高品質インターネット　1－11
（2001. 11. 21）

コンテンツスイッチ・プロキシサーバのための TCP 非対称スプライシング方式の提案

小林正好　村瀬勉

NEC ネットワーキング研究所
〒216-8555 川崎市宮前区宮崎 4 丁目 1-1
TEL:044-856-2123　FAX:044-856-2230
E-mail: {m-kobayashi@eo, t-murase@ap}.jp.nec.com

あらまし　コンテンツスイッチやプロキシサーバは，クライアントからのリクエストのフォワーディング先をアプリケーション層の情報を用いて決定している．既存のコンテンツスイッチや一部のプロキシサーバは，フォワーディング先の決定後に，TCP スプライシングによって TCP 終端処理を避ける事で，フォワーディング性能を向上させている．しかし，TCP スプライシングを行っている間は，アプリケーション層の情報を利用する事が出来なくなるため，ユーザの遅延を大幅に削減出来るパイプライン HTTP リクエストに対して正しくサーバ選択を行う事が出来ない．本報告では，この問題を解決する，TCP 非対称スプライシングとサーバ切替方式を提案する．TCP 非対称スプライシングは，クライアントからサーバへ流れるアプリケーション層情報をコンテンツスイッチやプロキシサーバが常に利用する事と，サーバからクライアントへのデータをパケットフォワーディングによって高速にフォワードする事を同時に実現する．また，TCP 非対称スプライシングと共に，提案するサーバ切替方式を用いる事によって，パイプライン HTTP リクエストに対して正しくサーバ選択を行う事が可能になる．性能評価では，従来のコンテンツスイッチに比べ，パイプラインリクエストを提案方式によって扱うコンテンツスイッチは，クライアントのコンテンツ取得遅延を大幅に削減出来る事を示した．
キーワード：　プロキシ，コンテンツスイッチ，レイヤ 7 スイッチ，サーバ負荷分散，TCP スプライシング

Asymmetric TCP Splicing for Content-Based Switches and Proxies

Masayoshi Kobayashi　Tutomu Murase
Networking Laboratories, NEC Corp.
1-1, Miyazaki 4-chome, Miyamae-ku, Kawasaki, Kanagawa 216-8555, Japan
TEL: +81-44-856-212 3　FAX:+81-44-856-2230
E-mail: {m-kobayashi@eo, t-murase@ap}.jp.nec.com

Abstract　Application layer proxies, such as, content-based switches make forwarding decisions (server selections) based on an application layer information and forward data in the application layer. After making forwarding decisions, some existing proxies and most content-based switches increase their forwarding performance by TCP splicing, which releases them from maintaining TCP endpoints and allows them to forward data by packet forwarding. However, TCP splicing prevents proxies and content-switches from using the application layer information for forwarding decisions. Thus the existing content-based switches cannot hand off pipelined HTTP transactions, which can greatly reduce client perceived latencies. This paper proposes an asymmetric TCP splicing and a method to hand off HTTP transactions between servers. Asymmetric TCP splicing allows the content-based switches to use all the application layer information in the TCP data stream from clients to servers, although it allows the switches to forward the TCP data stream from servers to clients by packet forwarding. The proposed handoff method, which users a TCP half-close to detect the boundary of server responses, enables content-based switch to support pipelined HTTP transactions in combination with asymmetric TCP splicing. Our evaluation shows that if a content-based switch supports pipelined HTTP transactions by asymmetric TCP splicing and our proposed handoff method, client -perceived latencies can be reduced substantially.
Keywords: Proxy, content-based switch, layer 7 switch, server load balance, TCP splicing.

1. はじめに

近年，アプリケーションプロキシのような，アプリケーション層の情報に基づいたデータフォワーディングを行うネットワーク機器が重要性を増している．従来のIPルータのようなネットワーク機器では，パケットヘッダ内の情報に基いたフォワーディング先の判断を行ってきたが，パケットヘッダ内の情報は限られており，多様なサービスの提供やサービスの差別化を実現するには不十分である．アプリケーション層で運ばれるコンテンツを意識した転送先の判断を行う必要になってきている．たとえば，WWWサーバ群の手間で，ユーザリクエスト情報に応じたサーバ選択を行うコンテンツスイッチ[1](レイヤ7スイッチ，あるいは，URLスイッチ[2]とも呼ばれる)は，この代表的な例である．パケット内の情報を用いて同様のサーバ選択を行えるスイッチに，レイヤ4スイッチがあるが，ユーザからのリクエスト内の情報に応じたサービス提供は不可能である．また，単純なサーバ負荷分散機能に限っても，リクエストされたURLに基づいたサーバ選択が不可能であるため，全ての選択され得るサーバが同一のコンテンツを保持している必要があるなど，制約も多い．

