平成 12 年度

学士学位論文

高速パケットデフラグメント自律化方式の制御に関する研究

An automatic control of packet defragment

1010389　　小林 寛征

指導教員　　島村 和典 教授

2001 年 2 月 5 日

高知工科大学 情報システム工学科

要 旨

高速パケットデフラグメント自律化方式の制御に関する研究

小林 寛征

インターネットとコンピュータの普及により，ネットワーク上に流れる IP パケットの数は増加の一途である．そのため，ネットワーク中継機器とパケット送受信機器の処理速度の向上が求められている．そのパケットサイズは，データリンク毎に定められた Maximum Transfer Unit(最大転送単位) に制限されている．基幹網においては，アクセス網におけるロス率や速度を考慮した MTU 制限から，MTU よりはるかに小さいパケットが大部分を占める．小サイズパケットが多いと基幹網における経路選択，転送処理のオーバヘッドが大きくなる．そこで，転送処理にかかるオーバヘッドを減らすためにエッジルータでパケットをデフラグメントしパケットサイズを基幹網の MTU に近づける方法を検討した．さらに，現在のネットワークではフラグメントが禁止されたパケットが大半であるので，デフラグメントではなく，複数個のパケットを結合する方式を考案した．この方式をパケットアセンブリと呼ぶことにする．本論文では，パケットアセンブリの紹介と，中継機器の CPU 負荷の変化を測定よりパケットアセンブリの有効性を証明することを行う．

キーワード　ネットワーク，デフラグメント，パケットアセンブリ，中継ルータ，基幹網，CPU 負荷

Abstract

An automatic control of packet defragment

Hiroyuki KOBAYASHI

The number of IP packets is increasing in according to Internet enlargement and computer popularity. Emerging networks requires high throughput to the routers and packet sending and receiving tools. The size of packet is subject to restriction of MTU(Maximum Transfer Unit) established every datalink. In backbone network, quite small size packets have the majority, because MTU of access link considering loss and link speed is small. Routing and forwarding overhead has become large because of small size packets. To solve the backbone network problem mentioned above, the author proposed a packet defragmentation on edge routers to decrease the forwarding overhead and examined its effect. The defragmentation makes the backbone network to handle the packet equivalent to almost the size of the backbone network MTU. But packets fragmentable don't have the majority in network at present. Therefore I devised a method to assemble some packets one packet and named this method 'Packet Assembly'. This thesis introduces packet assembly and proves the effectiveness of packet assembly by the measurement of CPU load in relay router.

key words network, defragment, packet assembly, router, backbone network, cpu load

目次

図目次

表目次

第 1 章

本研究の目的

この章では，本研究の目的について説明する．

1.1 デフラグメント自律化方式の目的

IP を用いたネットワークにおけるデータ伝送方式は，次の 3 つのステップを踏んでいる．

1. データ送信の前処理

   IP はプロトコル上位層から送信するデータを受け取る．そのデータが， 送信経路上にある次ホップまでのデータリンクの IP パケットサイズの制限，MTU(Maximum Transfer Unit:最大転送単位) を越える場合，制限以内に収まるようフラグメント (分割) する．そして IP ヘッダを付け送信する．
2. ネットワーク伝送

   IP ヘッダを参照し，次に転送するホップを決定 (ルーティング：経路選択) し，転送処理を行う．このとき， 送信処理の時と同じく次ホップまでの MTU による制限を越える場合，このホップでさらにフラグメントし， MTU に及ばない場合は，そのまま転送する．
3. data 受信の前処理

   受信した IP パケットからフラグメントしたデータを再構成するデフラグメントを行い，位層に渡す．

ネットワークの高速化のために，2 にある転送処理の効率化と 3 のデフラグメント処理の処理方式を研究しており，デフラグメント処理が最終的な目的である．デフラグメント処理の高速化は，ネットワーク伝送が効率的に行われて有効となる技術である．そこで，ネットワーク伝送の効率化のために次の点に重点を置き，研究を進めている．

1.2　エッジルータにおける IP パケットの再構成

現在のパケット伝送方式は，MTU の制限によってフラグメントされた IP パケットは，目的ノードに到達するまで，デフラグメントできないため，MTU の大きい基幹網内は小サイズのパケットが大半を占め，基幹網内の転送処理効率の低下を招いている．解決方法として，MTU の小さいアクセス網から大きい基幹網へ IP パケットの伝送を行う場合，複数のパケットを基幹網の MTU のサイズに再構成する (図 1.1)．これによって，基幹網内の高効率転送を実現する．

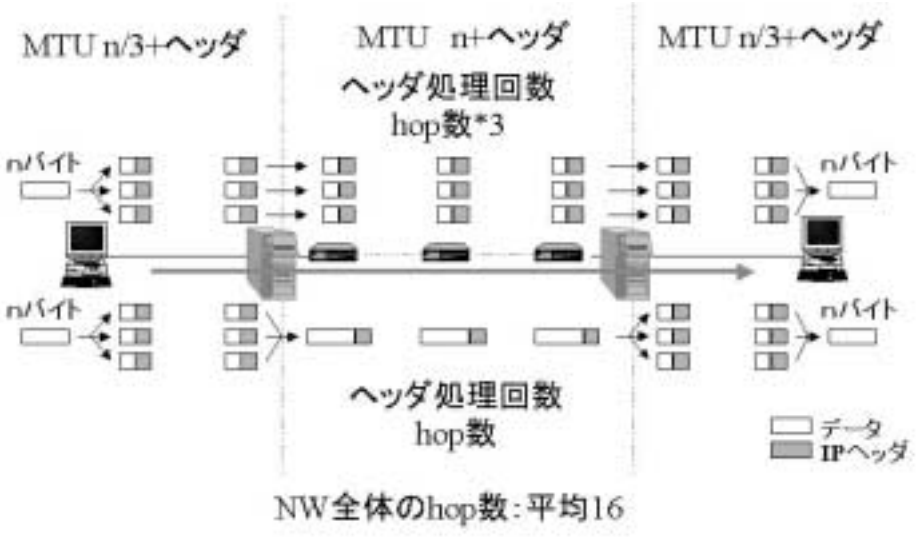

図 1.1　現在と研究する伝送方式の流れ

第 2 章

研究の背景

この章では，本研究，並びに，目的の中で触れた前段階の研究の背景，さらに類似する研究について説明する．

2.1 デフラグメント自律化を求める理由

現在，インターネットとコンピュータの普及により，ネットワークを通して扱われるデータの内容が変化してきている．映像・音楽の配信や，peer to peer 型を用いる Napster[1] のようなデータのやり取りなどで膨大なデータを扱うのである．また，ストレージ技術として登場した FiberChannel ら (IP と異なる) を用いた SAN(Strage Area Network) の標準化の失敗と，GigabitEthernet(1000Mbps のデータ転送速度は，FiberChannel の 850Mbps を越える）の急速な普及によって，SAN を用いる必要性が無くなった事もあり，Strage over IP という方式が登場し，企業の競争力を支援する情報の蓄積・アクセスなど膨大なデータを扱うものも現 IP ネットワークを利用するに至っている [2]．以上の理由から，インターネットが処理しなければならない IP パケット量が増加し続けている．ネットワークにおいては，ATM や Gigabit Ethernet など高速網が普及しつつあり，中継ルータにかかる負荷を除いては問題は少なくなっている．しかし，高速網に接続されているサーバや今後高速化される一般のアクセス網上の端末において，短時間に連続して送られてくる IP パケットをすぐに処理できるよう，受信した IP パケットをアプリケーション層が扱えるデータにデフラグメントする処理を高速化させなければ，ネットワーク全体の速度の向上は見込めなくなる．そこで，コンピュータを動作させているクロックに捕らわれず，この処理を実現できる処理方

