の効率や公平性に大きな影響を与える。したがって、実際にインターネットで使用されているアプリケーションやトランスポートについて、その伝送制御機構を検証する必要がある。

従来からインターネット上の伝送制御機構に関する研究は盛んに行なわれてきた。これらはシミュレーション、実測、エミュレーションの 3 つのアプローチに分類することができる。しかし、従来の研究はある特定の用途や目的に絞られて行なわれており、すべてのアプリケーションやトランスポートの伝送制御機構を検証する目的にはそぐわない。シミュレーションは限定された特定のモデルやアルゴリズムの検証しか行なえず、実測は特定のアプリケーションやトランスポートの実際のネットワークでの挙動や性能しか把握できない。また、エミュレーションでは特定の伝送路を模倣しているにすぎなかったり、特定のアプリケーションやトランスポートをエミュレートするのみに留まっている。

そこで、インターネットを使用するあらゆるアプリケーションやトランスポートの伝送制御機構を検証するシステムとして、インターネットをエミュレートし、その上で伝送制御機構を検証する方法を提案する。

まず、ルータにおけるパケットの処理方式、インターネットを伝送媒体の多様性、トラフィックの特性などからインターネットのモデル化を試みる。そして、1 段のルータを挟んだネットワークモデルを定式化し、そのモデルを通信トラフィック理論にもとづいて解析する。そして、その理論解析に基づいて、それを実現する PEER [1] と呼ばれるシステムを設計し実装する。PEER システムを用いることによりさまざまな種類の 1 段のルータ

を挟んだネットワークのエミュレートが可能になる。PEER システムは、帯域、遅延、パケットロス、キューサイズを自由に設定できるルータと各インターフェイスを通過したパケットに関する情報を収集するツール群から構成される。実装した PEER システムが正しく機能しているかどうかを実測値と理論値により比較検討を行なう。

最後にこれらをふまえ PEER システムを用いた伝送制御機構の検証の一例としてさまざまなオペレーティングシステムにおける TCP の検証を行なう。TCP は、現在インターネット上でもっとも使用されている伝送制御機能を備えたトランスポートの一つである。

本論文では、ネットワークをエミュレートし、その上で伝送制御機構を検証することの有効性および妥当性を示す。まず、第 2 章では、伝送制御機構の検証の必要性および既存の方法やシステムの問題点について述べ、伝送制御機構を検証するシステムとして求められる要素について述べる。第 3 章では、インターネットのモデル化、および 1 段のルータを挟んだネットワークモデルの定式化、そのモデルの通信トラフィック理論に基づいた解析について述べる。第 4 章では、さまざまな種類の 1 段のルータを挟んだネットワークをエミュレートすることができる PEER システムの設計および実装について述べる。第 5 章では、PEER システムを用いた伝送制御機構の検証の一例としてさまざまなオペレーティングシステムにおける TCP の検証について述べる。第 6 章では、今後の展望について述べ、最後に第 7 章でまとめとする。

第2章

伝送制御機構の検証

本章では、まずインターネットで使用されているトランスポートやアプリケーションの伝送制御機構の検証の必要性について述べる。そして、関連研究および従来の方法の問題点について述べ、最後に伝送制御機構の検証システムに求められる要素を述べる。

2.1 伝送制御機構

通信ネットワークの発展に伴って、さまざまな場面においてこの通信ネットワークを利用したデータのやり取りが行なわれるようになってきた。例えば、公衆電話網を利用したファクシミリの送受信、ATM 網を利用したテレビ会議、インターネットを利用した電子メールの交換など、数多くのものが挙げられる。

通信ネットワークを利用したデータ転送を効率的に行なうためには、伝送制御機構が必要であるが、この性能がデータ転送効率に大きな影響を与える。本論文における伝送制御機構とは、通信ネットワークを介してデータ通信を行なう両端で行なわれている伝送に関する制御のことを指す。伝送制御機構に関する研究は、通信ネットワークの形態によって求められる要件や特性などが異なるためすべてを同時には議論はできない。通信ネットワークの形態は大きく分けて、回線交換型、パケット交換型の 2 つに分類できる。回線交換型のネットワークでは、通信を行なう機器間で回線を占有できる。このためこれらの機器間のみでのデータ転送を考えれば良い。一方、パケット交換型のネットワークでは回線を占有できるわけではなく他の通信と共有するため他の通信も配慮する必要があり、回線交換型に比べて考慮しなければならない点が多い。

本論文では、インターネットを検討の対象とする。インターネットは、データ転送を行ないたい両端のホスト間における IP パケットの転送サービスを提供する通信ネットワークである。すなわちインターネットはパケット交換型のネットワークであり、さらに他の通信ネットワークに比べてさまざまな種類の通信媒体が使用されていること、利用の目的が多様化していることなどから、インターネットでのデータ転送では伝送制御機構が非常に重要であり、かつ難しい問題であるとされている。このような背景を踏まえ、ここではまずインターネットで使用されている伝送制御機構が難しい理由、およびインターネットで使用されている伝送制御機構への要件について述べる。

インターネットはリンク共有型のパケット交換型ネットワークであり、その最大の特徴はベストエフォート型の通信を提供していることである。すなわち、インターネットにおいてはパケットの到達保証やパケットの到達時間の保証が一切ないため、パケットの喪失、順序逆転、遅延の変動などが生じる。このようなベストエフォート型のネットワークにおいて確実かつ効率よくデータ転送を行なうために、輻輳回避制御、再送制御などの伝送制御機構が用いられている。さらに、インターネットはリンク共有型の通信ネットワークであるため公平性も重要な要素となる。

伝送制御機構としてインターネット上で用いられている具体的なトランスポートとしては、TCP[2] 、UDP[3]、VMTP[4] [5]、XTP[6]、RTP[7]、ST-2[8] などがあげられる。例えば、トランスポートの一つである TCP では広帯域でも高遅延である場合には帯域を十分に使えないことや、遅延の大きいものと小さいものが競合した場合に大きい方は帯域を取れないこと、また、 UDP などの通信と競合してしまった場合には十分に帯域を確保することができないなどの問題が指摘されている [9]。このように現在広く用いられているものについても問題が存在している。

また、インターネットの用途の多様化の面からも伝送制御機構の重要性が認識されている。例えば、最近では計算機の高速化および広帯域ネットワークの広域にわたる普及によって、音声や映像といった連続メディアを広域ネットワークで送受信するアプリケーションが出現し、実用的に使われることが期待されている。しかし、インターネットでは転送サービスの品質が保証されていないため、連続メディアを転送するアプリケーションはいかにして連続メディアの時間的制約を保ちつつデータを転送するかが大きな問題とされている。

上記のことからも分かるように伝送制御機構の性能がデータ転送の性能や効率に大きな影響を与える。特にインターネットにおける伝送制御機構では、以下のことが要求される。

- 帯域を公平に使い、他の通信を圧迫しないこと
- 輻輳が発生した場合には、回避を行ない輻輳を助長しないこと
- 利用可能な帯域を十分に使い効率のよい通信をすること

各トランスポートやアプリケーションの伝送制御機構は RFC(Request For Comments) や I-D(Internet Draft) において規定や提案がなされており、より性能や効率の良い方式が研究されてきた。しかし、現在実際に我々が日常使用しているトランスポートやアプリケーションの伝送制御機構に対して検証が行なわれることは少なかった。今後はさらにさまざまな種類のトランスポートやアプリケーションが開発されてくると考えられるため、インターネットの効率的かつ公平な利用のためにも、現在すでに使用されているトランスポートやアプリケーションも含めて伝送制御機能を検証する必要がある。

2.2 関連研究

伝送制御機構の検証に関係する研究は、その方法からシミュレーションによる方法、実測による方法、エミュレーションによる方法の 3 種に分類することができる。以下では、

インターネットにおける伝送制御機構の検証という点からそれぞれの方法について検討し、伝送制御機構の検証方法に関する議論を行なう。

シミュレーションによる方法

シミュレーションによる方法としては、A.Heybey らによる ns[10] や S.Keshav らによる REAL [11] などがある。シミュレーションによる方法の利点は、一台の計算機上で行なえるため手軽であり、また測定者の意図するネットワークや提案するモデルなどもプログラムを変更するだけで評価が行なえるという点である。しかし、シミュレーションによる方法では、データリンクやアーキテクチャの特性まではシミュレートできない、ソースコードの公開されていないものに対しての検証は難しい、またソースコードが公開されていたとしてもシミュレータへの移植作業が必要であるためこの点では手軽でないなどの欠点がある。そして、何よりも検証している対象は実際に我々が使うアプリケーションやトランスポートではなく、それを模倣した特定のモデルである。このためシミュレーションによる方法は、モデルやアルゴリズムに限定して検証する場合には良い方法であると考えられる。

実測による方法

実測による方法としては、T.C.Slattery による ttcp、R.Jonkman による netspec[12]、R.Jones による Netperf[13]、村山らによる DBS[14]、V.Paxson による Tcpanaly[15]、Tim Shepard による Xplot[16]、S.Ostermann による Tcptrace、G.Minshall による Tracelook などがある。実測による方法は、実際に我々が使用するアプリケーションやトランスポートを使用できること、また、実際のネットワークを使用できるため他のトラフィックからの影響も測定できること、アーキテクチャや実装などの影響も考慮した総合的な検証が可能なことなどが利点として挙げられる。しかし、さまざまなネットワーク環境を準備することが困難であるため、意図する全ての条件での検証を行なうことが難しい。また、自然のトラフィックの影響を受けるため、不具合の原因を特定することが困難になるなどの欠点がある。また、実測による方法を採用した上述のシステムは、それぞれのアプリケーションやトランスポートに特化したものであることが多いことや、検証できる内容が限られていることも欠点であると考えられる。

エミュレーションによる方法

エミュレーションによる方法としては、Luigi Rizzo による Dummynet[17]、N.C.Hutchinson らによる x-kernel[18]、E.Limin Yan による Emulated WAN[19] などがある。エミュレーションによる方法では、実際のアプリケーションやオペレーティングシステムが使えること、さまざまなネットワーク環境を意図的に操作できること、また既存のアプリケーションなどを使うことにより他のトラフィックを発生させることができることなどの利点がある。エミュレーションによる方法の欠点としては、実際のネットワークを正確にエミュレートすることが非常に難しいことがあげられる。しかし、既存のシステムでは、単にネットワー

クやトランスポートのみをエミュレートするにとどまっているものが多い。このため、それらは総合的な伝送制御機構の検証に用いるには不十分であると言える。

以上のように、それぞれの方法には長所および短所があり、これらそれぞれの方法によって実装されたシステムでは、特定の用途もしくは限定された内容のみを対象とした検証しかできないことが分かる。シミュレーションはより良いモデルやアルゴリズムのみのためであり、実測は特定のアプリケーションやトランスポートの実際のネットワークでの挙動や性能しか把握できない。また、エミュレーションでは特定の伝送路の模倣や特定のアプリケーションやトランスポートのエミュレートだけに留まっている。

以上から、インターネットで使用されている全てのトランスポートやアプリケーションの伝送制御機構を検証するシステムが存在しないことがわかる。そこで、インターネットで使用されている伝送制御機構を総合的に検証するシステムに対する必要要件を次節で検討する。

2.3 伝送制御機構の検証システム

インターネットで使用されている全てのトランスポートやアプリケーションの伝送制御機構を検証するシステムに求められる要件について検討する。必要な要件は以下の通りである。

- さまざまなネットワーク環境において検証できること
- 実際のアプリケーションやトランスポートの検証ができること
- ソースコードがなくても検証できること
- 特定の用途に特化しないこと
- さまざまな情報を収集できること

以下ではそれぞれについて詳しく述べる。

さまざまなネットワーク環境において検証できること

インターネットにおける通信では、通信相手や通信経路によって利用可能な帯域や到達までの遅延が異なる。また、これらはネットワークの混雑の具合にも依存するため時間によっても異なる。このためインターネットにおける伝送制御機構はこれらに柔軟に対応する必要がある。したがって、伝送制御機構を検証するシステムにおいてはさまざまなネットワーク環境における伝送制御機構を検証できる必要がある。

実際のアプリケーションやトランスポートの検証ができること

検証したいアプリケーションやトランスポートとは実際に使用しているものである場合が多い。それらを模倣したものを検証する方法も考えられるが、アーキテクチャとの相性や実装による影響までを模倣することは難しい。また、模倣するための手間を考えると必ずしも良い方法であるとは言えない。したがって、実際のアプリケーションやトランスポートをそのまま検証できることが望ましい。

ソースコードがなくても検証できること

検証したいアプリケーションやトランスポートには必ずしもソースコードが付属しているとは限らない。商用のアプリケーションやトランスポートには付属してない場合の方が多い。しかし、このようなソースコードの付属していないアプリケーションやトランスポートを使用することも多い。したがって、ソースコードがなくても検証できることが望ましい。

特定の用途に特化しないこと

伝送制御機構を検証する際に、特殊な機材を用いてネットワークをエミュレートしたり、伝送媒体に特化した方法により情報を収集する方法なども考えられる。しかし、インターネットは非常に多くの異なる伝送媒体から形成されている。伝送制御機構を検証する際には、これらのさまざまな伝送媒体において検証できることが望ましい。また、上記で述べたように特定のトランスポートやアプリケーションに特化した方法は望ましくない。従って、特定の伝送媒体や用途に特化した方法は望ましくない。

さまざまな情報を収集できること

伝送制御機構を検証する際にデータ転送に関するさまざまな情報を収集できる必要がある。検証したいアプリケーションやトランスポート、もしくは検証内容によって収集したい情報は異なると考えられる。このため目的に応じて柔軟にさまざまな情報を収集できることが望ましい。

上記の要素を満たす伝送制御機構を検証システムの実現方法について考えた。そして、インターネットで使用されているあらゆるアプリケーションやトランスポートの伝送制御機構を検証するシステムとして、図 2.1 のようにインターネットをエミュレートして、その上で伝送制御機構を検証する方法を提案する。

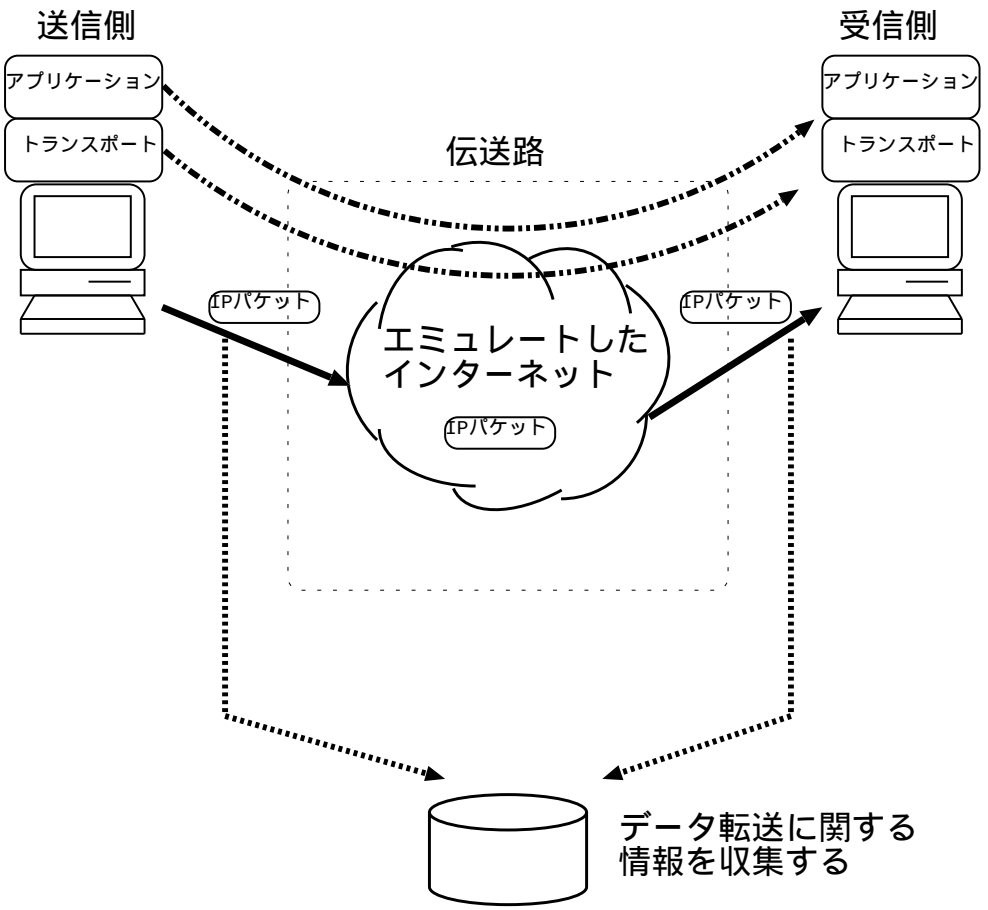

図 2.1 伝送制御機構を検証する方法

各アプリケーションやトランスポートのデータ転送の様子は図 2.1 のように送出直後および到着直前を観測することにより収集できる。また、実際のネットワークと異なる部分はネットワークをエミュレートしている部分のみであるから、あらゆるアプリケーションやトランスポートの検証ができることが分かる。

この方法ではインターネットをいかにしてエミュレートするかが最大の問題である。そのためには、まずはインターネットの網としての特性を検討する必要がある。その検討結果にもとづいてインターネットをエミュレートする方法を検討するのが妥当である。

次章では、インターネットの網としての特性の検討について述べる。

第3章

インターネットのエミュレーション

本章では、インターネットのエミュレーションに関する議論を行なう。

3.1 インターネットのモデル化

インターネットはさまざまな規模のネットワークがルータによって相互に接続された大規模なネットワークであり、通信相手とIPパケットのやり取りを行なうことによってデータ通信が行なわれる。いわゆるパケット交換型ネットワークである。