コンテンツスイッチに代表されるプロキシ型のネットワーク装置は，一方でクライアントと，他方でサーバとの間に，それぞれTCPコネクションを保持し，これらのTCPコネクションを流れるデータを装置内でアプリケーション層でフォワーディングしている．この動作にはTCPの終端処理とアプリケーション層でのデータフォワーディングとを必要とする．これらの処理は，IPルータやレイヤ4スイッチが，パケットヘッダ情報を基にパケットフォワーディングによってデータフォワーディングを行っているのに比べ負荷が非常に大きい．このため，2節で述べるように，TCPスプライシングと呼ばれる手法を用いてフォワーディング性能の向上を図っている．TCPスプライシングとは，サーバとクライアントの間に張られた，2つのTCPコネクションを流れるパケットを，サーバとクライアントが直接通信しているかのようにパケットヘッダ書き換えてフォワーディングする事によって，コンテンツスイッチやプロキシサーバをTCPの終端処理から開放し，フォワーディング性能を向上させるものである．TCPスプライシング処理は単純でハードウェア化可能であり，ワイヤスピードフォワーディングを実現する事は容易である．実際，近年のコンテンツスイッチ[7, 8]は，上記のヘッダ書き換え処理を行うハードウェアを搭載し，TCPスプライシングを使ってワイヤスピードのデータフォワーディング処理を実現している．

現在，インタネット上のコンテンツの大半は，TCP層の上位プロトコルであるHTTP(Hypertext Transfer Protocol)を用いて行われている．もともとHTTPでは，1つのHTTPトランザクション処理(リクエストとそれに対する応答)に対して1つのTCPコネクションを必要としていた．しかし，現在のバージョンのHTTP(HTTP/1.1[5])や，1つ前のバージョンのHTTP(HTTP/1.0[4])の多くの実装では，複数のHTTPトランザクションを1つのTCPコネクションで扱うpersistent HTTPが用いられている．Persistent HTTPでは，クライアントは任意の数のリクエストを1つのTCPコネクションに送信する事が可能で，サーバは同じTCPコネクションに，受け取ったリクエストの順に対応するレスポンスを返送する．さらにHTTP/1.1では，クライアントは任意の数のリクエストを，対応するレスポンスを受け取る前に送信可能な，パイプラインリクエストを用いるように推奨されている．パイプラインリクエストを用いると，サーバからの応答を待たずに次々とリクエストを発行出来るため，トランザクション毎に生じていたネットワークのラウンドトリップ遅延を削減出来，2倍近くの高い処理スループットを得ることが出来る[3]．以下ではパイプラインリクエストによって処理されるHTTPトランザクションをパイプライントランザクションと呼ぶ．

従来のTCPスプライシングを用いているコンテンツスイッチやプロキシサーバの問題点は，パイプライントランザクションを正しく扱うことが出来ない事である．これは以下の理由による．パイプライントランザクションを扱うためには，クライアントからのリクエストを全て認識し，各リクエストに基づいてサーバを選択し，クライアントとのコネクションを維持しながら，接続するサーバを切り替えなければならない．しかし，従来のコンテンツスイッチではクライアントからの最初のリクエストに基づいてサーバを選択した後は，TCPスプライシングによってクライアントとサーバの間に流れるデータをフォワードしており，2番目以降のクライアントからのリクエストを認識したサーバ選択が出来ない．TCPスプライシングによってパケットフォワーディングしている間は，クライアントのリクエスト内の情報等のアプリケーション層の情報を得ることが出来ないからである．従来のコンテンツスイッチでは，クライアントがパイプラインリクエストを送信しないと仮定し，この仮定に基づいてリクエスト毎にTCPスプライシングを中断する事によって，2番目以降のリクエストに基づいたサーバ選択を実現している．このため，パイプラインリクエストを扱えない．

本報告では，コンテンツスイッチ，あるいは，プロキシサーバにおいて，TCPスプライシングを行っている場合と同等のフォワーディング性能を維持しつつ，パイプラインリクエストを扱う事を可能にする方式を提案する．提案内容は，クライアントからのリクエストをアプリケーション層でサーバへフォワードしながら，サーバからのレスポンスをパケットフォワーディングによってクライアントへと転送するTCP非対称スプライシング方式，および，TCP非対称スプライシングを行いつつ，TCPハーフクローズを利用し，パイプラインリクエストに対して，サーバを切替える方式である．最後に提案方式の性能評価結果も示す．

2. 従来のコンテンツスイッチ

本節では，従来のコンテンツスイッチおよびプロキシサーバの，TCP スプライシングを用いたフォワーディング性能向上方法，および，サーバ切替方法について説明する．

2.1. TCP スプライシング

TCP スプライシングを説明するため，図 1 に従来のコンテンツスイッチにおける，クライアント，コンテンツスイッチ，サーバ間のパケットのやりとりを示したタイミングチャートを示す．ここで，クライアントはサーバの IP アドレスを XA であると認識していると仮定する．HTTP トランザクションでは，まずクライアントが，TCP SYN パケットをアドレス XA に対して送信し，TCP コネクションを張ろうとする．コンテンツスイッチは，このパケットを受け取り(Step 1)，TCP の 3-way ハンドシェイクを実行し，クライアントとの間に TCP コネクションを開設する(Step 2)．コネクション開設後，クライアントは最初の HTTP リクエスト(HTTP request 1)を，開設された TCP コネクションに送信する(Step 3)．コンテンツスイッチはこのリクエストを受け取り，リクエストに基づいて適切なサーバ（サーバ 1)を選択した後，選択されたサーバ 1 との間に別の TCP コネクションを張り(Step 4)，リクエストをそのコネクションにフォワードする．次に，サーバ 1 はリクエストを受け取り，レスポンスをコンテンツスイッチへと送り返す．コンテンツスイッチはサーバ 1 からのレスポンスを含んだ最初のパケットを受け取ると(Step 6)，TCP スプライシングを開始する．すなわち，クライアントとサーバ 1 に張られた 2 つの TCP コネクションの終端処理を行うことをやめて，ヘッダ書き換えとパケットフォワーディング処理を開始する．ヘッダ書き換え方法は表 1 に示した通りである．ここで，dSeq_u = SSEQ-XSEQ, dSeq_d=CSEQ-YSEQ, であり，CSEQ と SSEQ はそれぞれクライアント，サーバの TCP の初期シーケンス番号である．ヘッダ書き換えに加えて，IP ヘッダおよび TCP のチェックサムのインクリメンタルな更新も行われる．このヘッダ書き換えにより，サーバ 1 とクライアントは直接 TCP コネクションを張っているかのように通信を行い，コンテンツスイッチは TCP 終端処理から解放される．

