マルチメディア通信と分散処理 51－6
（1991. 7. 5）

大規模広域ネットワークにおけるRTTに関する考察

中村 修†, 北島 剛‡, 村井 純*
†東京大学大型計算機センター
‡慶應義塾大学理工学部計算機科学科
* 慶應義塾大学環境情報学部

共有資源の利用では、同時に利用するプロセスの数が増加すれば、必然的に個々のプロセスの応答時間が遅くなる。例えば、あるCPUを同時に利用するプロセスの数が増加すると、個々のプロセスの応答時間が遅くなる。ネットワークも多くの利用者によって共有される資源であるため、近年大規模広域ネットワークでは、利用者の数が増加し、各処理の応答時間が著しく遅くなってきている。このような共有資源の利用環境では、その負荷情報の提供が重要な項目となる。負荷情報が提供されることにより、この共有資源を利用するプロセスの実行形態をプロセス毎に決定することが可能となり、アプリケーションの実行形態の幅を広げることが可能になるとともに、共有資源を有効に利用することが可能となる。
本論文では、ネットワークの負荷情報を提供するための実験システムについて述べる。特にネットワークの負荷情報としてRTT (Round Trip Time) を選び、任意のホスト間における平均RTT値を提供するシステムと一般的なネットワークアプリケーションにおけるコミュニケーションを行なっている間のRTT値を提供するシステムについて述べる。

Experimental Systems for Providing Round Trip Time on Wide Area Network

Osamu Nakamura†, Tuyoshi Kitajima‡, Jun Murai *
†Computer Centre, The University of Tokyo
‡Faculty of Science and Technolgy, Keio University
* Faculty of Environmental Information, Keio University

When number of processes sharing a resource simultinouly is increased, the response time of each process should be delayed. Sharing CPU is a typical example. Because a network is a shared resource, the response time of applications over a wide area network is likely to be delayed. Using shared resources, the load information of the resources is very important. If load information of network is provided, an application can decide its own processing strategies. It is also realized good ulitization of network capacity.
In this paper, network load information providing systems are introduced. RTT (Round Trip Time) is selected as network load information. This value represents one of the network load information and is suitable for interactive applications. Two experimental systems for providing RTT are described in this paper. One is providing average RTT value between two hosts, and the other is the generic system for providing RTT value while communicating between applications using TCP.

1 はじめに

大規模広域ネットワークは、WIDE/Internet[1,2]やTISN, JAINなどの活動により近年盛んに利用されるようになってきた。現在では、国内で約100の組織が相互にIP接続され、WIDEのあるバックボーンでは、1日に約100Mbyteにものぼるデータが流れている。このようにネットワークの利用が盛んに行なわれるようになると、現在のネットワークアーキテクチャではいろいろなトラフィックが相互にネットワーク資源を消費することにより、処理速度の低下が起こってきている。これは有限の資源を複数の利用者によって共有することによって必然的に生じてくることがらである。例えば、CPUを時分割しているTSSでは、処理の数が増えてくれば、必然的にある処理に割れ当てられる時間は少なくなるように、ネットワークでも利用者の数が増加すると必然的にネットワーク上での処理が遅くなる。

このような共有資源の分配を行なう場合、システム全体として、また、各処理にとっても効率良く有限の資源を分配する必要がある。オペレーティングシステムでは、CPUの割り当て処理は一般にスケジューリングとして知られ、CPUという共有資源の効果的な割り当てを行なっている。例えば、スケジューリングは、各プロセスにプライオリティ付けを行ない、このプライオリティを適切に選択することにより、システム全体としてまた個々のプロセスにとって効果的な資源の分配を行なっている。

現在のネットワークは、パケットの到着順に逐次処理しているので、現在のようにネットワークの負荷が高くなると必然的に処理速度が低下し、アプリケーションによっては、使い勝手が著しく低下するものがある。例えば、telnetに代表される遠隔端末機能を提供しているアプリケーションでは、著しく応答時間が遅くなることが頻繁に発生している。このような応答性を要求されるアプリケーションに対して、FTPなどに代表される処理は、それほど処理時間を重視しない。もちろん、著しく転送速度が低下することは論外であるが、例えば、転送時間が1割程度増加してもそれほど使い勝手には影響しない。