ネットワークをモデル化するためには、そのネットワークの伝送路およびトラフィックをモデル化する必要がある。以下では、伝送路およびトラフィックのモデル化について述べる。

3.1.1 伝送路のモデル化

インターネットにおける伝送路は、ルータとルータ同士を接続する伝送媒体から構成され、これらが複雑に絡み合うことによって形成されている。伝送路をモデル化するに当たって、ここでは2つのホスト間におけるデータ転送に着目する。データ転送の最中に2つのホスト間のデータ転送経路が変化しないと仮定した場合、2ホスト間の伝送路は図3.1のようにいくつかのルータとそれぞれの間を接続する伝送媒体を一直線に繋いだ形にモデル化できる。さらに、この伝送路を細かく見ると図3.2の破線の部分のようにルータとそのルータがパケットを転送する方向の伝送媒体の組み合わせに分けることができる。本論文では、このルータとそのルータがパケットを転送する方向の伝送媒体を合わせたものを1ホップと呼ぶことにする。以下では、この1ホップのモデル化を考える。

この系に入れられたパケットが次のホップに渡されるまでの行程は次のようになっている。

1. この系に入ったパケットは、まずルータのキューに入れられ転送されるための順番待ちに入る。
2. 順番がきたら、そのパケットの宛先IPアドレスから出力すべきインターフェイスが決められる。

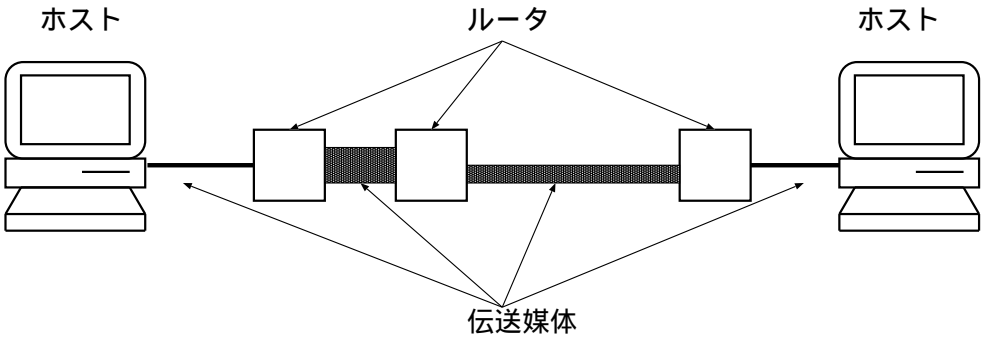

図 3.1 伝送路のモデル

3. 出力すべきインターフェイスの伝送媒体を通して次のホップに渡される。

ルータにおけるキューイング方式は、研究レベルでは RED [20] や CBQ[21] などのさまざまなキューイング方式が提案されているが、現段階ではほとんどはテイルドロップの FIFO キューイングが用いられている。したがって、現状のインターネットでは先着順の FCFS [1] の待ち行列としてモデル化できる。

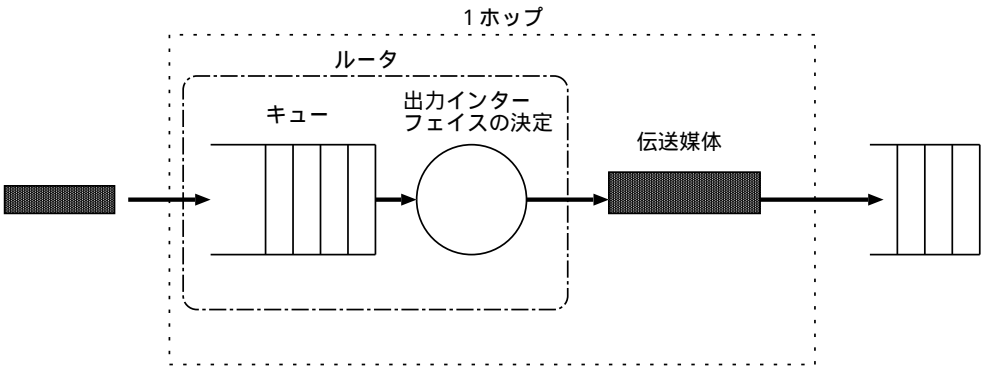

図 3.2 1ホップのモデル

また、経路の決定に関しては、ルーティングテーブルの大きさや探索方法に依存するが、各パケットごとにほぼ一定の時間で処理されると考えることができる。さらに、ここで処理されるパケットの量は出力先の伝送媒体の転送能力に依存することが分かる。したがってこれも伝送媒体に依存した一定時間サービスとみなすことができる。以上のことから、出力インターフェィスの決定から出力インターフェイスへの出力までの時間に関しては、パケットの種類によらず一定時間サービスとモデル化できる。

最後に、伝送媒体を通じて次のホップのキューに入れられるまでに伝送路による遅延の影響を受ける。これに関しても伝送路自体の遅延であるため一定時間サービスとしてモデ

[1] first come first serve.

ル化が可能である。

以上のことから、１ホップは「先着順処理の一定時間サービス」としてモデル化できることが分かる。さらに、インターネットにおける伝送路は多段化した一定時間サービスの組み合わせとしてモデル化できると考えられる。しかし、各ホップによって伝送媒体の帯域や遅延が異なるため解析は容易ではない。

3.1.2 トラフィックのモデル化

インターネットにおける伝送制御機構を検証するために、ネットワークのエミュレートを行なうに当たって、ネットワークを流れるトラフィック自体をエミュレートすることも必要である。

従来からインターネットのトラフィックをランダム到着とみなしてモデル化や理論解析などが行なわれてきた [22][23]。しかし、近年、Leland らは、実際のトラフィックが自己相似性的な分布を持つことを指摘し、従来のパケット到着モデルに疑問を投げて話題となっている [24]。従来のモデルではパケットの破棄率を一定以下に押えるためのバッファサイズが計算できるのに対して、自己相似なパケット到着モデルでは、いくらバッファサイズを大きくしても、バースト的な通信に対してパケットの破棄率を押えることが不可能である点が大きく異なる。実際のトラフィックは、これらのモデルの両側面を持つと考えられ、今後の理論的な解析の進展と実際のネットワークへの応用が期待されている。

また、FTP や TELNET や NNTP などの TCP を用いた各アプリケーションの広域ネットワークにおける発生トラフィックの解析も行なわれている。Paxson らによると、コネクションの到着間隔はポアソンモデルとして近似できるが、各コネクション内におけるトラフィックの発生間隔はポアソンとはやや異なり各アプリケーションに依存したものとなる [25][26]。このため tcplib[27] のような観測データに基づいたトラフィック発生システムも存在する。しかし、これらの論文では今の広域ネットワークのトラフィックの大半を占める WWW トラフィックに関する議論は行なわれていない。

以上のように、実際のネットワークのトラフィックは到着間隔がポアソン到着に近い形にはなるが、厳密にはポアソン到着ではないことがいくつかの論文で指摘されている。しかし、本論文の目的はインターネットにおける伝送制御機構の検証であり解析可能なモデルかつ再現性可能なモデルの方が好ましい。そこで本論文ではトラフィックモデルとして実際のネットワークにほぼ近いと従来から言われているパケット到着間隔がポアソンであるモデルを用いて検討を行なう。

3.1.3 まとめ

上記のことから１ホップは、ケンドール記号で $M/D/1$ と表せることになる。サービス時間はそのホップにおける帯域や遅延などに依存するため、必ずしもすべてのホップで同じになるとは限らない。以下では $M/D/1$ モデルの解析について述べる。

3.2 モデルの解析

上記でモデル化した 1 ホップの理論的解析について述べる。

ここでは、一定時間サービス $M/D/1$ における待ち時間の分布を求める [28][29] [30]。まず、1 ホップ系内に到着したパケットに番号を付け、n 番目のパケットの待ち時間を w_n、n 番目のパケットと $n+1$ 番目のパケットの到着間隔を t_n、n 番目のパケットのサービス時間を s_nとする。ここで、w_nの漸化式を考える。

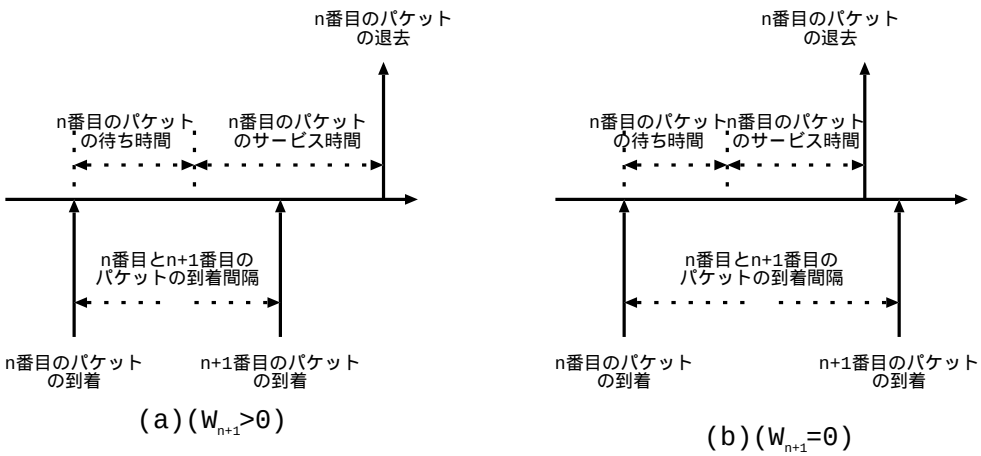

図 3.3 待ち時間の関係

$n+1$ 番目のパケットが到着したときに、窓口がふさがっていた場合には $w_{n+1} > 0$ であり、ふさがっていない場合には $w_{n+1} = 0$ である。それぞれの場合を図 3.3 の $(\mathrm{a})(\mathrm{b})$ に示す。

(a) の場合には以下の式が成り立つ。

$$w_{n+1} = w_n + s_n - t_n$$

また (b) の場合は、明らかに $w_{n+1} = 0$ である。これらを合わせて考えると $n+1$ 番目のパケットの待ち時間は以下のように表せる。

$$w_{n+1} = \max\left[0, w_n + s_n - t_n\right]$$

ここで $M/D/1$ であることからサービス時間は一定であり、サービス時間 s_nをτと置く。また、到着間隔を z と書けば、w_{n+1}は以下のように表せる。

$$w_{n+1} = \max\left[0, w_n + \tau - z\right]$$

いま、任意の正の整数 t に対して、$z = u$ が分かったという条件のもとでは、以下の式が成り立つ。

$$P_u\{w_{n+1} < t\} = P_u\{w_{n+1} < t + u - \tau\}$$

ただし、P_uは、$z = u$ という条件のもとでの条件付き確率を表す。しかも、z は n 番目のパケットと $n+1$ 番目のパケットとの到着間隔であるため、w_n は明らかに z とは独立である。したがって、$f(t) = P\{w_n < t\}$ とすると w_nと w_{n+1}が同一の分布であることを用いて、$f(t)$ は以下のように表せる。

$$\begin{aligned} f(t) &= \int_0^\infty \lambda e^{-\lambda u} P_n\{w_{n+1} < t\} du \\ &= \int_0^\infty \lambda e^{-\lambda u} P_n\{w_n < t + u - \tau\} du \\ &= \int_0^\infty \lambda e^{-\lambda u} f(t + u - \tau) du \end{aligned}$$

この $f(t)$ を微分可能であると仮定して、両辺を微分すると以下のようになる。

$$f\prime(t) = \int_0^\infty \lambda e^{-\lambda u} f\prime(t + u - \tau) du \quad (t \geq 0)$$

この両辺をさらに部分積分すると以下のように表せる。

$$f\prime(t) = \left[\lambda e^{-\lambda u} f(t + u - \tau)\right]_0^\infty + \lambda \int_0^\infty \lambda e^{-\lambda u} f(t + u - \tau) du$$

すなわち、以下の微分方程式が得られる。

$$f\prime(t) = \lambda \left[f(t) - f(t - \tau)\right]$$

この微分方程式を解くと以下のように待ち時間の分布が得られる (付録 A)。
$0 < t \leq \tau$ のとき

$$f(t) = (1 - \lambda\tau) e^{\lambda t}$$

$n\tau < t \leq (n+1)\tau$ のとき

$$f(t) = (1 - \lambda\tau) e^{\lambda t} \sum_{k=0}^{n} e^{-\lambda k \tau} \frac{(k\tau - t)^k}{k!} \lambda^k$$

だたし、n は任意の非負の整数であるとする。

λは、単位時間当たりのパケットの到着数でありネットワークの状態により決まる。τは、1 ホップにおける帯域および遅延、ルータの処理時間の影響を受け、これらによってサービス時間τが決定される。

ここで、到着するパケットのサイズが一定であると仮定し、ルータにおいて 1 パケットの処理にかかる時間をκ とすると、インターネットの 1 ホップにおけるサービス時間は以下のように求まる。

$$\tau = \frac{1000 \times \text{パケットサイズ}\,(Bytes)}{\text{帯域}\,(KByte/sec) \times 1024} + \kappa \tag{3.1}$$

従って、帯域を bKByte/sec、遅延を $d$msec とおくと、１ホップにおける PEER システム内における系内時間の分布は以下のようにして求めることができる。ただし、パケットサイズは Sbytes で一定とする。

$0 < t \leq \tau$ のとき

$$f(t) = \{1 - \lambda(\frac{1000 \times S}{b \times 1024} + \kappa)\}e^{\lambda(t-d)} \tag{3.2}$$

$n\tau < t \leq (n+1)\tau$ のとき

$$f(t) = \{1 - \lambda(\frac{1000 \times S}{b \times 1024} + \kappa)\}e^{\lambda(t-d)} \sum_{k=0}^{n} e^{-\lambda k(\frac{1000 \times S}{b \times 1024} + \kappa)} \frac{\{k(\frac{1000 \times S}{b \times 1024} + \kappa) - (t-d)\}^k}{k!} \lambda^k \tag{3.3}$$

だたし、n は任意の非負の整数であるとする。

3.3 理論値と実測値の比較検討

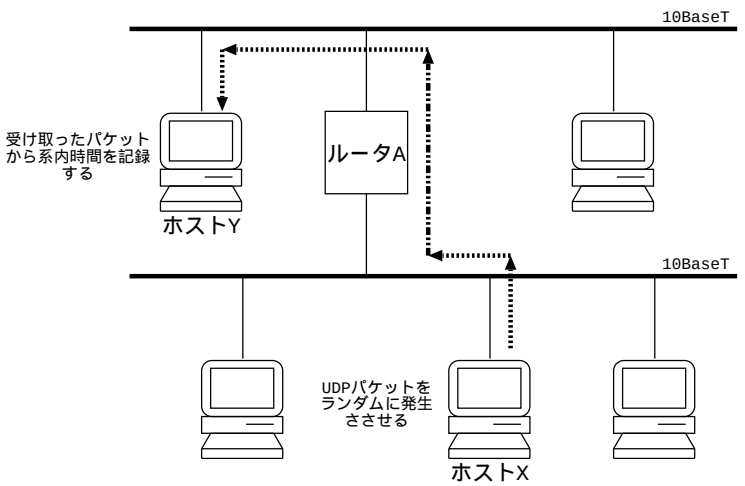

図 3.4 実験に用いたネットワーク

3.2 節の検討に対し、同じ条件で実際の１ホップのデータを測定した。実験には研究室内で実際に使用しているネットワークを用いて行なった。測定は、図 3.4 のホスト X からホスト Y へルータ A を介してデータ転送を行ない１ホップの系内時間を測定した。系内時間の分布を求めるために、ホスト X において UDP パケットをランダム間隔で発生させ、各パケットの１ホップ内における系内時間を求めた。各 UDP パケットのデータ部分にはシーケンス番号と送出時刻が入れられており、受信側ではそのデータと現在の時刻を比較することにより系内時間を求める。ただし、系内時間を求めるために両端のマシンの時刻が正確である必要があるため、NTP[2] を用いてホスト X およびホスト Y の時計を正確に合わせた

[2] Network Time Protocol

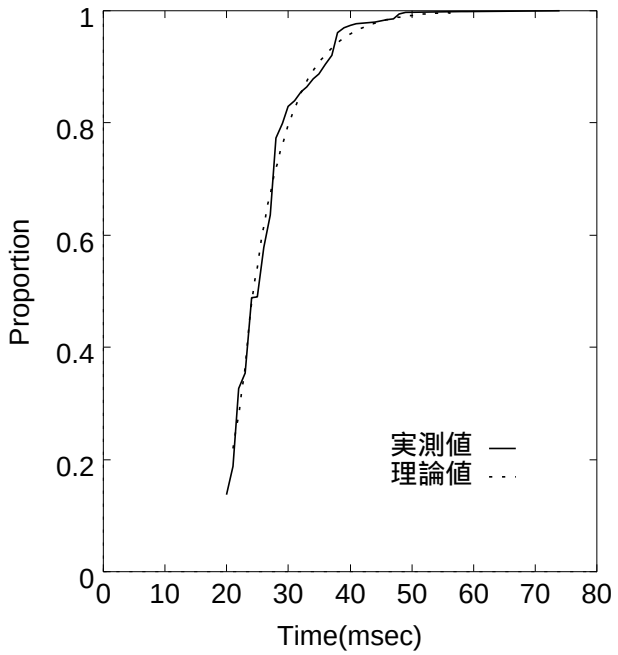

図 3.5 系内時間の比較

後に測定を行なった。UDP パケットの送出には単位時間当たりのパケットの送出量を調整でき、かつパケットの発生間隔をランダムにできるように変更を加えた ttcp を使用した。UDP パケットの発生率λを 0.26、UDP パケットのサイズを 1460Bytes にした。