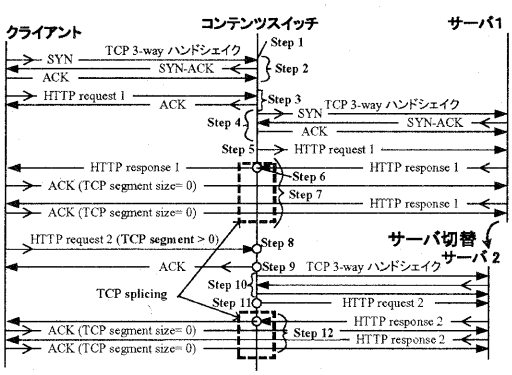


図 1: 従来のコンテンツスイッチでのタイミングチャート

表 1: パケットヘッダ書き換え規則

	クライアント→サーバ		クライアント→サーバ	
	書換前	書換後	書換前	書換後
宛先IPアドレス	XA	SA	YA	CA
送信元IPアドレス	CA	YA	SA	XA
宛先TCPポート番号	XP	SP	YP	CP
送信元TCPポート番号	CP	YP	SP	XP
TCPシーケンス番号	s	s - dSeq_u	s	s - dSeq_d
ACK番号	a	a + dSeq_d	a	a + dSeq_u

CA,CP: クライアントのIPアドレス、TCPポート番号
XA,XP: コンテンツスイッチのクライアント側コネクションのIPアドレス，TCPポート番号
YA,YP: コンテンツスイッチのサーバ側コネクションのIPアドレス，TCPポート番号
SA,SP: サーバののIPアドレス、TCPポート番号

次に，TCP スプライシングを，TCP のフィードバック制御の面から説明する．TCP は 2 つの TCP 終端点の間で，フロー制御および再送制御をフィードバック形式で行っている．TCP コネクションは全二重であり，片方向に流れるパケットは TCP データだけでなく反対方向のデータフローのフィードバック情報を TCP ヘッダ内に保持している．この情報には，ACK 番号と広告ウィンドウサイズが含まれる．ACK 番号は受信側がこれまでに正しく受け取った TCP データのバイト位置を示しており，再送制御に用いられる．広告ウィンドウサイズは，受信バッファの残り容量をしめしており，フロー制御に用いられる．

図 2(a)は，コンテンツスイッチが TCP スプライシングを開始する前のフィードバックループを示している．クライアントとスイッチ，および，スイッチとサーバの間に 2 つのフィードバックループがあり，コンテンツスイッチは 2 つの TCP 終端処理を行っている．これに対し，図 2(b)に示すように，TCP スプライシングを行った後では，フィードバック制御はクライアントとサーバにおいて行われ，スイッチは TCP 終端処理から解放されている．これによってスイッチはフォワーディング性能を向上させることが可能となっている．

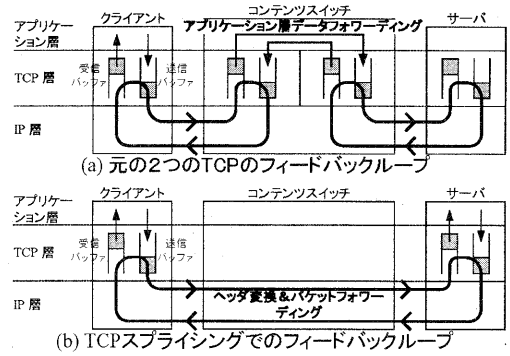


図 2: 従来のコンテンツスイッチでのフィードバックループ

2.2. サーバ切替方式

従来のコンテンツスイッチでのサーバ切替方式を図1を用いて説明する．従来のコンテンツスイッチはクライアントがパイプラインリクエストを送らないと仮定している．TCPスプライシングを行っている最中に，コンテンツスイッチが1バイト以上の長さのTCPデータを含むパケットを受け取った場合(図1のStep 8)，上記の仮定から，クライアントは1つ前のリクエストに対するレスポンスを受信完了し，このパケットは，次のクライアントからのリクエストを含んでいるという事になる．そこでスイッチは，TCPスプライシングしているコネクションを，元の2つのコネクションとしての処理に戻す．すなわち，TCPの終端処理を開始する．クライアントからリクエストを完全に受け取ると(Step 9)，リクエストに基づいて適切なサーバを選択する．ここで，選択されたサーバとの間にTCPコネクションが存在しない場合，スイッチは新たなTCPコネクションを選択されたサーバ(サーバ2)との間に開設し(Step 10)，リクエストをそのコネクションに送信する．サーバからのレスポンスを含んだ最初のパケットを受け取った時点で，サーバとのコネクションとリクエストを受け取ったクライアントとのコネクションについて再びTCPスプライシングを行う(Step 12)．Step 9で，選択されたサーバとの間にTCPコネクションが存在する場合（この場合は図1に示していない），新たにコネクションを開設せず，そのコネクションを使って同様の処理を行う．

