

SSL/TLS プロトコルにおける 圧縮アルゴリズムの実装とその評価

岡本無字[†] 木村成伴[‡] 海老原義彦[‡]

[†]筑波大学第三学群情報学類 [‡]筑波大学電子・情報工学系

筑波大学コンピュータネットワーク研究室

〒 305-8573 茨城県つくば市天王台 1-1-1 工学系学系 E 棟 105

Tel & FAX: 0298-53-5158

{okamoto, kimura, ebihara}@netlab.is.tsukuba.ac.jp

あらまし

代表的な暗号化通信の一つである SSL/TLS では、上位層から受信したデータを暗号化する前処理としてデータ圧縮を行う規定がある。しかし、SSL/TLS の規格書には無圧縮以外の圧縮アルゴリズムが規定されていないため、SSL/TLS では通信当事者同士の事前合意なしにデータ圧縮を行うことが出来なかった。

本研究の最終目標は SSL/TLS に特化した圧縮アルゴリズム開発することであるが、本論文ではその予備調査のため、HTTP レベル又は SSL/TLS レコードレイヤレベルで SSL/TLS に汎用圧縮アルゴリズムを導入する。両方式を実装したシステムでファイルの圧縮率と平均データ転送時間を測定し、その結果を比較する。

キーワード 圧縮アルゴリズム, レコードレイヤ, SSL, TLS, HTTP, 暗号化通信

Implementations of Compression Algorithms for SSL/TLS Protocols and Their Evaluations

Okamoto Nau[†] Kimura Shigetomo[‡] Ebihara Yoshihiko[‡]

[†]College of Information Sciences, Third Cluster of Colleges, University of Tsukuba

[‡]Institute of Information Sciences and Electronics, University of Tsukuba

Address: Computer Network Laboratory, University of Tsukuba 1-1-1 Tennoudai,

Tsukuba, Ibaraki 305-8573, Japan. Room 105, Bldg. E of 3rd Cluster

Tel & Fax: +81-298-53-5158

{okamoto, kimura, ebihara}@netlab.is.tsukuba.ac.jp

Abstract

In the SSL/TLS protocols, their specifications define that data from the upper layer may be compressed before the data are encrypted. Since no compression algorithm except non-compression is defined in the specifications, nobody can compress transmission data at the SSL/TLS protocols unless communication entities have a special agreement to use a compression algorithm.

This paper discusses two implementation systems which introduce compression algorithms into an SSL/TLS program at HTTP level or SSL/TLS record layer. For the both systems, we observe their compression ratio and average transfer time for several text and compressed graphic files.

key words compression algorithm, record layer, SSL, TLS, HTTP, encryption communication

1 背景

近年の急速なインターネットの普及に伴い、利用者が持つ情報の漏洩を防止することが重要な課題となっている。これを実現するために通信データの暗号化は不可欠な技術であり、現在、PGPやSSH、SSL[1] / TLS[2] 等の暗号化通信が利用されている。この内、SSL (Secure Socket Layer)/TLS (Transport Layer Security) はソケットを使う全てのアプリケーションに適用可能な暗号化通信方式である。特に HTTP では、ほとんどの WWW ブラウザに SSL/TLS の機能が組み込まれているため、SSL/TLS は一般的なインターネットユーザに広く用いられている。

SSL/TLS の規格では上位層のデータを暗号化する際、前処理として圧縮処理を行うと規定している。この処理により暗号化するデータをあらかじめ圧縮しサイズを減らし、負荷の大きい暗号化処理におけるオーバーヘッドを減らすという利点が得られることが期待される。しかしながら、この規格では無圧縮方式以外の圧縮アルゴリズムが規定されていない。このため、SSL/TLS でサーバ/クライアント双方が事前の合意なしに利用可能な圧縮アルゴリズム存在しない。