式が求められている．

2.2　エッジルータでのパケットの再構成を求める背景

前節で中継ルータにかかる負荷の問題と書いた．それは，次のような理由による．IP パケットの転送にはデータリングごとに定められた MTU 以下のサイズで行われている．また，中継ルータでパケットのデフラグメントを行うことが禁止されている．そのため，MTU の小さいネットワークから大きいネットワークへパケットを伝送する場合でもサイズは変わらない．実際，基幹網を流れるパケットは，小サイズのものが大半を占めている (図 2.1)[3]．小サイズパケットが基幹網に与える影響として IP ヘッダの経路選択と転送処理の負荷増大と帯域使用効率の低下を招いている．現在では，技術の発展に伴い高品質 (低 Loss・高速) の基幹網が背備され始めているため，基幹網内の伝送効率を向上させるためにもエッジルータでのパケットを再構成し，基幹網内のパケットを基幹網の MTU のサイズにして転送する方式を考案すべきである．

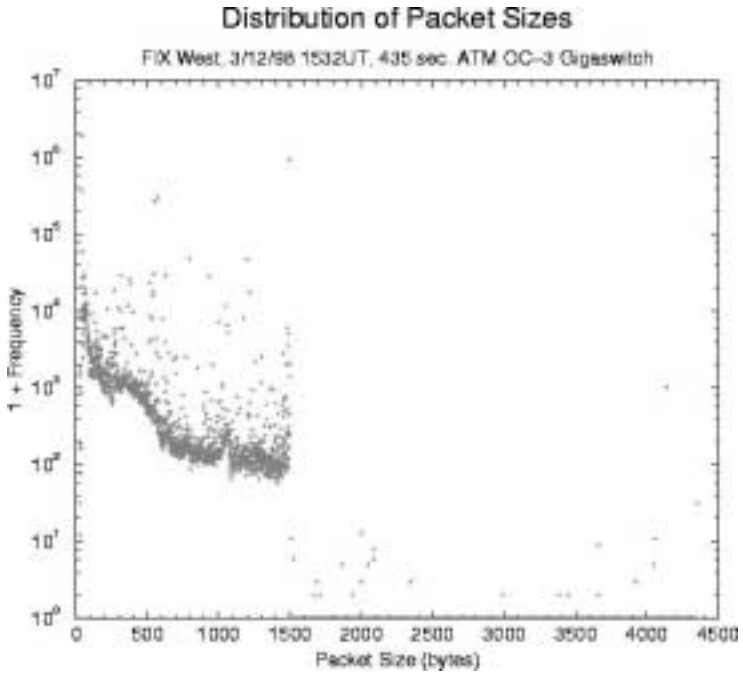

図 2.1　ネットワーク上のパケットサイズ分布図

2.3 類似技術とその現状

2.3.1 Framed ATM over SONET/SDH transport

ATM Forum が研究を進めている FAST は SONET/SDH(同期光ネットワーク) 上で，可変長の ATM frame を伝送する技術である．ATM は 5Bytes の header に，48Bytes の data を付けた合計 53Bytes の固定長 cell を使って伝送する．FAST を用いると，4Bytes の固定長 header と最大 64kBytes のデータフィールドで frame を構成する．IP packet は可変長の data field に格納される．大きな可変長 field を採用したことで，IP packet を短い ATM cell に分割する必要が無くなり効率の高い IP packet 伝送が期待できる．この技術と同様のものとして，米ルーセント・テクノロジーズの SDL(Simple Data Link) や NTT の MAPOS(Multiple Access Protocol over SONET) がある．どちらも FAST 同様 long frame を特徴とし，SDL の最大フレーム長は FAST とほぼ同じ 64kBytes である [4].

これが実現されると，再構成できる size は最大 64kBytes まで可能になる．ただ，IPv4 の IP over ATM の MTU が 9180Bytes なので，当分はそれまでだが，今後 IPv6 に移行が完了すれば最大 MTU が 65535Bytes(ヘッダを含めると 65575Bytes) となる．今後，より小サイズパケットの再構成が必要とされるかもしれない.

2.3.2 Jumbo Frame

「Jumbo Frame」は，米アルテオン・ネットワークスの技術である．これを用いると，TCP Resegmentation と MSS Spoofing と使うことにより，送信側のステーションがはるかに大きな MTU を使っていても，受信側のステーションは自らの特定の MTU に合った IP packet を受信できる．どういう事かというと MTU による制限で IP packet をフラグメントする必要が起きた場合，そのルータにおいて，MTU に合うサイズに TCP セグメントを resemgentation する．このとき必要となるのは，IP データ長の再計算とチェックサムの再計算である．この処理はルータのバックグラウンドで処理される．この技術を用いるに当たり必要とされるのは，この二つの技術をサポートしているネットワーク機器とサーバに用

いる NIC である [5]．この社の製品にこの技術は用いられている [6]．

2.3.3 高効率パケット多重化転送ネットワークの構築

工学院大学大学院の堀内氏と浅谷氏が研究している．この研究は，本研究と同様ルーティング方向が同じ IP パケットを多重化 (後に付ける) する形で，転送を行う．これによってスループットの向上を図っている．2000 年 10 月の段階では，シミュレーションを用いて伝送効率を評価しているものを電子情報通信学会 [7] の方へ提出されている．

我々の研究と異なることは，我々は，IP や TCP ヘッダを参照し，重複・不要部分を削除して結合する．こちらは，IP パケットをそのまま結合している点である．

第 3 章

パケットアセンブリ (Packet Assembly)

3.1 経路途中でのデフラグメント

パケットサイズを基幹網の MTU に合わせ伝送を行う方法として，先ず，既存の技術であるデフラグメント処理をエッジルータに用い，デフラグメント後，MTU に再度フラグメントして転送させる方法を検討した. 基幹網から抜ける場合，エッジルータで再度フラグメントを行う. そこで次の問題点が挙げられる.

- デフラグメント処理

  デフラグメントには，フラグメントされたパケット全てが必要となる. 一つのパケットでもロスした場合，もしくはタイムアウトの場合，デフラグメント処理を停止し，受け取っていたデータを破棄する. この時，パケットを破棄し ICMP メッセージを送信側に返すか，そのまま転送するか，どちらの方法を採るにしてもパケットを保持せねばならず，また，すべてのパケットがデフラグメント対象となるので，大容量のメモリが必要となる. また，オリジナルデータのサイズが大きい場合，フラグメントされた全てのパケットの受信からデフラグメントが完了するまでに時間を要し，逆に伝送効率は低下してしまう.

- IP の転送

  経路制御の研究が進められている. しかし，IPv4 における転送は基本的にはベスト・エ

フォートであるため，すべてのパケットが同一経路を通るとは限らない．そのため，デフラグメント出来ない可能性がある．

- 経路 MTU 探索 (Path MTU Discovery)

  IP 層でフラグメントし送信した場合，ロス発生時はパケットを全て再送しなければならない．そこでトランスポート層では，この効率の悪さを考慮し，送信時に伝送経路上の最小 MTU を探索し，その MTU のサイズに分割して IP 層に渡す方式を採っている．このため，ネットワーク上に流れるパケットは，フラグメントされてないものが大部分を占めている．そのため，デフラグメントを行えない．

■結論　以上から，エッジルータでデフラグメント処理を用いる利点は無い．また，送信元と送信先の両方が一致している場合 (シングルフロー) で且つオリジナルデータからフラグメントされたパケットのみと適用範囲が小さい．別の方式を考案する必要がある．

3.2　パケットアセンブリ

前節の問題から，より効率的に MTU に近いサイズで転送する方式，パケットアセンブリを考案した [8, 9]．本方式は，

- 送信元と送信先の IP アドレスが一致するパケット同士を結合するシングルフロー・アセンブリ
- 送信元と送信先のネットワークアドレスが一致するパケット同士を結合するマルチフロー・アセンブリ

の二つがある．本方式は，次の動作を採る．エッジルータで基幹網へと流れるパケットを保持し，一定時間内の後続のパケットのうちシングルフロー，又は，マルチフローのどちらかに当てはまるパケットを保持しているパケットの後ろに結合する．このとき，保持したパケットのヘッダ情報と結合するパケットのヘッダ情報の中から一致するフィールドは削除する．対象となるヘッダ情報は，

- どのヘッダも対象外 (どれも削除しない)
- IP ヘッダのみ
- IP ヘッダ+TCP ヘッダ
- IP ヘッダ+UDP ヘッダ (シングルフローのみ)

の 4 パターンとする．これにより，MTU に近いサイズ，もしくは，有る程度サイズを大きくしたパケットを基幹網へ転送できる．また，アクセス網へと抜けていく場合，取り除いたフィールドを複製し，オリジナルパケットへと復元する．結合する動作をアセンブリ，復元する動作をデアセンブリと呼ぶことにする．

これは，ACK パケットもアセンブリ対象に入てれいる．

以上がパケットアセンブリ方式の動作である．

3.3 パケットアセンブリ仕様

ここで，現在決められているパケットアセンブリのヘッダ操作を説明する．各ヘッダフィールドの扱いと，アセンブリ後のパケットフォーマットを示す．ただし，TCP においては，フィールドの削除に参照する Code Bit により存在する．そこで TCP においては，削除方法，削除対象となるヘッダフィールドの表示を示す．

3.3.1 シングル/マルチフロー・アセンブリ (ヘッダ削除無し)

アセンブリ IP パケットヘッダとなるフィールドの説明.