上記のサーバ切替方法は，クライアントがパイプラインリクエストを送信しない場合においてのみ正しい処理を行う．もし，クライアントがパイプラインリクエストを送信すると，すなわち，クライアントがサーバからのレスポンスの受信の最中に後続のリクエストを送信してしまうと，スイッチがTCP終端処理を開始してしまうため，該当のレスポンスはクライアントへ正しく届かない．この場合でも，アプリケーション層でデータをクライアントへフォワードする事は可能であるが，フォワーディング性能は，TCPスプライシングを用いないコンテンツスイッチと同等にまで下がってしまう．これらの問題の解決には，(1)サーバからのリプライを処理負荷の低いパケットフォワーディングによって転送しつつ，クライアントからのリクエストをアプリケーション層でサーバへと転送する方法，および，(2)接続先のサーバを(1)の転送方法を行いつつ，クライアントとのコネクションを保ったまま切り替える方法が必要である．

3. TCP非対称スプライシング

本節では，コンテンツスイッチがクライアントからのリクエストのアプリケーション層の情報を常に利用可能にしながら，サーバからクライアントへのレスポンスを処理負荷の低いパケットフォワーディングによって転送するTCP非対称スプライシング方式，および，それを使ったパイプライントランザクションを処理可能なサーバ切替方式を提案する．提案方式によって，サーバ，クライアントのソフトウェアに変更を加えることなく，コンテンツスイッチは，パイプライントランザクションを処理可能となる．

3.1. アイデア

一般にHTTPトランザクションでは，クライアントからサーバへの個々のリクエストは数百バイト程度であるが，サーバからクライアントへのレスポンスは平均で数十キロバイトであり，クライアントからサーバへのデータ転送量に比べ，サーバからクライアントへのデータ転送量が非常に大きい．このため，スイッチのフォワーディング性能はサーバからクライアント方向へのデータ転送能力が支配的である．一方，コンテンツスイッチが必要とする情報の観点からは，クライアントからサーバへ転送されるリクエストを得る必要はあるが，サーバからクライアントへのレスポンスについては内容を知る必要がない．以上の事から，コンテンツスイッチは，クライアントからのリクエストはアプリケーション層でサーバへフォワードし，サーバからのリプライはクライアントへTCPスプライシングのように，パケットフォワーディングで，処理負荷を低く保って転送する方法が適していると言える．これがTCP非対称スプライシングのアイデアである．

3.2. TCP非対称スプライシングにおけるフィードバックループ

3.1 節でアイデアを示したTCP非対称スプライシングの実現に必要とされるフロー制御，再送制御のフィードバックループを図3に示す．図2(b)で示した，従来のTCPスプライシングでの単一のフィードバックループと異なり，図3には，3つのフィードバックループLa, Lb, Lcが存在し，クライアントからサーバへのデータフローと，サーバからクライアントへのデータフローに対して非対称に働いている．

クライアントからサーバ方向のデータフローについては，コンテンツスイッチがクライアントからのリクエストをアプリケーション層で受け取れるように，図3のループLaで示されるようにACK番号と広告ウィンドウサイズをクライアントに対して返送する必要がある．また，コンテンツスイッチはクライアントから受け取ったリクエストをサーバへ信頼性をもっ

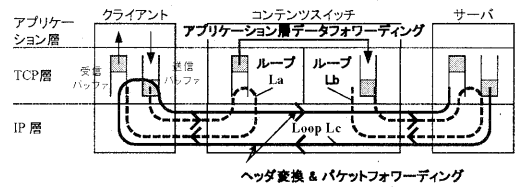


図3:コンテンツスイッチでのフィードバックループ

て転送するために，もう１つのフィードバックループ Lb をサーバーとの間に持つ必要がある．

サーバからクライアント方向のデータフローについては，コンテンツスイッチはその内容を知る必要がない．サーバとクライアント間で直接フィードバックループ Lc が維持すればよく，スイッチはパケットヘッダの変更とパケットフォワードを行えばよい．このヘッダ変更処理とパケットフォワード処理は従来の TCP スプライシングを行うスイッチが行っているものと同等であり，サーバからクライアント方向のデータフローについては，従来の TCP スプライシングを行うスイッチが行っているものと同等フォワーディング性能が出せる．

3.3. コンテンツスイッチにおける TCP 非対称スプライシング

コンテンツスイッチが TCP 非対称スプライシングをどのように実現するのかを詳細に説明する．

図４にクライアント，TCP 非対称スプライシングを実行するコンテンツスイッチ，サーバとの間でのパケットのやりとりのタイミングチャートを示す．図４の Step 1 から Step 5 までの処理は，2 節で図 1 を使って説明した，従来のコンテンツスイッチの処理と同様である．図４の Step 6 において，スイッチがサーバからのレスポンスを含む最初のパケットを受け取った時点で，コンテンツスイッチは TCP 非対称スプライシングを開始する．