しかし、コンピュータの処理能力は急速に向上している一方で、電話回線や ISDN 等の比較的低速な回線を利用しているインターネットユーザが多い現状では、このような圧縮処理は有効である。例えば、このような低速回線の下において、HTTP に gzip[3] 圧縮処理を付加することによりファイルの転送時間が改善するという報告がなされている [4]。特に、SSL/TLS を利用して暗号化されるデータは限定された形式を持つことが多く、これに特化した圧縮アルゴリズムを使うことで、更なる効率化が期待される。本研究の最終的な目標はこのような圧縮アルゴリズムを開発することにあるが、本論文ではその予備的な調査のために、gzip 等の汎用圧縮処理を SSL/TLS に実装した場合にどの程度の転送効率改善が見られるかについて実験を行い、実装後の性能向上について議論する。

本論文の構成は以下の通りである。まず、第2章で SSL/TLS 通信について概観する。3章では、SSL/TLS にデータ圧縮方式を導入する方式について検討する。4章では、3章で述べた方式を実装したシステムを構築し、その圧縮効率や、データの伝送効率について評価する。最後に5章で本論文の結論を述べる。

2 SSL/TLS 通信の概要

本章では、本論文の SSL/TLS 通信について、特にその圧縮方式を中心に説明する。

2.1 SSL/TLS 通信ハンドシェイクプロトコル

SSL/TLS 通信では暗号化されたデータ通信を行うために、最初に暗号方式、圧縮方式、暗号化に用いる共通鍵を作成するための情報等についてサーバ・クライアント間で合意しておく必要がある。この処理を担うのがハンドシェイクプロトコルである。暗号化通信路を開設するまでの典型的な処理手順を図1に示す。

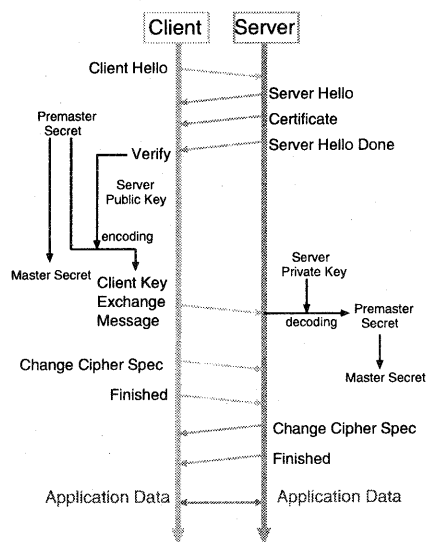


図1: SSL/TLS 通信での典型的なハンドシェイク処理

1. クライアントはサーバに HELLO メッセージ (Client Hello) を送る。この HELLO メッセージにはクライアントが対応できる暗号化方式と圧縮方式のリスト等が含まれる。
2. サーバはクライアントに HELLO メッセージ (Server Hello) を送る。クライアントの HELLO メッセージ中のリストからデータ通信に用いる暗号化方式および圧縮方式等を選択し、これをこのメッセージに入れる。

3. これに続けて、サーバは自分の公開鍵を含む証明書をクライアントに送り (Certificate), HELLO メッセージの送信終了 (Server Hello Done) をクライアントに告げる。
4. 証明書の確認 (Verify) 後、クライアントは通信データの暗号化に使用する共通鍵等 (Master Secret) の作成に必要な情報 (Premaster Secret) をサーバの公開鍵で暗号化してサーバに通知する (Client Key Exchange)。
5. サーバとクライアントは Premaster Secret から Master Secret を作成する。
6. 合意に至った暗号化方式および圧縮方式を用いて以降のデータを圧縮、暗号化することをクライアントがサーバに告知する (Change Cipher Spec). さらに、暗号化通信路を開設するためのハンドシェイクが完了 (Finished) したことをクライアントがサーバ側に告知する。同様のことをサーバもクライアントに対して行う。ここで、サーバおよびクライアントの Finished は圧縮および暗号化が既になされた状態で相手側に送信される。
7. 以上の処理により暗号化通信路が開設されたので、これを用いて SSL/TLS 上位層のデータを送受信する (Application Data)。

