月刊「インターフェース」95 年 9 月号
特集企画「マルチメディア時代のネットワーク技術」
第 3 章

多目的 proxy サーバ DeleGate

佐藤豊

DeleGate

はじめに

昨年 3 月に開発が開められ、国内外の 680 以上の組織に配布され使われるようになり、ソースコード 25000 行を越える中規模のシステムに成長した現在でも、 DeleGate は全く未完成のソフトウェアであり、日々修正や拡張を受けながらむくむくと成長を続けています。

DeleGate とは何であるか。開発者自身、このような問に対する明瞭な答えを持ちません。 WWW の proxy サーバとしての機能を持ち、そのような用途で利用されていることは確かですが、それがこのシステムのゴールではありません。何でありたいか？　それも固定して考えたくはないのですが、強いて言うなら、クライアントとサーバの仲立ちをする proxy サーバという存在を積極的にとらえたとき、そこでどのような付加価値機能が実現できるかを実験しつつ探り出して行くための道具だと言えます。また、あるプロトコルを理解する上で、じめじめと仕様を眺めているよりも、そのプロトコルを中継するプログラムを実際に作成して動かしながら理解するのは効率的で楽しい方法ですので、開発者自身の勉強の道具としても役立っています。

実用のシステムとして利用される方の立場からは、開発者の意図はともかく、それが具体的に何物なのかがはっきりしなくては使うことができません。仕様も実現も手探りしながら「とりあえず」で走り続けて来たシステムを「こうだ」と断定的に書き記すのには少なからず痛みを感じますが、世間に送り出してしまったからにはそのように説明する責任も感じます。以下では、現在までの DeleGate で実現されている機能について概説し、特に DeleGate に特徴的な機能について実現の概要を説明することにします。

3.1　DeleGate とは

DeleGate は，インターネット・プロトコル (IP) 上の各種アプリケーション・プロトコルをアプリケーション層で中継するシステムであり、いわゆる proxy サーバです。 WWW の文脈で proxy サーバと言えば、 CERN の Proxy-httpd (以下、Proxy と略す) と互換なもので、様々なプロトコルのサーバと通信した結果を、 HTTP プロトコルのクライアントに対して提供するものを指します。 DeleGate は Proxy 互換機能を持っていますが、それだけでなく、 HTTP プロトコル以外のクライアントに対してそのプロトコルのままにサーバとの間を仲介する機能も備えた、汎用の proxy サーバです。 DeleGate が中継可能なプロトコルは表 1 に示すように多岐にわたり、またその用途も firewall におけるアクセス制御だけでなく、以下のように多様な付加価値機能を提供する多目的の中継サーバとなっています。

- 経路制御

アクセス元 / アクセス先のホストのドメイン名や IP アドレスによって (そのパターンマッチングを行い) 、さらに別の proxy への中継を行ったり、障害のある経路を迂回する機能。

- アクセス制御

アクセス元 / アクセス先のホストおよびユーザ名によって，中継の許可 / 不許可を制御する機能。

- プロトコル変換

FTP, WAIS, Gopher などのプロトコルで提供されている情報資源 (resource) を， HTTP プロトコルに変換してクライ

アントに提供する機能。

- 文字コード変換

中継データに含まれるテキストの文字コードを，中継先のサーバやクライアントが受理可能なものに変換して中継する機能。

- MIME コード変換

NNTP,SMTP,POP を中継する際に、 MIME 形式に基づいて，メイル / ニュースのヘッダ部 / ボディ部の非 ASCII テキスト / バイナリーデータのエンコード / デコードを行う機能。

- データキャッシュ

サーバからクライアントへ一度中継したデータを保持し，その後の同一のデータへのアクセスの際に再利用する機能。

- コネクションキャッシュ

DeleGate から一旦接続した FTP サーバへの接続を一定時間維持し，複数の HTTP クライアント間でその接続を共有・再利用する機能。

- 既存ソフトのための中継

既存の FTP や Telnet のクライアントソフトウエアに対して変更を加えることなく中継を行う機能。

- マウント機能

サーバ / クライアント間の双方向の通信に含まれる情報資源の識別名 (URL) を書き換えて中継し，実際の資源の所在位置を秘匿してクライアントに提供する機能。

- トンネル機能

IP 的に接続されていないホスト間、あるいは、利用したいプロトコルが通過できないホスト間を DeleGate で結び、そのプロトコルを通過させる機能。

- 外部フィルタ機能

DeleGate とクライアントとの間に、外部のコマンドをフィルターとして挟んで、クライアントからの要求、クライアントへの応答を加工する機能。

これらは、現在の DeleGate の版 (DeleGate2.7) で実現されているものですが、皆さんがこれを読まれる頃には、さらに別の機能や、他のプロトコルの中継がレパートリーに加わっているかもしれません。全ての機能を、適用可能な全てのプロトコルにおいてサポートしたいと考えています。また、これまで TCP/IP 上のプロトコルの中継を中心に開発を行ってきましたが、最近の版で UDP/IP 上の CU-SeeMe の中継機能が加わり、今後 archie や RPC などの UDP/IP ベースのプロトコルへも対象を広げて行く予定です。

3.2 proxy サーバの基本的な枠組

以下では proxy サーバを実現するための基本的な枠組を説明しますが、ここではソケット (socket) を扱うためのシステムコール群についての知識を前提とします。もしご存知ない場合にも、 Unix のオンライマニュアルをはじめ、ソケットに関するドキュメントや手引き書は容易に入手できると思います。

さて、 TCP/IP 上のプロトコルを固定されたホストのサーバに単純に中継するだけならば、そのプログラムはごく簡単に実現できます。例えば、ホスト H の上のポート P (H:P) にある目的のサーバに対して、ホスト I のポート Q (I:Q) を介して中継する proxy は以下のように実現できます (図 2)。

(1) socket() でクライアントからの接続を受理するソケット e を作る

(2) bind() でそのソケット e を I:Q に束縛する

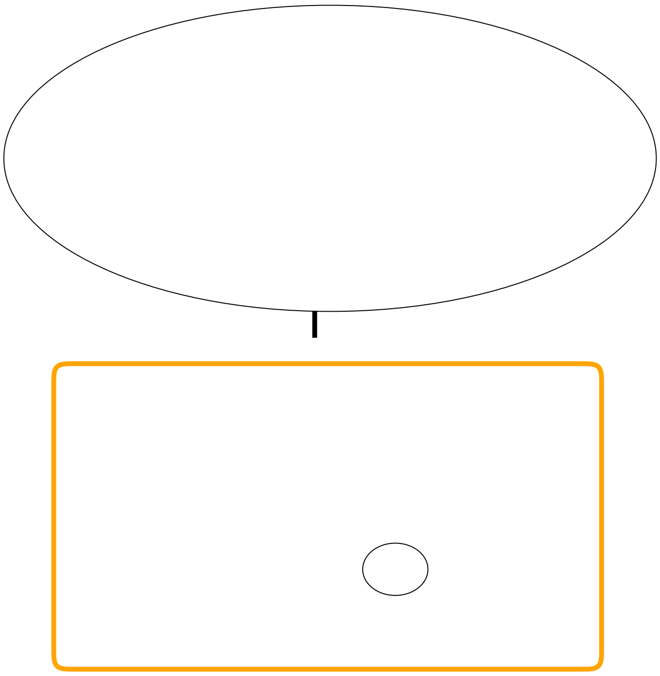


図 1: DeleGate の主要機能

表 1: DeleGate が中継するプロトコルとデータ形式

名称	機能概要	規格名
HTTP	WWW の通信プロトコル	Internet-Draft
Gopher	広域分散情報探索	RFC1436
WAIS	広域分散情報検索	RFC1625
Whois	人名ディレクトリサービス	RFC954
FTP	遠隔ファイル転送	RFC959
Telnet	遠隔ログイン	RFC854
X	遠隔ウィンドウシステム	RFC1013
NNTP	ニュース配送	RFC977
SMTP	メール配送	RFC821
POP	遠隔メールボックス管理	RFC1460
CU-SeeMe	簡易遠隔会議システム	
MIME	多言語多目的メール形式	RFC1521
HTML	ハイパーテキスト記述言語	Internet-Draft
URL	汎用広域情報識別名形式	RFC1738