図５に TCP 非対称スプライシングを実行するコンテンツスイッチのブロック図を示す．各 HTTP トランザクションに関して，コンテンツスイッチは２つの TCP 終端点，C および S を持つ．TCP 終端点 C および S は，従来の TCP 終端点とほぼ同等の処理を行うが若干の処理の違いがある．TCP 終端点 C はクライアントからのデータは受け取るが，クライアントへデータを送信しない．また，TCP 終端点 S はサーバへデータを送信はするが，サーバからデータを受信しない．これら２つの TCP 終端点は，図４の機能ブロック Ba, Bb, Bc とともに，以下のように TCP 非対称スプライシングを実現する．

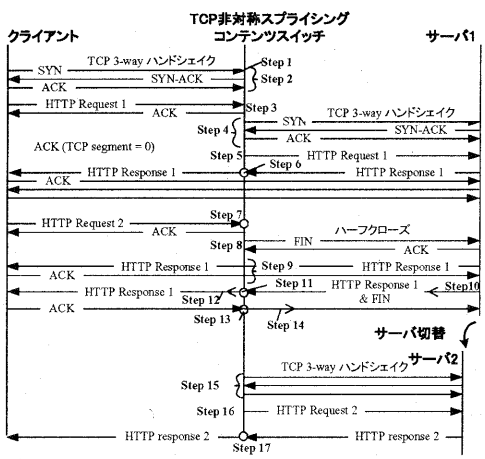


図 4: 非対称スプライシングを行うコンテンツスイッチの場合のタイミングチャート

機能ブロック Ba:

TCP 終端点 C がパケットをクライアントから受け取ると，クライアントへ返送すべき ACK 番号と広告ウィンドウサイズが，機能ブロック Ba へと渡され，以下の処理が行われる．

A-1: 機能ブロック Bb から送られてきたパケットがあれば，そのパケットの ACK 番号と広告ウィンドウサイズを，TCP 終端点 C から渡されたもので上書きし，パケットをクライアントへ送信する．

A-2: 機能ブロック Bb から送られてきたパケットがない場合，TCP 終端点 C から渡された ACK 番号と広告ウィンドウサイズを持つ TCP ACK パケット（データセグメントはない）を生成し，クライアントへ送信する．

以上の処理によって，図３で示したフィードバックループ Lc が確立され，TCP 終端点 C は常にリライアブルにクライアントからのリクエストを受け取る事が出来る．

機能ブロック Bb:

機能ブロック Bb は，サーバからパケットを受信した時，以下の処理が行う．

B-1: パケットの ACK 番号，広告ウィンドウサイズを読みとり，TCP 終端点 S へと渡す．

B-2: パケットが TCP データを含む場合，そのパケットをヘッダ変換処理をして機能ブロック Ba へとフォワードする．このヘッダ変換処理は，２節の表１の「サーバ→クライアント」の欄のものと同じである．パケットが TCP データを含まない場合にはパケットを破棄する．

B-1 の処理は図３に示したフィードバックループ Lb を確立するために必要である．また，B-2 において，TCP データを含まないパケットを破棄する理由は，もしこのパケットを，ヘッダ変換してクライアントへ送ってしまうと，クライアントにとっての重複 ACK パケットとなってしまい，クライアントのデータ転送速度を落としてしまうからである．

機能ブロック Bc:

機能ブロック Bc がクライアントからのパケットを受け取ると，その ACK 番号，および，広告ウインドウサイズが読みとられ，TCP 終端点 S へと受け渡される．この際に，ACK 番号については，２節の表１の「サーバ→

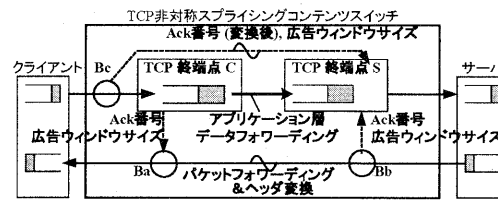


図 5: TCP 非対称スプライシングを行うスイッチの内部ブロック図

クライアント」の列の，ACK 番号の行に書かれた変換が行われる．TCP 終端点 S では，S で維持している，サーバへ返送されるべき ACK 番号と広告ウィンドウサイズの値を，Bc から受け渡されたものに置き換える．これらのいずれかの番号に変化があった場合，そのときにサーバへ送信するパケットが存在していなければ，これらの ACK 番号，広告ウィンドウサイズを含む ACK パケットを生成し，サーバへと送信する．以上の処理によって図 3 に示したフィードバックループ Lc が確立される．

3.4. TCP 非対称スプライシングでのサーバ切替処理

次に，TCP 非対称スプライシングを用いて，パイプラインリクエストに対してサーバ切替処理を実現する方法を示す．2.2 節で述べたとおり，図 4 の Step 1～Step 6 は，どのようにコンテンツスイッチが TCP 非対称スプライシングを開始するかを示している．ここでは，Step 7 以降で提案するサーバ切替方式を説明する．

コンテンツスイッチがクライアントからの 2 番目のリクエスト(HTTP request 2)を，サーバから 1 番目のレスポンスを受信中に受け取ったとする (Step 7)，すなわち，クライアントがパイプラインリクエストを送信したとする．このとき，コンテンツスイッチは，TCP 非対称スプライシングを行っているので，サーバからのレスポンスをクライアントへとフォワードしつつ，HTTP request 2 を正しく受信できる．スイッチは HTTP request 2 を受信後，リクエストに応じて適切なサーバを選択する．このとき，選択されたサーバは，1 つ前のリクエストをフォワードしたサーバである場合と，そうでない場合との 2 つの場合があり得る．