2.2 SSL/TLS レコードレイヤ

SSL/TLS は二つの副層から成っており、上位層のハンドシェイクレイヤでは前節で示したように暗号化通信路の開設やエラー処理、合意された暗号化通信の開始告知を行う。下位層のレコードレイヤでは SSL/TLS の上位層、たとえば HTTP などのアプリケーション層から受け渡されたデータに対し細分化 (Fragmentation)、圧縮 (Compression)、暗号化 (Encryption) の順に処理を行い、処理済のメッセージを下位層の TCP 等に渡す。

図 2 はアプリケーション層から受け渡されたデータに対してレコードレイヤが行う処理の一例を示したものである。レコードレイヤでは、ハンドシェイクレイヤまたはアプリケーション層からデータ (Application Data) を受信すると、これを細分化 (Fragmentation) する。規格では、一つのフラグメントが 16k バイト以下となるように細分化することが推奨されている。細分化されたデータは 5 バイトのヘッダが付加され、図 3 の一番上に示す Plaintext 構造体となる。このヘッダには、

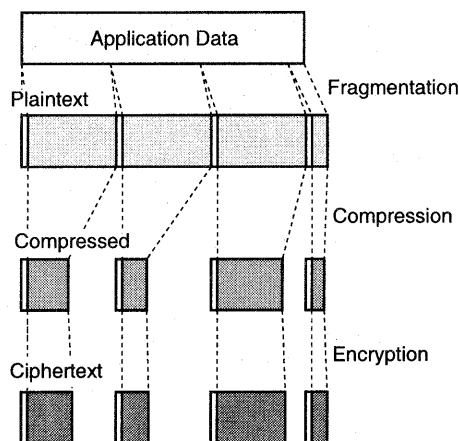


図 2: メッセージの細分化および圧縮および暗号化

SSL/TLS のバージョン番号、データの種別 (制御用またはアプリケーションデータの別)、構造体に格納されるデータ (fragment) のサイズを含む。そして、Plaintext 構造体の fragment 部分に細分化されたデータが格納される。さらに、レコードレイヤは細分化されたデータに圧縮処理を行い、Compressed 構造体を生成する。図 3 に示すように、Compressed 構造体も Plaintext 構造体と同様なヘッダを持つ。また、圧縮後のデータサイズは 17k バイト以下になることが期待される。この大きさは、Plaintext 構造体よりも大きい。これは既に圧縮されたデータや図 2 の 4 分割された最後のデータのように極端にサイズが小さいデータは冗長性に乏しいため、圧縮処理を行ってもほとんどサイズが変わらないか、かえってサイズが増加してしまう場合があることを考慮したものである。

最後に暗号化処理を行い、Ciphertext 構造体を生成する。この構造体にも図 3 下に示した 5 バイトのヘッダを付加し、データサイズは 18k バイト以下になることが期待されている。

3 SSL/TLS におけるデータ圧縮の導入

ISP と、ユーザとをつなぐいわゆるラスト・ワンマイルの部分は公衆電話回線などの比較的低速な回線が利用されることが多い。このような回線を利用するユー

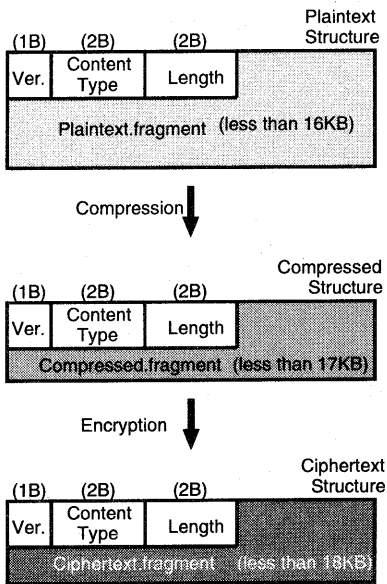


図 3: レコードレイヤで扱われる SSL/TLS 構造体

ザの通信効率を改善するため、データを圧縮して送信する方法がある。本章では、SSL/TLS で送信するデータを圧縮する方法について検討する。

3.1 HTTP 圧縮を用いた実現方法

HTTP1.1[5] では、データを圧縮して伝送することが考慮されており、その仕様書には gzip や compress, deflate といった圧縮アルゴリズム名が IANA によって規定されている旨が示されている。これにより、サーバがファイルを圧縮して送信し、クライアントが圧縮ファイルを受信して展開することが可能になっている。