- Version

  現在 IPv4 を対象にしているため，4 の時はアセンブリ　6 は非アセンブリ．
- Internet Header Length

  5 の時アセンブリ，6 以上 (オプション有り) は非アセンブリ．
- Type Of Sservice

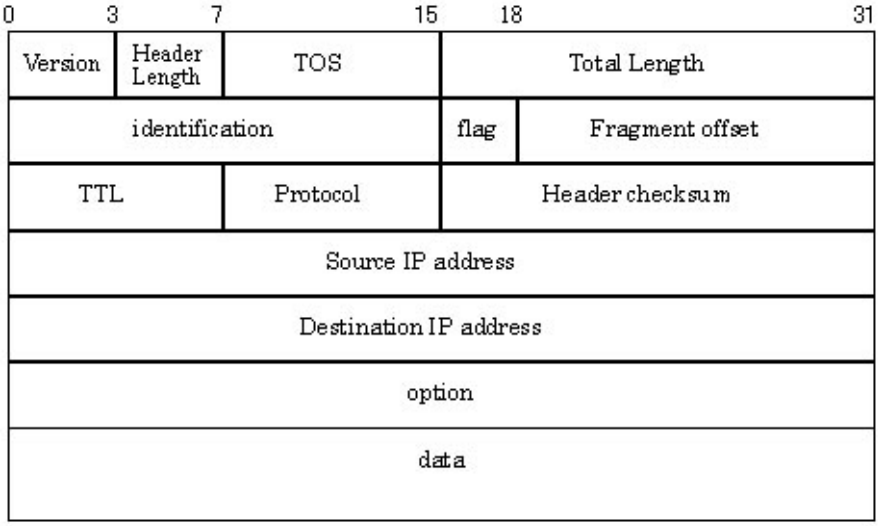

図 3.1 IP ヘッダ

優先度 3bit	D	T	R	C	U

優先度 3bit 8 レベルの優先度を表す.

D requests low Delay

T requests high Throughput

R requests high Reliability

C requests a route with a lower Cost

U Unused

先頭のアセンブリヘッダに，使用されていない最終 bit をアセンブリフラグとして用いる．これは，どのアセンブリ処理でも共通とする.

- total length

  アセンブリされた全てのパケットの total length と元ヘッダ情報の確保用 32bit 分の 1 を足し合わせた数値に再計算.

- identification(ID)

  先頭のパケットの値をコピーする.

- flags

DF フラグをセットした状態で他の値はコピーする.

- Fragment offset

  データグラムがフラグメントされたときの元のデータグラムでの位置を表す. ルータでは用いないので，先頭のものをコピーする.

- Time To Live

  本方式では，ヘッダを丸ごとアセンブリする. 差分を取る必要があるので，アセンブリ時に TTL record へ先頭の TTL を保存する.

- protocol

  トランスポートのプロトコルを指すためのものでルータではしようされない. 先頭パケットのものをコピー.

- checksum

  ルータを通過するたびにヘッダのためだけに計算するものなので，エッジルータで普通の IP ヘッダとしての扱いを行って計算する.

- Source IP address

  エッジルータでデアセンブリ出来ればよいので，先頭のものをコピーする.

- Destination IP address

  先頭のものをコピーする.

- Option

  今回は考慮していない. IHL を 5 に限定しているのもオプション有りのパケットをアセンブリ非対応にするため.

3.3.2 シングルフロー・アセンブリ

IP ヘッダフィールド削除

アセンブリ IP パケットヘッダとなる各フィールドの扱い. シングルフロー・アセンブリのヘッダを削除しない場合のヘッダフィールドと異なる扱いをするフィールドの説明をする.

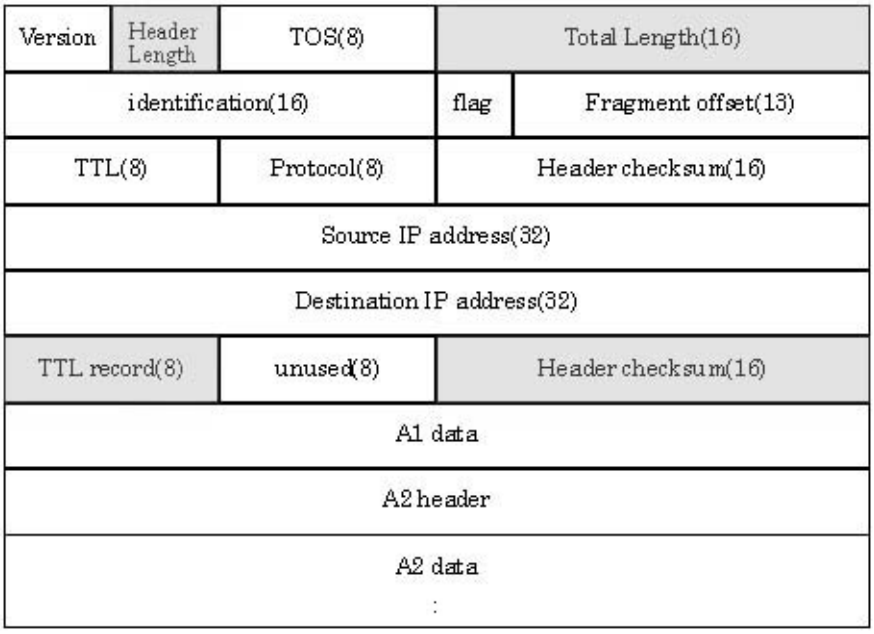

図 3.2 シングル/マルチフローアセンブリ，ヘッダ削除無しパケットフォーマット

- total length

  アセンブリされた全てのパケットのパケットデータの長さと (元ヘッダ情報の確保用 64bit)*(アセンブリしたパケット数) を足し合わせた数値に再計算.

各ヘッダの元情報確保用 64bit へ保存するフィールドの説明.

- total length(16bit)
- identification(16bit)
- Fragment offset(13bit)
- MFbit(1bit)
- Unused(18bit)

