DeleGate: A General Purpose Application Level Gateway

Yutaka Sato

Computer Science Division, Electrotechnical Laboratory,
1-1-4 Umezono, Tsukuba, Ibaraki, 305 Japan

Abstract. DeleGate is a general purpose application level gateway which relays a variety of application protocols on the Internet, including HTTP, FTP, Gopher, WAIS, NNTP, SMTP, POP, Telnet, X Window and CU-SeeMe. It overcomes a variety of obstacles on the Internet caused by barriers for security, insufficient band width and mismatching among protocol specifications and implementations.
An application level gateway mediates communication between clients and servers recognizing the syntax and semantics of application protocols it is relaying. Therefore it can supply various functions like caching and translation to an application protocol, for its procedure and data, without changing clients and servers.
This paper presents an overview of DeleGate and its two unique features termed mounting and connection cache. Mounting is a mechanism which provides a virtual view for a name space of networked resources by filtering, aliasing and merging names from original name spaces. Resource names transferred as data on each protocol are recognized and rewritten by DeleGate. Connection cache allows established connections to servers to be resources which are sharable and reusable among clients. It reduces the cost of repetitive connection establishments to the same server, where the repetition is introduced by an application protocol gateway which relays between different protocols.

1 Introduction

An application level gateway is not new, but is a traditional technology to mediate between clients and servers where they cannot communicate directly for various reasons. They may be isolated by security *firewalls* placed at the network entrance at each organization to avoid intrusion from the outer world[1]. They cannot communicate physically because a necessary path for packet exchange is closed. Another typical application of it is a protocol gateway which mediates clients and servers that use different application protocols. In this case, they cannot communicate logically because they cannot speak nor understand the peers language. An application level gateway solves such problems acting as a *proxy server* which behaves as a server for the client while behaving as a client to the real server. It is an interpreter which can communicate with both sides physically and logically.

The evolution of the technology was not so active until the appearance of the World Wide Web (WWW). WWW dramatically increased the activity and

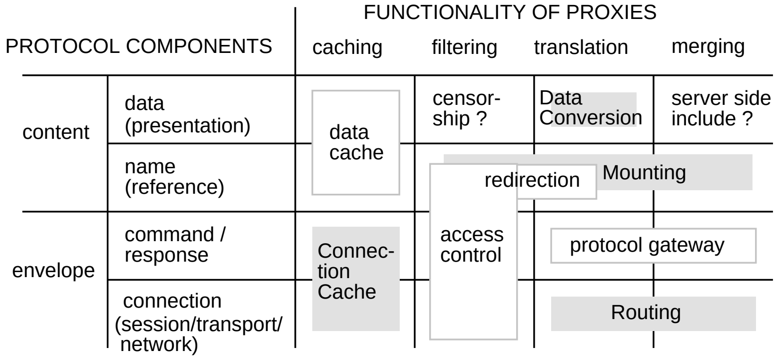


Fig. 1. Possible utilization of protocol components by proxies

altered the application of the Internet, and eagered solutions to overcome the firewalls, narrow band width and long latency of wide area networks. The solution adopted was a *caching proxy* for HTTP[8], which is an application level gateway conveying HTTP protocol beyond the firewall and making retrieved data sharable and reusable in a cache. Amongst the standardized application protocols in the Internet, the HTTP protocol was the first to introduce the proxy feature in its specifications, making the complete functionality of the protocol available via a proxy server, gatewaying to other application protocols, and supporting cache control. Therefore, nowadays just saying proxy implies such an application level gateway devoted to HTTP.

Many HTTP proxies have appeared in the last three years and have been used widely [3][17][4][5]. Through the practical use of proxies, many problems have been recognized and subsequently been improved and extended both in its specification[9] and implementations.

In 1994, the author started development of a caching proxy server named DeleGate [17][18][19] compelled by his own practical necessity. Starting as a proxy for Gopher, it was immediately extended to support a variety of application protocols including HTTP. From its inception, DeleGate was not a proxy devoted to HTTP protocol, but was intended to support clients of multiple protocols, while other HTTP proxies can only communicate with clients in HTTP protocol. While it has been widely and practically used as an HTTP proxy, the main purpose of DeleGate has been as a testbed to explore "What can a proxy do ?"

An application protocol carries information resources contained in carrier envelopes based on a clearly specified procedure and data structure. The envelope consists of procedures to exchange the content and protocols in lower layers to transport the content. The content contains data on information resources and names or references pointing to other resources. An application gateway relays such protocol recognizing the syntax and semantics of the envelope and its content. Thus the question is "How can a proxy utilize information recognized from the protocol components ?"