この他に，Ident(利用者認証プロトコル，RFC1413) 使用

---- コラムＡ： DeleGate の名前の由来 ---

DeleGate という名前は英語の delegate から来ています。名詞としては「代理、代表者、使節」などの意味を持ちますが、 proxy よりちょっと偉そうな感じがします。また、ネットワークの gateway を表す gate という単語を部分列として含んでいるのもマッチすると思い、このような命名になりました。固有名詞であることを明確にし、 archie などで検索する時などのためにも、 Gate を大文字にしています。（開発最初の一週間は、 Gopher 中継の機能だけを持つ小さなプログラムで、 go-far と呼んでいました）なお、 DeleGate は「デリゲート」または「デリゲイト」と読みます。英語が苦手な筆者が言うのも何ですが、決して「でれげえと」ではありません。

図 2: proxy サーバの枠組

(3) listen() で e の上のバックログの数を指定する（必須ではない）

(4) accept() でクライアントからの接続を受理しソケット c を得る

(5) socket() でサーバに接続するためのソケット s を作る

(6) connect() で、ソケット s を H:P の上の目的のサーバに接続する

(7) s と c の間を双方向に中継する

(8) s または c の接続が切れたら (4) に戻る（あるいは終了する）。接断は、受信時の EOF や送信時の SIGPIPE により知る。

このような基本的な枠組の上で、 DeleGate では以下のような様々なバリエーションや機能を実現して来ました。

A. 複数の中継を並列に処理するために (3) の accept() の後に fork() し、子プロセスは (5) 以降へ、親プロセスは (3) に戻る。 DeleGate2.4 まではこの方法のみで実現していました。

B. (3) の段階で複数のプロセスに fork() しておき、各プロセスで (4)-(8) を並行して行うことで、並列中継を行う。これは、任意数の並列中継を実現するときに実現がやや複雑になりますが、 A. に比べて接続ごとの fork() を無くせる利点があり、 DeleGate2.4 以降の HTTP 用 DeleGate はこれを基本にしています。

C. inetd から nowait モードで起動された場合には、 (4) で accept() したソケット c が標準入出力に割り当てられた状態で DeleGate が起動される。 DeleGate2.7 からは、このような起動方法に対処しています。

D. 目的サーバへの connect() を、 SOCKS を利用して (SOCKS ライブラリの提供する Rconnect() 関数により) 行う。

E. (5)-(6) で一旦サーバに接続したら、そのソケット s を維持して、以降の繰り返しにおいて再利用する。この方法は、 DeleGate2.0 以降に導入された、 HTTP クライアントと FTP サーバの間の中継の実現に使われています（3.8に紹介する「コネクションキャッシュ」がこれです）。

F. (7) の双方向の中継を 2 つのプロセスで実現する、あるいは、 1 つのプロセスで select() や poll() を用いて実現する。当初の DeleGate は 2 つのプロセスでの中継を基本として実現していましたが、中継の終了時の処理などに問題があるため、最近は 3.10で紹介するような単一プロセスによる中継に実装を切り替えています。

G. (4) の段階で、ソケット c から getpeername() や Ident サーバを使ってクライアントのホストやユーザを調べ、アクセス制御を行う。また、 3.3で紹介するように、既存の FTP サーバを利用者認証のためのサーバとして利用しています。

H. c から s へ中継するリクエストを見て、そこに含まれる実際の目的サーバの IP アドレスやポート番号に基づき、そのサーバへ接続する。この場合、 (7) の中継を一部行った後で、 (5)-(6) が行われます。具体的な例として 3.4で HTTP の Proxy の実現法を説明します。

I. H. のように c から s へ中継するリクエストの中で目的サーバのプロトコルが p であると指定されていて、クライアント側のプロトコル q と異なる場合、目的サーバとは s を通じてプロトコル p により通信し、その結果をクライアントには c を通じてプロトコル q で返す。 FTP,Gopher,WAIS と HTTP の間の変換中継は、このようにして実現されています。具体例として 3.7 で、 HTTP クライアントと FTP サーバの間の中継の実現を説明します。

J. c から s への要求、 s から c への応答のそれぞれに含まれる情報資源の識別名を書き換え、情報資源の実際の物理的な識別名と異なる識別名にマップする。これによって、 3.5で説明するマウント機能を実現しています。

K. c から s に要求 req を中継し、応答として res を s から c に中継した時、 req と res を対にしてキャッシュし、それ以降に同一の req を要求された時に、目的サーバに中継せずにキャッシュされた res を応答としてクライアントに送る。

L. s から c、あるいは c から s への通信に含まれるデータに対して、文字コードの変換や、 MIME エンコーディングに関する変換を行う。

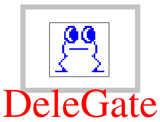
M. 目的のサーバに直接接続するのでなく、更に別の proxy サーバ（Proxy や DeleGate）に中継する。

N. M で中継先のサーバの候補が複数ある場合、あるいは直接でも間接でも目的サーバに到達できる場合には、どの目的サーバにどのような経路で到達を試みるかについて制御を行う（経路制御）。これによって、 3.6に説明するような障害経路の自動的な迂回が実現されています。

O. 通進路のセキュリティを確保して接続するプロトコルや暗号化された通信の中継。 DeleGate2.7 では、 Netscape で採用している HTTPS/SSL の中継を実現しています。

P. クライアントと DeleGate、 DeleGate と目的サーバの間の通信に外部のコマンドをフィルタとして挟んで、要求や応答を加工する機能を DeleGate に外付けする。 2.7 で実現した、クライアントと DeleGate 間の外部フィルタについて、 3.10 に紹介します。

---- コラムＢ： DeleGate のロゴの由来 ---



DeleGate のロゴは囲いの中のカエルです。このカエルは筆者がメールやニュースの署名中に数年前から愛用して来たもので「筑波のガマ」に由来します。

```
   @ @
  ( - )
 _<   >_
```

これを囲うことで、「井の中の蛙」、つまり firewall で囲まれた組織を表し、 DeleGate を使って井の中を飛び出して自由にインターネットの大海原に泳ぎ出したいという気持ちを込めました。（脱水してイカロスになるかも知れませんが）

\------------------------

3.3　アクセス制御

アクセス制御では、誰が、どの情報資源にアクセスできるかを制御します。例えば、 firewall を越えて外部のサーバにアクセスできるようにするための proxy サーバにおいて、最も簡単な制御のルールは、 firewall 内部のユーザに対してのみアクセスを許し、対象の情報資源は制限しないというものです。一方、 firewall を越えて内部のサーバを外部に公開する場合には、アクセスは任意の利用者に対して許す一方、アクセス先のホストやプロトコルを限定することになります。

アクセス制御の実現方法は基本的に、

(1) クライアントがどのホストのどのユーザであるか識別する

(2) 利用者が要求している情報資源を識別する

(3) そのユーザからその資源へのアクセスを許すか否か判定する

という手順によります。最も単純な方法は、 getpeername() によって取得したクライアントのホスト名だけを利用して判定するものです。よりきめ細かなアクセス制御をするには、ユーザ名を用います。特に、複数のユーザで一つのホストを利用する Unix システムのような場合には、これが有効です。