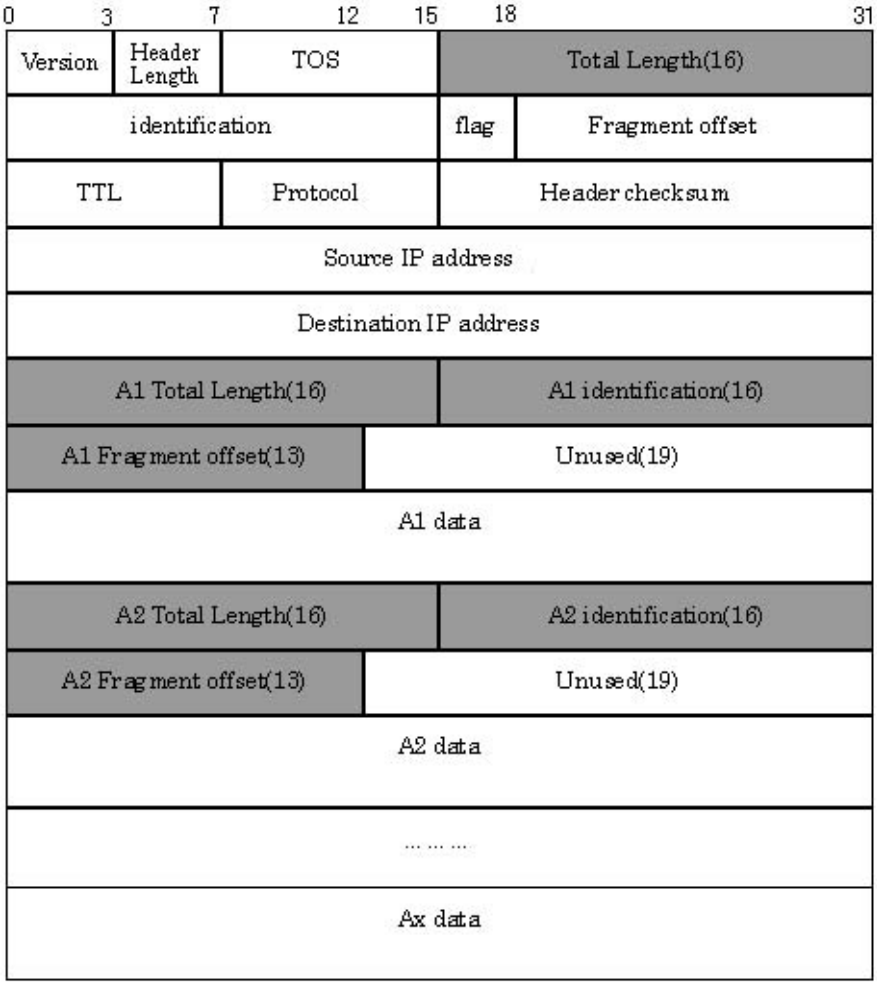

図 3.3　シングルフロー・アセンブリ IP 削除パケットフォーマット

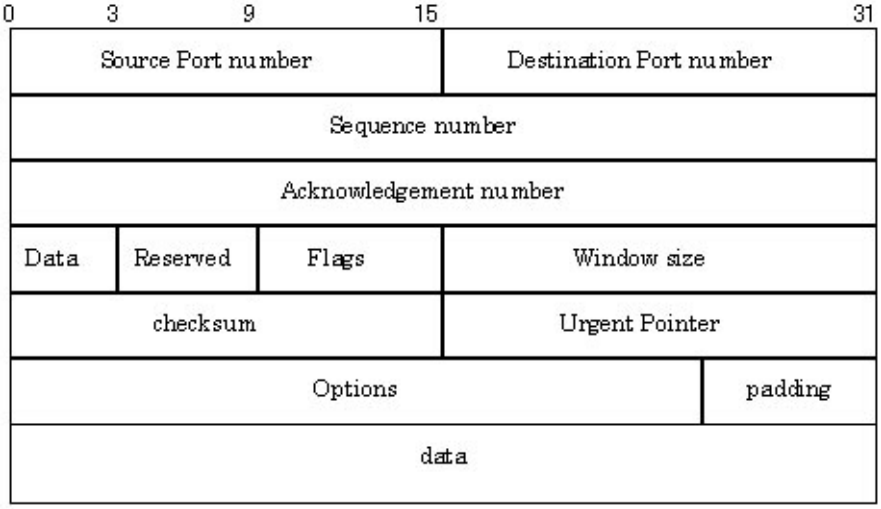

図 3.4　TCP ヘッダ

IP ヘッダ+TCP ヘッダフィールド削除

TCP ヘッダの削除は，他のヘッダの削除とは若干異なり，共通の値を持つフィールドを削除するだけでなく，使用されていないフィールドも削除を行う．共通の値を持つとき削除できるフィールド

- Source Port Number
- Destination Port Number
- Window size

使用されていない場合削除できるフィールド

- Acknowledgement number

  ACK がセットされている場合は使用される．セットされていないときは削除可.
- Windows Size

  先頭の TCP ヘッダの値と一致するときも削除可．Reset(RST) がセットされている場合，Window Size は値を持たないので削除可.
- Urgent Pointer

  Code bit 中の URG がセットされているとき使用される．セットされていないときは削除可

以上に挙がらなかったフィールドは削除できない．また，削除可能なもののうち，削除したフィールドを調べるために，Assembly によって省略された IP ヘッダに生じる Unused の領域を利用する．まず，Unused 中 6bit を用い，TCP Assembly Flag とする．6bit のフラグは次の通り.

1. Source Port を省略した場合 1 をセット
2. Destination Port を省略した場合 1 をセット
3. Window Size を省略した場合 1 セット

4. ACK をコピーしたもの
5. RST をコピーしたもの
6. URG をコピーしたもの

省略対象となるフィールドは次の図で示す.

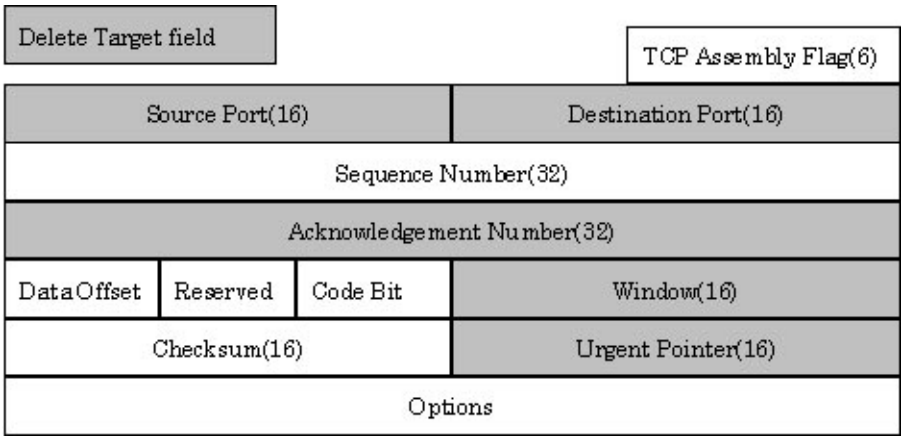

図 3.5　シングルフロー，TCP ヘッダの削除対象フィールド

IP ヘッダ+UDP ヘッダフィールド削除

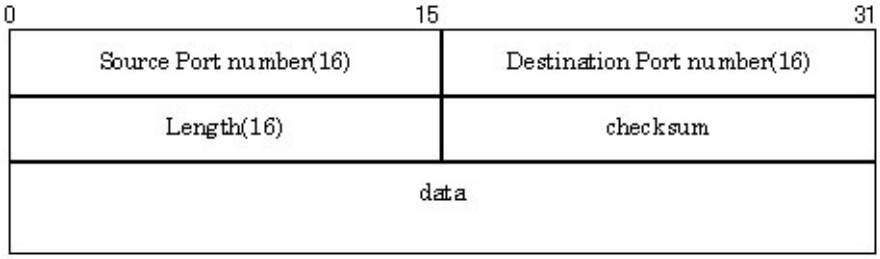

図 3.6　UDP ヘッダフォーマット

UDP ヘッダは削除できるフィールドがシングルフローの場合しかない．削除する項目は次の通り.