A possible utilization of protocol components in proxies is shown in Fig.1. In this figure, it can be seen that proxies can perform caching, filtering, translation and merging operations. Almost all HTTP proxies support access control, caching for data, and protocol gateway. DeleGate also supports them, as well as performing other tasks, such as translation (e.g. Japanese character code conversion), translation and merging of names (Mount), routing at the application level, and caching for connection (Connection Cache).

1.1 Where is it applicable ?

There are many situations where proxies are necessary to solve problems or useful in advanced usage of networks (Fig.2). Generally, HTTP proxies are utilized at the client side to cross the firewall of the organization it belongs to. Some servers with heavy accesses make *mirror servers* (i.e. clones) which serve exactly the same content as the original servers. As such, proxies are applicable to many situations. Some typical examples are to:

- Cross firewalls transparently without breaking security
- Provide a virtual view for servers hiding their actual existence
- Translate transferred data into a format usable by the client or user
- Translate protocols to one which is recognizable by the client
- Cache transferred data to revise response time and reduce traffic
- Reuse a connection to a distant host which is expensive to establish
- Routing among networks which are unreachable from each other at the IP level

These situations exist independently from application protocols, thus an application gateway which supports multiple application protocols can fulfill these requirements with a single common mechanism.

Ordinarily there are several routes available for a client to reach a destination server. Routes via Socks gateways[13] or via other proxies may be available as well as several direct IP routes. The use of some routes may be restricted to only privileged users or special application protocols. To make such a higher level routing possible, DeleGate can control which route should be used for which clients accessing which servers in which protocol.

1.2 What can be relayed ?

The protocols supported by DeleGate are shown in Fig. 3. One of the major features of DeleGate is HTTP proxy which is compatible with CERN-HTTPD as a caching proxy server, relaying a variety of protocols other than HTTP for clients of HTTP protocol. So far, conversion from FTP, Gopher, NNTP, WAIS and HTTPS/SSL to HTTP is supported.

DeleGate also provides proxy functions for most major application protocols on the Internet, including HTTP, FTP, Gopher, WAIS, X Window, NNTP,

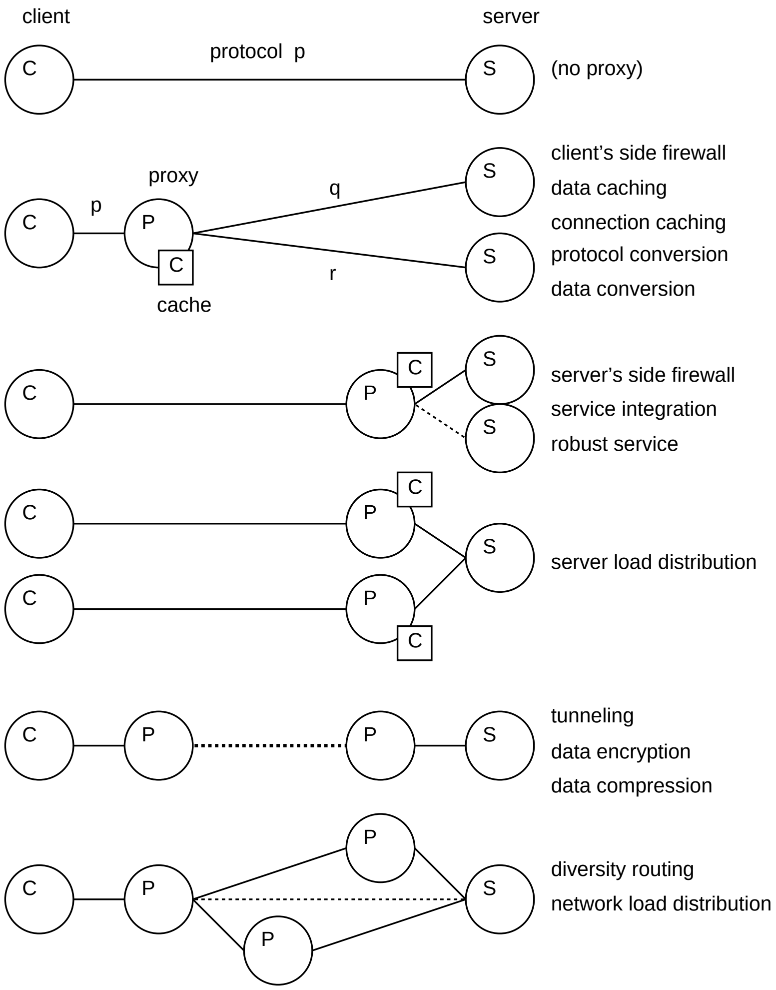


Fig. 2. Applicable situations of proxies

SMTP, POP, Telnet, Socks and CU-SeeMe. Caching is supported for HTTP, Gopher, FTP, and NNTP. Caches for Gopher, FTP, and NNTP are shared among HTTP clients and other clients native to these protocols.