実測値および理論値のグラフを図 3.5 に示す。理論値はサービス時間τを 3.0、遅延時間dを 21 として上記の式 (3.2) および式 (3.3) から求めたものである。

以上から、3.2 の式 (3.2) および式 (3.3) と実験値がおおまかに一致することから、伝送路の $M/D/1$ 近似は妥当であることが分かる。

3.4 まとめ

本章では、インターネットをエミュレートするために通信路の特性およびトラフィックの特性からインターネットのモデル化を行なった。その結果、1 ホップは $M/D/1$ でモデル化できることを導き出し、実際の測定結果と比較検討することによりこのモデルの妥当性を示した。このモデルにもとづいたさまざまな種類の 1 ホップを作り出す装置を実現することにより、インターネットをエミュレートして、その上で伝送制御機構を検証することが可能となる。

次章では、上述の 1 ホップのモデルを実現する装置の設計と実装について述べる。

第4章

PEERシステムの設計と実装

前章では、インターネットの1ホップを $M/D/1$ モデルで近似することの正当性を述べた。そこで、本章では $M/D/1$ モデルをもとに、さまざまなネットワークをエミュレートして伝送制御機構を検証するPEERシステムを提案し、その設計と実装について述べる。

4.1 PEERシステムの特徴

PEERシステムでは、$M/D/1$ モデルにもとづいたさまざまなネットワーク環境を実際にエミュレートし、エミュレートした各ネットワーク環境を用いて実際にデータ転送を行い、その上でデータ転送を観測することによって伝送制御機構を検証する。PEERシステムの特徴は以下の通りである。

- 帯域、遅延、喪失、キューサイズのそれぞれをエミュレートできる。また、それらのパラメータを自由に変更できる。
- データ転送における各種情報を収集できる。
- データ転送の様子をリアルタイムに可視化できる。

4.2 PEERシステムの構成

PEERシステムの構成について述べる。PEERシステムは図4.1のように以下の3つの部分から構成される。

- ネットワーク部
- 制御部
- 情報収集部

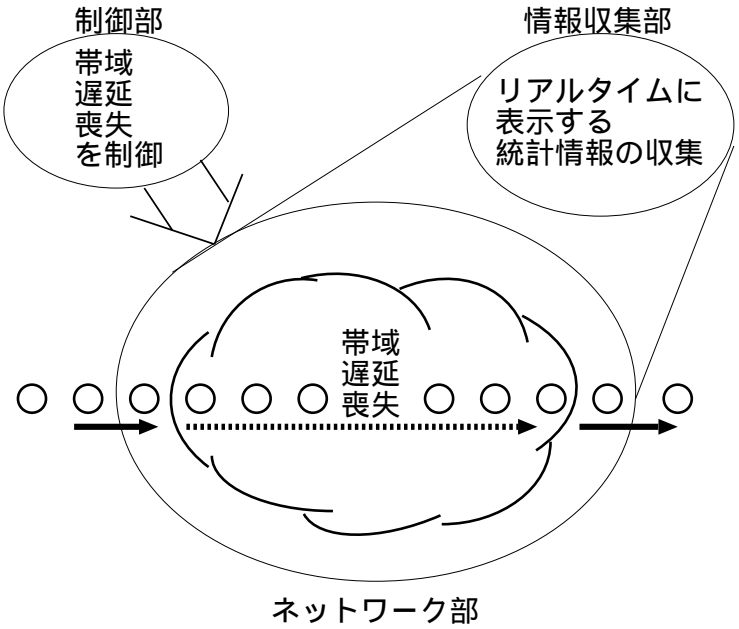

図 4.1 PEER システムの構成

まず、はじめに PEER システムを用いた検証の手順を具体的に述べる。本システムの一番簡単な構成例を図 4.2 に示す。PEER システムは少なくとも 2 つ以上のネットワークインターフェイスを持った PC1 台で構成される。以下では、この PC を PEER ルータと呼ぶ。伝送制御機構を検証したいアプリケーションやトランスポートが実装された 2 台の計算機を用意して、PEER ルータを介して通信が行えるように設定する。そして、実現したいネットワーク環境を PEER ルータ上で設定した後に、実際に通信実験を行い、PEER ルータから観測した情報を取り出す。以上のようにして、利用者は PEER システムを使用して伝送制御機構の検証を行なう。上記の例では一台の PEER ルータしか使用していないが、数台組み合わせて使用することも可能である。

4.3 動作原理

PEER システムの動作原理について述べる。PEER システムは各インターフェイスにおける「帯域」「遅延」「パケットロス」「キューサイズ」のそれぞれをエミュレートするために、 dummynet[17] を参考にして各インターフェイスごとに図 4.3 のような 2 つのキューを使用した。 以下では、このキューをそれぞれ Limit_Queue、Delay_Queue と呼ぶ。

- まず PEER ルータはパケットを受け取ると経路表を参照して、出力すべきインターフェイスを決める。そして、そのパケットを該当するインターフェイスの Limit_Queue に入れる。Limit_Queue では、ルータにおけるキューの大きさをエミュレートしている。もし、Limit_Queue に指定されたサイズより多くのパケットが入れられる場合に

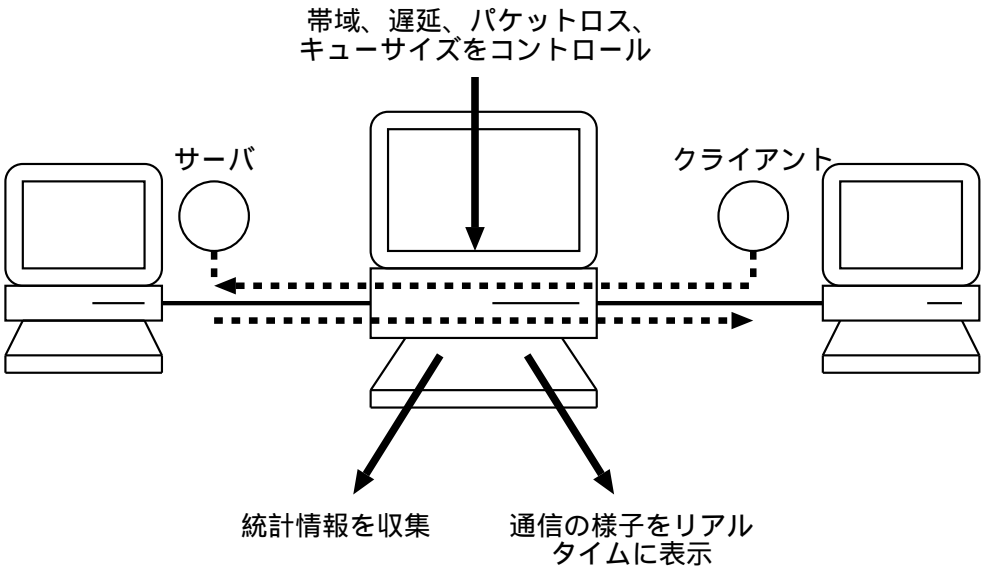

図 4.2 PEER システムの使用方法

は、そのパケットは破棄される。

- 次にパケットを Limit_Queue から Delay_Queue に移動する。この際に帯域の制御を行っている。Limit_Queue から Delay_Queue へのパケットの移動は一定時間をおいて定期的に行い、移動するパケットの量は設定された帯域から決める。例えば、帯域を xKbytes/sec に制限するために、y msec ごとのタイマを使用する場合には一回の移動に $\frac{x}{y \times 1000}$K バイトのパケットを移動する。
- Delay_Queue では、遅延の制御を行っている。Delay_Queue では、設定された時間の間、パケットを保持する。このようにすることによって遅延のエミュレートを行っている。最後に Delay_Queue から出力する際にパケットロスをエミュレートする。

以上のような動作原理を用いて各インターフェイスごとに「帯域」、「遅延」、「パケットロス」、「キューサイズ」をエミュレートしている。

以下では、ネットワーク部、制御部、情報収集部の各部についてそれぞれ詳しく述べる。

4.4 ネットワーク部

4.4BSD 系の UNIX では、IP パケットを処理する部分を初期化するために関数 ip_init() を実行する [31]。PEER システムでは、まずこの部分に変更を加えた。ここは、図 4.3 に示したようなキューを各インターフェイスごとに用意するように変更を加えた。

次に、パケットの転送処理する部分に変更を加えた。4.4BSD 系の UNIX では図 4.5 に示すように、インターフェイスから渡されたパケットを関数 ip_intr() に渡す。ここではパ

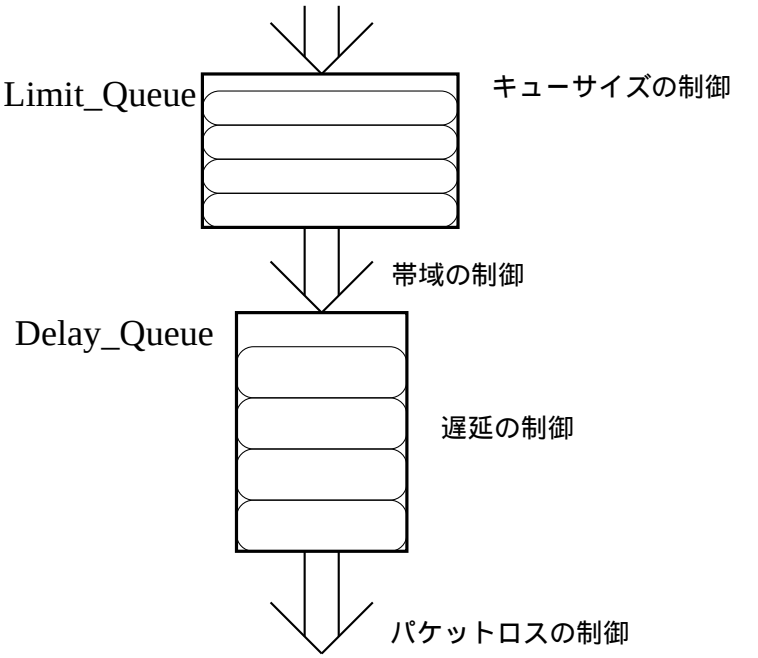

図 4.3 PEER システムの動作原理

ケットヘッダを見て自分宛のパケットかどうかを判断し、自分宛であればトランスポート層の処理部分に渡し、自分宛でなければ関数 ip_forward() に渡し、転送処理を行う。関数 ip_forward() は渡されたパケットを出力すべきインターフェイスを決めた後に、しかるべきインターフェイスに出力する。PEER システムでは、関数 ip_forward() から関数 ip_output() に引き渡す際に、前述のキューにパケットを入れるように変更した。

関数 ip_forward() から Limit_Queue に入れる際にすでに Limit_Queue に入れられているパケットの数を確認して、設定された数より大きくなるようであれば、そのパケットを破棄する。そうでなければ Limit_Queue に入れる。次に、Limit_Queue に蓄えたパケットを定期的に Delay_Queue に移動する。この際に動作原理のところで述べたように、一回に移動するパケットの量は帯域とタイマーから計算する。これを実現するためにタイマー関数である関数 timeout() を使用している。このため帯域制御の精度の限界は、このタイマー関数の精度に依存している。そして、Delay_Queue において指定した時間だけパケットを保持することによって遅延を生じさせている。この遅延の制御にも前述のタイマーを使用しているため、精度はタイマーに依存している。最後に、Delay_Queue から関数 ip_output() に渡す。この際にパケットロスをエミュレートしている。

4.5 制御部

制御部は、PEER ルータの制御を行なう部分である。PEER システムでは、各インターフェイスの設定を行うために isysctl コマンドを用意した。このコマンドは PEER ルータ

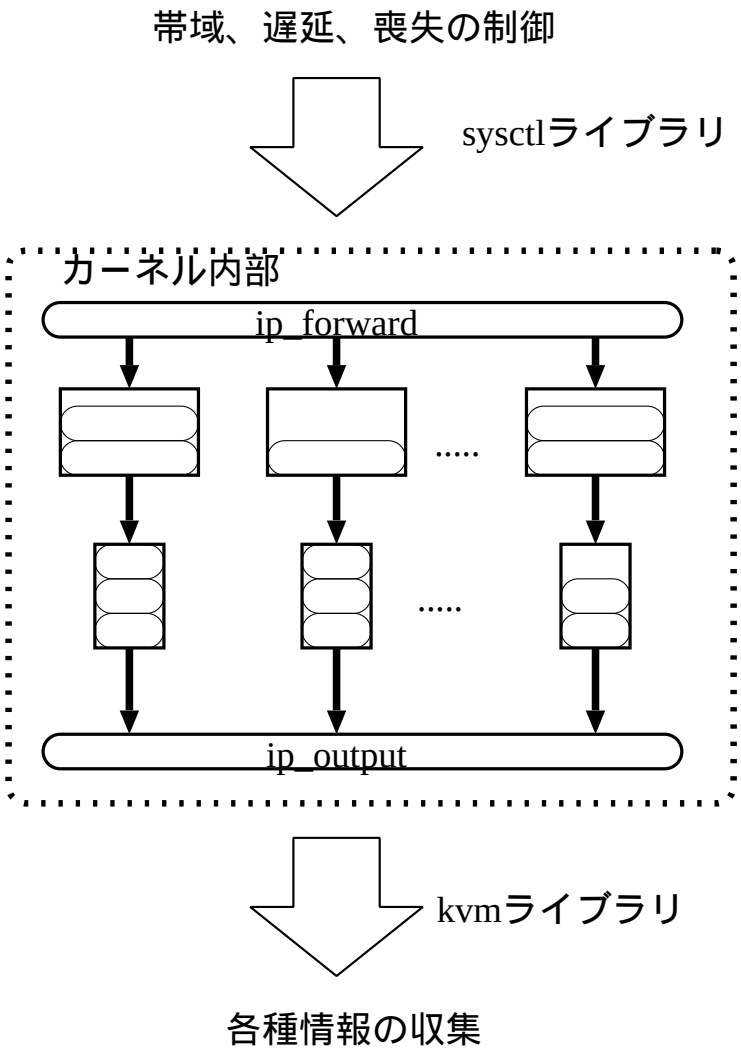

図 4.4 PEER システムの全体像

をコントロールするためのコマンドであり、統計情報の初期化なども行なう。 このコマンドには sysctl ライブラリ [32] を使用している。isysctl コマンドの使用例を図 4.6 に示す。ここでは、3 番目のインターフェイスにおいて喪失を 10%、帯域を 100KByte、キューサイズを 6 個、遅延を 30ms に設定している。

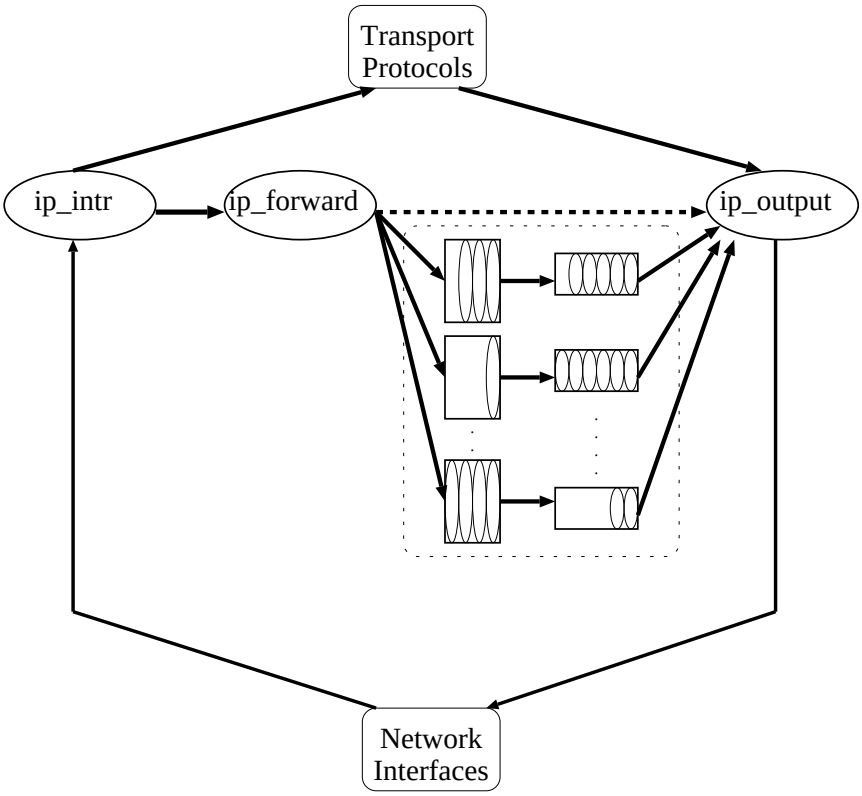

図 4.5 パケットの流れ

4.6 情報収集部

情報収集部は、PEER ルータから各種情報を収集する部分である。PEER ルータから各インターフェイスの情報を取り出すために inetstat および ivisual コマンドを用意した。このコマンドは kvm ライブラリ [32] を使用して、各種の統計情報をカーネル内部から読み出す。

各インターフェイスの統計情報をキャラクタで見るためには inetstat コマンドを利用する。inetstat ではコマンド実行時点の各インターフィスにおける以下の情報を情報を取り出すことができる。inetstat コマンドの出力例を図 4.7 に示す。

- 通過したパケットの数
- Limit_Queue の長さの平均、分散、最大値
- Limit_Queue に入ったパケットの総バイト数
- パケットに与えた遅延の平均、分散、最大値
- 破棄したパケットの数およびバイト数
- 連続して破棄したパケットの数およびバイト数

```
#./isysctl -help

usage: ./isysctl [-s <flag>]

        flag:           ON=1,OFF=0

usage: ./isysctl [-i <link> ][-l loss(%)][-b bandwidth(KB/s)]
                    [-k queue_size][-d delay(*10ms)]

        link:           0 -- 16 (netstat -i)
        loss:           0 -- 99(%)
        bandwidth:      0 -- 999(KB/s)
        queue size:     0 -- 99
        delay:          0 -- 99(*10ms)

#./isysctl -i 3 -l 10 -b 100 -k 6 -d 3
#./isysctl -s 1
```