この機能を用いた HTTP セッションの概要を以下に示す。

1. クライアントはサーバに対し GET や POST 等のリクエストを行う。リクエストヘッダ中に Accept-Encoding ヘッダフィールドを含めることにより、圧縮方式のうち使用可能なものをサーバに通知する。
2. サーバはクライアントのリクエストに対しレスポンスを返す。その際、Accept-Encoding に提示された方式が使用可能であれば、その方式を適用し

たことを示す Content-Encoding ヘッダフィールドをレスポンスヘッダに含め、その方式でデータを圧縮する。

3. クライアントは受けとったエンティティヘッダの Content-Encoding に示されている方式で圧縮されたデータを展開する。

この機能は HTTP サーバでは Apache や IIS に、HTTP クライアントでは Internet Explorer や Netscape Navigator に実装されている。また、圧縮方式に gzip を用いて圧縮画像を含む CNN のページへアクセスした場合、ISDN の 128kbps 回線では 5.5%, 28.8kbps 回線では 30% の伝送時間の低下が見られたとの実験結果が報告されている [4]。

SSL/TLS を HTTP 上で利用する場合、この機能を利用すれば疑似的に SSL/TLS で伝送するデータを圧縮することができる。すなわち、図 2 において、HTTP サーバから渡されるデータ (Application Data) が圧縮されていれば、Plaintext 構造体を改めて圧縮する必要がなくなり、SSL/TLS の実装を変更することなしに、SSL/TLS のデータ圧縮が行われることになる。本方式は HTTP 上でしか利用できないという欠点があるが、細分化前に圧縮するため、細分化後に圧縮した場合よりも圧縮率が高くなることが予想される。

3.2 SSL/TLS における圧縮アルゴリズムの実装

SSL/TLS において、HTTP といったアプリケーションに依存せずにデータ圧縮を行う必要がある場合は、SSL/TLS に圧縮アルゴリズムを導入する必要がある。以下では、無圧縮以外の圧縮アルゴリズムを TLS/SSL サーバおよびクライアントに実装する方法について検討する。

まず、圧縮アルゴリズムとして gzip および bzip2 を採用した。gzip[3] は UNIX で広く用いられている圧縮技術で、LZ77 符号化法および適応型ハフマン符号を組み合わせたアルゴリズムを用いている。また bzip2[6] はブロックソーティングおよび LZ 系圧縮およびハフマン圧縮によるアルゴリズムを用いており、一般に gzip と比較して圧縮率が高いが、圧縮/展開に要する負荷が gzip より大きい。この gzip と bzip2 はともに C 言語のライブラリがフリーで提供されており、これらを用いることで圧縮機能をアプリケーションに実装することが容易に可能となる。

本論文における SSL/TLS への圧縮機能の実装もこのライブラリを用い、これを OpenSSL[7] に組み込む。すなわち、SSL/TLS のレコードレイヤで図 2 の Plaintext 構造体のデータを合意されたアルゴリズムで圧縮し、Compressed 構造体に格納する。3.1 節の方式と異なり、細分化を行ってから圧縮するため、細分化のサイズが小さいと圧縮率が悪くなる可能性がある。

ところで、SSL/TLS の仕様書には無圧縮 (ID=0) 以外の圧縮方式の ID が規定されていない。そこで本論文では、gzip の ID を 16 に、bzip2 の ID を 17 に割り当てる。