Unlike HTTP, most of the other protocols have no standardized interface from clients to proxies. Therefore DeleGate introduces its original proxy mechanism for them, which, however, is designed to be as universal among protocols as possible. It does not require changes to existing client software, either. It also has adopted proxy interfaces of other systems. For example, it is compatible with the TIS Firewall Toolkit[2] for Telnet, X and FTP.

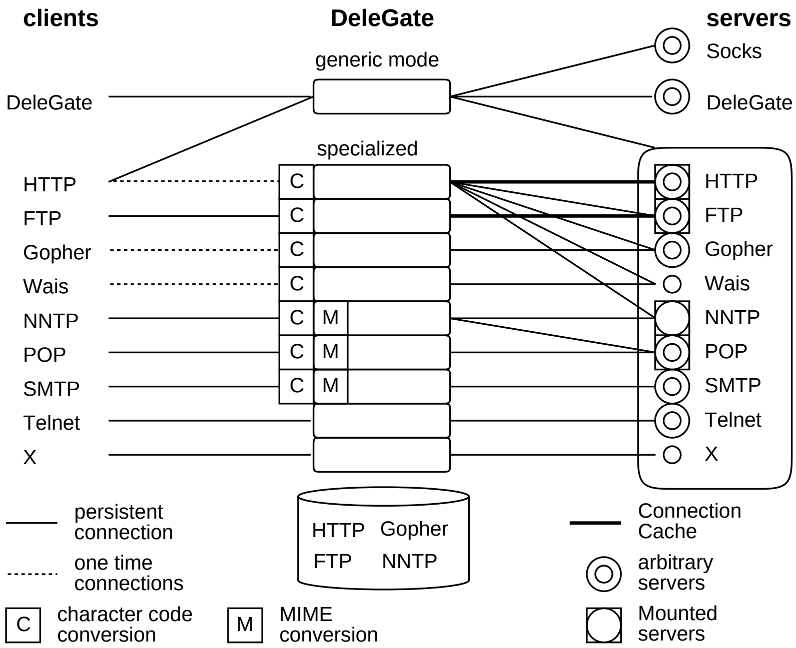


Fig. 3. Protocols supported by DeleGate

1.3 How Is It Constructed ?

Another purpose of DeleGate is to find out "What basic functions are necessary to construct clients, proxies, or servers of application protocols." Historically, many programs are created from scratch. Each is specialized to a single application protocol, or specialized to its platform, although they have very similar functions to each other. If a set of commonly used components are provided as library functions, developing a program for a specific protocol becomes easy by sharing and reusing them.

The overall construction of DeleGate is shown in Fig. 4. Its components are classified into two categories: common parts independent from a specific protocol and parts specific to each protocol. The components of DeleGate consist of about 67,000 lines of source code written in C, 70% are for common parts and 30% are for protocol specific parts. Less than 2% of the program concerns the lower network layers, TCP/IP and UDP/IP via the "socket" interface. DeleGate has compatibility modules to cope with a variety of UNIX platforms and the Windows system. The total size of them is less than 1% of the whole source program.