- Source Port number

- Destination Port number

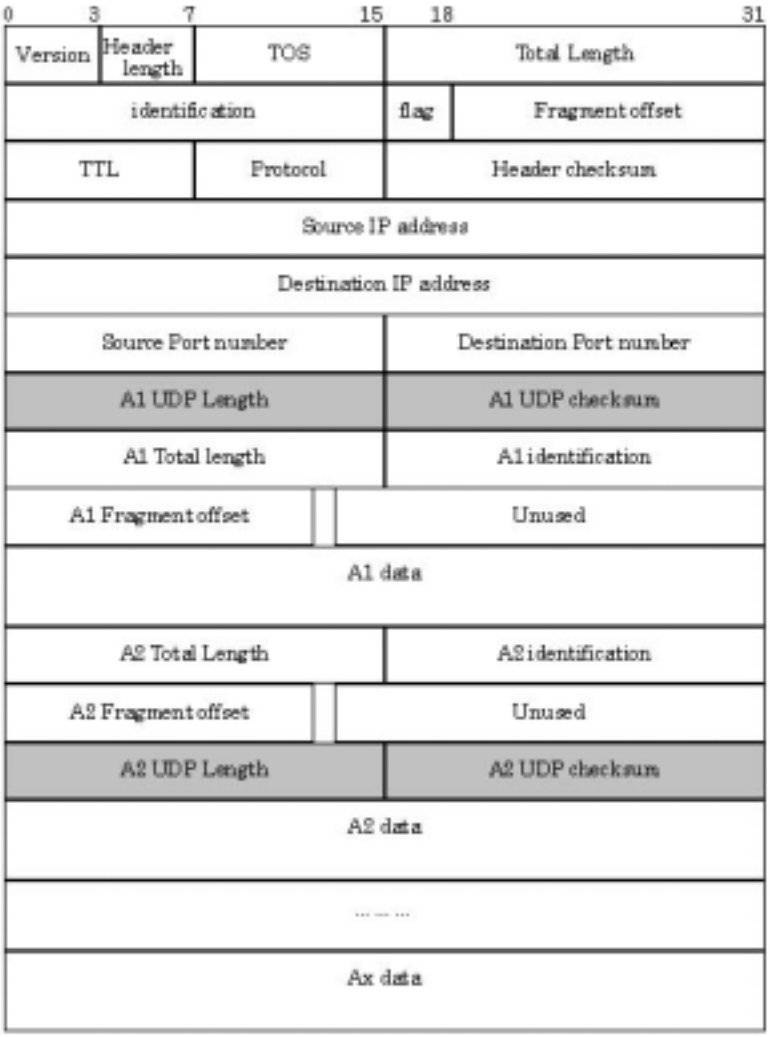

図 3.7　シングルフロー・アセンブリ IP+UDP 削除パケットフォーマット

3.3.3　マルチフロー・アセンブリ

IP ヘッダフィールド削除

アセンブリ IP パケットヘッダとなる各フィールドの扱い．シングルフロー・アセンブリのヘッダを削除しない場合のヘッダフィールドと異なる扱いをするフィールドの説明をする．

- IHL

ネットマスクを保存するために 32bit オプションを追加する.

- total length

  アセンブリされた全てのパケットのパケットデータの長さとネットマスク用の 32bit と (元ヘッダ情報の確保用 96bit)*(アセンブリしたパケット数) を足し合わせた数値に再計算.

各ヘッダの元情報確保用 64bit へ保存するフィールドの説明.

- total length(16bit)
- identification(16bit)
- Fragment offset(13bit)
- MFbit(1bit)
- Unused(18bit)
- Source HOST address(max 16bit)
- Destination HOST address(max 16bit)

IP ヘッダ+TCP ヘッダフィールド削除

シングルフローで用いたものの，共通した値のフィールドの削除を除いたものである．Assembly によって省略された IP ヘッダに Unused の領域が生じる．これを利用する．まず，Unused 中 3bit を用い，TCP Assembly Flag とする．3bit のフラグは次の通り.

1. ACK をコピーしたもの
2. RST をコピーしたもの
3. URG をコピーしたもの

省略対象となるフィールドは次の図で示す.

3.3　パケットアセンブリ仕様

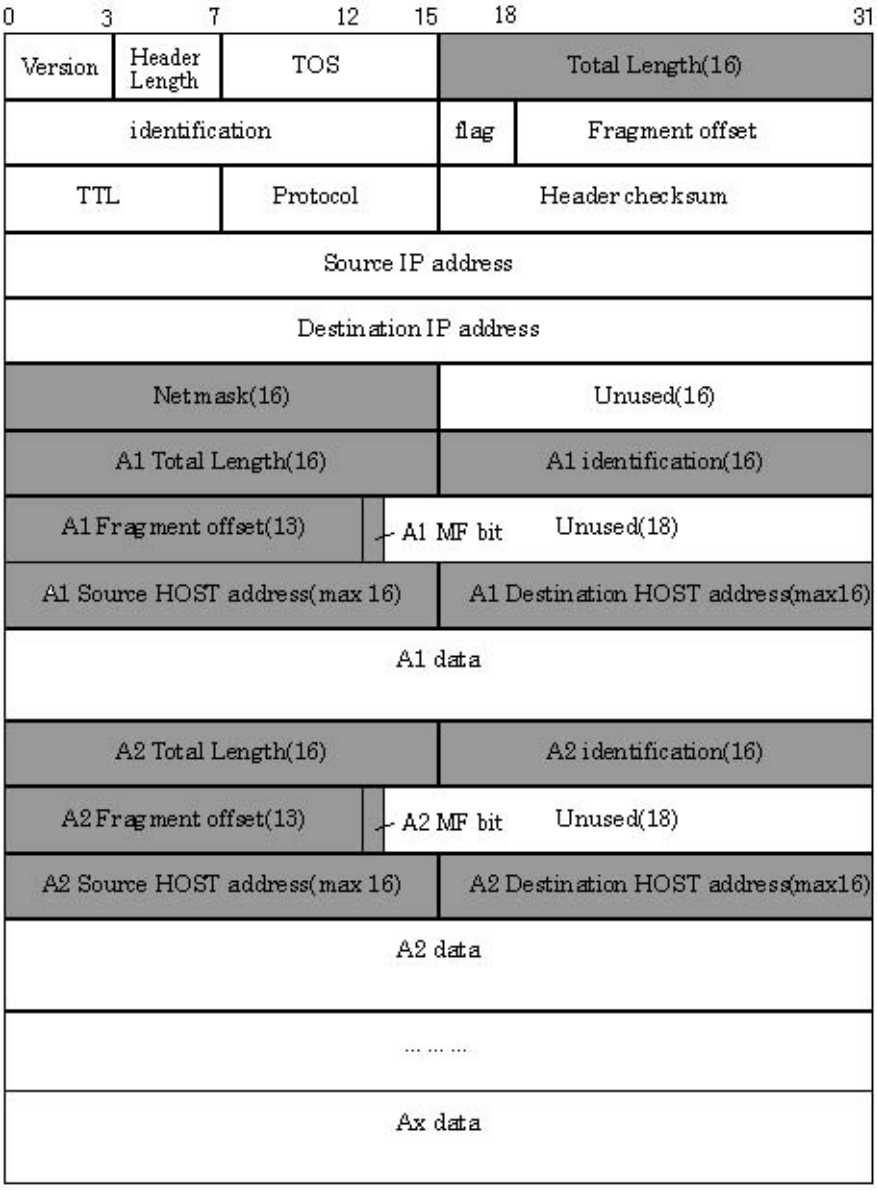

図 3.8　マルチフロー・アセンブリ パケットフォーマット

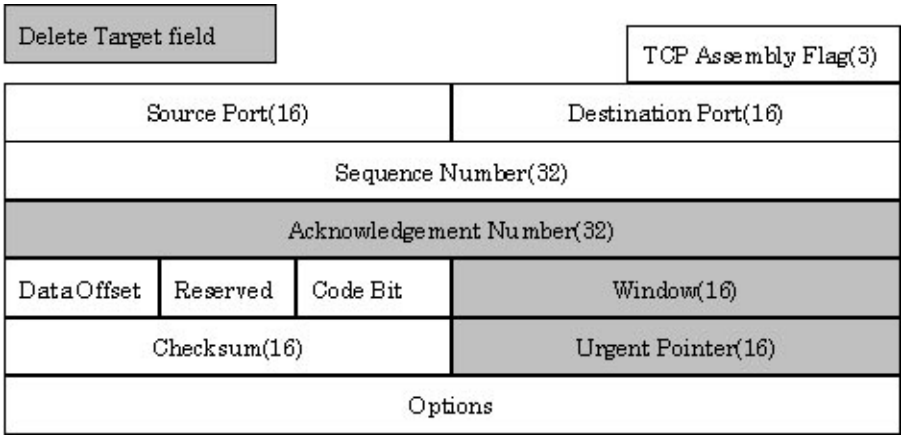

図 3.9　マルチフロー，TCP ヘッダの削除対象フィールド

第 4 章

パケットアセンブリ時の有効性

4.1 アセンブリ評価方法

本方式の有効性を示す評価項目として次のものが挙げられる.

4.1.1 シミュレータを用いて測定する項目

- 基幹網上の中継ルータ CPU 負荷測定

  経路選択と転送処理にかかる CPU 負荷が，中継ルータを通過するパケット数による経路選択負荷，パケットサイズによる転送処理負荷による変化の調査.