現在広域ネットワークで使用しているネットワークプロトコルであるIPプロトコルには、このような問題を解決する1つの方法としてTOS (Type of Server) という概念を持っている[3,4]。各サービスの種類、例えば、応答性を要求するアプリケーションであるとか信頼性を要求しているなど、通信の特徴を各パケット毎に指示するためのフィールドが用意されている。しかし、実際には、TOSフィールドを参照し、処理方法を変更する機能はほとんど実現されていない。

このような問題を解決するためには、実際にどのような原因によって処理速度が低下するか、また、個々のアプリケーションは、どのようにネットワークを利用しているのかを解析し、システムにとってもまた個々の処理にとっての効率良い資源の分配方法を決定する必要がある。

また、現在の広域ネットワークではネットワークの負荷情報などを提供するような機能を持っていない。しかし、ネットワークの負荷情報を提供することが可能であれば、利用者にとってもまた種々のアプリケーションにとっても柔軟にネットワークを利用することが可能となる。例えば、UNIXオペレーティングシステムにおけるLoad Avarageは、CPUの割当を待っているプロセスの数の平均値を示している単純な値であるが、このような情報が提供されていることにより、利用者にとっても安心してシステムを利用することが可能となっている。また、アプリケーションによっては、この値を見てシステムの負荷が重い時間を避けて処理をするような実行が可能となっている。

本論文では、ネットワークの負荷情報、特にデータ転送における遅延時間に着目し、遅延時間が生じる原因を解析するとともに、ネットワーク負荷情報の提供について議論する。

2 負荷情報の種類

ネットワークの負荷情報の提供を考える場合、時系列で大きく3つの状態に分割して考えることができる。

1. 処理を始める前に得る負荷情報
   実際にネットワークを使用するアプリケーションを実行する前に必要とする負荷情報。UNIXにおけるLoad Avarageの値がこれに相当する。

2. 実際にネットワークを利用している間の負荷情報
この場合には、負荷情報と言うよりは実際にネットワークを用いてデータ転送を行なっている時の処理速度となる。

3. 長期的なネットワークの負荷情報
ネットワーク管理などにおける回線速度の次期計画を行なう場合などに用いられるネットワークの利用率

最後のネットワークの長期的な混雑度は、回線の増強や接続形態の変更を行なう場合などに用いられる情報である。本論文でこのようなネットワークの負荷情報は特に議論しない。

1 の事前にネットワークの負荷情報が提供されることは、非常に重要である。もし、事前にネットワークの負荷情報を得ることが可能であれば、以下のような処理を行なうことが可能となる。

1. バッチ的な処理の効率良いスケジューリング
ある程度実際にネットワークを利用する時間を移動することが可能な処理は、ネットワークの負荷が軽い時間帯に処理をすることが可能となる。

2. 効率良いサーバ選択
同様なサービスを複数のホストが提供しているようなサービスの場合ネットワーク的に状態の良い（近い）ものを利用することによって、ネットワーク全体を効率良く利用することが可能となる。

3. 任意のホスト間に複数の経路が存在する場合、アプリケーションのネットワーク利用の特徴にあった経路を選択することが可能となる。

4. ネットワークが混んでいることを、利用者が事前に知ることができれば、他の仕事を行なうなどの選択が可能となる。

2 番目の実際にネットワークを利用している時の負荷情報は、一般的にはネットワークの負荷情報と言うよりは実行転送速度的な値である。一般的にこのような値を必要とするアプリケーションとしては、実時間処理を要求されるアプリケーションなどが考えられる。しかし、ネットワークを利用する上でこのような情報は、実時間処理などを要求されない一般的なアプリケーションでも有効に利用することができる。例えば、ファイル転送をしている間にネットワークのルーティングが変化し、処理の前半では 1.5Mbps で転送していたのに、後半では、9.6Kbps の回線に変化したような場合、現在のネットワークアーキテクチャでは、このような情報は提供されないため、利用者は、いつまでたっても処理が終らないのに待ち続けなければならないし、またどような原因によって処理時間が掛かっているのかすら理解することができない。もし、このような時に、ネットワークの実行転送速度が期待した値よりも桁違いに異なる場合には、処理を中止することが望まれるような場合もある。