図 4.6 isysctl の使用例

また、各インターフェイスを通過するパケットの様子をリアルタイムに観測するには ivisual コマンドを利用する。 このコマンドでは、kvm ライブラリの他に、Tcl/Tk[33] および Tk の拡張ライブラリである BLT ライブラリを使用した。ivisual コマンドでは各インターフィスにおける以下の情報の変化をリアルタイムに観察することができる。また、これらの情報はコマンド終了時にテキストファイルに残すこともできる。ivisual を実行した様子を図 4.8 に示す。

- スループットの変化
- Limit_Queue の長さの変化
- パケットに与えた遅延の変化
- シーケンス番号の変化
- Limit_Queue に入ったパケットのバイト数の変化
- 破棄したパケットのバイト数の変化

```
# ./inetstat -i 3
=======================<link 3>=======================
Total Through Packets            795              packets
Queue Length Average             3.652198         pieces
Queue Length Varience            1.813926
Queue Length Maximum             7                pieces
Total Packet Bytes               1167044          bytes
Packet Delay Average             77.544655        ms
Packet Delay Varience            1174.892090
Packet Delay Maximum             192.000000       ms
Total Drop Packet                45               packets
Total Drop Bytes                 63652            bytes
Drop Burst Average               2.000000         packets
Drop Burst Varience              0.000000
Drop Burst Maximum               2                packets
=======================<link 3>=======================
#
```

図 4.7 inetstat コマンドの出力例

4.7 実装に使用したオペレーティングシステムとツール

PEER システムのネットワーク部の実装には、4.4BSD 系の UNIX である BSD/OS2.1[1] を使用した。このため、現時点では、BSD/OS2.1 のみでの動作しか確認していない。しかし、BSD/OS3.0 および 4.4BSD 系列の FreeBSD[2] や NetBSD[3] に移植することは容易であると思われる。また、PEER システムの情報収集部では、可視化するために以下のツールを使用した。

- Tcl version7.6
- Tk version4.2
- BLT version2.1[4]

4.8 PEERシステムの性能評価

PEER システムの性能評価を行なった。性能評価実験に用いた実験環境を図 4.9 に示す。PEER ルータには、CPU に Pentium133MHz を使用し、メモリを 40MB 搭載したラップトップ PC を用いた。Host1 および Host2 には 486DX2 66MHz を使用したデスクトップ PC を用いた。今回の性能評価実験では、特に PEER ルータの帯域制御機能および遅延制御機能の性能評価を行なった。

[1] http://www.bsdi.com
[2] http://www.freebsd.org
[3] http://www.netbsd.org
[4] ftp://ftp.neosoft.com/tcl/extensions/BLT2.1.tar.gz

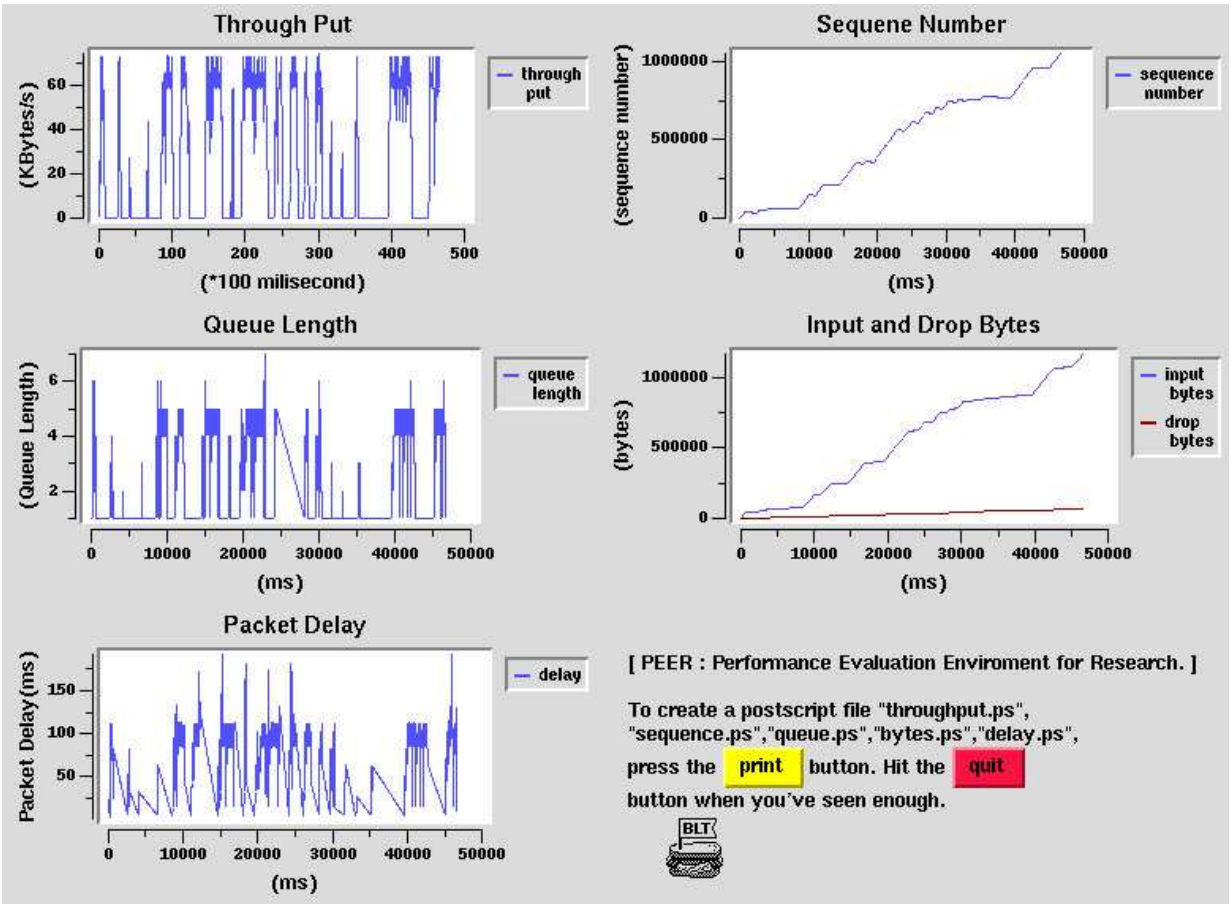

図 4.8 ivsual コマンドによる表示例

帯域制御機能の評価では ttcp を用いてデータ転送を行ない、そのトラフィックを観測した。各インターフェイスにおいて適宜帯域を設定した後に、ttcp の TCP のバッファサイズを 1024Bytes、メッセージサイズを 1024Bytes に設定し、1MBytes のデータ転送を行なった。各インターフェイスにおいて帯域を設定する際に、残りの遅延、パケットロス、キューサイズはそれぞれ 0ms、0%、99 個に設定した。測定結果を図 4.10 に示す。また、PEER システムにおいて帯域を最大に設定した場合および PEER システムを使用しない場合の比較も行なった。PEER システムにおいて帯域、遅延、パケットロス、キューサイズをそれぞれ 999KBytes/sec、0ms、0%、99 個に設定した場合は 369KBytes/sec であり、PEER システムの機能を使用しない場合は、430KBytes/sec であった。

遅延制御機能の評価では ping コマンドを用いて RTT[5] を測定した。各インターフェイスにおいて遅延を設定した後に、ping コマンドを用いて RTT を測定した。各インターフェイスにおいて遅延を設定する際に、帯域、パケットロス、キューサイズはそれぞれ 999KBytes/sec、0%、99 個に設定した。測定した結果を図 4.10 に示す。

以上の測定結果から帯域制御機能および遅延制御機能の両者とも意図した通りに機能し

[5] Round Trip Time

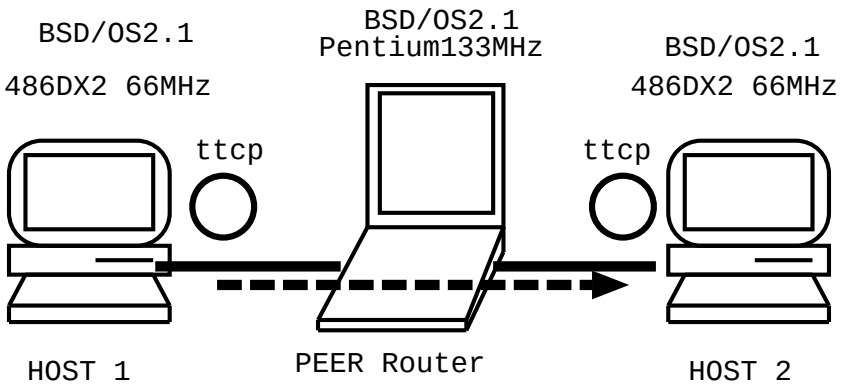

図 4.9 実験環境

ていることが分かる。これらの機能の実現にはカーネルのタイマーを使用しているため現状では 10msec 未満の精度によるパケットの制御はできない。このため PEER システムの限界はハードウェアの性能の他にもカーネルのタイマーに依存していることになる。

以下では $M/D/1$ モデルとして PEER システムが正しく機能しているかを検証する。

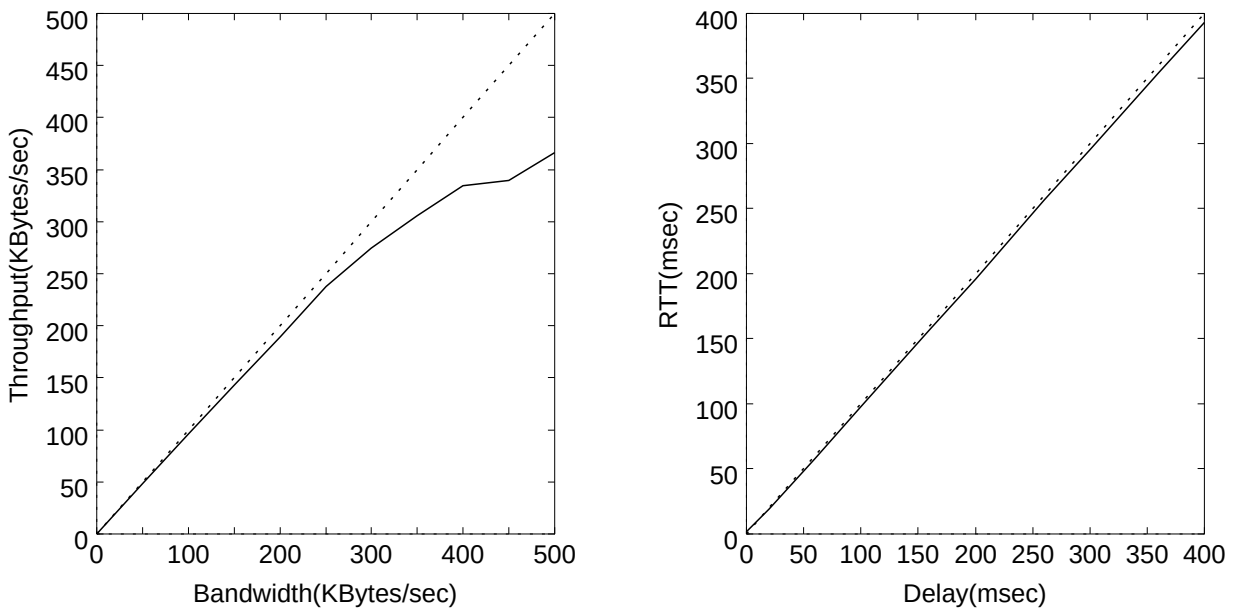

図 4.10 PEER の性能評価

4.9 実測結果と理論値の比較

PEER システムを伝送制御機構の検証システムとして使用するためには、PEER システムが意図した通りに 1 段のルータを介したネットワークをエミュレートできることを保証する必要がある。そこで、PEER システムの前章で述べた $M/D/1$ モデルの再現性を検証した。

4.9.1 系内時間の測定

PEER システム自体の検証を行なうために、理論値による系内時間の分布と実測値による系内時間の分布を比較検討する。

系内時間を求めるために、PEER システムにおいて各種の帯域および遅延を設定した後に、UDP パケットをランダム間隔で発生さるプログラムを用いて、PEER システムを介した通信を行ない系内時間の測定を行なった。各 UDP パケットのデータ部分にはシーケンス番号と送出時刻が入れられており、受信側ではそのデータと現在の時刻を比較することにより系内時間を求める。実験環境には、4.8 で述べた PEER システムの性能評価のときに用いた環境と同一の環境を使用した。ただし、系内時間を求めるために両端のマシンの時刻が正確である必要があるため、NTP[6] を用いて両端のマシンの時刻を正確に合わせた後に測定を行なった。

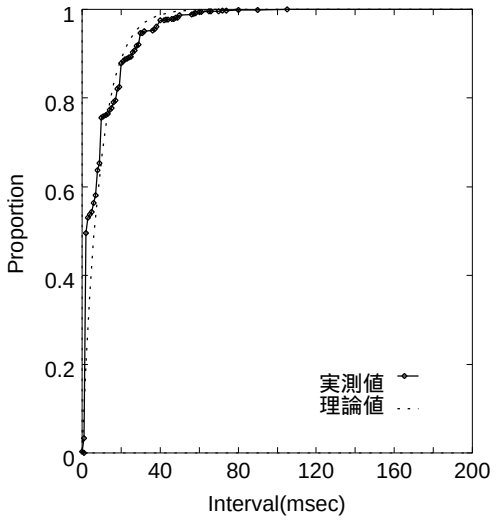

図 4.11 UDP パケットの到着間隔の分布

UDP パケットの送出には単位時間当たりのパケットの送出量を調整でき、かつパケットの発生間隔をランダムにできるように変更を加えた ttcp を使用した。パケットの送出量はパラメータを変えることにより変更できるようにし、今回の実験では 4 種類の異なるパラ

[6] Network Time Protocol

メータを使用した。事前にこの 4 種類のパラメータにおける UDP パケットの到着間隔を測定し、それぞれの単位時間あたりのパケット発生率λを求めた。一例としてパラメータを 2 に設定したときの理論値および実測値の到着間隔の分布を図 4.11 に示す。理論値は、λを 1.10×10^{-1} として $H(t) = 1 - e^{-\lambda t}(t \geq 0)$ により求めたものである。上記の 4 種のパラメータに関して、UDP パケットの到着間隔がランダムであることとを確認すると同時に各パラメータにおけるパケット発生率λを求めた。各パラメータにおけるパケット発生率λは以下の表の通りである。

パラメータ	パケット発生率λ
1	6.07×10^{-2}
2	1.10×10^{-1}
3	1.61×10^{-1}
4	2.02×10^{-1}

上述の方法によりさまざまな帯域および遅延における系内時間を測定した。それぞれパケットサイズが 1460Bytes である UDP パケットを 2000 個使用して測定を行なった。帯域に関しては 100KBytes/sec から 400KBytes/sec の間で 100KBytes/sec おきに測定を行ない、それぞれ ttcp のパラメータには 1,2,3,4 を使用した。また、各帯域において遅延を 0msec,50msec,100msec, 150msec にして 4 種類に対して測定を行なった。

4.9.2 実測結果と理論値の比較

上記の実験により求めた実測結果と理論値の比較検討について述べる。まず、実測結果から PEER システムの系内時間の分布を求めた。各帯域について求めた系内時間の分布を図 4.12 に示す。各グラフの最も左側に位置する太線が理論値である。この理論値は 3.2 の式 (3.2) および式 (3.3) において遅延時間 d を 0、ルータにおけるパケット処理時間κ を 1.12、パケットサイズ S を 1460 として求めたものである。ただし、パケットの発生率λおよび帯域 b はそれぞれ実験に用いた値を使った。

図 4.12 における理論値と実測値の比較について述べる。理論値は遅延時間 d を 0 として計算したものであり、実測値は、PEER システムにおいて遅延時間を 0msec,50msec,100msec, 150msec として求めたものである。したがって、理論値と実測値の比較は、理論値のグラフを時間軸に沿って平行移動することにより行なう。伝送媒体の遅延および両端のプロセスの処理時間を考慮に入れると、図 4.12 から理論値と比較してほぼ正しく PEER システムが動作していることが分かる。割合が 0.9 を超えたあたりからの誤差が非常に大きくなっているが、これは理論値では到着間隔を無限大まで考えていることによるものであり、無視しても問題はないと考えられる。