もし，選択されたサーバが，1 つ前のリクエストをフォワードしたサーバと同じ場合（この場合は図 4 には示していない)，スイッチは単純にそのリクエストを選択されたサーバへと転送するだけでよい．

そうでない場合(図 4 の Step 7 以降にこの処理を示す)，サーバを切り替える必要がある．コンテンツスイッチは現在のレスポンスをクライアントが受信完了した後，HTTP request 2 に対するレスポンス(HTTP response 2)がクライアントへ届き始めることを保証しなければならない．提案手法では，TCP ハーフクローズを用いて以下のようにこの保証を行う．コンテンツスイッチは TCP FIN パケットをサーバ 1 へと送信し，現在のサーバとのコネクションに対して TCP ハーフクローズを実行する．こうすることによって，サーバ 1 は，現在のレスポンスをクライアントへ送信し終えた後すぐに，TCP FIN パケットをクライアントへと送るようになる．このサーバからの TCP FIN パケットを現在のレスポンスの終了の印として用いる事が出来る．サーバからのレスポンスに含まれる Content-length フィールドを用いてレスポンスの終了を知る方法もあるが，Content-length フィールドはアプリケーション層の情報であり，サーバからのレスポンスをパケットフォワーディングによってクライアントへと転送することでフォワーディング性能を向上させる TCP 非対称スプライシングでは，この情報を利用出来ない．FIN パケットは，TCP ヘッダのステータスビットで識別可能であり，TCP 非対称スプライシング実行中でも判別する事が可能である．

図 4 の Step 8 において，コンテンツスイッチが TCP FIN パケットをサーバに送信している．サーバ 1 は現在のレスポンス(HTTP response 1)の残りを送信後(Step 9)，TCP FIN パケットを送信する(Step 10)．コンテンツスイッチは，サーバからの FIN パケットを受信すると(Step 11)，以下の処理を行う．もし，FIN パケットが TCP データを含まない場合，FIN パケットに対する ACK パケットをサーバへと返信し，FIN パケットを破棄する．そうでない場合，パケットの FIN ビットを落とし，ヘッダ変換後，ヘッダ内のシーケンス番号(FSEQ)，および TCP セグメントサイズ(FLEN)を記憶し，パケットをクライアントへとフォワードする(Step 12)．

次に，コンテンツスイッチはクライアントからのパケットを監視し，その ACK 番号が FSEQ+FLEN であるパケット（HTTP response 1 の最後のバイトに対する ACK）の到着を待つ．コンテンツスイッチが該当パケットを受信すると (Step 13)，TCP 終端点 S(図 5)は対応する ACK パケットをサーバ 1 へと送信する(Step 14)．ここまでの処理によって HTTP request 1 に対するトランザクションが完了したことになる．

次にスイッチは，サーバ 2 へコネクションを開設し(Step 15)，HTTP request 2 をサーバ 2 へと送信する(Step 16)．このコネクション開設は，サーバ 1 がレスポンスをクライアントへ返している間に行う事も可能である．サーバ 2 はレスポンス(HTTP response 2)をスイッチへと返し，スイッチがこのレスポンスの最初のパケットを受け取ると，ヘッダ変換処理を行ってクライアントへとパケットを転送する．このヘッダ変換処理は，表 1 のヘッダ変換処理と同等であるが，表 1 の SA および SP をサーバ 2 の IP アドレスとポート番号に読み替える必要がある．また，dSeq_u および dSeq_d も，dSeq_u=SSEQ2-Seq_x, dSeq_d=Seq_c - YSEQ2 と再定義される．ここで，Seq_x, Seq_c, YSEQ2, SSEQ2 はそれぞれ以下のように定義される．

Seq_x: クライアントが最後にコンテンツスイッチから受け取った TCP シーケンス番号(FSEQ+FLEN).
Seq_c: コンテンツスイッチが HTTP request 1 を受信中に最後にクライアントから受け取ったシーケンス番号.
YSEQ2:サーバ 2 との間に開設した新しいコネクションの，コンテンツスイッチの初期シーケンス番号.
SSEQ2: サーバ 2 との間に開設した新しいコネクションの，サーバ 2 の初期シーケンス番号.

以上のヘッダ変換処理に加え，IP ヘッダおよび TCP チェックサムのインクリメンタルな更新を行う．

以上の提案したサーバ切替方式により，スイッチはパイプライン HTTP トランザクションをクライアントとの間のコネクションを保ったまま扱う事が可能になる．

4. 性能評価

本節では，ここまでに提案したTCP非対称スプライシングとサーバ切替方式を用いたコンテンツスイッチに関して，クライアントのコンテンツ取得にかかる平均遅延に関する性能評価を行う．比較対象は従来のTCPスプライシングを用いているコンテンツスイッチ（パイプラインリクエストを扱えない）である．提案のコンテンツスイッチに接続されたクライアントはパイプラインリクエストを送ると仮定し，従来のコンテンツスイッチに接続されたクライアントはパイプラインリクエストを送らないと仮定する．