2.1 解放型アーキテクチャの問題点

前節で示したようにネットワークの負荷情報は非常に重要な値であるにも関わらず、現在のネットワーク環境で提供されていない最大の理由は、解放型ネットワークアーキテクチャを用いて現在のネットワークが構築されている点にある。

解放型ネットワークアーキテクチャは、プロトコルをレイヤ構造として実現し、各レイヤでは、下位のレイヤによって提供されるサービスを用いてレイヤ間の通信を行なう。レイヤ構造によるプロトコルの実現は、各レイヤの独立性を実現することができるとともに、下位レイヤが内部的に行なっている処理を意識することなくレイヤ間の通信を行なうことが可能となるなど多くの利点がある。実際にネットワークの経路情報を管理し、どのような経路を使ってパケットを転送するかなどの決定は、ネットワークレイヤによって実現されている機能であるため、アプリケーションレイヤでは、どのような経路を通ってパケットが転送されているかを一切意識せず通信を行なうことができる。

しかし、このように各レイヤでの処理を各レイヤ内に隠蔽してしまっているために、柔軟な処理ができなくなるという問題点もある。

近年データ通信技術の発展には目を見張るものがある。ISDN などに代表される高速な間欠リンクを提供可能なデータリンクも利用可能となってきているし、衛星通信のように広いバンド幅を持った通信も利用可能となってきた。データリンクの特徴が単純にデータ転送速度という一次的な尺度だけでは評価できなくなってきている。個々のデー

タリンクが種々の特徴を持ってくると、このような特徴を効果的に利用するためには、データリンクレイヤが扱っている情報をアプリケーションレイヤに対して提供する必要が生じてくる。

現在の解放型ネットワークアーキテクチャは多くの点で利点はあるが、より柔軟なネットワーク環境を実現するためには、各レイヤが協調した新しいネットワークアーキテクチャが必要となってきている。

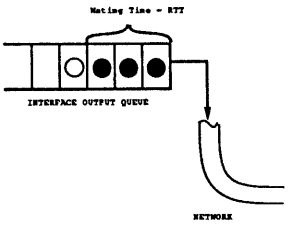

Figure 1: 待ち行列による RTT の増加

3 事前の遅延情報

この節では、アプリケーションを実行する前にネットワークの状況、特に RTT(Round Trip Time) を取得する方法について述べる。

一般的な任意のホスト間における RTT の値を得る方法としては、ping コマンドによる RTT 値の測定方法が用いられる。しかし、一般に RTT の値は、ネットワークの負荷によって動的に変化することが知られているため、平均的なネットワークの負荷情報として RTT の値を用いる場合には、ある程度の時間パケットをネットワーク上に送出しなければならない。

本節では、ping コマンドの様に不必要なトラフィックを発生させないで、任意のホスト間における RTT の値を取得する方法について考察する。

3.1 RTT の解析

大規模広域ネットワークでの RTT の値を構成する要因としては、以下のようなものをあげることができる。

1. ネットワークを構成しているリンクの物理的な遅延
2. ゲートウェイにおけるパケットのフォワーディングにかかる遅延
3. 送信側アプリケーションエンティティおよび IP 層に渡すための処理時間
4. 受信側アプリケーションエンティティおよび IP 層の処理時間

これらの要素の内、ネットワークの負荷に依存する値とそうでない値をわけると、RTT とは、以下のような式で表現できる。

$$RTT = \sum_{l} \{f_l(\text{ネットワークの負荷}) + C_l\}$$

ネットワークの負荷によって変わる値としては、各 IP 層およびデータリンク層での処理時間などをあげることができ、また、負荷によって変化しない値としては、ネットワークを構成しているリンクの物理的な遅延時間などをあげることができる[1]。

ネットワークの負荷によって変化しない値の場合には、どのゲートウェイを通って通信が行なわれるかが決定された段階で、その合計を計算することができるが、ネットワークの負荷によって変化する値の場合には、何らかの方法によってその値を算出する方法が必要となる。

3.2 ネットワークの負荷と RTT の実験

RTT は、ネットワーク上を流れる他のトラフィックに影響されるが、どのような関係で影響されるかを把握するために以下のような実験を行なった。

2 台の Sun Workstation を 64Kbps の同期式通信によって接続し IP プロトコルによるポイントーポイントのネットワークを構築した。このリンクを用いてネットワークの負荷と RTT の関係を調べた。データリンクプロトコルとしては、Xerox PPP[5] を用いた。他のトラフィックが一切ないこのリンクに一定の負荷をかけながら RTT を測定した。