- 経路上のルータの pps(packet per second) 測定

  アクセス網上，基幹網上でパケットアセンブリによるスループットの影響の調査.

- エッジルータの CPU 負荷測定

  アセンブリとデアセンブリによる負荷が許容範囲を超過すると，逆にエッジルータがボトルネックになるため測定が必要.

- ネットワーク伝送時間，遅延，揺らぎ測定

  アセンブリとデアセンブリを含めた伝送が，他のトラフィックやアプリケーション層などの上位層に与える影響の調査.

- 現実のネットワークモデルでパケットアセンブリによる影響を測定

  パケットアセンブリ対応エッジルータ作成し，実際のネットワークで用いた測定結果の期待値を調査.

実際のパケットアセンブリ対応エッジルータを用いて測定する項目

上記のシミュレーションを用いた測定項目以外に次の項目を挙げる.

- パケットアセンブリ仕様の問題点

  現在の仕様のままでは，問題が生じる可能性がある. その問題点を調査.
- 実ネットワークに接続し，エッジルータ前後でのパケットをキャプチャ

  現実に近いネットワークモデルを用いたシミュレーション結果と比較する.

4.2 測定

上記に挙げた測定項目のうち，基幹網上の中継ルータ CPU 負荷について 2 種類の測定を行った. 通常転送の場合と UDP ソケットを用いた疑似パケットアセンブリ転送の場合で，基幹網上の CPU 負荷変化について次の条件下で測定を行った.

1. 512Bytes のパケットサイズで送信し，送信時の pps の変化させた場合の CPU 負荷変化の測定
2. MTU の調節により 4 種類のサイズのパケットを連続させ，アセンブリ時には 1500Bytes まで結合した場合の CPU 負荷変化の測定

4.2.1 共通環境

この 2 種類の測定で共通する環境を先に紹介する.

測定に用いたアセンブリしたパケットのフォーマット

第 3 章でパケットアセンブリの仕様を紹介した. しかし，今回は実験のため，次の簡単なパケットフォーマットを用いる. この連続するパケットは，UDP ソケットを用いたオリジナルデータをフラグメントしたもので，エッジルータで一度 UDP ソケットで受信し，次 (基幹網に当たる) のホップまでの MTU で送信を行う. このため，IP ヘッダは基幹網を抜ける

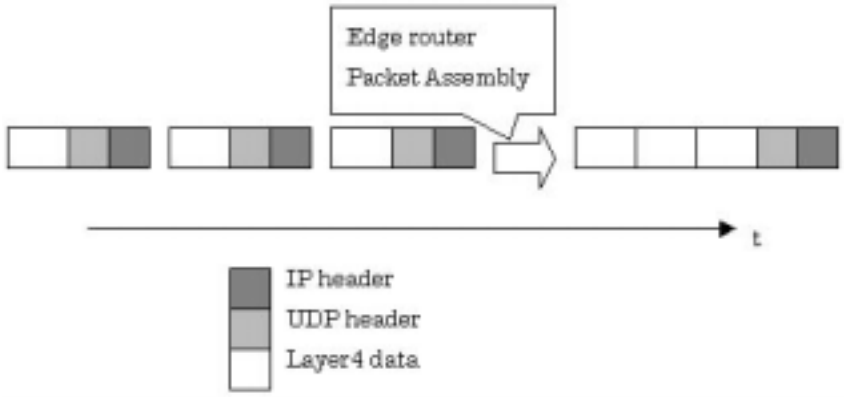

図 4.1　測定用アセンブルパケットフォーマット

エッジルータの情報を載せた新しい物になる．また，UDP ヘッダも全体の情報を載せた新しい物になる．

測定環境

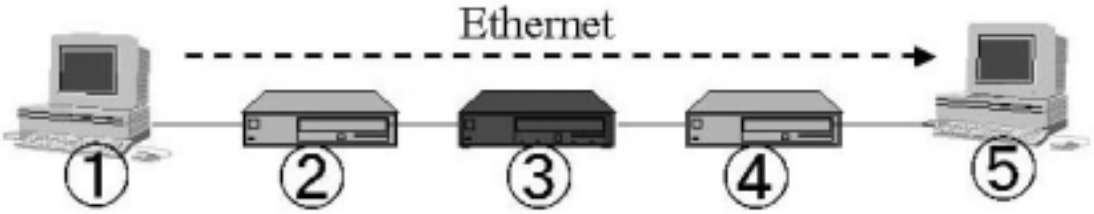

図 4.2　測定用ネットワーク

測定に用いた一経路のネットワークは共に同じで次の設定がある．

1. パケット送信端末 (Solaris2.6)

   後に説明する条件下のサイズのパケットを UDP ソケットで送信する．総送信パケット数と，送信開始時と送信終了時に gettimeofday 関数を用い取得した時間より，送信時の pps を算出する．

2. エッジルータ (FreeBSD4.0R)

   送信端末より送られてきたパケットを UDP ソケットで受信し，オリジナルデータに復

元する. その後，4 までの MTU にまで分割し，再び UDP ソケットを用いて送信する.

3. (基幹網の) 中継ルータ (WindowsNT). ここを通過するパケット転送処理にかかる CPU 使用率 (負荷) を図る. 用いた CPU 測定ソフトは，WindowsNT 付属のパフォーマンスモニタ. 精度は毎秒で少数点第 1 位までの使用率を表示する. 測定対象の値は，全転送処理終了後の平均値である.
4. エッジルータ (FreeBSD4.0R)

   2 のエッジルータより送信されたパケットを UDP ソケットで受信する. その後，5 までの MTU に分割し，再び UDP ソケットを用いて送信する.
5. パケット受信端末 (Solaris2.6)

   4 より送信されたパケットを UDP ソケットで受信する. これで受信処理終了. 1 と同様に総受信パケット数と受信開始時と受信終了時を gettimeofday 関数で取得. 送受信パケット数よりロス率を割り出す. しかし，今回の測定ではロスが生じない条件下で測定したため用いていない.

以上が共通環境である.

4.2.2 pps の変化による基幹網上の CPU 負荷測定

測定の目的

上記の疑似パケットアセンブリ環境を用い，同じサイズのパケットを送信し，送信時に連続させるパケット数を調整して，通常転送とアセンブリ転送での転送処理の負荷変化による基幹網内のルータ CPU 負荷を測定する.

測定

- 測定条件

  512Bytes のパケットを 1000pps，1500pps，2000pps，2500pps の 4 レベルで送信を行

う．この時図 4.2 の 2. エッジルータにおいて，通常転送時はそのまま転送し，アセンブリ転送時は 3 パケットを結合し，合計サイズを 1500Bytes にして転送を行う．このためアセンブリ転送時に図 4.2 の 3. 中継ルータに実際に流れるパケット総数は，通常転送時の 1/3 となる．

- 測定結果

  結果 (図 4.3) の読み方は，各 pps において通常転送時は 512Bytes パケット*pps の転送処理にかかる CPU 負荷を表し，アセンブリ転送では，1500Bytes パケット*pps/3 の転送処理にかかる CPU 負荷を表す．

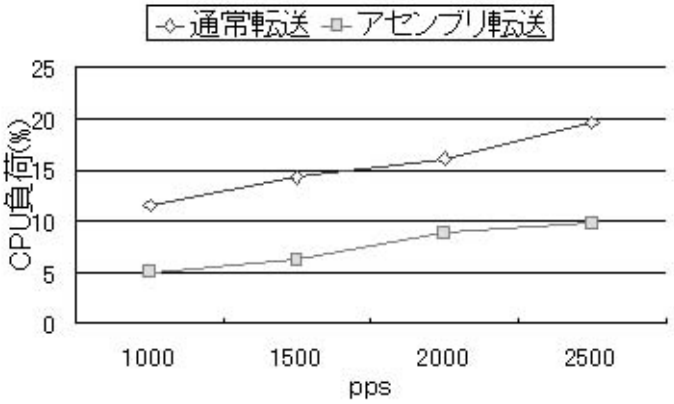

図 4.3　pps における基幹網内の中継ルータ CPU 負荷

- 考察

  パケットサイズが 3 倍に，転送パケット数が 1/3 倍に減少したとこで，中継ルータの CPU 負荷がどの pps の場合も半分程度になった．これは，パケットサイズの大きいを転送するより，転送パケット数を減らす方が CPU 負荷を軽減できる証明し，負荷が変化しても同様の効果が得られることが証明できる．よってパケットアセンブリするこでパケット数を減少させることで基幹網の中継ルータ CPU の負荷軽減に繋がることがわかる．

4.2.3 パケットサイズの変化による基幹網上の CPU 負荷測定

測定の目的

上記の疑似パケットアセンブリ環境を用い，パケットサイズを変更して送信し，通常転送とアセンブリ転送のパケットサイズとパケット数の変化による基幹網内のルータ CPU 負荷を測定する.