Ident サーバを利用した自動認証の手順は、図 3の様に表すことができます。

(0) ホスト C の上でユーザ名 ysato の利用者がクライアントを使っている

(1) クライアントが H:P にあるサーバに接続する

(2) サーバは、 accept() したポ-トから getpeername() により、クライアントのホスト: ポートが C:Q であることを知る

(3) サーバはクライアントと同じホストにある Ident サーバ (identd) のポート、つまり C:113 に接続し、ポート Q の使用者を問い合わせる

(4) Ident サーバが、そのポートの使用者が ysato であると答える

Ident サーバによる自動認証は、エンドユーザに余分な手数をかけずに認証ができるので便利なのですが、残念ながら Ident サーバはそれほど普及していません。そこで DeleGate では、 Ident サーバが無い場合のために、 FTP サーバを認証サーバとして流用するという方法も実現しています。この方法による HTTP における認証の手順を図 4に示します。

(1) クライアントから url の指す資源の取得を要求する

(2) DeleGate サーバは、その url の取得には利用者の認証が必要であることを通知する

(3) クライアントはユーザに対して認証情報を請求し、ユーザは必要なユーザ名とパスワードを入力する。この時、ユーザ名として与えた ysato@hostD は、 hostD 上の FTP を利用するためのユーザ名であり、パスワードはそのためのパスワードである。

(4) クライアントはユーザに与えられた認証情報とともに、 url の指す資源の取得を再要求する

(5) DeleGate サーバは hostD の FTP サーバに接続し、与えられたユーザ名とパスワードでログインが可能かどうかを試す。

(6) FTP サーバは、ログインが成功したか否かを応答する

(7) ログインが成功したなら認証が成功し、 url で指される資源が返される。

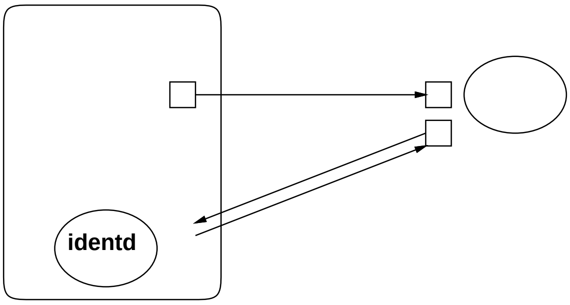


図 3: Ident サーバによる利用者の自動認証

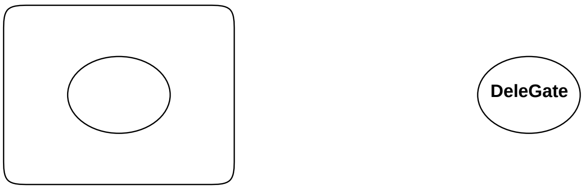


図 4: FTP サーバを利用した認証

このような形で認証サーバとして流用できるサーバは、 FTP サーバの他にもありそうです。このような認証サーバはどのホストにあっても良く、また proxy 経由でも利用できます。そのため、利用者の認証情報という強い保護の必要な情報を firewall などの危険度の高いホストの上に置かずに済みます。 DeleGate では、 DeleGate サーバを遠隔のホストから管理する際の管理者の認証手段としてもこれを利用しています。

3.4　Proxy 方式によらない中継

DeleGate では「proxy サーバ」の概念を持たないプロトコル，あるいは中継機能をサポートしていないクライアントプログラムに対しても，中継機能を提供しています。 CERN の Proxy によって proxy サーバが普通に用いられるようになった HTTP にしても、最初の規格から proxy という概念があったわけではなく、後に拡張されたものです。 HTTP 以外の各種のプロトコルには、もともとの規格上 proxy の機能を定めているものはほとんど無いと言って良いでしょう。

CERN の Proxy による中継は、図 5に示すような形で実現されています。ユーザはまず環境設定により、 Proxy サーバを指定します。設定の方法は環境変数や X Window のリ

---- コラム C ：第 1 版での失敗 (1) -----

第 1 版では、アクセス制御のデフォルト値が緩く、そのまま使うとセキュリティ上危険な場合があり、第 2 版でデフォルトのアクセス制限をより厳しく設定しました。例えば、 HTTP 中継用の DeleGate では以下のようになっています。

	source	destination	protocol
第 1 版	誰からでも	何処へでも	何でも
第 2 版	同一ネットから	何処へでも	WWW 関連のみ

同一ネットからというのは、 DeleGate サーバが動作しているホストと IP アドレスのネットワーク部分が同一のホストからという意味です。クラス A,B,C で規模は違いますが、同一のネットワークであれば同一の組織に属して同一の方針でアクセス管理されている場合が多いと考えて、このようなデフォルト設定にしています。

たとえ firewall を設けていないサイトでも、ニュースサーバのアクセスなどは組織内のホストに限定しているのが普通ですが、 Proxy や DeleGate を経由すると全てのアクセスがそれら proxy サーバのホストから来るように見えますから、注意が必要です。

開発開始当時は、現在のように firewall 上で実用される場合のことを真剣に考えていなかったのが、第 1 版の敗因でした。この仕様変更の結果、「第 1 版では大丈夫だったのに、 2 版にしたらアクセスが拒否されるようになった」という苦情はやはりありましたが、恐れていたほどにはありませんでした。

\----------------------

図 5: Proxy 方式による中継

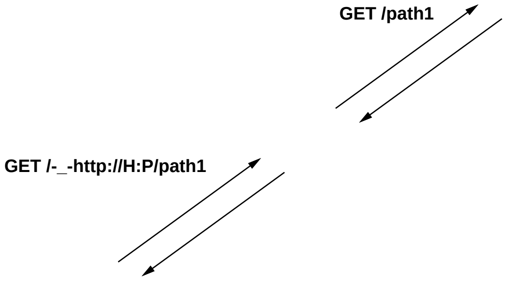


図 6: DeleGate 中継記法による HTTP の中継

ソースなど、クライアントによって異なります。 Proxy サーバ I:Q が指定されると、クライアントは要求を、実際の目的サーバでなく、指定された I:Q の Proxy に送ります。その際、実際のサーバと直接通信する場合には除かれる http://H:P/ の部分も含む、完全な URL を送り出します。 Proxy は、この情報を用いて実際の目的サーバ H:P に接続するわけです。

これに対して、 DeleGate 方式による中継は、 HTTP の場合、図 6のような形で実現されています。この方式では、クライアントソフトウェアは中継のための特別な機能を持たず、従って環境設定もありません。ですから、 DeleGate サーバへの接続のきっかけは、ユーザが与えることになります。

(1) ユーザは、 http://H:P/path1 に対して I:Q の DeleGate を経由してアクセスしようとするとき、クライアントに対して以下のような URL を指示する。

```
http://I:Q/-_-http://H:P/path1
```

すなわち、目的の URL の前に「DeleGate サーバの URL -_-」を前置して指示します。

(2) するとクライアントは、 http://I:Q/ の部分の情報に基づいて I:Q にある DeleGate サーバに HTTP プロトコルで接続し、この部分を除いた部分をリクエストとして DeleGate サーバに送る。

```
GET /-_-http://H:P/path1
```

(3) HTTP 中継用の DeleGate サーバは、このような「-_-」の含まれる URL を受け取ると、その後に実際の目的サーバの URL が続くと解釈する。実際の目的サーバ H:P に対して接続し，以下のような要求を送る。

```
GET /path1
```