[1] リンクの物理的な遅延時間は、リンクの提供方式によっても異なるが、本論文ではネットワークの負荷に依存しない値とする。

リンクに対する負荷は、任意のパケット長と任意のパケット送出間隔を設定し、一定の負荷をリンクにかけ、負荷が安定してからRTTを測定した。

RTTの測定には、ping(ICMP Echoを用いてリンクのRTTを測定するユーティリティ）を用いて測定した。RTTの値は100回の測定値の平均によってその値とした。結果を図2に示す。

ポイントーポイントでのRTTの値は、リンクの処理能力を越えるまでは、指数関数的な変化をして、その後は、データリンクで処理することができないパケットを破棄するため、パケットの消失が多発する。この時のRTTは、リンクの負荷に対して比例して増加する。この結果から、ネットワークの負荷に対するRTTの値とは、リンクに対する待ち行列のために発生する遅延時間であると考えることができる。すなわち、図3.1に示すようにあるリンクを利用しようとするパケットによってリンクを利用するための待ち行列が形成され、新しく到着したパケットは、この待ち行列がFIFOで処理されるため前のパケットが処理される間待たされることによっておこる遅延時間が、トラフィックの増加によるRTTの増加につながると考えられる。

実際に待ち行列理論によってリンクのトラフィックの増加がRTT及ぼす影響を計算してみる。

リンクに対してのパケットの出力要求の処理は、M/M/nの待ち行列理論を適応する。パケットは、ポアソン到着し、パケットの転送サービスを指数分布サービスとする。ここで、平均パケット長をP_l、回線速度をL_c、パケットの平均到着間隔をA_t、待ち行列の最大長をNとする。

$$\text{到着率}: \lambda = \frac{1}{A_t}$$

$$\text{サービス率}: \mu = \frac{L_c}{P_l}$$

M/M/nのモデルにおいてシステム中で費やされる全時間の平均ωは、以下の式で表現される。

$$\omega = \frac{1}{\lambda}\rho\frac{1-(n+1)\rho^N + N\rho^{N+1}}{(1-\rho)(1-\rho^{N+1})}$$

$$\rho = \frac{\lambda}{\mu}$$

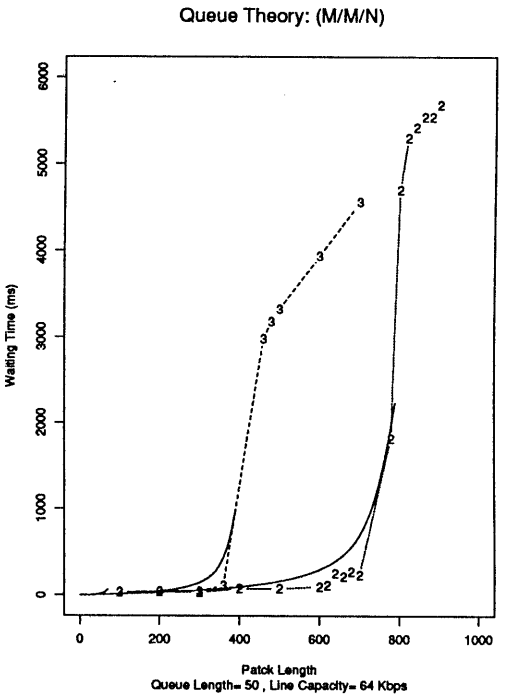

Figure 2: M/M/Nによる計算結果 (到着間隔100ms, 50ms,10ms) 実線（計算値)、点線（実測値: 2: 100ms 3: 50ms)

この関係を用いて回線速度64Kbps、最大待ち行列長を50[2]として上記の実験結果をモデルの解析によって計算すると図2内の実線で示された値となる。

この結果からもRTTの変化は、各リンクに対する待ち行列の中で待たされる時間の影響が大きいことがいえる。また、その他の遅延要素、例えば、ホスト内でのパケットの生成時間やIP層での各種処理による遅延時間は、リンクに対する待ち行列による遅延時間に対して充分に無視できるといえる。