測定

- 測定条件

  1500Bytes のパケットを送信時に 256Bytes，512Bytes，1024Bytes，1500Bytes にフラグメントし連続して送信を行った．この時図 4.2 のアクセス網にあたる 1-2，4-5 間 MTU1 を基幹網にあたる 2-4 間 MTU2 に対し，通常転送時は MTU1=MTU2，アセンブリ時は MTU1<MTU2 とすることで 3 を流れるパケットサイズを調節する. パケットサイズは比較しやすい値を選択し，現ネットワーク上に流れるパケットサイズ分布と関係ない.

- 測定結果

  結果 (図 4.4) の読み方は，各 pps において通常転送時は (条件サイズのパケット)*(総送信パケット数) の転送処理にかかる平均 CPU 負荷を表し，パケットアセンブリ転送では，(1500Bytes パケット)*(総送信パケット数)/1500/(条件サイズ) の転送処理にかかる平均 CPU 負荷を表す.

- 考察

  この結果より，小サイズパケットをアセンブリすることで CPU 負荷の軽減を実現が証明できる．また，1024Bytes の通常転送時の CPU 負荷とアセンブリ時の CPU 負荷を比べると，2 パケットでもアセンブリすることの有効性が証明できる.

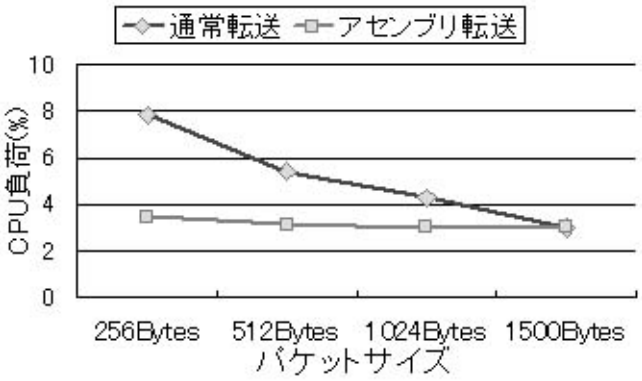

図 4.4 パケットサイズによる基幹網内の中継ルータ CPU 負荷

4.3 考察

以上の測定は全てのパケットをアセンブリ対象とした一経路上での測定である. しかし, pps(中継ルータにかかる負荷) が変化した場合でも同様の CPU 負荷を軽減でき，数多くアセンブリすることで CPU 負荷をより軽減の実現を示した. このことは，実際のネットワークでは全パケットをアセンブリすることは不可能であるにしても，少なからず中継ルータの CPU 負荷を軽減できることは表したと推測する. ただし，この測定は測定要素が少なく，また，中継ルータの CPU 負荷はわずかな物である. そこでより確かな結果を得るため，今後, 本測定で利用したネットワーク環境を用い測定する内容を挙げる.

4.3.1 今後予定している測定内容

1. 図 4.2 の 3 へ高負荷を与えた場合の CPU 負荷を測定する.
2. 現在のネットワークを測定し，パケットの割合 (シングル/マルチフローなど) やアセンブリ対象となる或る一定時間に通過するパケット数，また，アセンブリ対象となるパケットの存在数など基幹網を流れるパケット分布を調査する. その結果より，図 4.2 の 3 へ通常転送時とアセンブリ転送時のトラフィックを与え，負荷を測定する.
3. 4.1 節で挙げた項目の測定

などである.

第 5 章

まとめ

5.1 前章までのまとめ

インターネットの普及によってネットワークの大規模化や回線の高速化が進む中で，問題の一つに中継機器と送受信機器の処理速度がある．ネットワーク上のパケットが増加しているためである．そのパケット伝送にかかる負荷が大きくなっている. 特にアクセス網からパケットが流れる基幹網に負荷がかかる. その負荷とは，パケットの経路選択と転送処理である. そのうち転送処理についての研究を行った. パケットサイズは，MTU によって制限され基幹網内では，小サイズのパケットが大半を占めている (図 2.1). そのため，基幹網の MTU サイズで転送する場合と比べ，ヘッダ情報が多くなり，転送処理にかかるオーバーヘッドが大きくなる．この問題を解決するために，アクセス網と基幹網を繋ぐエッジルータでパケットデフラグメントを用いようとした．しかし，経路 MTU 探索などパケットの大半がフラグメントされていない状態で伝送されている．そこで，パケットアセンブリという方式を考案した．これは，エッジルータで基幹網へと伝送されるパケットの内，送信先 IP アドレス又は，送信先ネットワークアドレスが一致するものを，結合するものである．本方式における詳細を仕様として紹介し，有効性を示すための測定を行った．一経路上における pps の変化，パケットサイズの変化から，通常転送時とアセンブリ転送時の基幹網内の中継ルータ CPU 負荷を測定した. 共に，アセンブリ転送時の CPU 負荷の軽減が証明できた．今後，実際のネットワークに近いネットワーク環境を構築し，測定を行っていく．また，受信機器の処理速度の向上のための方式も考案しなければならない.

5.2　今後の研究展開

4.3.1 節でも示したように，パケットアセンブリの有効性を示すために，測定出来ていない項目を調査する．また，パケットアセンブリ用のボードを FPGA を用いて作成できる環境が整っているので，パケットアセンブリ対応エッジルータの作成の為に，アセンブリ回路とデバイスドライバの作成を行っていく．本稿でも記したように，IPv4 でパケットで MFbit のセットされているものが大半を占めているため，デフラグメント処理を高速化に有効性が見いだせない. そこで，IPv4 環境においては，IP 層ではなくトランスポート層に重点を置き，TCP の desegmentation の高速化処理を検討する. もしくは，IPv6 でもフラグメント処理は存在し，送信端末でのみフラグメントが行われるようになっている．IPv6 では，経路制御など多くの情報を使って伝送を行うため，IP 層でデフラグメント処理を用いるアプリケーションが増える可能性がある．そこで，IPv6 のデフラグメント処理の高速化も検討項目に入れていきたい.

謝辞

本研究を行うに際し, 多大なるご指導, 御鞭撻を頂いた, 本情報システム工学科の島村 和典 教授, ならびに通信・放送機構高知トラヒックリサーチセンターの神田 敏克 研究員に深く感謝致します. 測定環境を提供して下さった通信・放送機構高知トラヒックリサーチセンターの皆様に感謝致します. また, 本講座院生 中平 拓司氏, ならびに本講座学部生 稲葉 昭好君, 浦西 慶規君, 岡村 俊弘君, 川崎 道雄君, 坂田 青児君, 辻 貴介君, 野中 健史君, 橋本 江里子さん, 吉村 百子さんに感謝致します. さらに在学中, 親切なる御助言を頂いた清水 明宏 助教授, 福本 昌弘 講師 をはじめとする諸先生方に心より感謝致します.