議論のために，以下を定義する．

RTT: クライアントサーバ間のラウンドトリップタイム

Ts: 1つのリクエストの平均送信時間

Tr1: TCPスロースタート中の1つのレスポンスの平均送信時間

Tr2: TCPスロースタートが終わった後の1つのレスポンスの平均送信時間

p: あるリクエストが1つ前のリクエストと異なるサーバへと転送される確率．

また，以下を仮定する．

- リクエストやレスポンスの送信前にTCPのハンドシェイクは完了している．
- あるリクエストが1つ前のリクエストと同じサーバへと転送された場合，レスポンスはスロースタートが終わったTCPによって転送され，平均Tr2時間で送信できる．そうでない場合レスポンスはスロースタート中のTCPによって送信され，平均Tr1時間で送信される．

以上の仮定のもと，まず，従来のコンテンツスイッチの場合のクライアントの平均コンテンツ取得遅延を評価する．従来のコンテンツスイッチでは，クライアントはサーバのレスポンスを待ってから，次のリクエストを送信する．図6(a)に従来のコンテンツスイッチにおいて，クライアントが3つのコンテンツを取得する場合のタイミングチャートの例を示す．この例では，最初の2つのコンテンツは同一のサーバへと転送され，3つめのリクエストがそれとは異なるサーバへと転送されている．図6(a)に示すように，リクエストが1つ前のリクエストと同じサーバへと転送される場合，1つのリクエスト処理にかかる平均遅延は，2Ts+RTT+Tr1となる．そうでない場合，平均遅延は2Ts+RTT+Tr2である．よってN個のリクエストの処理の平均遅延D_tcps(N)は，以下の式で表される．

$$D\_tcps(N)=2Ts+RTT+Tr1+(N-1)\{p(2Ts+RTT+Tr1)+(1-p)(2Ts+RTT+Tr2)\}$$

次に，提案するTCP非対称スプライシングとサーバ切替方式を用いるコンテンツスイッチの場合のクライアントのコンテンツの平均取得遅延を評価する．この場合，クライアントはパイプラインリクエストを送信する事が可能である．従来のコンテンツスイッチの評価に使ったのと同じ例について，提案方式の場合のタイミングチャートを図6(b)に示す．提案方式の場合，k個のリクエストが同じサーバに転送される場合，これらのk個のリクエストの処理遅延は，2kTs+RTT+kTr2となる．もしリクエストが1つ前のリクエストとは異なるサーバへ転送された場合，その処理遅延は，従来のコンテンツスイッチの同じ場合と同等で，2Ts+RTT+Tr1である．よって，k個のリクエストの平均処理遅延であるD_atcps(k)は，以下のように表される．

$$D\_atcps(k)=D\_atcps(k-1)+p(2Ts+Tr1)+(1-p)(2Ts+RTT+Tr2)\ (k=2,3,..)$$

$$D\_atcps(1)=2Ts+RTT+Tr1$$

上記の漸化式を解く事により，N個のリクエストの処理の平均遅延

$$D\_atcps(N)=(2Ts+RTT+Tr1)+(N-1)\{p(2Ts+Tr1)+(1-p)(2Ts+RTT+Tr2)$$

を得る．

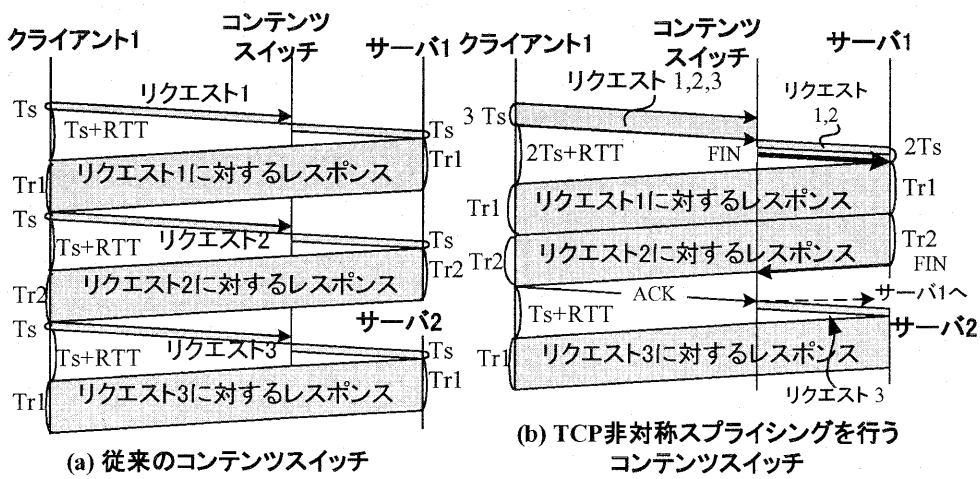


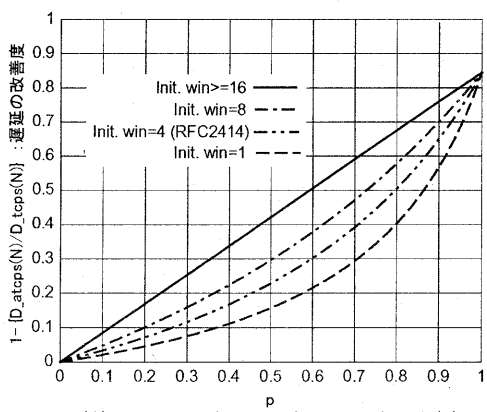
図6: 従来のコンテンツスイッチとTCP非対称スプライシングを行うコンテンツスイッチでのサーバ切替方式の違い