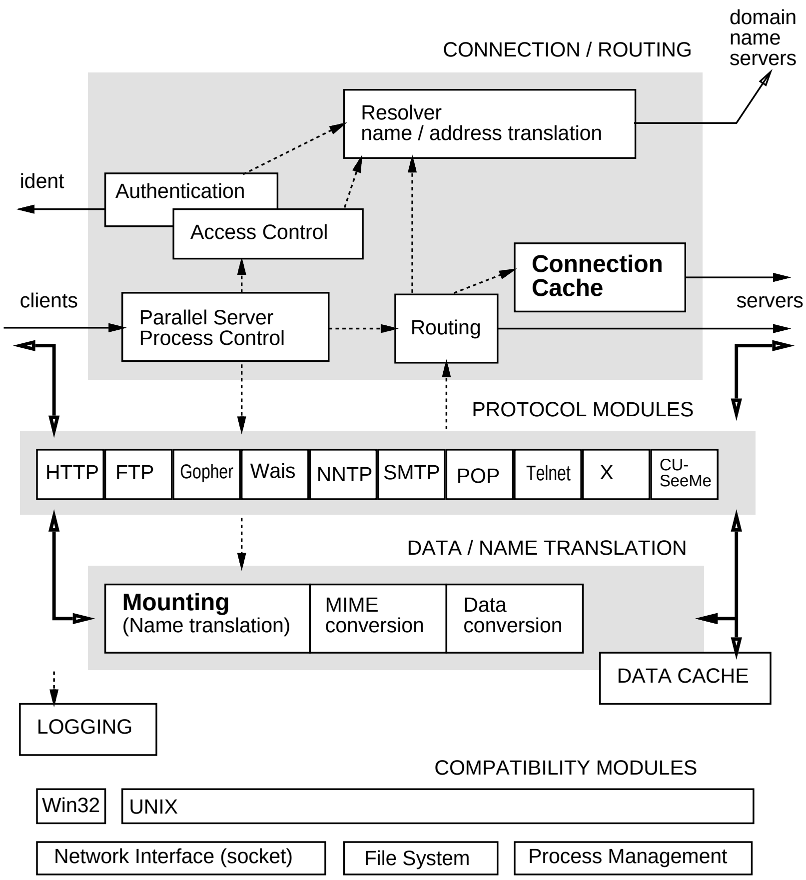


Fig. 4. Overall construction of DeleGate

2 Mounting – Creating a Virtual View For Resources

A DeleGate server can merge resources provided by other original servers, and then provide them to clients acting as if it is the original server. This mechanism is named "mounting" because it acts like a mounting mechanism in the Network File System, where files physically residing on remote hosts are uniformly accessible as if they reside on the local host.

In this case, clients recognize the DeleGate not as a proxy but as the original server which provides all the information. Because the existence of servers behind the DeleGate is hidden, this is very useful in providing information from servers within a firewall, to beyond its firewall, to the Internet.

As shown in Fig.5 the mounting mechanism provides three functions: filtering, aliasing, and merging. Resources on servers are identified by names as URLs

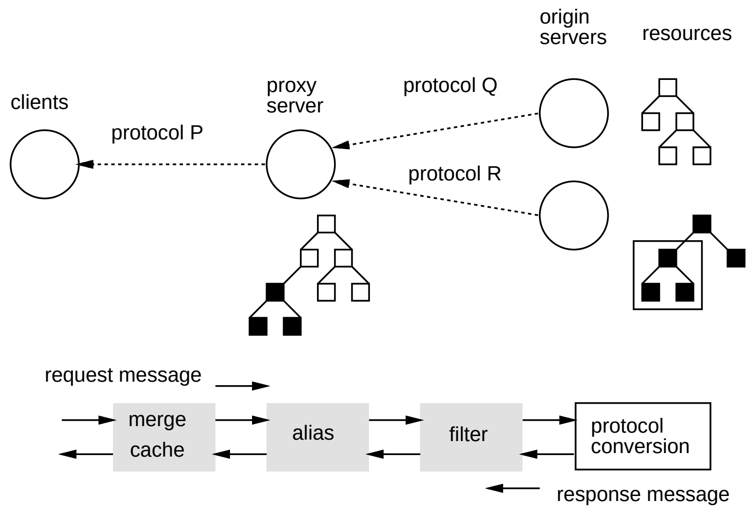


Fig. 5. Resource mounting

in HTTP, news group names in NNTP, directory names in FTP, and so on. Sets of names from the original servers are merged into a single set of names on a DeleGate server. Names which should be merged into the set are selectable, and names in the set can be mapped to other names.

In protocols like NNTP, most existing clients expect that all the information is available from a single connection to a single news server. Because switching the news server is not easy, it is best if users don't have to switch between several servers. Otherwise each NNTP server would has to hold the same full set of information (news groups) by exchanging them with the other servers. But maintaining such a large server requires much time and labor, and it introduces a risk that internal news groups, which should be available only to the local domain, may leak out by accident. The mounting mechanism of DeleGate solves this problem providing a virtual news server which merges news groups from several news servers. Local news servers can then be maintained in isolation without exchanging articles with the other servers.