しかし、グラフからは理論値と一致しない部分もいくつか見られる。そこで、これらの原因として考えられるものを列挙しておく。

- UDP パケットの到着間隔の理論値との誤差による影響
- PEER システムにおけるカーネルタイマーの粒度による影響

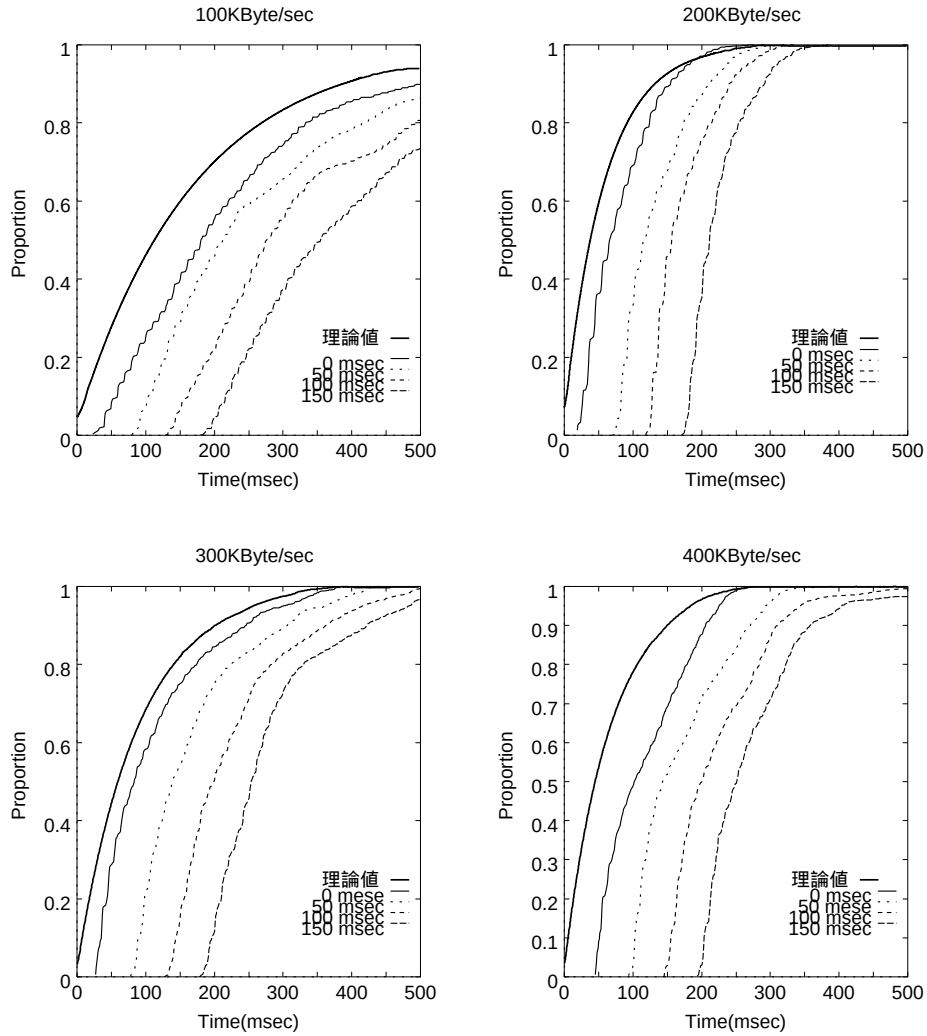

図 4.12 系内時間の分布

- PEER システム外の伝送媒体による影響
- ルータにおけるパケットの処理時間のばらつきによる影響

以上のように今回実装した PEER システムは意図した 3 章で述べた 1 段のルータを介したネットワークをほぼ正しくエミュレートでき、伝送制御機構の評価方式として妥当であることが示された。

4.10 考察

ここでは、PEER システムの考察について述べる。

4.10.1 実トラフィックの影響

今回の実装では、1 ホップの伝送路のエミュレートを実現した。しかし、モデル化のところで述べたようにネットワークに流れるトラフィックのことも考慮する必要がある。それらの影響も考えて伝送制御機構を検証できることが望ましい。そこで、tcplib[34][27] などを参考にして、トラフィックを発生させるメカニズムも今後は検討する必要がある。

4.10.2 伝送制御機構の検証結果の比較方法

PEER システムを用いることによりさまざまな 1 ホップにおける伝送制御機構を検証することが可能になった。現在の PEER システムでは、各トランスポートやアプリケーションの実装のデータ転送を観測するだけに留まっている。今後、さらにさまざまな伝送制御機構を検証することを考えると結果を比較できるようにすることが必要である。そこで主に以下の点を考慮して、伝送制御機構の検証結果を定量的に扱えるようにする必要がある。

- 利用可能な帯域をどれだけ効率的に使えているか。
- 帯域を越えた量のパケット送出によるパケットロスの発生がいかに少ないか。

さらに、各実装の伝送制御機構を検証するために測定の基準も決める必要がある。今後は、測定の基準となるネットワーク環境を決める必要があると考えられる。

4.10.3 ユーザインターフェイス

現在の PEER システムでは、テータ転送の様子に関しては ivisual コマンドを使用することにより視覚的に観察することができる。さらに詳しく解析するためには結果をダンプしたファイル編集した後に解析することが可能である。今後、検証を手軽に行なうためにも、この結果を自動的に処理して分かりやすく表示するユーザインターフェイスが必要であると考えられる。

また、現在の PEER システムでは、PEER ルータを制御するコマンドラインによるインターフィスしか用意されていない。ユーザの利便性を考えると GUI による操作法も用意することが望ましいと考えられる。

4.10.4 機能の拡張

伝送制御機構の検証のための機能の拡張として以下の 2 つの機能を追加する必要があると考えられる。一つ目はパケットの複製機能である。インターネットでは、パケットの重複が起こる可能性がある。このためこれらを用いた検証を行なうためにパケットの複製機能を PEER に組み込む必要がある。二つ目はパケットのデータ操作機能である。現在のインターネットはセキュリティーに関してはあまり配慮されていないため、パケットの盗聴や改ざんは比較的容易にできる状態にある。このため、これらにまで対応したアプリケー

ションやトランスポートの検証まで行なうことを考えると、この機能が必要であると考えられる。

第5章

PEERシステムを用いたTCPの検証

本章では、PEER システムを用いたさまざまなオペレーティングシステムにおける TCP の検証について述べる。

5.1 PEERシステムによる検証

前章までに述べたように、PEER システムでは 1 段のルータを介したさまざまなネットワークを正しくエミュレートできる。すなわち、一定の固定的な帯域および遅延をもった伝送路を正しくエミュレートできる。さらにパケットロスをエミュレートできる。そこで、これらの特性を利用して、さまざまなオペレーティングシステムにおける TCP の検証を行なう。

5.2 TCPの検証

TCP はインターネット上で最も重要なトランスポートプロトコルである。現在のインターネット上のサービスの大部分は、この TCP を使用して構築されている。UDP がコネクションレスの信頼性のない通信を提供するの対して、TCP は信頼性のあるコネクション指向のストリーム型通信を提供する。TCP の特徴は以下の通りである。

- ストリーム通信
- コネクション指向
- バッファ付き転送
- 非構造化ストリーム
- 全二重コネクション

TCP は信頼性のある通信を効率的に行なうための伝送制御機構をである。 TCP は、RFC793 [2] で標準化されたトランスポートプロトコルである。しかし、 TCP は従来から

用いられている低速なネットワーク上で発達してきたため、TCP の利用が想定されていなかった高速なネットワーク上でのさまざまな問題点が指摘されている。具体的には、データリンクがどんなに広帯域になっても限られた帯域でしかデータ転送を行なわないことや、ネットワークの帯域が十分に余裕があっても間欠的にしかデータが送出されない場合があることなどが明らかにされている。このような問題に対処するために RFC793 以降も改良を目的とした RFC1122 などの数多くの RFC や I-D が発行されている [35][9]。

このように、RFC793 以降も TCP にはさまざまな仕様の追加や改訂が行なわれており、また、仕様の中でもすべて厳密に規定されているわけではなく実装者に任されている部分も多い。そのため、このようなことからさまざまな実装における TCP を RFC に準拠しているかどうかや各実装による違いを検証する必要がある。

5.3 関連研究

TCP に関する研究は従来から多く行なわれている。

シミュレーションによる方法の代表的なものとしては A.Heybey らによる ns[10] や S.Keshav らによる REAL[11] などが挙げられる。これらは主に TCP のアルゴリズムや伝送メカニズムの検証に用いられており、例えば SACK[36] や RED[20] などのような新しい手法を採り入れた場合に、どのような転送状況になるかを手軽に観測することができる。

実測による方法では、主に 2 つに分類することができる。一つは通信のパケットのトレースを行なうものである。これには V.Paxson による Tcpanaly[15]、S.Ostermann による Tcptrace、G.Minshall による Tracelook、Tim Shepard による Xplot[16] などが挙げられる。この方法では実際のネットワークにおける TCP のデータ転送の様子を観測することができる。そして、もう一方はデータ転送のスループットを測定するものである。これには T.C.Slattery による ttcp、R.Jonkman による netspec[12]、R.Jones による Netperf[13]、村山らによる DBS[14] などが挙げれられる。この方法では実際のネットワークにおいて TCP を使用してデータ転送を行なった場合のスループットを測定することができる。

また、x-kernel[18] のような新たなプロトコルや新しい手法を実験および開発するためのプラットホームを提供するシステムもある。この x-kernel を利用したプロトコルの検証システムとして ORCHESTRA[37] [38][39] と呼ばれるシステムもある。この ORCHESTRA では、x-kernel 上においてスクリプト言語を使用してパケットの操作を行ない、そのプロトコルの検証を行なうことができる。

また、TCP は大変複雑なプロトコルであるため実装も難しく、間違いも起こしやすい。このため実装上の問題点も指摘されている [40]。また、新しい手法や、性能向上を目指した研究としては TCP Vegas [41] や SACK などが挙げられる。

これらの TCP に関係する研究を伝送制御機構の検証という観点からみると、x-kernel を除くと全てが TCP に特化しており、さらに検証できる機能もそれぞれ限定されていると言える。また、ORCHESTRA に関しても、x-kerenel 上で動作するアプリケーションやトランスポートを使わなければならないため、この点で比較しても筆者のアプローチの方が優れていると考えられる。

5.4 検証項目

TCP にはフロー制御、再送制御、輻輳回避制御の 3 つの制御機構があり、これらは TCP の性能を決定する大きな要因となっている。フロー制御は、終点間で最大のスループットが得られるように制御する機構である。再送制御は、セグメントが喪失したかどうかをできるだけ素早く予測してセグメントの再送を制御する機構である。そして、輻輳回避制御はネットワークの混み具合に合わせてセグメントの送信量を制御しセグメントの喪失を防ぐ機構である。これらの 3 つの制御機構が協調して動作することにより TCP はネットワークを有効に利用し、かつ最大限のスループットを得る努力をする。そこで、これらの 3 つの制御機構に関して以下の 6 つの代表的な機能の検証を行なう。これらの 6 つの機能は TCP を実装する際の問題点としても指摘されている [42]。各機能の詳しい説明はそれぞれのところで述べる。

1. 再送制御
   - Retransmission Timeout
   - Retain Above-Sequence Data
2. フロー制御
   - Initial Slow Start
   - Zero Window Probe
3. 輻輳制御
   - Slow Start After Retransmission
   - Fast Retransmit and Fast Recovery

5.5 実験に使用した機材とツール

今回の TCP の検証実験では、BSD/OS3.0、SunOS4.1.4、Solaris2.5.1、Windows95 の 4 つのオペレーティングシステムに関して検証を行なった。実験に用いた機材は以下の通りである。

オペレーティングシステム	本体	CUP	Memory
BSD/OS3.0	DOS/V	486DX2 66MHz	16MB
SunOS4.1.4	SparcStation10	SuperSparc 40MHz	32MB
Solaris2.5.1	DOS/V	Pentium200MHz	32MB
Windows95	DOS/V	Pentium100MHz	24MB

実験環境は 4.8 節で述べた PEER システムの評価実験のときと同じである。ただし、Host1 の部分に検証するオペレーティングシステムに応じた計算機を配置した。データ転送には ttcp を使用した。基本的にはメッセージサイズを 1460Bytes に設定し、1000 個のパケットを転送した。各検証のために ttcp に変更を加えた部分もあるが、変更内容に関しては各検証のところで必要に応じて説明する。

5.6 Retransmission Timeout (RTO)

5.6.1 検証内容

TCP では信頼性のある通信を行なうためにセグメントが喪失した場合には再送を行なう。 TCP では、各セグメントが送出されるたびにそれぞれ再送タイマがセットされる。このタイマが切れるまでにそのセグメントに対する ACK が来ない場合にはそのセグメントは喪失したと仮定する。そして、 TCP はそのセグメントを再送した後に、また新たに再送タイマをセットする。これらのタイマの値はその通信の end-end 間の RTT を元にして計算される。TCP の仕様では、同じセグメントの再送間隔は指数的に増加するように設定することとされている。この再送間隔のことを RTO と呼ぶ。RFC1122 では、RTO の計算には Jacobson のアルゴリズム [43] を使うようにとされている。また、この RTO には上限が設定される場合もあるとされているが、上限の値自体は決められていない。このように RTO は、RTT を元にして経路に応じた適切な値に決められなくてはならない。さらに、再送間隔は指数的に増加しなければならない。また、再送回数に関する規定は全くない。本検証では、これらの点を検証する。

5.6.2 検証方法

上記の RTO を検証するために、次のような実験を行なった。ACK 側のパケットに遅延を入れて、ACK パケットが 30 個通過するまでは通常通り通信を行わせた。 この間に各オペレーティングシステムの TCP が、通信路の RTT に基づいて RTO の計算を行うことを期待している。PEER での遅延はそれぞれ 0,3,6,9 秒に設定した。その後に、送信側からのパケットを全て PEER ルータにおいて喪失するようにした。そして、同じシーケンス番号のパケットの再送される様子を観測した。

5.6.3 検証結果

上記の検証方法に基づいて実験した結果が以下の図 5.1 である。図 5.1 のグラフはそれぞれ下から遅延が 0 秒、3 秒、6 秒、9 秒の場合である。全てのオペレーティングシステムに関して再送間隔は上限に至るまで指数的に増加していることが分かった。上限の値は、BSD/OS3.0 および SunOS4.1.4 が 64 秒、Solaris2.5.1 が 56 秒、Windows95 が 263 秒であった。また、最初の再送までの間隔は RTT をもとにして適正な値が使われるべきである。しかし、グラフからも分かるように各オペレーティングシステムによってかなりばらつきがあることが分かった。最初の再送までの間隔を以下の表に示す。

オペレーティングシステム	遅延 0 秒	遅延 3 秒	遅延 6 秒	遅延 9 秒
BSD/OS3.0	1.4 秒	6.5 秒	7.9 秒	20 秒
SunOS4.1.4	0.5 秒	4.5 秒	6.7 秒	9.8 秒
Solaris2.5.1	0.24 秒	0.50 秒	0.48 秒	0.48 秒
Windows95	5.0 秒	7.3 秒	20 秒	27 秒

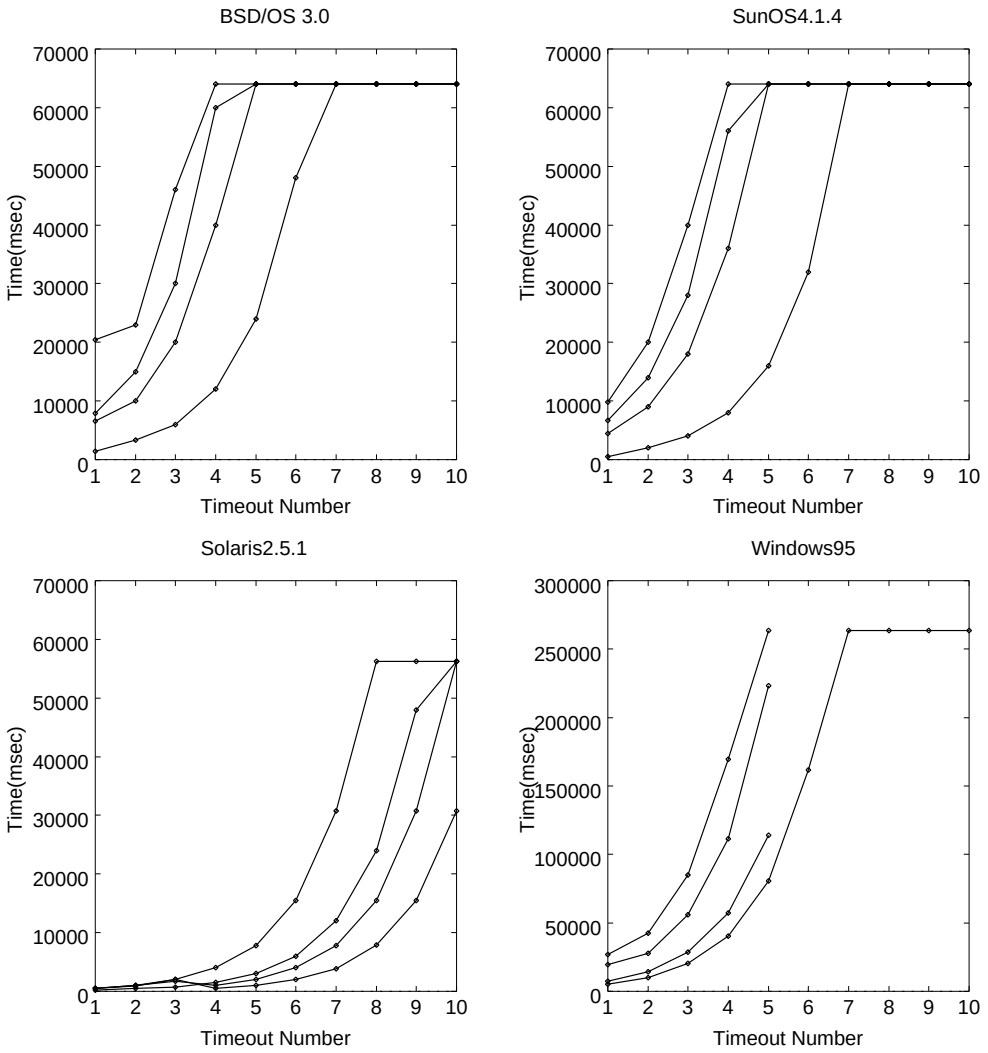

図 5.1 タイマーを用いた再送制御

この表からも分かるように SunOS4.1.4 は RTT から適切な値を使用していることが分かる。しかし、BSD/OS3.0 は、やや RTT に比べて長い値を使用している。さらに Windows95 では、BSD/OS3.0 よりも長い値を使用していることが分かる。一方、Solaris2.5.1 は、すべてにおいて極端に早い間隔で再送を行なっている。これは、再送間隔の計算に RTT を適切に使用できていないことが分かる。