また、実際に大規模広域ネットワークで利用している回線速度は、64Kbpsから1.5Mbpsのように低速なリンクを用いて構築されている場合が多い。このような場合には、ゲートウェイの処理能力はリンクのそれに対して充分早いといえる。一般にゲートウェイの処理能力は、3000から10000 packet/secであるので、例えば、平均パケット長を141Byte[3]とすると、その能力は以下のように換算することができる。

$$2000pk/sec \times (141byte \times 8) = 2.256Mbit/sec$$

$$10000pk/sec \times (141byte \times 8) = 11.28Mbit/sec$$

このことからも、64Kbpsのように低速なリンクの場合には、RTTの増加の要因としては、そのほとんどがリンク上をパケットを送出するための処理に依存し、T.2などの高速リンクで始めて同等な割合となることが分かる。

3.3 平均遅延時間取得方法

前節の解析の結果、ネットワークの負荷によるRTTへの影響は、主に各ゲートウェイにおける回線への出力待ち行列中に待たされる時間であることが分かった。そこで、各ゲートウェイの出力待ち行列の長さを収集することによって、ある時点での任意のホスト間のRTTを得られることになる。

各ゲートウェイにおける各回線に対する待ち行列の長さは、SNMP[6]によって収集することが可能である。以下にSNMPによるデータ収集の流れを示す。

```
/*
 *
 * Caluclate RTT from destination Address
 * using SNMP Protocol
 *
 * Argument:
 *       DestinationAddr: Destination Network
 *                              IP Address
 */

FUNCTION GetRTT (DestinationAddress)
DNADDR = GetNetworkAddr(DestinationtAddress);
N = 0;
BEGIN
     N = N + 1;
     RouterIPAddress =
        SNMPGET(RouterIPAddress,
              ip.ipRoutingTable.ipRouteEntry
                        .ipRouteNextHop.DADDR);
     if (CEHCK(DestinationAddress,
                        RouterIPAddress)) break;
     INDEX = SNMPGET(ip.ipRoutingTable
           .ipRouteEntry.ipRouteIfIndex.DADDR);
     QLEN(N) = SNMPGET(interfaces.ifTable
           .ifEntry.ifOutQLenAV.INDEX);
     CAP(N) = SNMPGET(interfaces.ifTable
           .ifEntry.ifSpeed.INDEX);
END
RTT = CAL(QLEN, CAP)
END FUNCTION
```

ただし、現在のMIB[7]によって定義されている各回線に対する待ち行列の長さ(.ifOutQLen)は、SNMPによって問い合わせた時点での待ち行列長となるため、平均的な遅延時間を得るためには、この値も平均的な待ち行列長を示す値に拡張する必要がある。

4 通信中の遅延情報

実際に通信を行なっている最中の遅延情報を提供することが可能であれば、アプリケーションにとって様々な処理が可能であることは始めに述べた。本節では、セッションプロトコルとしてTCP[8]による一般的なアプリケーションに対する通信中の遅延情報の提供システムについて述べる。

4.1 TCPにおける遅延情報

現在多くのアプリケーションが利用しているTCPは、双方向の信頼性のある通信を提供しているプ

[2]SunOS 4.0.3におけるリンクに対する最大待ち行列IFQ_MAXLENの値

[3]パケット平均長は、WIDE/Internetのあるバックボーンをながれるパケット長の平均値である。

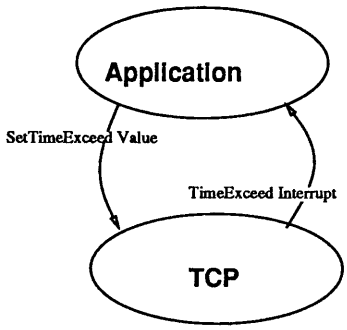

Figure 3: TCP における RTT 情報提供システム

ロトコルである。このため、内部的にはすべてのパケットの到着確認を行なうと同時にネットワークの混雑度に対応したパケットの送出を行なうためにウインドウ制御を行なっている [9] 。そのため、実際には、すべてのパケットの到着確認をすると同時に RTT の値を取得している。