2.1 Technical Details

In the case of HTTP, a name for identifying an information resource is a Uniform Resource Locator (URL)[10]. They are carried in messages over the HTTP protocol exchanged between clients and server. We now consider how to make a real URL like (2) appear to client as if it is URL (1).

(1) http://delegate/path1/*
(2) http://host1/*

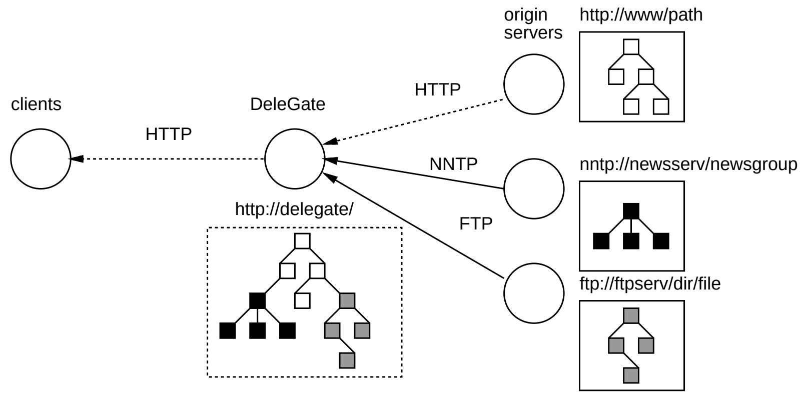


Fig. 6. Example of mounting

To make it work, URLs contained in a request message, sent from a client to a server, must be renamed from (1) to (2), and URLs contained in a response message, sent from the server to the client, must be renamed from (2) to (1).

A URL may appear at several places in messages; each message consists of a header part and a body part following the MIME format[12]. The header includes meta information about what type of data is contained in the body part. The syntax of a header is defined by the HTTP specification including where URLs may appear in a header. When the body data type is HTML, URLs may appear as attribute values of some tags defined in the HTML specification[11].

A simple example of messages rewriting is shown in Fig.7, which includes such URLs (or components of it) in each line. From the viewpoint of the target server, it must not be aware whether it is accessed directly from a client or indirectly via a DeleGate. Therefore an informational field like "Referer" should also be rewritten.

2.2 A Proposal for Standard Protocol

To make mounting successful, any resource identifiers carried on the protocol should be recognized and rewritten if necessary. To make the mechanism more universal in their functionality, and more simple in its implementation, the way they are expressed in data should be uniform, and they should appear only where defined by a protocol specification.

Unfortunately, URLs carried in HTTP protocol are used in various ways and scattered in transferred data, especially in HTML. It's not so easy to find out where URLs are contained in HTML text. Also some URLs are passed as user defined data which are not defined in the protocol specification. Some proposed headers for HTTP like "Set-Cookie" includes components of URLs which don't follow the standard expression of URL.

– a request message from a client –
GET /path1/path2 HTTP/1.0
Host: delegate:80
Referer: http://delegate/path1/path3

– a request message forwarded to the server –
GET /path2 HTTP/1.0
Host: host1:80
Referer: http://host1/path3

– a response message from the server –
HTTP/1.0 200 OK
Location: http://host1/path2
Set-Cookie: name=val; domain=host1; path=/path2
Content-Type: text/html; charset=US-ASCII

<A HREF="http://host1/path4">
<IMG SRC="http://host1/icons/img.gif"> </A>

– a response message forwarded to the client –
HTTP/1.0 200 OK
Location: http://delegate/path1/path2
Set-Cookie: name=val; domain=delegate; path=/path1/path2
Content-Type: text/html; charset=US-ASCII

<A HREF="http://delegate/path1/path4">
<IMG SRC="http://delegate/path1/icons/img.gif"> </A>

Fig. 7. An example of rewriting HTTP messages

2.3 Automatic Mounting

Like automatic mounting in the Network File System, DeleGate provides automatic mounting for arbitrary resources in the whole URL space. For a resource identified by a *URL* in the space, a DeleGate server provides another URL to access it via DeleGate as follows (3),

(3) http://*delegate*/-_-*URL*
(4) *URL*

DeleGate interprets URLs like (3) in a request message as *URL* (4), and *URL*s (4) are transformed to (3) in the response message.