(4) 目的サーバから、情報資源へのポインタを含む応答メッセージが返される。

書き換えが必要なのは、 text/html 型のメッセージのボディ部分に含まれる URL （ < A HREF=URL >, < IMG SRC=URL > など）と、 URL を含む HTTP のヘッダフィールド（Location: URL など）です。ただし、相対パスによる部分 URL

の場合（< A HREF="path2" >のような場合）には、この書き換えは不要です。

(5) DeleGate サーバは、応答メッセージに含まれる (a) のような URL を (b) のように書き換えて、クライアントに中継する。

```
（a） http://H:P/path2
（b） http://I:P/-_-http://H:Q/path2
```

すなわち、(1) でユーザが手動で行ったのと同様に、実際の URL の前に「DeleGate サーバの URL -__-」を挿入します。

このような要求と応答の書き換え中継が繰り返されることによって，一旦手動で DeleGate に接続すると、それ以降のクライアントから任意のサーバへのアクセスは自動的に I:Q の DeleGate サーバを経由することになるわけです。

Proxy 方式に対する DeleGate 方式の最大の欠点は、実際の URL と異なる中継記法の URL がユーザに見えてしまうことです。一方、それによって、proxy サーバを個々のクライアントで固定的に設定するのではなく、サーバ側で設定して自動的に切り替えられるという利点を持っています。例えば、HTML の中にもともと DeleGate 中継記法による、

```
<A HREF="http://hostX:portX/-_-http://H:P/path">
```

のようなアンカーが含まれていれば、DeleGate サーバはこれを書き換えません。そして、このリンクをユーザが辿ると、その後は hostX:portX（にある DeleGate サーバ）が proxy として使われるようになります。このようなアンカーをリストにしておいて、複数の DeleGate を切り替えるスイッチテーブルを作ることもできます。

Deleate 中継記法の URL がユーザに見えてしまうという欠点は、クライアントのソフトウェアに変更を加えて、「-__-」の前の部分をユーザに対して表示しないようにすることで克服することができるでしょう。ただ、ヒストリなどに永続的に記録する時に、proxy サーバがひとつの DeleGate サーバに固定され記録されてしまうという問題の解決は難しいかも知れません。

DeleGate 方式による中継は，HTTP に関しては CERN の Proxy 方式に対応したクライアントの普及に伴いその役割をほぼ終えました。しかし，Gopher、WAIS などの専用クライアントのための中継のためには，それらのプロトコルが proxy の機能を規定していないため、依然としてこの中継方式が有用です。DeleGate の現在の版では，Gopher の他にも Whois に対して、この方法を用いた中継機能を実現しており、今後 WAIS に対してもこの方法での中継を実装することを予定しています。

DeleGate ではここに説明した方法の他にも、Telnet, FTP, POP などの既存のクライアントに対応する proxy 機能も提供しています。これら，目的サーバへの「ログイン」を行うプロトコルには，クライアントからサーバに対してログインのためのユーザ名を送信するコマンドがあります。具体的に，FTP, POP にはともに「USER username」という形式のコマンドがあります。ここで，username として通常は目的サーバ内のローカルなユーザ名を送るわけですが、これを拡張して，目的サーバのホスト名やポート番号を埋め込むことができます。DeleGate は、FTP や POP の proxy サーバとして使用される場合、「USER username@host:port」の形式のコマンドを受理するようになっています。クライアントプログラムの利用者は DeleGate に接続した後，このように拡張されたログイン名を送ることによって実際の目的サーバに接続することができるわけです。

---- コラムＤ：第 1 版での失敗 (2) -----

一旦使われ始めてしまった仕様は、簡単には取り消すことができません。DeleGate 第 1 版では、Proxy 機能に対応していないクライアントのための中継を実現するために、URL の拡張記法として「＝@＝中継記法」を導入しました。これは、目的サーバのポートを H:P、DeleGate サーバのポートを I:Q としたとき、

http://H:P/Path
↓
http://I:Q/=@=:H:P/Path

gopher://H:P/TypePath
↓
gopher://I:Q/TypePath=@=gopher:H:P:Type

ftp://H:P/Path
↓
ftp://I:Q/Path=@=ftp:H:P
...

というように、対象プロトコルごとに異なる中継記法を用いており、憶えるのが面倒でした。また、＝ や @ という特殊文字を URL の中で用いることによって中継に不都合が生じる場合もありました。この反省から第 2 版では、プロトコルによらない統一的な「-__-中継記法」に変更しました。

URL
↓
http://I:Q/ -__- URL

つまり、本来の URL の前に DeleGate サーバの URL を前置する、という単純な規則です。今となっては、=@= 記法は過去の遺物なのですが、第 2 版となって半年以上が過ぎた現在でも、この記法が使われていたり、何かの書き物の中で解説されていたりするのを散見して、胸が痛みます。

3.5 マウント機能

一般に proxy サーバは firewall を越えて，内部のクライアントが外部のサーバへアクセスする手段として使われます。DeleGate も主にこのように使われています。一方その逆に、firewall の内部のサーバを外部のクライアントに公開したいこともあります。この場合セキュリティ上、内部にどのようなホストがあるかという情報を外部に漏らさず、また内部でプライベートな IP アドレスが用いられている場合にも対応できる必要があります。

DeleGate では「マウント機能」によりこのような利用方法を可能にしています。マウント機能はその名の通り，ファイルシステムのマウント機能を模したものです。これは、別

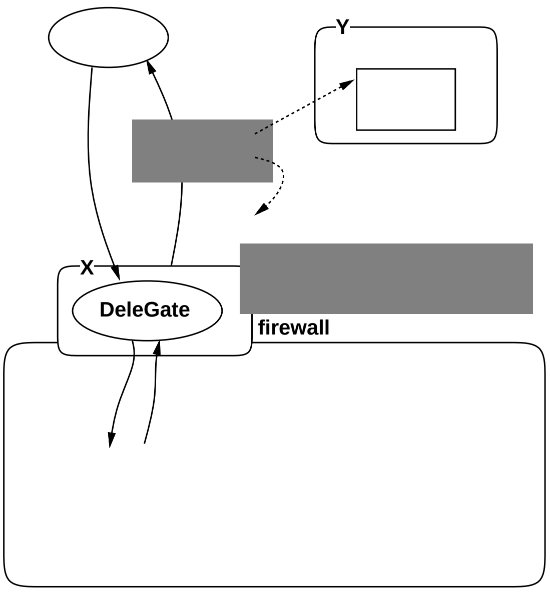


図 7: 実際のサーバ構成と要求 / 応答の流れ

の（複数の）サーバ上にある資源が，クライアントから見ると（全て）DeleGateサーバの上にあるかのように見えるようにするものです。

マウント機能は、前節に述べたDeleGate方式の中継方式をベースに実現されています。「-__-中継記法」による中継は、以下のような一律の書き換え規則に基づく、一種のオートマウントです。

要求→
`http://I:Q/-_-URL` ←→ `URL`
←応答

これに対して、この書き換え規則を、もとのURLに応じて複数定義できるようにしたのがマウント機能です。例えば、図7のように、firewall内部にあるサーバ、AとBを外部に公開したい場合、以下のように書き換え規則を「マウントテーブル」に定義します。

`http://X/a/*` ←→ `http://A/*`
`http://X/b/*` ←→ `http://B/*`

ここで、'*'は、任意の文字列を表します。クライアントから内部のサーバへのアクセスは、以下のような手順で中継されます。

(1) クライアントが、http://X/a/a1 というURLの指す資源を要求する

(2) DeleGateはマウントテーブルを見て、要求中のURL http://X/a/a1 を http://A/a1 に書き換える。その結果、サーバAに接続して /a1 の取得要求を送る。