参考文献

[1] http://www.napster.com/, Napster (Feb,2001)

[2] http://www.zdnet.co.jp/eweek/0008/03/00080301.html (Feb, 2001)

[3] http://www.caida.org/outreach/resources/learn/packetsizes/ (Feb, 2001)

[4] http://www4.nikkeibp.co.jp/NCC/nccnews/ncc577.html (Feb,2001)

[5] http://www.zdnet.co.jp/pcweek/archives/981130/981130p4021.html (Jan, 2001)

[6] http://www.alteon.co.jp/cobalt/, alteon web systems (Feb, 2001)

[7] 堀内 晋也 浅谷 耕一, 高効率パケット多重化ネットワークの検討, 電子情報通信学会 信学技報 CQ2000-38(2000-19)

[8] Toshikatsu Kanda Kazunori Shimamura, A Throughput improvement on Routers by the control of packet size,Technical report of IEICE SSE2000-114, IN2000-65, CS2000-45(2000-09)

[9] Toshikatsu Kanda Kazunori Shimamura, A study on the rate of packet reachability and RTT in real networks,Technical report of IEICE CQ2000-3 (2000-05)

付録 A

本文中の技術紹介

A.1　各データリンクの MTU

表 A.1　データリンクごとの MTU(Bytes)

データリンク	MTU	Total Length
IP の最大 MTU	65535	-
Hyperchannel	65535	-
IP over HIPPI	65280	65320
16Mb IBM Token Ring	17914	17958
IP over ATM	9180	-
IEEE802.4 Token Ring	8166	8191
IEEE802.5 Token Ring	4464	4508
FDDI	4352	4500
Ethernet	1500	1514
PPP	1500	-
IEEE802.3 Ethernet	1492	1514
IP の最小 MTU	68	-

A.2　デフラグメント処理手順

IP パケットの reassemble は，IP データグラム 3.1 のヘッダにあるフラグ/フラグメント・オフセットフィールドを用いて行われる.

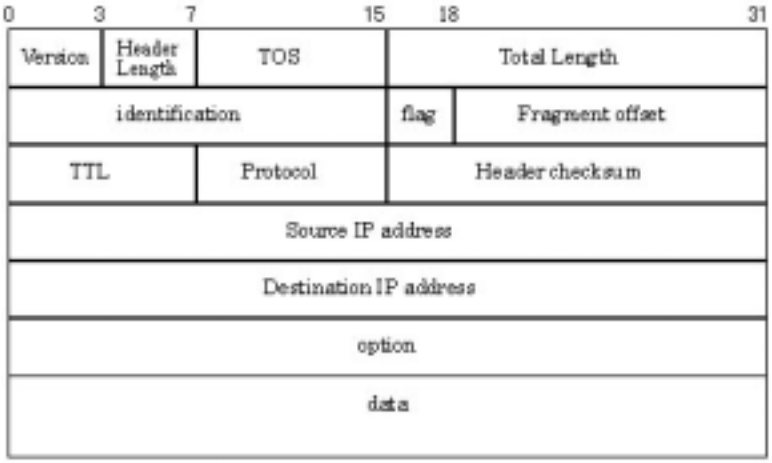

図 A.1　IP ヘッダ

flags は 3bit から成り，unsed:fragment 禁止フラグ:More Fragment フラグ (0 の場合最終フラグメント) となっている. フラグメント・オフセットは，13bit から成り，8Bytes を 1 ブロックとして考えたとき，何ブロック目からのデータであるかを示す. これによりオリジナルのデータの順番が維持される. 処理手順は次の表 A.2 である. 同じフラグメントが来た場合には最後のものを適用する.

A.3　経路 MTU 探索 (Path MTU Discovery)

RFC1191 に経路 MTU 探索 (Path MTU Discovery) という技術が定められている. 探索に際し以下の方法が採る. ホストが DF(Don't Fragment) フラグをセットした IP データグラムを送信し，これを受信したルータで，そのデータグラム長が次のホップの MTU を越える場合には，ICMP 宛先到達不能メッセージを送信側に返信する. このメッセージは「Datagram Too Big」を示し，コードは「フラグメントが必要にも関わらず DF フラグがセットされている」を意味する. このとき，RFC1191 は，ICMP ヘッダの未使用領域 32bit

の下位 16bit に，次のホップの MTU をセットして返信する．このメッセージを受け取ったホストは，DF フラグをセットし，送られてきた次のホップの MTU のサイズで再び送信する．こえを繰り返し送信先に届いたとき，探索された経路上で送信できる最大 MTU(Path MTU) を取得できたことになる．以降のパケットは，この PMTU に分割され送信される．

表 A.2　reassemble の処理手順

1	バッファID に「送信 IP アドレス/プロトコル/識別子」をセット
2	もし，フラグメント・オフセットと MoreFragment フラグ= 0 ならば処理終了 (非 fragment)
3	バッファが未設定ならば，バッファ/タイマー/全長を初期化して次に継続
4	フラグメントデータをデータファイルに格納
5	フラグメントビットテーブル上の該当ブロックビットをセット
6	MF フラグが 0(最終フラグメント) ならば，全データ長を計算
7	フラグメント・オフセットが 0(第１フラグメント) ならば，ヘッダをヘッダバッファに格納
8	全データ長が 0 でなく，且つ，ビットテーブルのそのブロックまでがセットされている (そのブロックまでが受信済) ならば，全長を更新し，データグラムをインターネットモジュールの再構成の次の処理に送り，全ての再構成リソースを開放し，このバッファID の処理は終了
9	タイマーに，現在のタイマー値と TTL の大きい方をセット
10	次のフラグメントまたはタイムアウト待ち
11	タイムアウト時は，全ての再構成処理を停止

付録 B

プログラムの紹介

B.1 擬似アセンブリプログラム

送信用，受信用，アセンブリ用，デアセンブリ用の 4 つ必要となる．しかし，どのプログラムも似ているため，例としてアセンブリ用のプログラムを記す.

```
/*
** virtual assembly router 1
** 2000/08/04~
** H.Kobayashi + T.Kanda
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <syslog.h>
#include <varargs.h>
#include <string.h>
```

```
/* normal packet data size */
#define BUFFSIZE2     484
/* assemble packet data size*/
#define BUFFSIZE3     1452
/* packet receive udp port number */
#define ATTACK_PORT   1285
/* packet send udp port number */
#define REATTACK_PORT 1290

main(argc,argv)
     int argc;
     char *argv[];
{
  char    buff2[BUFFSIZE2];
  char    buff3[BUFFSIZE3];
  char    *msgfmt = "Data is %s\n";
  int     sockdsc2,sockdsc3;
  int     loop;
  int     fromlen;
  int     count = 1;
  struct  hostent  *hent2,*hent3;
  struct  sockaddr_in      sockadd2,sockadd3;

  /* UDP socket create for packet send */
  if((sockdsc2 = socket(PF_INET, SOCK_DGRAM, 0)) < 0)
```

```
          exit(1);

  /* header infomation set */
  sockadd2.sin_family      = AF_INET;
  sockadd2.sin_addr.s_addr     = htonl(INADDR_ANY);
  sockadd2.sin_port      = htons(ATTACK_PORT);

  if(bind(sockdsc2, (struct sockaddr *)&sockadd2,
          sizeof(sockadd2)) < 0) exit(2);

  /* UDP socket create for packet receive */
  if((sockdsc3 = socket(PF_INET, SOCK_DGRAM, 0)) < 0)
          exit(1);

  /* header infomation set */
  if(argc < 2 || NULL == (hent3 = gethostbyname(argv[1])))
          exit(2);
  memcpy(&sockadd3.sin_addr, hent3->h_addr, hent3->h_length);
  sockadd3.sin_family      = AF_INET;
  sockadd3.sin_port        = htons(REATTACK_PORT);

  /* receive three packets, packet data assembly,
          send one packet */
  for(;;){

    /* reset data buffer */
```

```
    bzero(buff2,sizeof(buff2));
    bzero(buff3,sizeof(buff3));

    /* receive packet and packet data save to buffer */
    while(count <= 3){
      fromlen = sizeof(struct sockaddr_in);
      if(recvfrom(sockdsc2, buff2, sizeof(buff2), 0,
         (struct sockaddr *)&sockadd2, &fromlen)
          == -1) exit(3);
      memcpy(&buff3[BUFFSIZE2*count],buff2,sizeof(buff2));

      bzero(buff2,sizeof(buff2));
      count = count++;
    }

    /* send packet to edge router */
    if(sendto(sockdsc3, buff3, sizeof(buff3), 0,
       (struct sockaddr *)&sockadd3, sizeof(struct sockaddr))
       == -1) exit(3);

    count = 1;
  }

  close(sockdsc2);
  close(sockdsc3);
}
```