また、再送回数に関しては、BSD/OS3.0 および SunOS4.1.4 が 12 回、Solaris2.5.1 が 15 回、Windows95 が 5 ~ 14 回再送したのちに通信を切断することが確認できた。

5.7 Retain Above-Sequence Data

5.7.1 検証内容

インターネットでは、パケットの損失や逆転が起こることがあるため、必ずしも順番通りにパケットが到着するとは限らない。TCP は、その点も考慮して効率を挙げるために先行到着パケットを保持すべきであるとされている。先行到着パケットとは、RCV.NXT[1] 以上 RCV.NXT+RCV.WND[2] 以下のシーケンス番号を持ったパケットである。RFC793 では、「保持する場合もある」とされていたものが RFC1122 では「保持すべきである」と変更になっている。先行到着パケットを保持しない場合には、パケットの損失がある環境において通信を行なう場合、それ自身および他の通信にも効率の面で悪影響を与える。

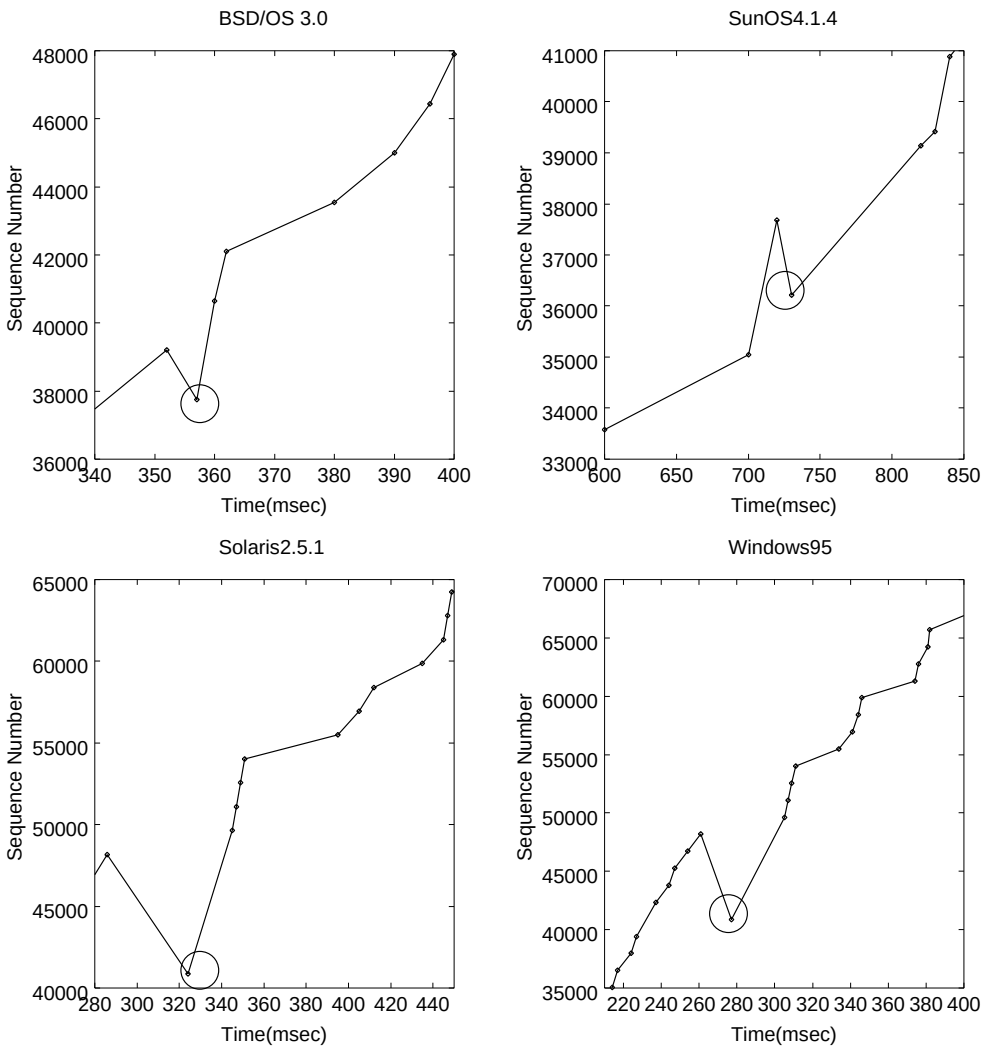

図 5.2 セグメントの順番制御

[1] 次に受け取るべきパケットのシーケンス番号
[2] 受信側のウィンドウサイズ

5.7.2 検証方法

上記の先行到着パケットを保持しているかどうかを検証するために、次のような実験を行なった。PEER ルータにおいて送信側から送出された 30 番目のセグメントと 31 番目のセグメントを順序を入れ換えて送り出すように設定した。その後に 31 番目のパケットと同じシーケンス番号のパケットが再送されたかどうかを観測する。

5.7.3 検証結果

上記の検証方法に基づいて計測した結果が以下の図 5.2 である。

図 5.2 から今回の実験に使用した全てのオペレーティングシステムにおいて先行到着パケットが保持されていることが確認できた。

5.8 Initial Slow Start

5.8.1 検証内容

TCP では、データの送信を開始する際にスロースタートと呼ばれる手法を使うことが求められている。スロースタートとは、まず最初に 1 パケット送出して、その後 ACK を受け取るたびに送出するパケットの数を増やしていく手法である。そしてパケットの喪失が発生するまで送出するパケットの数を増やしていく。 RFC1122 では、スロースタートを使うように要求している。また、RFC2001[44] ではスロースタートの詳細な仕様が述べられている。もしスロースタートを行なっていない場合には、輻輳が起きている環境の下での通信において他の通信に性能の面で大きな悪影響を与える。さらに自分自身の性能にも悪影響を与える。

5.8.2 検証方法

上記のスロースタートを使用しているかどうかを検証するために次のような実験を行なった。スロースタートの様子をはっきり観測するために ACK 側の遅延を 30ms に設定した。その後にデータ転送を開始し、送信側のパケットの送出状況を観測した。

5.8.3 検証結果

上記の検証方法に基づいて計測した結果が以下の図 5.3 である。各オペレーティングシステムにおいてパケットの送出間隔にばらつきがあるものの、すべてにおいてスロースタートを使用していることが確認できた。

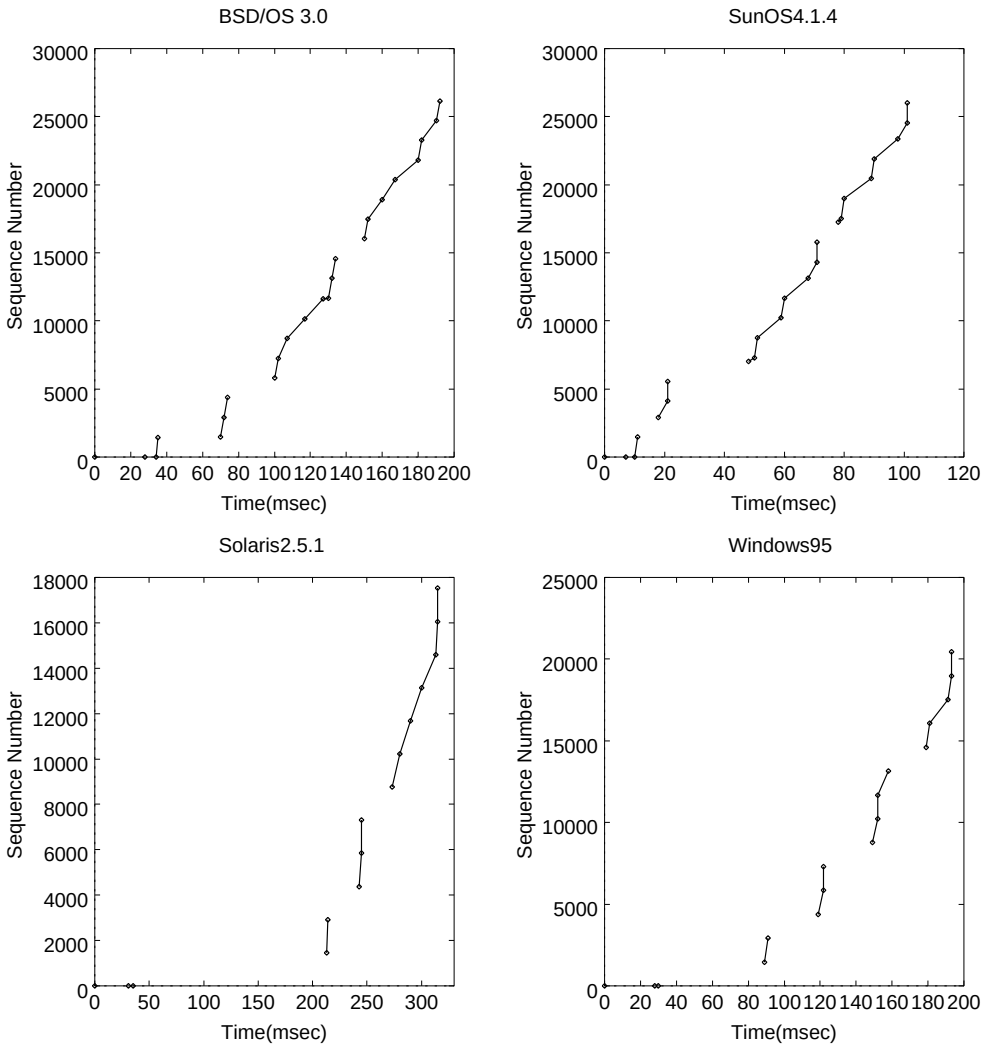

図 5.3 スロースタート

5.9 Zero Window Probe

5.9.1 検証内容

TCP では、受信側のバッファを溢れさせないように送信する量を調整することも行なわれている。 TCP の仕様では、受信側があと何オクテットのデータを受信できるかを ACK のヘッダに書き込んで送信側に伝えるようにしている。もし、受信側が受け取り可能な量以上のデータを送信側が送出すると受信側では、それらは破棄されてしまう。ゼロウィンドウプローブとは、受信側のバッファがいっぱいになっても送信側からは何らかのセグメントを送出して ACK を受け取りバッファが空くのを待つ機能である。RFC では、ゼロウインドウプローブは必ずサポートしなくてはならないとされている。なぜならばウィンドウが再び開いたことを示す ACK セグメントは喪失する可能性があるからである。もしゼ

ロウインドウプローブがサポートされていない場合には ACK セグメントが喪失した場合に、永遠に通信がハングアップすることになる。

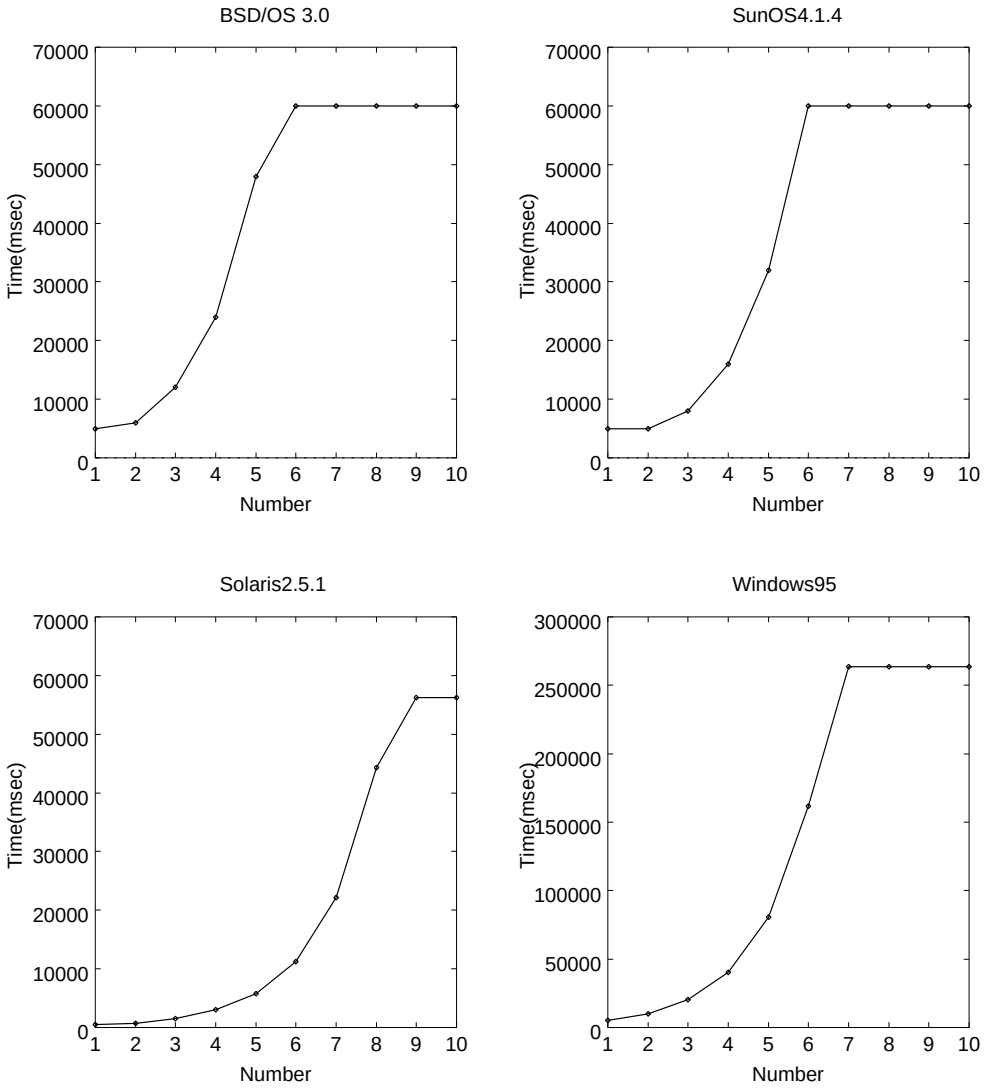

図 5.4 ゼロウィンドウプローブ

5.9.2 検証方法

上記のゼロウィンドウプローブを使用しているかどうかを検証するために次のような実験を行なった。受信側のプロセスにおいて read システムコールを実行しないようにした。このようにすることによって送信されたパケットはすべて TCP のバッファに溜められていき、そのうち TCP のバッファが一杯になる。その後の送信側のゼロウィンドウプローブの送信状況を観測した。

5.9.3 検証結果

上記の検証方法に基づいて計測した結果が以下の図 5.4 である。全てのオペレーティングシステムにおいて、ゼロウィンドウプローブの間隔は指数的に増加していることが確認できた。この点では RTO と同じ計算方法が用いられていることが分かる。また、このゼロウィンドウプローブの間隔にも上限があることが確認できた。BSD/OS3.0 および SunOS4.1.4 はともに 60 秒であった。また、Solaris2.5.1 は 56 秒、Windows95 は 263 秒であった。Solaris2.5.1 と Windows95 に関しては RTO の上限と全く同じであることが確認できた。また、ゼロウィンドウプローブの再送回数に関しては Solaris2.5.1 を除く全てのオペレーティングシステムは 60 分以上再送し続けることを確認した。Solaris2.5.1 では 14 回再送した後にコネクションを切ることが確認された。

5.10 Slow Start After Retransmission

5.10.1 検証内容

TCP では輻輳回避のためにいくつかのメカニズムを用意されている。その一つとして、タイムアウト後にセグメントの再送を行なった場合にはスロースタートを行なうようにしている。このメカニズムが用意されていない場合には輻輳が発生した状況において、他の通信に性能の面で大きな悪影響を与える。さらに自分自身の性能にも悪影響を与える。

5.10.2 検証方法

上記のタイムアウト後の再送後にスロースタートを使用しているかどうかを検証するために次のような測定を行なった。まず送信側の 10 番目のセグメントを故意に喪失してその後の送信の様子を観測した。ただし、次に述べる早期回復や早期再送による影響を受けないようにするために、同じ ACK シーケンス番号をもった ACK セグメントは送信側に送らないようにした。

5.10.3 検証結果

上記の検証方法に基づいて計測した結果が以下の図 5.5 である。すべてのオペレーティングシステムにおいて、タイマーを用いた再送を行なった後はスロースタートを使用していることが確認できた。RTO のところでも述べたように各オペレーティングシステムによって再送までの間隔が違うことも分かる。