(3) サーバAからの応答メッセージに、内部サーバBへのポインタ http://B/b1 が含まれている。

(4) DeleGateはマウントテーブルを見て、http://B/b1 を http://X/b/b1 に書き換えて、クライアントに中継する。

このように行われている中継の実際はクライアントからは見えず、全ての資源がfirewall上のホストXにあるように見えます。つまり、クライアントから見ると図8のようにサーバが構成されているようにしか見えず、firewall内部のサーバA, Bの存在を知る事はできません。

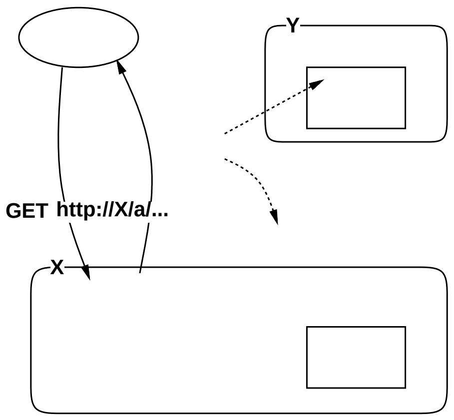


図 8: クライアントから見たサーバ構成

マウント機能はこのようなfirewall越えのための利用だけでなく，一般に、複数のサーバの提供する資源を統合してサービスする一方、その中継をクライアントに意識させないようにするのに有効な方法です。上の例ではHTTPプロトコルに関する書き換え規則を説明しましたが、これを例えばFTPやNNTPに適用することはとても興味深い応用です。

3.6 経路制御

あるDeleGateから、さらに別のProxyまたはDeleGateへ、多段に中継を行うことができます。あるホストから目的のホストへ達する通信経路が複数とおり存在する状況では，DeleGateの経路制御機能により、障害のある経路を自動的に迂回したり，各経路への負荷の分散を行うことができます。

多段にDeleGateを接続する場合、DeleGate間は「DeleGateプロトコル」によって通信します。DeleGateプロトコルにより、複数の経路の中から、最終目的のサーバへの接続が成功するものが見つかるまで、可能な経路が試されます。アプリケーションレベルでの接続が失敗する原因としては、

(a) ドメイン名として指定された目的ホストの名前を(DNSなどにより) IPアドレスに変換することが出来ない(unresolvable)

(b) 目的ホストに対してIP的に到達できない(unreachable)

(c) サーバへの接続が失敗する(connection refused)

(d) 上位のアクセス制御での拒否

などがあります。ここで(a),(b)に関しては、目的サーバに達するための別の経路を選択することで、回避できる可能性があります。DeleGateプロトコルでは、接続の開始にあたり、手前のDeleGateから先方の(目的サーバに近い)DeleGateに向けて、目的サーバのプロトコル、IPアドレスまたはドメイン名、およびポート番号が送られます。これに対して、先方のDeleGateは手前のDeleGateに対して、接続が成功したか否かを知らせます。失敗を知らされた場合には、さらに別の経路を試行するわけです。例えば図9では、あるDeleGateが複数のDeleGateに向けた冗長経路を持っている状況で，目的のホスト名からIPアドレスへの変換を行うことの

---- コラムE：胃の痛む話 --------

昨年度、筆者は霞ヶ関の某庁に出向していましたが、DeleGate を一般に公開する以前、まだよちよち歩きの DeleGate をそこで利用していました。そんな6月のとある午後、いきなり「1時間後に長官が来るので WWW のデモをしろ」ということになりました。こういう場合、ふだんなら何とか動いていたシステムも何故か動かなくなるのが常です。ISDN 回線の接続はしぶる、DeleGate はなかなか立ち上がらない...。第1版の初期まで、DeleGate サーバの socket の REUSEADDR オプションを立てていなかったために、DeleGate を停止して再起動できるまで長時間待たなくてはならないと言うバグがあったのでした。その後、今年の2月になりますが、プレスも呼んで有名なT長官が某ネットワークのこけら落としをするデモがありました。その前日、か細い専用線のネックを避けるために各地の WWW のページを DeleGate にキャッシュして廻っていたら、それまで普通にアクセス出来ていたページが次々に壊れてしまうではありませんか（ひーっ...）。これは、If-Modified-Since 付きの HTTP の要求に対して返される「変更無し」を表す応答メッセージが条件によって実際のデータの変わりにキャッシュされてしまうバグでした。

昨今の WWW の流行で、似たような不幸な境遇に陥った経験のある方は少なくないのではないかと想像します。もしそれが DeleGate のせいだったらと思うと...

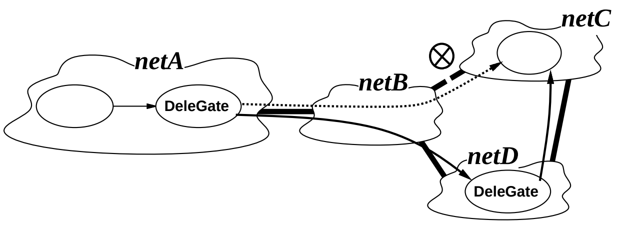


図 10: 障害経路の自動的な迂回

---- コラムF：身の縮む思い -------

開発開始以来、DeleGate は膨大な数のバグを生み出し、そして退治して来ました。バグの発見と退治には、DeleGate メーリングリストの参加者の皆さんを中心として、多くの方からの協力をいただいて来ました。思い出深いバグは多いのですが、中でも発見時に一番身の縮む思いをしたのが、「DeleGate を root 権限で走らせていると時々システムが勝手にシャットダウンする」というものです。原因は、双方向の中継のために fork() した親子プロセスの間で、中継終了時に親プロセスから子プロセスに SIGTERM を送っていたのですが、システムのプロセステーブルが満杯になった時などに fork() が失敗して返される -1 を、そのままプロセス ID と見てシグナルを送っていたためでした。つまり、

```
kill(-1,SIGTERM)
```

で皆殺し...

できる DeleGate への経路が自動的に選ばれている様子を示します。

先方の DeleGate が複数ある場合の選択だけでなく、目的サーバへの接続の試行の順序を制御することもできます。デフォルトでは、

(a) キャッシュにあればそれを用いる

(b) 先方のサーバ（DeleGate や Proxy）へ転送可能ならばそうする

(c) 直接接続を試みる

という順序になっていますが、この順序は制御することができます。

図10のようなネットワーク構成において，client から server にアクセスする場合，通常は netA → netB → netC のように直接通信します。ここでもし netB と netC の間が IP レベルで不通になった場合に，自動的に netD に置いた DeleGate に迂回して netC に中継させるようにすることができます。

実際には遠隔地のネットワークの切断を直接知る事は容易ではありませんから、接続の手続きにタイムアウトを設けることによってこれを近似的に判断します。一定時間内に接続が成立しなかった時に，別経路による接続の試行に移るわけです。従ってこれは，過負荷状態のネットワーク経路を自動的に迂回して負荷分散を実現する方法ともなっています。

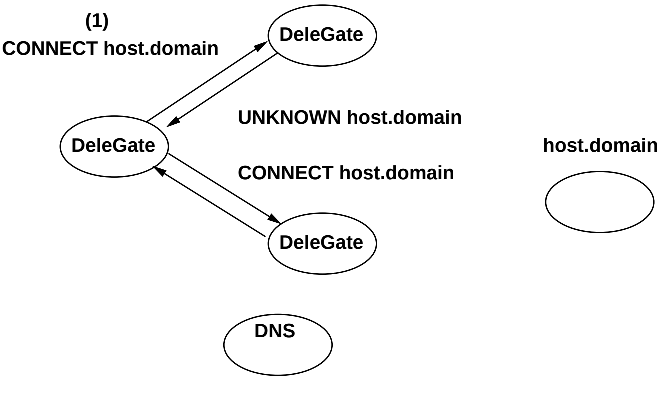


図 9: 経路の再試行と自動的な選択

3.7 プロトコル変換

DeleGate は Proxy と同様に、HTTP クライアントからの、FTP,WAIS および Gopher のサーバへのアクセスを中継します。この場合の中継の基本的な枠組は以下のようになります。