次に、圧縮処理を実装した SSL/TLS クライアントおよび SSL/TLS サーバ間での圧縮に関するハンドシェイクの概要を以下に示す。

1. クライアントは、自分がサポートする圧縮方式 (gzip, bzip2, 無圧縮) の ID を自分の好ましい順に並べ、そのリストを HELLO メッセージで SSL/TLS サーバに提示する。
2. サーバは提示された圧縮方式から合意できる方式を選択し、これをサーバ HELLO メッセージでクライアントに告知する。
3. 合意に至った圧縮方式は、Change Cipher Spec メッセージ以降に送信されるメッセージに適用される。

次章では、本章で検討した SSL/TLS に対する 2 種類のデータ圧縮方式について評価する。

4 SSL/TLS におけるデータ圧縮方式の評価

本章では、3 章で述べた 2 つの圧縮方式を比較するため、これらのシステムを実装したシステムを用意し、HTTP によるファイル転送を行う。以下では、3.1 節に述べた方式を HTTP 圧縮方式、3.2 節で述べた方式を SSL/TLS 圧縮方式とそれぞれ呼ぶことにする。

ここで、HTTP 圧縮方式の HTTPS サーバは、代表的な HTTP サーバである Apache 1.3.20[8] に OpenSSL 0.9.6a を用いて SSL/TLS 機能を実装する mod_ssl 2.8.4[9] モジュールと、HTTP 圧縮を行う mod_gzip 1.3.19.1a[10] モジュールを併用することにより実現した。また、SSL/TLS 圧縮方式の HTTP サーバは、レコードレイヤにおいて gzip と bzip2 で圧縮処理を行えるように修正した OpenSSL ライブラリを作成

し、これに加えて Apache 1.3.20 と mod_gzip 1.3.19.1a モジュールを用いて実現した。

両方式の HTTPS クライアントは、OpenSSL 0.9.6a を用いて作成した SSL/TLS でのファイル転送プログラムを、両方式の圧縮機能に対応できるよう変更したものを用意した。

これらの HTTPS サーバおよびクライアントは SSL と TLS のいずれにも対応しているが、紙面の都合により、以下では SSL Ver.3.0 を用いたときの実験結果のみを示す。なお、本実験で用いた鍵交換方式は DHE-RSA、暗号化方式は DES-CBC3 である。

4.1 ファイル圧縮率の比較

実験では、日本語 (シフト JIS) を含む HTML 文書を 5 種類 (4381~116075 バイト) と画像ファイル (GIF 形式) を 4 種類用いた。これらのファイルは、インターネット上で用いられているものを採取している。ここで、HTML 文書は未圧縮であり、高い圧縮率が期待できる一方、GIF 画像は圧縮処理済のファイルであり、再圧縮による大幅なファイルサイズ減少は期待できない。

さて、転送実験に先立ち、これらのファイルに HTTP 圧縮方式および SSL/TLS 圧縮方式を適用するどのように細分化および圧縮されるかを観測した。この内、各種類毎の最小と最大のファイルサイズで測定した結果を表 1 から表 4 に示す。

method	Packets(B)	Rate(%)
mod(N)	277,4381	100.0
mod(Y)	2128	42.3
ssl(0)	277,4381	100.0
ssl(16)	231,1816	41.5
ssl(17)	270,2019	46.1

表 1: 4381 バイトの HTML ファイル

Method	Packets(B)	Rate(%)
mod(N)	280,8192x14,1387	100.0
mod(Y)	303,4000,8000x2,968	18.1
ssl(0)	280,16384x7,1387	100.0
ssl(16)	234,3967,3509,3132,...,498	21.6
ssl(17)	277,3868,3354,3072,...,622	21.0

表 2: 116075 バイトの HTML ファイル

Method	Packets(B)	Rate(%)
mod(N)	1434	100.0
mod(Y)	1434	100.0
ssl(0)	1434	100.0
ssl(16)	1446	100.0
ssl(17)	1773	124.0

表 3: 1434 バイトの GIF ファイル

Method	Packets(B)	Rate(%)
mod(N)	278,8192x2,3254	100.0
mod(Y)	302,4096x4,3165	99.6
ssl(0)	278,16384,3254	100.0
ssl(16)	232,16303,3266	99.7
ssl(17)	269,16819,3714	105