次に，コンテンツの平均取得遅延について，従来方式に対する提案方式の改善度，1-D_atcps(N)/D_tcps(N) の数値例を示す．数値例では，一般に Ts は，RTT, Tr1, Tr2 に比べて十分小さい事から Ts の項は無視して評価した．また，Tr2 については，十分開いた輻輳ウィンドウでデータが転送されることから，Tr2=B/Ls (Ls: リンク速度，B: 平均コンテンツサイズ) として評価している．また，Tr1 については，次のように評価した．TCP の輻輳ウィンドウが w からスタートする場合，スロースタート中は輻輳ウィンドウサイズは RTT 毎に 2 倍される．よって，MSS を TCP の最大セグメントサイズとすると，B に関して以下が成り立つ．

- $B \leqq w\,MSS$ ならば $Tr1 = B/Ls$
- $w\,MSS < B \leqq (w+2w)MSS$ ならば $Tr1 = RTT + B/Ls$
- $(w+2w)MSS < B \leqq (w+2w+4w)MSS$ ならば $Tr1 = 2RTT + B/Ls$
- ...
- $(w+2w+..+2^{n-1}w)MSS < B \leqq (w+2w+..+2^{n}w)MSS$ ならば $Tr1 = n\,RTT + B/Ls$

よって，M を B/MSS 以上の最初の整数とすると，n を $w\{2^{n-1}-1\} < M \leqq w\{2^{n}-1\}$ を満たす n として，$Tr1=(n-1)RTT+B/Ls$ と表せる．

これらの評価を基に，図 7 にコンテンツの平均取得遅延の改善度(1-D_atcps(N)/D_tcps(N))の数値例を，様々な p と w に関して示す．ここで，WAN 回線経由でクライアントがアクセスする場合を想定し，N=10, RTT=100ms, MSS=512 バイト，Ls=1Mbps, B=8K バイトとした．これらの数値を選んだ場合，Tr2=6.5536ms である．Tr1 は，w の値によって変わるが，w=1 の場合(現在の多くの TCP の実装がこの値を使っている)，Tr1=406.5536ms であり，w=4 の場合 (RFC2414[6]の推奨値)，Tr1=2-6.5536ms, w=8 の場合 Tr1=106.5536ms, w が 16 以上の場合，Tr1=6.5536ms である．図 7 において，横軸は p を示す．この図において p=1.0 の場合は各リクエストが必ず前のリクエストと同じサーバへ転送される (100%の正の相関がある) 事を意味しており，p=0.0 の場合は，必ず異なるサーバへと転送される事を意味する (100%の負の相関がある場合)．p=0.1 の場合は，例えば 10 台のサーバがランダムに選ばれる場合を示している．図 7 から分かるとおり，我々の提案方式は，常に従来の方式よりも短い処理遅延を実現しており，特に，連続するリクエストのサーバ選択に関して正の相関が高い (同じサーバが連続して選ばれる確率が高い) 場合に大きな効果を持つ事が分かる．実際のサーバへのコンテンツ配置では，パイプラインでまとめて取得されるであろうコンテンツを同一サーバに配置することで，p の値を 1 に近づける事が可能と考えられ，本方式は高い (最大 80%の改善) 性能を発揮出来ると期待できる．



(連続したリクエストが同じサーバへフォワードされる確率)

図 7: 提案方式によるコンテンツの平均取得遅延の改善度

5. 結論

本報告では，コンテンツスイッチがクライアントからのリクエストをアプリケーション層でサーバへとフォワードしつつ，サーバからのレスポンスを処理負荷の低いパケットフォワーディングによってクライアントへとフォワードする，TCP 非対称スプライシング方式を提案した．また，HTTP のトランザクションに関するサーバ切替方式も提案した．提案したサーバ切替方式は，サーバからのレスポンスの終了を TCP ハーフクローズを使うことによって検知し，コンテンツスイッチがパイプラインリクエストを扱う事を可能にするものである．性能評価では，提案方式によってパイプラインリクエストを扱うコンテンツスイッチが，従来のパイプラインリクエストを扱えないコンテンツスイッチと比較し，高い処理性能を持つことを示した．特に，パイプラインでリクエストされるコンテンツを同一サーバに配置する事によって，大きな効果(最大 80%の改善) が得られる事を示した．

参考文献

[1] G. Apostolopoulos, D. Aubespin, V. Peris, P. Pradhan and D. Saha, "Design, Implementation and Performance of a Content-Based Switch," *In Proc. IEEE Infocom 2000*, March 2000.

[2] A. Cohen, S. Rangarajan and H.Slye, "On the Performance of TCP Splicing for URL-aware Redirection," *In Proc. 2nd USENIX Symposium on Internet Technologies and Systems*, October 1999.

[3] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie and C. Lilley, "Network Performance Effects on HTTP/1.0, CSS1, and PNG," *In Proc. ACM SIGCOMM'97*, 1997.

[4] T. Berners-Lee, R. Fielding and H. Frystyk, "Hypertext Transfer Protocol - HTTP/1.0," *RFC1945*, May 1996.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frysyk and T. Berners-Lee, "Hypertext Transfer Protocol - HTTP/1.1," *RFC2068*, January 1997.

[6] A. Allman, S. Floyd and C. Partridge, "Increasing TCP's Initial Window," *RFC2414*, September 1998.

[7] Cisco Systems, Inc., "Content-Based Switch," available from http://www.cisco.com/

[8] Alteon Websystems, Inc., "Alteon ACE Director," available from http://www.alteonwebsystems.com/