Historically, this mechanism of proxying was introduced to DeleGate before the static mounting described above. Originally it was made to relay Gopher protocol which has no standardized proxy protocol, later it was applied to HTTP because standardized proxying was not yet so widely implemented in clients and servers at the time. Because the HTTP proxy protocol was standardized in

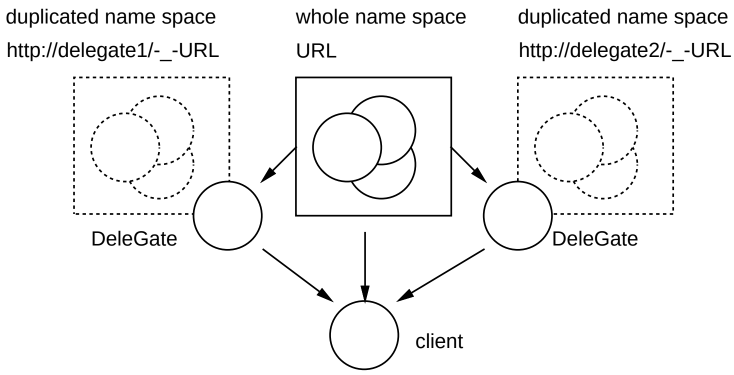


Fig. 8. Automatic mounting

HTTP/1.0 and was widely implemented in clients and proxies, it becomes obsolete as a HTTP proxy mechanism for usual use. But yet this mechanism is useful for making use of an arbitrary number of proxies temporary, or automatically switching proxies from the server side.

This proxy notation "-_-"of the DeleGate duplicates the whole URL space over the world under "http://delegate/" (Fig. 8). Unfortunately, this caused a serious problem when WWW robots began to collect information about resources by walking around URL space because they strayed off into duplicated URL spaces too. This problem was solved by making DeleGate provide /robots.txt, a file supplied by each HTTP server to control robots, which includes "Disable: /-_-" to declare any URLs under /-_- must not be traversed by robots.

3 Connection Cache – Connection as a Reusable Resource

In addition to ordinary caching of data, DeleGate also has another special type of cache named "connection cache". Data cache is effective mainly in reducing the long transfer time for large amounts of data. Connection cache reduces the time in getting small amounts of data from distant servers, where the time in establishing a connection may take up much of the total communication time.

It is known that the hit rate is not so high in WWW caches, at most 50% in several literatures[6][7]. This result is supported by observations on a heavily used public HTTP cache server (with about 400,000 HTTP requests per day) maintained by the author. When missed in cache, requested resources must be loaded from origin servers. Therefore it's important to shorten the turn around time necessary to load resources from origin servers to a proxy server.

In a situation where a server is a long way from a client, namely where the round trip time (RTT) between them is long, the delay in establishing a TCP

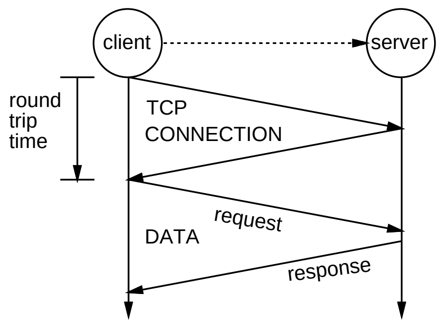


Fig. 9. Latency of a TCP based application protocol

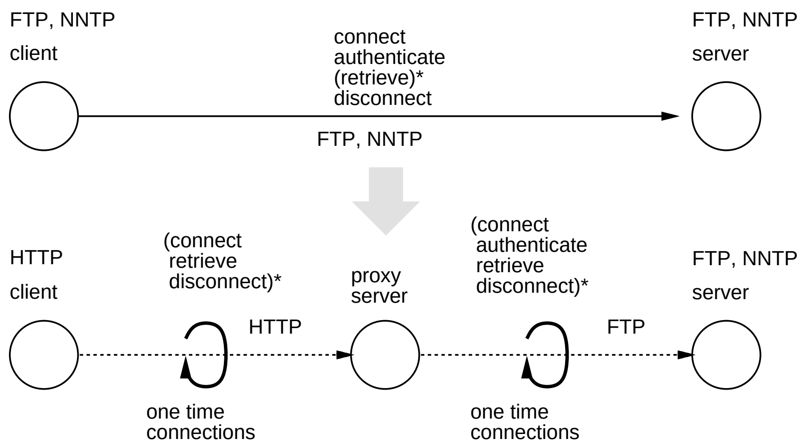


Fig. 10. Overheads introduced by protocol gateways

connection becomes serious[15]. The total time necessary for communication is the sum of the time for establishing connection and that of data transfer. When only a small amount of data is transferred for each established connection, the total communication time becomes approximately twice that of RTT at worst (Fig.9).