(1) HTTP クライアントは、Proxy サーバに対して以下のような HTTP の要求を送る。

```
GET protocol://server:port/path HTTP/1.0
```

(2) Proxy サーバは、server:port にある目的サーバに接続し、protocol で指定されたプロトコルを用いて path で指される資源を取得する。

(3) Proxy サーバは、取得したデータを MIME メッセージの形式にして、HTTP の状態コードとともにクライアントに返送する。

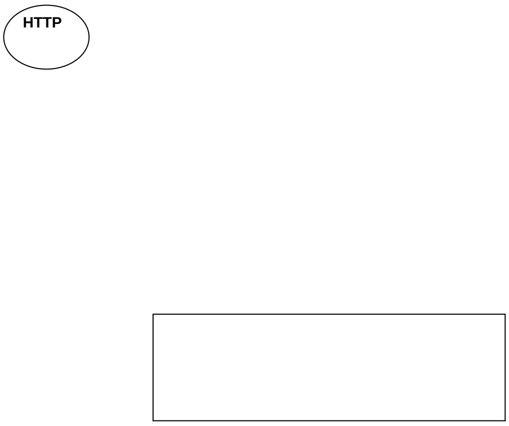


図 11: FTP と HTTP の中継

MIME メッセージのヘッダには、 path の名前や取得した資源の内容に基づいて推定した Content-Type （データ型）や Content-Transfer-Encoding （データエンコード形式）を設定します。例えば、 7 ビットのプレーンテキストなら、

```
Content-Type: text/plain
Content-Transfer-Encoding: 7bit
```

GIF 形式のファイルなら、

```
Content-Type: image/gif
Content-Transfer-Encoding: binary
```

といったヘッダになります。

他の資源へのポインタを含む資源の場合、例えば FTP のディレクトリの内容 (LIST コマンドの結果) などは、ヘッダに text/html 型と表示し、ボディ部分を HTML 形式で表します。図 11に、その例を示します。

3.8 コネクションキャッシュ

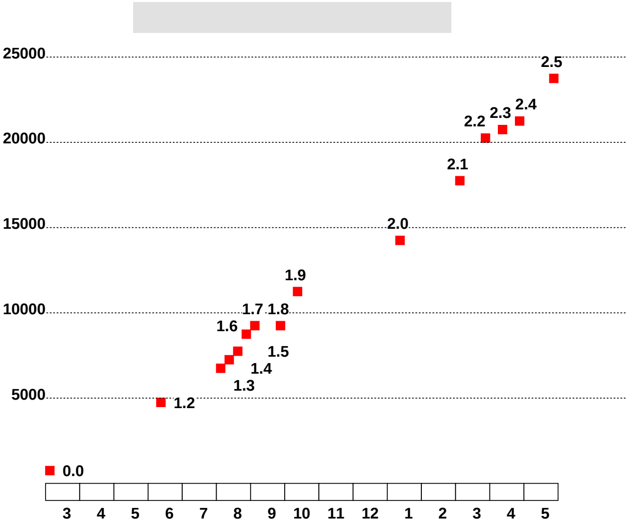
広域ネットワークを利用するとき，ユーザは，データの転送に時間がかかる問題以前に，サーバへの接続そのものにかかる待ち時間を意識させられます。特に HTTP では，一つの資源の取得ごとにこの接続作業が繰り返され、一方個々の資源のデータ量は小さいことが多いので，ユーザの待ち時間は接続待ちの時間で支配されることになります。（HTTP/1.0 の規格上では，一つの接続で複数の資源を取得する方法にも言及されているのですが、現時点では実用化されていません）

特に， FTP、 NNTP、 WAIS のように本来は一つの接続で繰り返し資源を取得できるプロトコルにおいて、この問題は深刻です。クライアントがサーバと直接接続する場合には、一旦目的サーバとコネクションを確立したら、その後の同一サーバへのアクセスにはそのコネクションを繰り返し利用できます。しかし、 DeleGate や Proxy のような HTTP の proxy サーバを介してアクセスすると，クライアントと proxy サーバの間の HTTP プロトコルによる接続が資源の取得毎に作り直されるため， proxy サーバと目的サーバとの間のコネクションも毎回作り直されてしまうからです。すなわち、前節の図 11のように、一つのファイルや記事の取得ごとに，接続，要求，応答，切断という作業が繰り返されてしまうわけです。

---- コラムG： DeleGate の成長記録 ----

昨年の 9 月に、とあるネットワーク関係の協会の委員会に呼ばれて DeleGate のお話をしたのですが、その時「現在どのくらいまで完成したと考えるか」という問に、筆者の答えは「利用者に要求されている機能は 9 割実現したと思うが、開発者としてやりたいことは 1 割くらいしかできていない」というものでした。最終的なゴールを決めずに開発を続けていますし、やりたいことは次から次に出て来ますから、現在でもこのような問を受けたら、答えは同じようなものです。それでも「やりたいことの 2 割位」まで、前進したでしょうか...

プログラムの行数にさほど意味はありませんが、それでも何かの目安にはなります。開発開始以来のプログラムの規模の変遷をプロットしてみると、ほとんど直線的に拡大して来ていました。



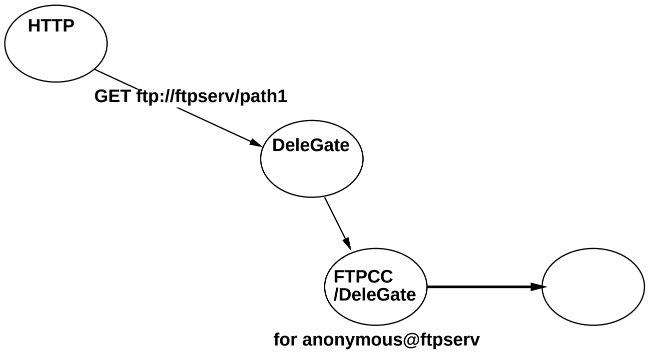


図 12: FTP コネクションキャッシュ

これは極めて非効率であり、ユーザに対する応答性が低下するばかりでなく，ネットワークや目的のサーバに余計な負荷をかけるという点でも問題です。接続ごとに繰り返される、接続のための通信、FTP サーバの起動，ユーザの認証、ログファイルへの書き込みなどの処理コストが問題になります。

そこで DeleGate では，一旦 DeleGate が目的サーバとの接続を確立すると，その接続 (コネクション) を一定時間保持して再利用する「コネクションキャッシュ」呼ぶ方式を実現しています。名前が示すようにこの方式は，データのキャッシュと同様な効果をコネクションに対して適用しようとするものです。すなわち、データキャッシュがデータの保持・共有・再利用によってデータ転送に関するコストを削減するのと同様に，コネクションの再利用によって接続に関するコストを削減することを意図したものです。