表 4: 19638 バイトの GIF ファイル

これらの表において、mod(N) と mod(Y) はそれぞれ HTTP 圧縮方式において圧縮処理をした場合としない場合を表している。値はバイト単位であり、それぞれが1パケットを表している。ssl(*n*) は SSL/TLS 圧縮方式において、ID が *n* であるような圧縮処理をした場合を表している。3.2 節で述べたように、*n*=0 は無圧縮、*n*=16 は gzip、*n*=17 は bzip2 による圧縮処理を行うことを示す。次に、Packets に示す値は、ハンドシェークレイヤから実際に送出されたパケットのサイズを個々に示したものである。

例えば、表 1 において mod(N) では 227 バイトと 4381 バイトのパケットが送られている。ここで 227 バイトのパケットは HTTP のヘッダである。mod(Y) では、2つのパケットがまとめて圧縮され、2128 バイトのパケット1つが送られている。このときの、全パケットに対する総圧縮率は Rate の項に示された通り、42.3%である。これに対して、ssl(16) などでは HTTP ヘッダを含むパケット毎に細分化と圧縮が行われていることがわかる。表 2 についても同様の傾向が見られるが、細分化後に圧縮される SSL/TLS 圧縮方式の圧縮率が HTTP 圧縮方式よりも 3~4% 低下している。また、表 2 において HTTP 圧縮では 4k バイトまたは 8k バイトに細分化されているのに対し、SSL/TLS 圧縮方式では 16k バイトで細分化されている。この挙動の違いの原因は調査中であるが、OpenSSL 0.9.6a のデフォルトの細分化サイズは 16k バイトであることから、mod_gzip 1.3.19.1a

による影響と思われる。

次に、表 3 と表 4 から、GIF ファイルは圧縮が困難であることがわかる。また、表 4 の細分化のサイズに関しては、表 2 と同じ傾向があることがわかった。

4.2 ファイル転送時間の比較

次に、図 4 に示すネットワーク上でファイル転送実験を行う。伝送回線の通信速度として、低速モデムを想定した 28.8kbps、ISDN 回線を想定した 64kbps、LAN を想定した 10Mbps を用いた。これらの通信速度は KAME プロジェクトによるプロトコルスタックである ALTQ[11] の CBQ を利用し、10Mbps の回線に対して帯域制限を行うことで実現している。ただし、28.8kbps はモデムを想定しているため、同期のためのスタートビットとストップビットが 8 ビットあたり計 2 ビット入るものと仮定し、回線速度を 20% 低下させた 23kbps にしている。

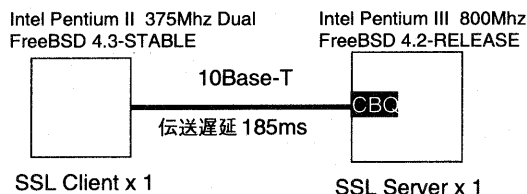


図 4: 実験に使用するネットワーク構成

このネットワーク環境において、4.2 節で示した HTML 文書と GIF 画像ファイルを 100 回転送させたときの平均転送時間を図 5 から図 10 に示す。ここで、ファイルの転送時間とは、SSL セッションの開始からデータ受信完了までにかかった時間のことを示す。

通信速度が 28.8kbps のとき (図 5)、圧縮しなかった場合と比べ、HTTP 圧縮方式は最大 78.2%、SSL/TLS 圧縮方式は最大 56.5% の平均転送時間が減少した。特に、ファイルサイズが 116075 バイトの場合、HTTP 圧縮方式と SSL/TLS 圧縮方式の差は 18.3~19.3 秒となった。SSL/TLS 圧縮方式では細分化後に圧縮することから、転送されるパケット数は圧縮前後で同じになる。従って、表 2 から、この方式は合計 9 パケットの転送が行われている。これに対して、HTTP 圧縮方式では、同表より 5 パケットの転送が行われていることがわかる。すなわち、SSL/TLS 圧縮方式には 4 パケッ