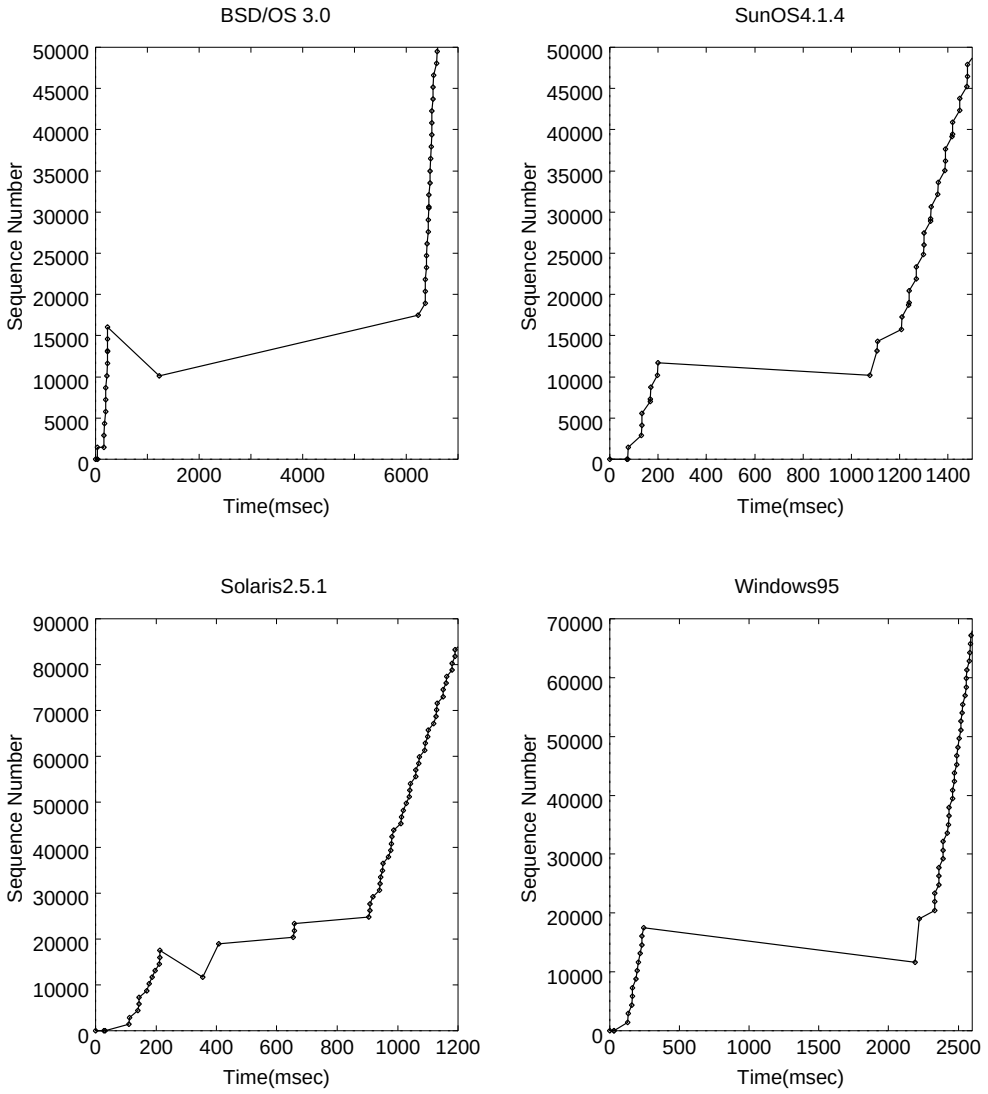

図 5.5 タイマーを用いた再送後のスロースタート

5.11 Fast Retransmit and Fast Recovery

5.11.1 検証内容

TCP におけるデータ転送の性能を上げるために RFC2001 において早期再送および早期回復と呼ばれるメカニズムが提案されている。早期再送は、3 個以上の重複した ACK (duplicate ACK) が続けて返ってきたら、それはセグメントが失われた可能性が高いので TCP は再送タイマーが切れるのを待たずに失われたと思われるセグメントの再送を実行するメカニズムである。また、早期回復は失われたと思われるセグメントを再送した後には、スロースタートではなく、輻輳回避を実行するメカニズムである。この制御によりそこそこの輻輳の下で (特にウィンドウサイズが大きい時に) 高いスループットが実現可能と

なる。そのため、に早期再送および早期回復が実装されていると輻輳時においてもデータ転送性能の効率が良いことが予想される。

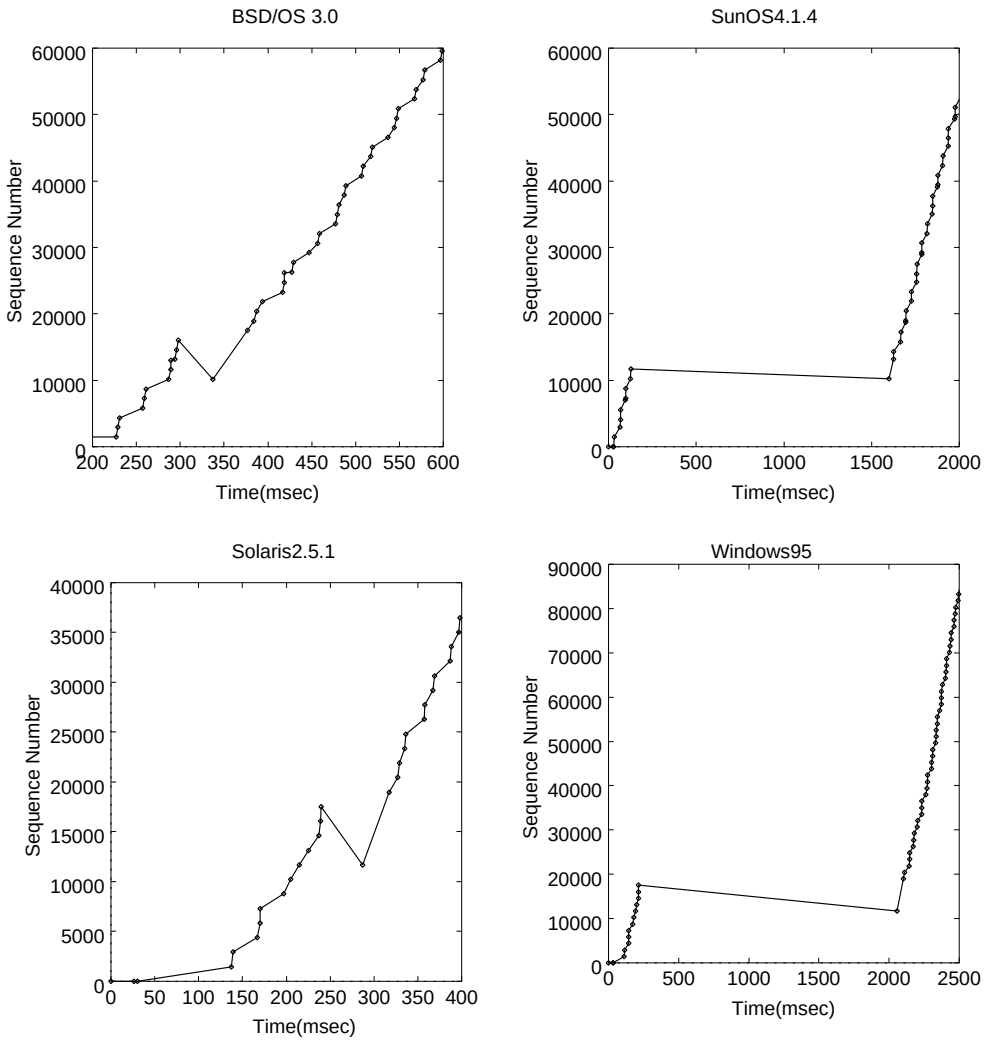

図 5.6 早期回復

5.11.2 検証方法

上記の早期再送および早期回復を使用しているかどうかを検証するために次のような実験を行なった。まず送信側の 10 番目のセグメントを故意に喪失してその後の送信の様子を観測した。また、さらに受信側からは 3 個以上の重複した ACK(duplicate ACK) を送信していることを確認した。

5.11.3 検証結果

上記の検証方法に基づいて計測した結果が以下の図 5.6 である。BSD/OS3.0 および Solaris2.5.1では早期再送および早期回復が実装されていることが確認できた。しかし、SunOS4.1.4 および Windows95 に関しては図 5.5 と似た結果になっており、早期再送および早期回復が実装されていないことが確認できた。

5.12 TCPの検証結果

ここでは、上記の検証結果についてまとめる。本章では４種類のオペレーティングシステムに関して、その TCP の検証を行なった。検証は項目は TCP の重要な機能であるフロー制御、再送制御、輻輳回避制御の中から６つを厳選した。検証結果を以下の表に示す。その機能が正しく機能しているものには○ 、正しく機能しているが各オペレーティングシステム間で違いがあるものには△ 、機能していないものには× を付けた。

TCP の機能の種別	BSD/OS3.0	SunOS4.1.4	Solaris2.5.1	Windows95
Retransmission Timeout	△	△	△	△
Retain Above-Sequence Data	○	○	○	○
Initial Slow Start	○	○	○	○
Zero Window Probe	△	△	△	△
Slow Start After Retransmission	○	○	○	○
Fast Retransmit and Fast Recovery	○	×	○	×

以上の検証結果から今回の実験に使用したオペレーティングシステムはすべての機能において比較的良く機能していることが確認できた。しかし、オペレーティングシステム間の結果の違いから以下のようなことが推測できる。

- Solaris2.5.1は、再送までのタイムアウトを RTT をもとにして正しく計算できていないため、RTT の大きい相手と通信する場合には、無駄な再送を多く行なっていると予想できる。このため帯域の無駄使いをしていることが予想できる。(付録 D)
- Windows95は、再送までのタイムアウトが RTT に比べて非常に長いことが分かった。さらに、早期再送および早期回復の機能を持っていないため、パケットの喪失がおこる環境における通信では、帯域が十分広くても非常に効率悪いことが予想される。同様のことが BSD/OS3.0 にも言えるが、こちらは早期再送および早期回復の機能を持っているために Windows95 ほどは効率が低下しないと思われる。

第6章

今後の展望

本章では本研究の今後の展望について述べる。PEER システムを実験のプラットホームとして考えた場合のいくつか有効な適用例を挙げ、広域分散環境におけるさまざなま実験への本提案方式の有効性、可能性を検討する。

6.1 トランスポートやアプリケーションの開発

トランスポートやアプリケーションの開発段階において、実際の回線を使用すると転送効率などの低下がプロトコルの問題もしくは回線やネットワークの転送状態の変化のいずれによるものなのかを考慮しなければならず、開発効率を下げる要因となる。また、開発段階の不手際によるネットワークへの影響も考える必要がある。しかし、このような開発段階において PEER システムを使用すると、上記の問題を全て解決でき、しかもデータ転送の様子も簡単に観測することができる。また、x-kernel の利用と合わせることにより新しいトランスポートの開発に非常に有用な手段となると考えられる。

6.2 衛星通信

近年では、衛星通信を使ったインターネットの実験が盛んに行なわれている。衛星通信の特徴の一つとして広帯域、高遅延というものが挙げられる。これらの実験では従来はシミュレーションによる方法、もしくは実際に衛星通信を使った検証が行なわれてきた [45][46][47]。しかし、これらの衛星通信を使って実際に実験ができるのは一部の限られた人のみである。そこで、PEER システムを用いることによって、実際のネットワークを使わなくても誰でも簡単に実験が行なえるようになる。

6.3 非対称リンクの実験

通信の経路として、上りと下りの異なるリンクを非対称リンクと呼ぶ。このような通信が最近注目されている。このような通信の例としては、大容量のデータを送る方向のリン

クに衛星通信や無線通信を使い、逆方向のリンクに細い有線やPHSなどを使うものが主流である。このような伝送路は帯域幅の違いや伝送遅延の違いなどさまざまな特徴があり、これらの実験を行なうために従来では、実際のリンクを使用したりイーサネットとシリアル通信を混同させて疑似的な環境をつくる必要があった[48][49]。しかし、PEERシステムを用いることによって、実際のネットワークを使わなくても簡単に実験が行なえるようになる。

6.4 マルチメディア実験

インターネットを使った音声や画像などを交換するマルチメディアアプリケーションと呼ばれる応用が考えられてる。これらの通信は特に遅延や損失などの影響を受けやすく、それに対する対策をしているものが多い。しかし従来の方法では実際のネットワークを使っての評価が多かったために定性的な評価を行なうことが出来なかった[50]。本提案方式のPEERシステムを用いれば、意図的な環境を作ることが可能であるため定性的な実験を行なうことが可能である。

第7章

まとめ

本論文では、ネットワークをエミュレートし、その上で伝送制御機構を検証することの有効性および妥当性を示した。

まず、はじめにインターネットを使用するアプリケーションやトランスポートの伝送制御機構を検証する必要性について述べ、インターネットをエミュレートして、その上で検証する方法を提案した。この方法ではインターネットをいかにしてエミュレートするかが最大の問題である。そこで、まずインターネットの網としての特性を検討し、1ホップは$M/D/1$でモデル化できることを示し、理論解析を行なった。実際のネットワークと比較することによりこのモデルの妥当性を示した。このモデルにもとづいてインターネットの1ホップをエミュレートするPEERシステムを提案し、設計および実装を行なった。PEERシステムによりさまざまなネットワークをエミュレートして伝送制御機構を検証することが可能になった。そして、これらをふまえ、PEERシステムを用いた伝送制御機構の検証の一例としてさまざまなオペレーティングシステムにおけるTCPの検証を行なった。

最後に本研究への将来への希望について述べる。現在のインターネットではWWWや電子メールなどの比較的通信量の少ない通信が主流ある。しかし今後、計算機の高速化および広帯域ネットワークの広域にわたる普及によって、音声や映像といった連続メディアを広域ネットワークで送受信するアプリケーションが出現し、実用的に使われることが予想される。このような時代には、今よりもさらに伝送制御機構の果たす役割が重要になると考えられる。本研究が将来においてもインターネットの公平性および有効利用に役立つことを望む。

謝辞

本研究を進めるにあたり構想段階から多大な支援をいただいた国立がんセンターの中川晋一氏およびソニーコンピュータサイエンス研究所の長健二朗氏に感謝します。理論解析から論文作成にいたるまで多大な御協力をいただいた東京工業大学の山岡克式氏に感謝します。また、このような研究の機会を与えて下さったWIDEプロジェクトにも感謝します。日頃からいろいろとお世話になった大野研究室および西七号館の皆さんに感謝します。特に、この3年間共に研究活動を行なってきたしてきた門間君、野田君、田中君にも感謝します。

そしてなにより、本当に熱心に指導していただいた大野浩之講師に心から感謝します。

参考文献

[1] 本庄利守, 中川晋一, 大野浩之, 長健二朗. データ転送性能評価システム PEER の構築. インターネットコンファレンス'97 論文集. 日本ソフトウェア科学会, 日本 UNIX ユーザ会,WIDE プロジェクト, December 1997.

[2] J. Postel. Transmission Control Protocol. Technical report, Request for Comments 793, September 1981.

[3] J. Postel. User Datagram Protocol. Technical report, Request for Comments 768, August 1980.

[4] D.R.Cheriton. VMTP: A transport protocol for the next generation of communication system. In *Computer Communication Review*. ACM, August 1986.

[5] David R. Cheriton. VMTP: Versatile Message Transaction Protocol Specification. Technical report, Request for Comments 1045, February 1988.

[6] W.Timothy Strayer, Bert J. Dempsey, and Alfred C. Weaver. *XTP: The Xpress Transfer Protocol*. Addison Wesley, 1992.

[7] Henning Schulzrinne. RTP: The real-time transport protocol. In *MCNC 2nd Packet Video Workshop, Vol. 2*. Research Triangle Park, North Carolina, December 1992.

[8] Claudio Topolcic et al. Experimental Internet Stream Protocol, Version 2(ST-II). Technical report, Request for Comments 1190, October 1990.

[9] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. Technical report, Request for Comments 1323, May 1992.

[10] A.Heybey. The network simulator. Technical report, September 1990.

[11] S.Keshav. REAL: A Network Simulator. Technical report, 1988.

[12] Roel Jonkman. NetSpec: A Network Performance Evaluation and Experimentation Tool, 1995.

[13] Hewlett-Packard Company. Netperf: A Network Performance Benchmark Revision 2.1, 1995.

[14] 村山公保, 門林雄基, 山口英. TCP 性能評価システムDBS の構築. インターネットコンファレンス'96 論文集, pp. 39–44. 日本ソフトウェア科学会, 日本 UNIX ユーザ会,WIDE プロジェクト, July 1996.

[15] Vern Paxson. Automated Packet Trace Analysis of TCP Implementations. In *ACM SIGCOMM '97*, September 1997.

[16] Tim Shepard. TCP Packet Trace Analysis. Technical report, MIT Laboratory for Computer Science MIT-LCS-TR-494, February 1991.

[17] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. In *ACM Computer Communication Review, Vol.27 No.1*, pp. 31–41, January 1997.

[18] N.C.Hutchinson and L.L.Peterson. The x-kernel: An Architecture for Implementing Network Protocols. In *IEEE Transactions on Software Engineering 17(1)*, pp. 64–76, january 1991.

[19] Elliot Limin Yan. The Design and Implementation of an Emulated WAN. Technical report, Computer Science Department University of Southern California Los Angeles, CA 90089, 1995.

[20] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. In *IEEE/ACM Transactions on Networking, Vol.1 No.4*, pp. 397–413, August 1993.

[21] Sally Floyd and Van Jacobson. Link-sharing and resource management models for packet networks. In *IEEE/ACM Transactions on Networking, Vol.3 No.4*, August 1995.

[22] Leonard Kleinrock. *Queuing Systems - Theory, Vol.1.* Wiley-Interscience, New York, 1995.

[23] Leonard Kleinrock. *Queuing Systems - Compute Applications, Vol.2.* Wiley-Interscience, New York, 1996.

[24] Will E.Leland, Murad S. Taqq, Walter Willinger, and Daniel V. Wilson. On the Self-Similar Nature of Ethernet Traffic. In *IEEE/ACM Transactions on Networking, Vol.2 No.1*, pp. 1–15, February 1994.

[25] Vern Paxson and Sally Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. In *IEEE/ACM Transactions on Networking, Vol.3 No.3*, pp. 226–244, June 1995.

[26] Vern Paxson. Empirically-Derived Analytic Models of Wide-Area TCP Connections. In *IEEE/ACM Transactions on Networking, Vol.2 No.4*, pp. 316–336, August 1994.

[27] Peter B. Danzig and Sugih Jamin. tcplib: A Library of TCP Internetwork Traffic Characteristics. Technical report, USC-CS-91-495,Computer Science Department, 1991.

[28] 森村英典, 大前義次. 応用待ち行列理論. 日科技連, 1975.

[29] 秋山稔. 通信網工学. コロナ社, 1981.

[30] 川島幸之助, 町原文明, 高橋敬隆, 斎藤洋. 通信トラヒック理論の基礎とマルチメディア網. 社団法人 電子情報通信学会, 1995.

[31] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated, Volume2, The Implimentation.* Addison Wesley, 1996.