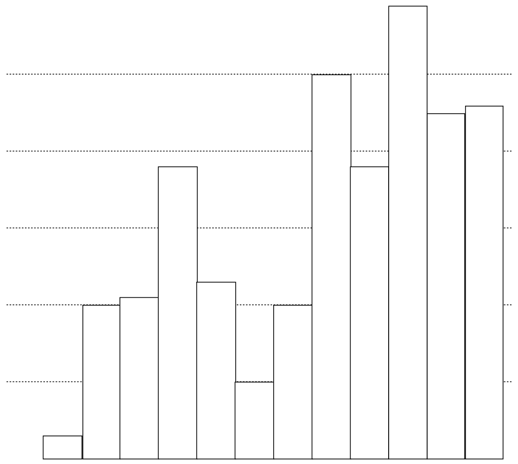
図 12に、 FTP と HTTP を中継する際のコネクションキャッシュの動作を示します。 FTP 用のコネクションキャッシュを行うサーバを FTPCC と呼んでいます。

(1) 最初の接続では、目的の FTP サーバ ftpserv への接続、ログインを行った後、データを取得する。

(2) クライアントからの要求を処理し終えた後も、 FTPCC は FTP サーバとのコネクションを一定時間維持し、次の要求を待ち受ける。 DeleGate は、 ftpserv の FTP サーバへのコネクションが FTPCC に保持されていることを記憶している。

---- コラムH : DeleGate の配布状況 ----

DeleGate の最新版は ftp://etlport.etl.go.jp/pub/DeleGate/ から anonymous-FTP で配布しています。このディレクトリに置かれた配布ファイル (delegate*.tar.gz) に対する転送状況の統計を以下の図に示します。 94 年 6 月 12 日配布開始~ 95 年 7 月 23 日現在までで、海外 13 カ国 136 ドメインを含む 687 ドメインに配布されています。



\-----------------------

(3) （別の）クライアントから ftpserv 向けの別の要求が来た時、このサーバ向けの FTPCC が残存している場合には、その FTPCC 経由で（既存のコネクションを利用して）データを取得する。ただし、 FTPCC が他のクライアントによって使用中の場合には、新たな FTPCC サーバを作成する。終了後、 (2) に戻る。

いずれかのクライアントが，DeleGate を経由してある目的サーバに対して接続を行うと，そのコネクションは複数のクライアントから共有して再利用できるため，この方法は個々のクライアントが直接 FTP サーバに接続するよりも効率的にすぐれています。

この FTP のコネクションキャッシュによる効果はテキメンです。例えば電総研からインターネットを通じて遠隔の FTP サーバに接続するのには，数秒間かかるのが現状ですが，コネクションキャッシュによりこの待ち時間は無くなります。 HTTP クライアントを用いて、 FTP のディレクトリをブラウズする場合，FTPCC 無しではディレクトリ移動のたびに数秒要するものが，FTPCC によりほとんどの場合一秒以内で移動できるようになりました。

3.9 DeleGate トンネル

互いに IP 的に接続されていないホスト間を DeleGate によって接続し、その上で TCP/IP 上のアプリケーションプロト

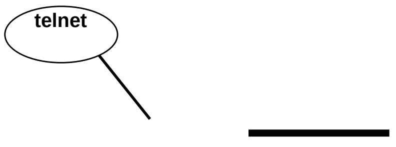


図 13: DeleGate トンネル

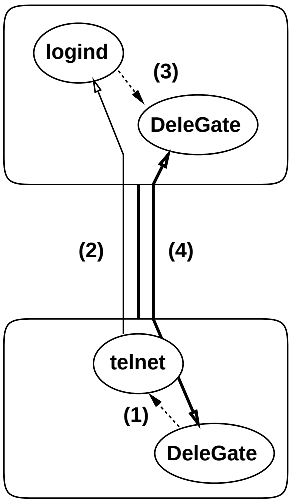


図 14: DeleGate トンネルの掘削手順

コルを中継する機能があります (図 13)。。これを「IP トンネリング」にならって「DeleGate トンネリング」と呼んでいます。DeleGate トンネルは、初期に (DeleGate1.3 で) 試験的に導入しましたが、最近 DeleGate2.7 において「Teleport プロトコル」と呼ぶより一般的なプロトコルを導入した上でその上に再構築しました。

DeleGate トンネルは、例えば図 14のような手順で自動的に作成されます。

(1) クライアントのホスト上で DeleGate から telnet クライアントを起動し、

(2) proxy サーバのホストに（モデム経由などで）ログインして、

(3) そのホスト上に DeleGate を起動すると、

(4) 2 つの DeleGate の間が双方向に接続される。

ここで、DeleGate 間の通信は Teleport と呼ぶ独自のプロトコルの上で運ばれています。Teleport プロトコルは、双方向の多重化通信を運ぶためのシンプルなプロトコルです。

この Teleport プロトコルを利用して、互いに firewall で囲まれたサイト間での通信を DeleGate で仲介することができます。図 15(1) のように、互いのサイトが firewall で遮蔽されていて、相手のサイトの中のホストにアクセス出来ない状況を考えます。これは良くある状況です。通常、互いの通信を通過させるには、firewall の設定により、サイトＡはサイトＢのホスト b からのアクセスを受け入れる、サイトＢはサイトＡのホスト a からのアクセスを受け入れる、というような設定をする必要があります。このような設定の手間は小さくないですし、firewall が多段階にある場合や、内部でプライベート IP アドレスを用いている場合などには、対処が複雑あるいは不可能になります。

ここで、(2) のようにサイトＡからもサイトＢからもアクセスが可能なホストＸがあったとします。典型的には、ホストＸがサイトＡまたはＢの firewall 上にある場合です。ホストＸに Teleport サーバを起動して置き、サイトＡ、サイトＢのそれぞれの内部で DeleGate サーバを起動し、Teleport プロトコルによりホストＸの Teleport サーバに接続します。すると、それぞれの DeleGate サーバは互いに双方向に接続され、(3) のように双方向に通信を中継することができるようになります。

ホストＸの Teleport サーバにおいて、「ホスト a とホスト b の間の通信だけを通過させる」というアクセス制御を行うことにより、サイトＡ、Ｂの firewall の各々でアクセス制御を行うのと同等のアクセス制御が行え、かつそれを一箇所で管理することができるわけです。

3.10 外部フィルタ

ユーザが用意したフィルタプログラムをクライアントと DeleGate との間に挿入して、クライアントからの要求やサーバからの応答を書き換えることができます (図 16)。これによって、DeleGate の本体に変更を加える事無く、利用者の用意した機能を DeleGate に追加することができるわけです。

まず、(A) のようにクライアントから DeleGate へのフィルタと、DeleGate からクライアントへのフィルタを独立のプロセスとして実現する場合には、これらのフィルタプログラムはいわゆるフィルタ、すなわち標準入力から入力して加工した結果を標準出力に出力するものです。例えば、クライアントからの要求によらず、常に応答データを加工する、たとえば文字コードの変換を行うようなフィルタは、この方法で実現できます。

一方、クライアントからの要求に応じて応答の加工方法を変えたい場合には、(B) のように一つのフィルタプログラムで双方向の中継を行う必要があります。この場合フィルタプログラムは、ファイルディスクリプタ 0 にクライアント向けの入出力のためのソケット、ファイルディスクリプタ 1 に DeleGate 向けの入出力のためのソケットが接続された状態で起動されます。このようなプログラムでの双方向の中継は、poll() あるいは select() を用いて実現します。図 17に、poll() を用いた簡単な双方向フィルタの実装例を示します。これは単に、双方向のデータ送信の様子を標準エラー出力に表示するだけのフィルタです。また、簡略化のためにエラー処理や出力の再試行処理などは省きました。