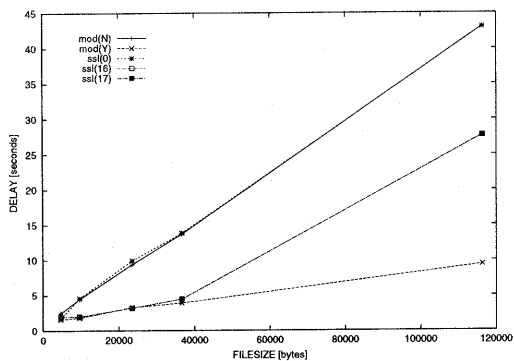


図 5: HTML ファイル, 28.8kbps

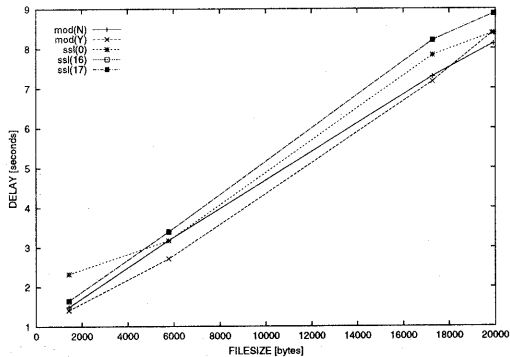


図 8: GIF ファイル, 28.8kbps

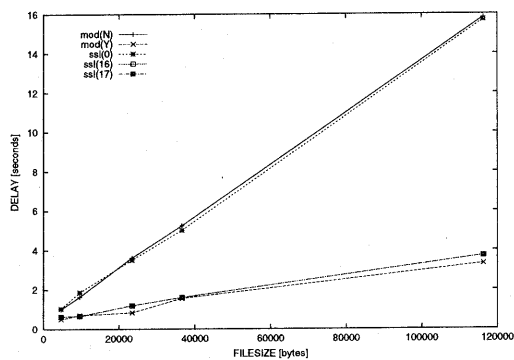


図 6: HTML ファイル, 64kbps

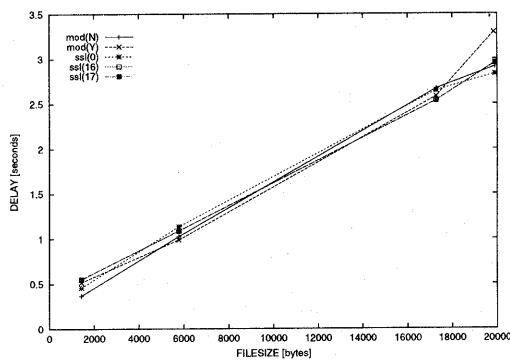


図 9: GIF ファイル, 64kbps

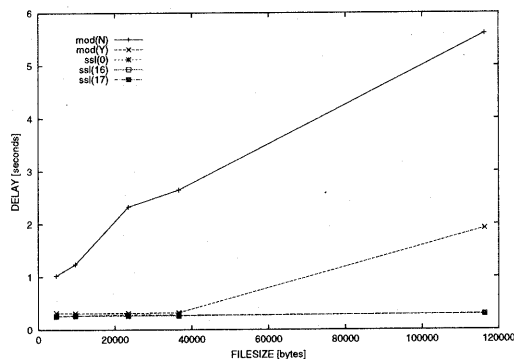


図 7: HTML ファイル, 10Mbps

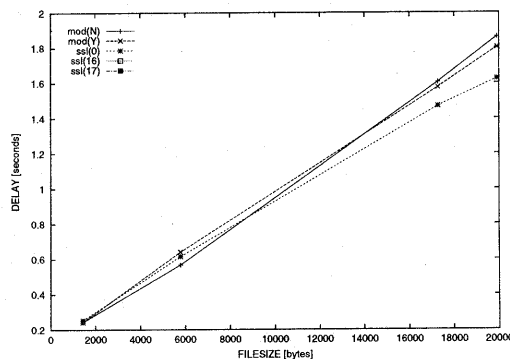


図 10: GIF ファイル, 10Mbps

ト分のヘッダ情報のオーバーヘッドがあり、これにより、転送速度に差が出たものと思われる。

通信速度が64kbpsのとき(図6)は、HTTP圧縮方式とSSL/TLS圧縮方式の差はほとんど見られず、両方式とも、圧縮しなかった場合と比べ、69.3~78.8%平均転送時間が減少した。通信速度が10Mbpsのとき(図7)は、HTTP圧縮方式とSSL/TLS圧縮方式による平均転送時間の差はほとんどなかった。しかし、圧縮しなかった場合に差が見られ、SSL/TLS圧縮方式はほとんど平均転送時間が変わらないのに比べ、HTTP圧縮方式は195%~648%平均転送時間が増加している。通信速度が高くなるに従い、両圧縮方式の差異は十分無視できることがわかった。

これに対して、GIF画像ファイルは、圧縮によりファイルサイズが減少しない。このため、圧縮に要するオーバーヘッドが生じることから、圧縮しなかった場合と比べ、通信速度に依らず、HTTP圧縮方式は0.0%~21.4%、SSL/TLS圧縮方式は0.0%~4.8%転送時間が増加した。

5 結論と今後の課題

SSL/TLSにデータ圧縮を導入するため、HTTP圧縮方式とSSL/TLS圧縮方式を考察し、その実装方法について検討した。また、これらを実装したシステムを用いて、HTTP文書とGIF画像ファイルの圧縮率と平均ファイル転送時間を測定し、それぞれの結果を比較した。これより、HTTP圧縮方式とSSL/TLS圧縮方式による大きな差異は見られなかった。また、両方式共に、HTML文書に関しては、データ圧縮による平均ファイル転送時間が減少し、特に、通信速度が低いほど減少する割合が高かった。一方、GIF画像ファイルに関しては、平均ファイル転送時間が通信速度に依らず増加した。従って、SSL/TLSにおいても、文献[4]と同様な結果が得られることがわかった。

さて、SSL/TLSによる暗号化通信では、画像ファイルなど圧縮されたファイルよりも、フォームなどのテキストデータを転送することが多いと予想される。また、ここで送付されるデータの形式は限定しやすいことから、これに特化した圧縮アルゴリズムを提案し、更なる転送時間の低減を目指すことを今後の課題としたい。