This problem is introduced by a proxy server which relays request from clients of a protocol with non-persistent connection like HTTP to the servers of protocols based on persistent connection like NNTP or FTP (Fig.10). Because the connection between the client and the proxy is established and closed for each HTTP request/response, naively implemented proxies may establish and close the connection for the server, too.

This not only makes the response time worse for the client's side, but it also

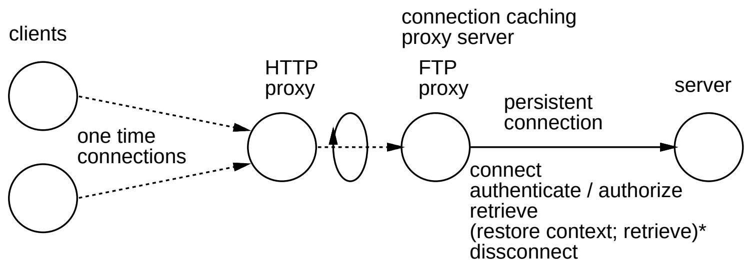


Fig. 11. Connection reuse and sharing with connection cache

causes another serious problem on the server's side; the waste load caused by repeating invocation and initialization of servers.

The concept of "connection cache" for DeleGate was introduced to solve this problem. Like cached data shared by a number of clients, once established, the connection to a server is held by DeleGate for a certain period, and re-used by subsequent clients which access the same server (Fig.11).

Connection cache introduces a merit that is not available where clients and servers are connected directly. Connection to the distant server, which could be a resource limited and/or expensive time-wise, is now sharable and re-usable among different users. This mechanism can be extended to allow *prefetching* too.

To solve the latency problem within the HTTP protocol, a protocol to keep an established connection alive through multiple requests/responses has been proposed and adopted as a standard mechanism of HTTP protocol version 1.1 [16]. The current version of DeleGate also supports this mechanism, but this mechanism does not provide for the re-use of a connection among multiple users.

3.1 The effect of FTP connection cache

Connection cache significantly improves the response to HTTP users browsing around directories in an FTP server. Without this mechanism, before each data transfer, a connection should be established, and several commands and responses must be exchanged between the FTP server and the proxy for the login procedure. With the mechanism, the response time for moving to and browsing another directory is decreased by a factor of two to four, for example, from several seconds to just a second when accessing a distant server. The effect of connection cache for FTP measured about turn around time for directory retrieval is shown in Table.1 and Fig.12. Each proxy issues a series of FTP commands[14] for a directory retrieval as shown with "(...)*" in Table.1. The result shows complex commands for repetitive login and for getting richer information make the turn around time worse significantly under long round trip time.

Table 1. Measured turn around times for local FTP directory retrieval via HTTP proxies

proxy server / version	turn around time (seconds)
CERN-HTTPD / 3.0A	1.25
Squid / 1.1.6	0.41
Netscape-Proxy / 2.5	0.18
Apache / 1.2b6	0.23
DeleGate / 4.0.7	0.24
DeleGate / 4.0.7 (nocc)	0.29

HTTP Request:
  GET ftp://localhost/ HTTP/1.0
  Pragma: no-cache

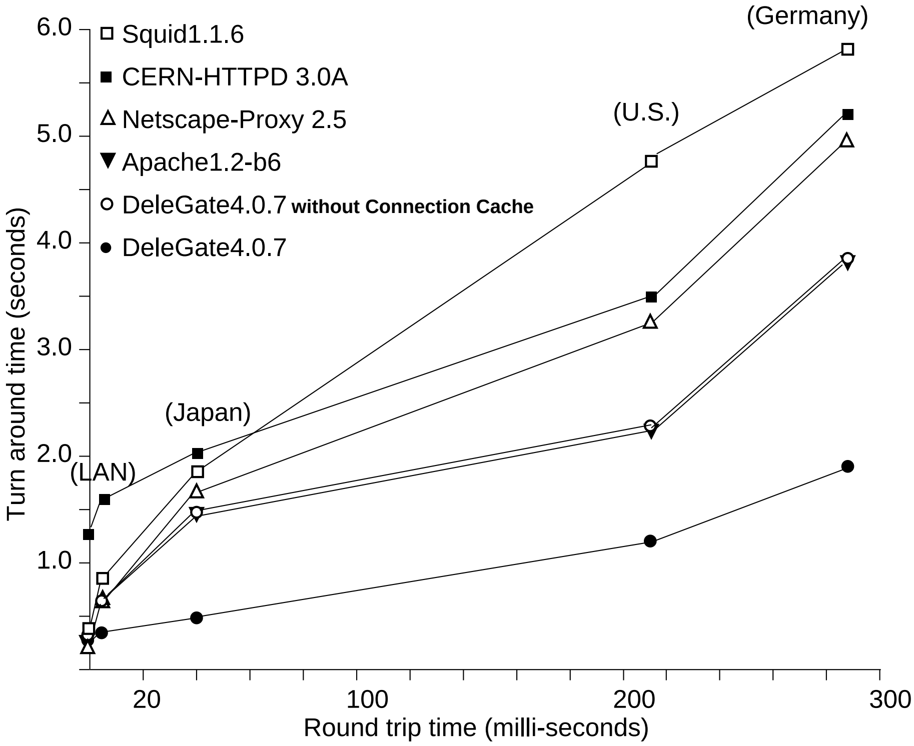
Platform:
  SunOS5.5 + SparcStation20/70x2 (2CPU)