この RTT の値をアプリケーションに通知する枠組を構築した。基本的な機能は以下に示す 2 つの機能を実現した。

1. アプリケーションからの平均 RTT の問い合わせに答え、TCP 内で保持している平均 RTT 値を返す機能

2. 指定された RTT 値の上限を越えた場合のアプリケーションへの通知

アプリケーションが TCP プロトコルを利用する時に、アプリケーション側から RTT に対する TimeExceed 値を設定する。TCP は、内部的に収集している平均 RTT の値が TimeExecced 値を越えた場合に、アプリケーションに対して割り込みを発生する。

Sony News に Tahoe 版のネットワークコードを移植し、この環境を実現した。

新しく以下の 2 つの ioctl エントリを追加した。

```
#define TCP_RTTEXC      0x03
        /* set time exceed value */
#define TCP_RTTCUR      0x04
        /* get current rtt value */
```

これらの ioctl は、以下のように socket インターフェイスからは、socket に対するオペレーションとして呼び出すことができる [10] 。

```
if ((err = getsockopt(net,
   IPPROTO_TCP, TCP_RTTCUR, &rtt, &len))<0){
        perror("Get Current RTT");
        return 1;
}
printf("Current rtt: %d (ms)\n", rtt);
```

この枠組を利用してい、telnet アプリケーション [11] を変更し、RTT の値がある値を越えた場合に利用者に通知する機能を付加した。

4.2 まとめ

Tahoe 版の TCP のコードでは、RTT の値を 500ms 単位で取得しているため、設定できる RTT 値も 500ms 単位となってしまい、あまり実用的な環境を提供することができない。しかし、TCP の RTT の値をより細かな単位で得るように変更すれば、いろいろな制御が可能となり、アプリケーションの幅が広がることを示すことができた。

5 おわりに

ネットワークを用いたアプリケーションを作成する場合、現在のネットワークは、多くの情報を隠蔽し、アプリケーションに対して提供される情報が少ない。特にネットワークの状態などがアプリケーションに対して提供されない現在のネットワークでは、より細かな制御を行なうアプリケーションを実現することができない。

本論文では、ネットワークの状態を示す値として RTT に注目し、ネットワークの状態、特に、ネットワークの負荷による RTT 値の変化について解析した。また、RTT の値をアプリケーションに対して提供するための実験を行なった。これらの結果から、既存のネットワークアーキテクチャでもある程度の情報を収集できることを示すとともに、このようなネットワークの状態をアプリケーションい提供することによって、より多彩なアプリケーションを構築できることを示した。

References

[1] J. Murai, H. Kusumoto, S. Yamaguchi, and A. Kato, "Construction of Internet for Japanese Academic Communities," in *Proceedings of Supercomputing '89*, ACM, November 1989. Reno, Nevada.

[2] J. Murai, A. Kato, H. Kusumoto, S. Yamaguchi, and T. Sato, "Construction of the Widely Integrated Distributed Environment," in *Proceedings of TENCON '89*, IEEE, November 1989. Bombay, India.

[3] J. P. (ed.), *Internet Protocol - DARPA Internet Program Protocol Specification.* Defense Advanced Research Projects Agency, 9 1980. RFC 791.

[4] W. Prue and J. Postel, *A Queuing Algorithm to Provide Type-of-Service for IP Links.* USC/Information Sciences Institute, 2 1988. RFC 1046.

[5] Xerox Corp., *Synchronous Point-to-Point Protocol.* 12 1984. XNS Standard 158412.

[6] J. Case *et al.*, *A Simple Network Management Protocol (SNMP).* SNMP Research, 5 1990. RFC 1157.

[7] K. McCloghrie and M. Rose, *Management Information Base for Network Management of TCP/IP-based Internets.* Hughes LAN Systmes, 5 1990. RFC 1156.

[8] J. P. (ed.), *Transmission Control Protocol - DARPA Internet Program Protocol Specification.* Defense Advanced Research Projects Agency, 9 1981. RFC 793.

[9] D. D. Clark, *Window and Acknowledgement Strategy in TCP.* RFC 813.

[10] *An Advanced 4.3BSD Interprocess Communication Tutorial.* University of California, Berkley, Computer Science Research Group, April 1986. UNIX Programmer's Supplementary Documents, Volume 1 (PS1).

[11] J. R. J. Postel, *Telnet Protocol Specification.* 5 1983. RFC 854.