参考文献

[1] Alan O. Freier, Philip Karlton and Paul C.

Kocher, "The SSL Protocol Version 3.0," draft-freier-ssl-version3-02.txt, 1996.

- [2] Tim Dierks, Philip L. Karlton, Alan O. Freier and Paul C. Kocher, "The TLS Protocol Version 1.0," Request for Comments: 2246, 1999.
- [3] Peter Deutsch, "GZIP file format specification version 4.3," Request for Comments: 1952, 1996.
- [4] John Giannandrea and Eric Bina, "performance: HTTP Compression," <http://www.mozilla.org/projects/apache/gzip/>, 1998.
- [5] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach and Tim Berners-Lee, "Hypertext Transfer Protocol - HTTP/1.1," Request for Comments: 2616, 1999.
- [6] Julian Seward, "The bzip2 and libbzip2 official homepage," <http://sourceware.cygnus.com/bzip2/>, 2000
- [7] The OpenSSL Project, "OpenSSL: The Open Source toolkit for SSL/TLS," <http://www.openssl.org/>, 2001
- [8] The Apache Software Foundation, "The Apache Software Foundation," <http://www.apache.org/>, 2001
- [9] Ralf S. Engelschall, "mod_ssl: The Apache Interface to OpenSSL," <http://www.modssl.org/>, 2001
- [10] Kevin Kiley, "HSC's mod_gzip for the Apache Web Server," http://www.remotecomunications.com/apache/mod_gzip/, 2001
- [11] Kenjiro Cho, "ALTQ: Alternate Queueing for BSD UNIX," <http://www.csl.sony.co.jp/kjc/software.html#ALTQ>, 2000