おわりに

本稿では、具体的な利用の方法については書くことができませんでしたが、これは DeleGate の配布セットに含まれるドキュメント (./doc/Manual.txt) を読んでいただければ、概ね理解していただけると思います。もし不明な点がありましたら、DeleGate メーリングリストでお尋ね下さい。筆者自身、あるいは他の参加者の方々から説明が受けられるはずです。

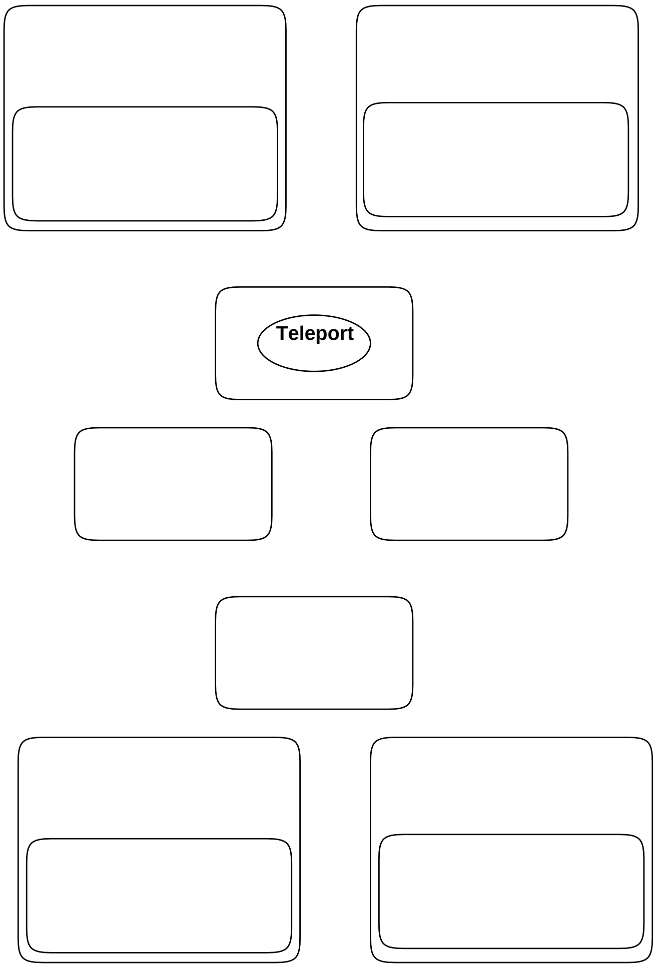


図 15: DeleGate が Vehicle で Teleport

---- コラムI： DeleGate メーリングリスト –

DeleGate に関するバージョンアップ情報の通知，障害報告，改良のための議論などは， DeleGate メーリングリスト < *delegate@etl.go.jp* > で行われています。 95 年 7 月 24 日現在、 215 のドメインから 443 名の参加者があり、一日平均 5 通くらいの記事が流れています。このメーリングリストでに出された質問や要求には、必ず開発者から何らかの応答があります。

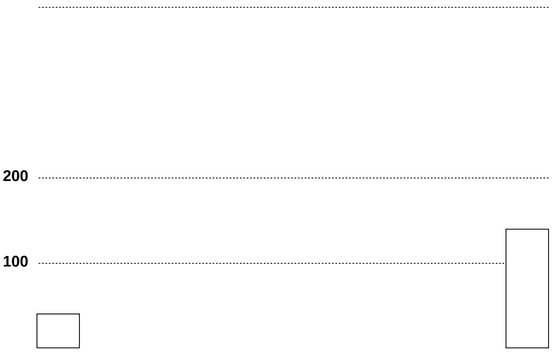


図 16: 外部フィルタ

```
#include <stdio.h>
#include <poll.h>

main(){
  struct pollfd fds[2];
  int timeout;
  int nready;
  char buf[1024];
  int rcc;

  fds[0].fd = 0; /* クライアント側ソケット */
  fds[0].events = POLLIN;
  fds[1].fd = 1; /* サーバ側ソケット */
  fds[1].events = POLLIN;
  timeout = -1; /* タイムアウト無し */

  while( 0 < (nready = poll(fds,2,timeout)) ){
    if( fds[0].revents & POLLIN ){
        /* クライアントからサーバへの中継 */
        rcc = read(fds[0].fd,buf,sizeof(buf));
        if( rcc <= 0 )
          break;
        /* ... リクエストの加工 ... */
        write(fds[1].fd,buf,rcc);
        fprintf(stderr,"client->server: %d\n",rcc);
    }
    if( fds[1].revents & POLLIN ){
        /* サーバからクライアントへの中継 */
        rcc = read(fds[1].fd,buf,sizeof(buf));
        if( rcc <= 0 )
          break;
        /* ... レスポンスの加工 ... */
        write(fds[0].fd,buf,rcc);
        fprintf(stderr,"server->client: %d\n",rcc);
    }
  }
}
```

図 17: 双方向フィルタの骨組み

---- コラム J ： DeleGate の配布条件 ----

工業技術院電子技術総合研究所で開発された DeleGate は、その著作権が工業技術院に帰属し、工業技術院職務作成プログラム等管理規定に基づいて 1994-ETL-E8715-01 という管理番号を与えられています。ただし、研究の途上にあり実際の利用を通じたフィードバックが研究開発の遂行に不可欠であるため、無償かつ無保証で配布を行っています。非営利目的であれば、電総研の FTP サーバからの取得・利用を自由に行うことができます。

DeleGate はこれまで、第 1 版では proxy サーバとしての基本機能の試行的実現、第 2 版では性能強化および、より多目的の中継システムに向けた機能拡張を行って来ました。今後、仕様の整理と、コンフィギュレーションファイルの整備を行った後、今年中に第 3 版に移行する予定です。その時には、少なくとも DeleGate の利用者の皆さんに対しては、「これが DeleGate の仕様です」とすっきり示せるような状況にしたいものだと思っています。

第 3 版以降では、ひとかたまりのシステムとしては成長の限界の見えてきた DeleGate を、多細胞のシステムとして成長させて行きたいと考えています。 DeleGate は何でありたいか？ それは結局、 WWW のような総合的なソフトウェアシステムを、複数の開発者が分担あるいは競争して開発する作業を支援できる共通のプラットフォームでありたいということです。その布石として第 2 版で導入した外部フィルタ機能では、利用者の手で、 DeleGate の外部から新たな機能を容易に付加することのできるインターフェースを導入しました。この機能を利用して DeleGate を踏台とし、皆さんの手で新たな proxy 機能を実験・開発していただければと思います。そして、このインターフェースの仕様についての不足な点の指摘や改良の要求をどんどんぶつけていただき、その改良を通じて、ソフトウェア開発者同士としての実りある協力関係を実現することが筆者の夢です。

参考文献

[1] 佐藤豊: プロトコル中継システム DeleGate の開発, 電総研研究速報 TR-94-17, (1994).
<URL:ftp://etlport.etl.go.jp/pub/DeleGate/ETL-TR-94-17.ps.gz>

[2] 佐藤豊: DeleGate に見るインターネット上の公共ソフトウェアの成長過程, 機械振興 **28**, **4**, (1995) 96-101.

[3] 佐藤豊: 多目的プロトコル中継システム DeleGate, 電子技術総合研究所彙報 **59**, **6**, (1995).
<URL:ftp://etlport.etl.go.jp/pub/DeleGate/ETL-BULLTIN-95-06.ps.gz>

なお、表 1 に示した RFC、 Internet Draft については、
ftp://ds.internic.net/rfc/
ftp://ds.internic.net/internet-drafts/
から、 anonymous FTP で入手できます。

さとう・ゆたか　電子技術総合研究所
E-mail: ysato@etl.go.jp
WWW: <URL:http://www.etl.go.jp/People/ysato/>