[32] Marshall Kirk McKusick, Keith Bostic, Michael j. Karels, and John S. Quarterman. *The Design and Implementation of the 4.4BSD Operating System.* Addison Wesley, 1996.

[33] J. Ousterhout. *Tcl and Tk Toolkit.* Addison Wesley, 1994.

[34] Peter B.Danzig, Sugih Jamin, Ramon Caceres, Danny J.Mitzel, and Deborah Estrin. An Empirical Workload Model for Driving Wide-Area TCP/IP Network Simulations. In *The Journal of Internetworking: Practice and Experiences*, 1992.

[35] R. Braden. Requirements for Internet Hosts – Communication Layers. Technical report, Request for Comments 1122, October 1989.

[36] Matthew Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP Selective Acknowledgement Options. Technical report, Request for Comments 2018, October 1996.

[37] Scott Dawson and Farnam Jahanian. Probing and Fault Injection of Distributed Protocol Implementations. Technical report, University of Michigan CSE-TR-217-94,EECS,Department, 1994.

[38] Scott Dawson, Farnam Jahanian, and Todd Mitton. ORCHESTRA: A Fault Injection Environment for Distributed Systems. Technical report, University of Michigan CSE-TR-318-96,EECS,Department, 1996.

[39] Scott Dawson, Farnam Jahanian, and Todd Mitton. Experiments on Six Commercial TCP Implementations Using a Software Fault Injction Tool. Technical report, University of Michigan CSE-TR-298-96,EECS,Department, 1996.

[40] L.S.Brakmo and L.Peterson. Performance Problem in BSD4.4 TCP. Technical report, Department of Computer Science University of Arizona Tucson, AZ 85721, 1994.

[41] L.S.Brakmo, S.W.O'Malley, and L.Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proceedings of SIGCOMM'94*, pp. 24–35, August 1994.

[42] V. Paxson and S.Dawson. Known TCP Implementations Problems. Technical report, draft-ietf-tcpimlp-prob-01.txt, Internet Draft, July 1997.

[43] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of SIGCOMM'88 ACM*, pp. 314–332, August 1988.

[44] W. Stevens. TCP Slow Start, Congestion Avoidance,Fast Retransmit, and Fast Recovery Algorithms. Technical report, Request for Comments 2001, January 1997.

[45] Mark Allman, Chris Hayes, Hans Kruse, and Shawn Ostermann. TCP Performance over Satellites Links. In *Proceedings of the Fifth International Conference on Telecommunications Systems*. Nashville, TN, March 1997.

[46] WIDE プロジェクト. 1994 年度 WIDE プロジェクト研究報告書. Technical report, WIDE, 1994.

[47] 西田佳史, 峯尾淳一, 寺岡文男, 中村修. Wish におけるインターネットトラフィックの解析. 情報処理学会第 49 回 (平成 6 年後期) 全国大会論文集 (1). 情報処理学会, September 1994.

[48] 加藤紀康, 鎌形映二, 伊藤隆治. 非対称移動通信リンクのための TCP/IP. 情報処理学会研究報告. 情報処理学会, May 1997. 97-MBL-1.

[49] 加藤紀康, 鎌形映二, 伊藤隆治, 斧澤陸. マルチメディア非対称無線ネットワーク (AWL) システムの試 作と評価. マルチメディア, 分散, 協調とモーバイルワークショッ プ論文集. 情報処理学会, July 1997.

[50] 中川晋一, 本庄利守, 岡村耕二, 石川光一, 小山博, 大野浩之, 山口直人. 病院内情報システムにおける mobile 端末の通信要求量の推定. 情報処理学会第 4 回モーバイルコンピューティング研究会. 情報処理学会, February 1997. 97-MBL-4.

付録 A

待ち時間分布の計算

ここでは、本文で省略した $M/D/1$ における待ち時間の計算の微分方程式の解法について述べる [28] [29][30]。

本文で省略したのは以下の微分方程式の解法である。

$$f\prime(t) = \lambda[f(t) - f(t-\tau)] \tag{A.1}$$

まずはじめに以下のような関数 $g(t)$ を導入する。

$$f(t) = e^{\lambda t} g(t)$$

この $g(t)$ によって置き換えることにより式 A.1 は以下の微分方程式に変換することができる。

$$g\prime(t) = -\lambda e^{-\lambda\tau} g(t-\tau)$$

ここでまず $0 < t \leq \tau$なる t について考える。このとき $t > 0$ であることから今まで導入した式はすべて成り立つ。ある時刻γを、平衡状態とすると $t \leq \tau$ のときには以下が成り立つ。

$$f(t-\tau) = P\{\gamma < t - \tau\} = 0$$

したがって、$g\prime(t) = 0$、すなわち $g(t) = c$(定数) となる。このことから $f(t)$ は以下のように表せる。

$$f(t) = ce^{\lambda t}$$

ここで、定数 c を決定する。$c = f(+0)$ および $f(-0) = 0$ であることから $c = f(+0) = f(-0) = P\{r = 0\}$ となる。これは任意に選んだ時点において、到着したパケットが待ち時間なしで済む確率なので $1 - \lambda\tau$となる。したがって、$0 < t \leq \tau$において f(t) の分布は以下のように表すことができる。

$$f(t) = (1 - \lambda\tau)e^{\lambda t}(0 < t \leq \tau)$$

次に $t \geq 0$ なる t に対しての分布関数を導く。そのために任意の非負の整数 n に対し区間 $n\tau < t \leq (n+1)\tau$ において関数 $g(t)$ が以下の式によって表されることを帰納法を用いて示す。

$$g(t) = (1-\lambda\tau)\sum_{k=0}^{n} e^{-\lambda k\tau}\frac{(k\tau - t)^k}{k!}\lambda^k$$

[証明]
$n = 0$ のときは上記で証明したので、成り立つことが分かる。まずある $n \geq 0$ に対して正しいと仮定し、この場合に $n+1$ の場合にも成り立つことを示せばよい。

いま、$n\tau < t \leq (n+1)\tau$ の t に対して正しいと仮定し、$(n+1)\tau < t \leq (n+2)\tau$ の t について考える。まず、以下のことが成り立つ。

$$\begin{aligned} g\prime(t) &= -\lambda e^{-\lambda\tau} g(t-\tau) \\ &= -\lambda e^{\lambda\tau}(1-\lambda\tau)\sum_{k=0}^{n=0} e^{-\lambda k\tau}\frac{(k\tau - t + \tau)^k}{k!}\lambda^k \end{aligned}$$

この式を t について $(n+1)\tau$ から t まで積分すると以下のようになる。

$$g(t) - g\left[(n+1)\tau\right] = -\lambda e^{-\lambda\tau}(1-\lambda\tau)\sum_{k=0}^{n=0}\frac{e^{-\lambda k\tau}\lambda^k}{k!} \times \int_{(n+1)\tau}^{t}\left[(k+1)\tau - u\right]^k du$$

さらに最後の因子の積分を実行すると以下のようになる。

$$\int_{(n+1)\tau}^{t}\left[(k+1)\tau - u\right]^k du = -\frac{1}{k+1}\{\left[(k+1)\tau - t\right]^{k+1} - \left[(k-n)\tau\right]^{k+1}\}$$

ゆえに

$$\begin{aligned} g(t) - g\left[(n+1)\tau\right] &= \lambda e^{-\lambda\tau}(1-\lambda\tau)\sum_{k=0}^{n}\frac{e^{-\lambda k\tau}\lambda^k}{(k+1)!}\{\left[(k+1)\tau - t\right]^{k+1} - \left[(k+1)\tau - (n+1)\tau\right]^{k+1}\} \\ &= (1-\lambda\tau)\sum_{k=0}^{n}\frac{e^{-\lambda(k+1)\tau}\lambda^{k+1}}{(k+1)!}\{\left[(k+1)\tau - t\right]^{k+1} - \left[(k+1)\tau - (n+1)\tau\right]^{k+1}\} \\ &= (1-\lambda\tau)\sum_{r=1}^{n+1}\frac{e^{-\lambda r\tau}\lambda^r}{r!}\{\left[r\tau - t\right]^r - \left[r\tau - (n+1)\tau\right]^r\} \\ &= (1-\lambda\tau)\{\sum_{r=0}^{n+1}\frac{\lambda^r e^{-\lambda r\tau}}{r!}(r\tau - t)^r - \sum_{r=0}^{n+1}\frac{\lambda^r e^{-\lambda r\tau}}{r!}\left[r\tau - (n+1)\tau\right]^r\} \end{aligned}$$

よって、仮定から $t = (n+1)\tau$ において以下のことが言える。

$$g\left[(n+1)\tau\right] = (1-\lambda\tau)\sum_{r=0}^{n}\frac{\lambda^r e^{-r\lambda\tau}}{r!}\left[r\tau - (n+1)\tau\right]^r$$

ここで、$r = n + 1$ に田応答する項は明らかに 0 であるため n+1 まで含めることができる。上の式と合わせて $(n + 1)\tau < t \leq (n + 1)\tau$における任意の t に対して以下の式が成り立つ。

$$g(t) = (1 - \lambda\tau) \sum_{r=0}^{n+1} \frac{\lambda^r e^{-\lambda r\tau}}{r!} (r\tau - t)^r$$

よって、任意の $t > 0$ に対して以下の式がなりたつことが示せた。

$$f(t) = e^{\lambda t}(1 - \lambda\tau) \sum_{k=0}^{n} e^{(-\lambda k\tau)} \frac{(k\tau - t)^k}{k!} \lambda^k$$

ただし、非負整数 n は以下の不等式によって定められる。

$$n\tau < t \leq (n + 1)\tau$$

証明終り.

付録 B

カーネル内部関数の関係図

ここでは、PEER システムでカーネル内部に追加したモジュール間のつながりを図 B.1 に示す。

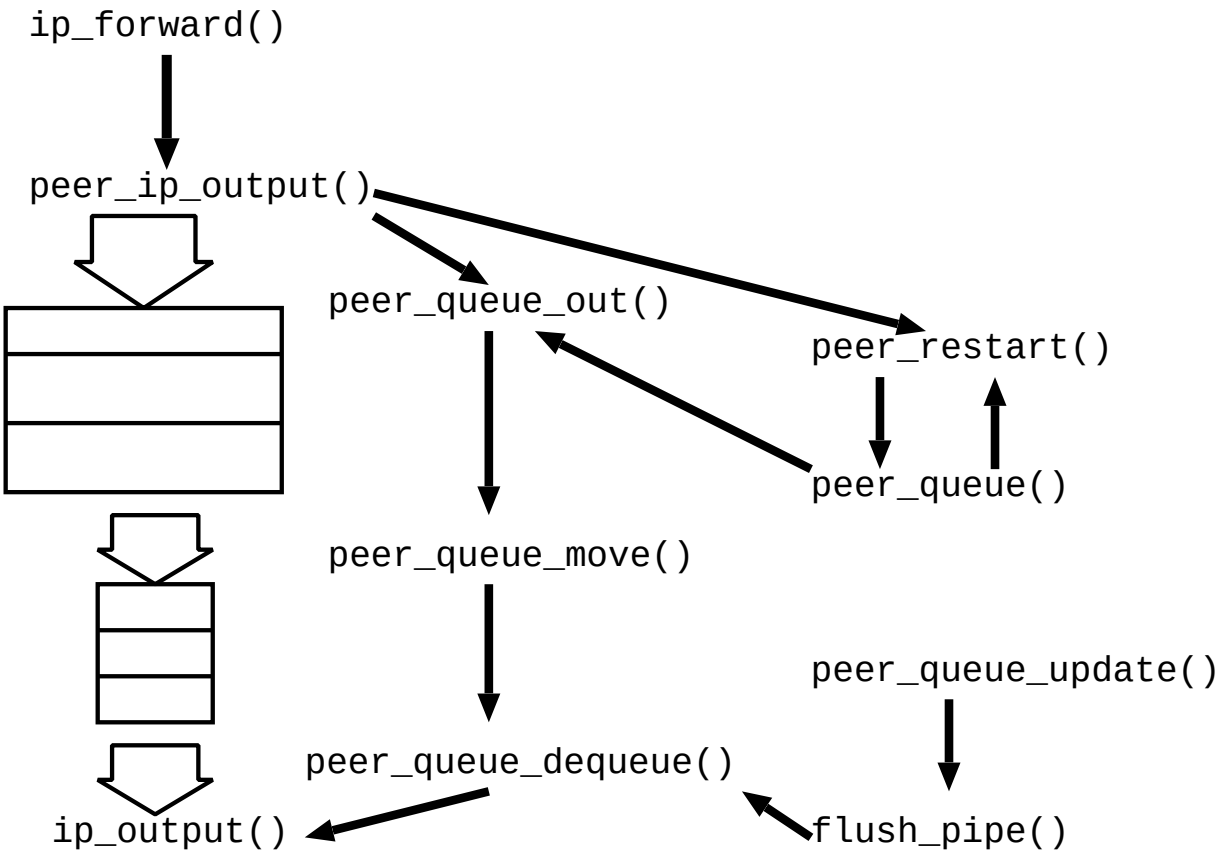

図 B.1 カーネル内の関数の関係

付録 C

TCPにおける帯域および遅延とスループットの関係

ここでは、本文で述べたTCPの検証以外に行なった実験結果の一例を示す。図C.1は、BSD/OS3.0のTCPを使用したときのスループットの変化をグラフにしたものである。帯域や遅延の変化によってスループットが急激に変化していることが分かる。

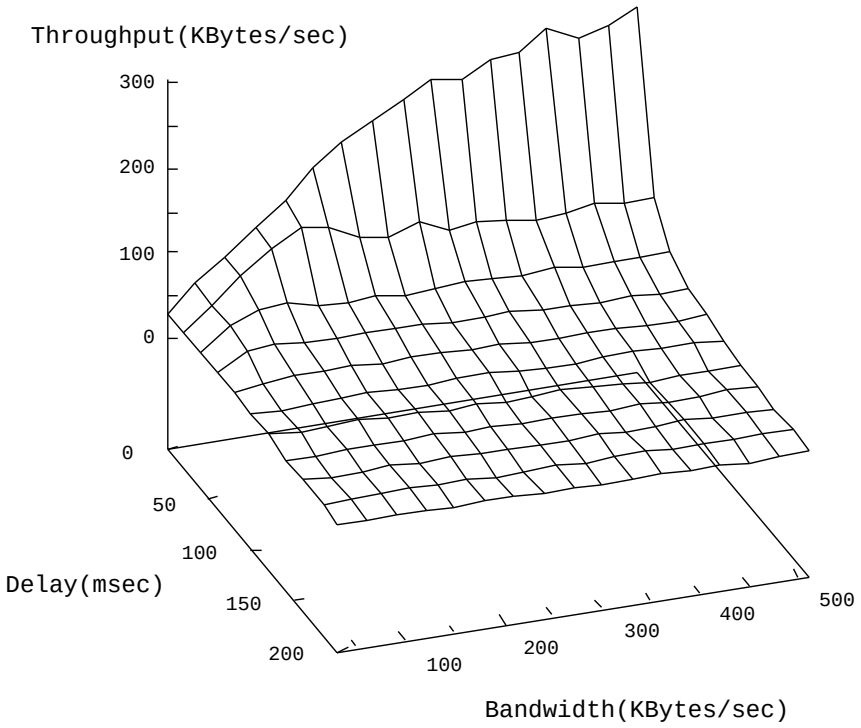

図 C.1 帯域、遅延、スループット

付録 D

Solaris2.5.1のTCPによるデータ転送の様子

ここでは、Solari2.5.1 の TCP によるデータ転送について述べる。第 5 章の RTO の検証結果から遅延の大きな通信路で通信を行なう場合には Solaris2.5.1 は不必要な再送が行なわれているのではないかと予想された。その結果を示すのが図 D.1 である。これは遅延を 3 秒にしたときの Solaris2.5.1 のデータ転送の様子をグラフにしたものである。確かに不必要な再送が行なわれていることが確認できる。

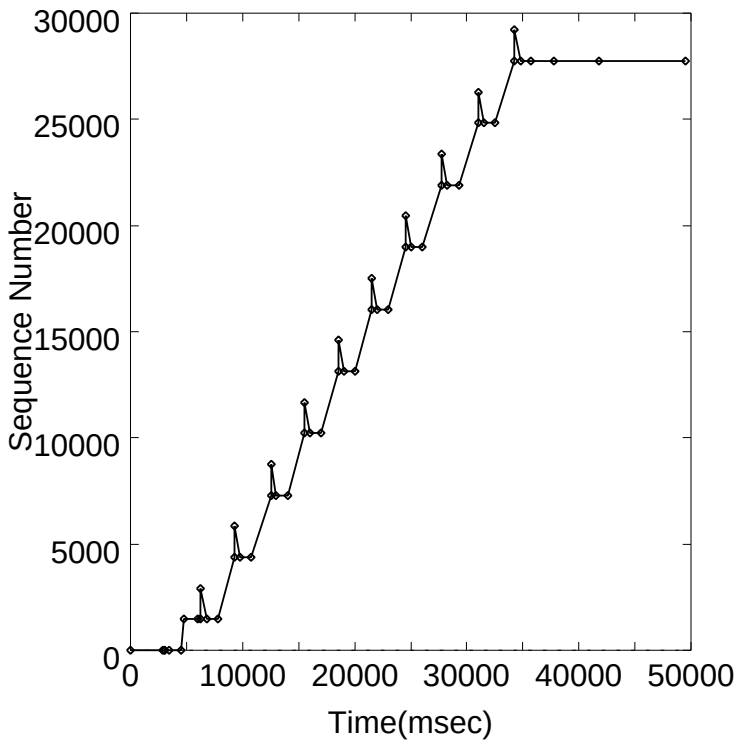

図 D.1 Solaris2.5.1 における再送の様子