Commands Issued:
CERN-HTTPD: (USER,PASS,PASV,SYST,LIST,QUIT)*
Netscape-Proxy: (USER,PASS,SYST,PASV,TYPE,SIZE,CWD,LIST)*
Squid: (USER,PASS,TYPE,MDTM,SIZE,PASV,REST,RETR,
  PASV,RETR,PASV,LIST,QUIT)*
Apache: (USER,PASS,SYST,PWD,PASV,TYPE,SIZE,CWD,LIST)
  + (USER,PASS,TYPE,PASV,LIST,QUIT)*
DeleGate: (MODE,USER,PASS,TYPE,SYST) + (CWD,PWD,PASV,LIST,NOOP)*
DeleGate(nocc:without-connection-cache):
  (USER,PASS,TYPE,SYST,PWD,PASV,LIST,QUIT)*

3.2 Implementation

A proxy process which established a connection to a destination server to relay a request bound for it becomes a connection caching proxy after it finished the request. For later proxy processes, it provides an entrance of a tunnel in the local host which is connected to the remote destination server. A later proxy which is going to connect to a server first looks up such a port bound for the server. Those ports are registered in a common directory in the current implementation. If such a port is found, it connects to the port instead of connecting to the destination server. The routing mechanism of DeleGate which controls the sequence of trials to connect the server is extended naturally to support the connection cache, adding it as the first alternative route to be tried.

In the current implementation, one persistent connection is held by one server process. For more efficient relaying and less consumption of processes, *file descriptor passing* technique should be adopted in future, holding all established connections in a single connection cache server process.



HTTP Request:
GET ftp://*ftp-server*/ HTTP/1.0
Pragma: no-cache

Fig. 12. Measured turn around times for remote FTP directory access

3.3 Time sharing of connection cache

Connection cache technique is applicable to clients of protocols which depend on a persistent connection to the server during a session. Since such a client expects a connection to a server is held until it is closed explicitly, the connection from such a client must be retained even when the client is idle for long time. It is indeed a waste of connections as limited and/or expensive resources.

This problem is solved by introducing a client side connection cache proxy (Fig.13). A DeleGate process at client side retains a connection from the client, while closing the connection to the server and become suspended when there is no request from the client for a specified period. When such a suspended DeleGate receive a request from the client, it resumes re-connecting to the server and restoring its context (e.g. the current newsgroup and article in NNTP protocol).

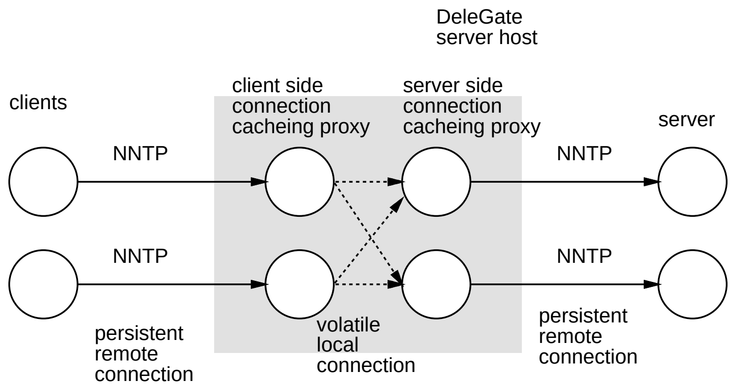


Fig. 13. Time sharing of connection cache

4 Current Status

DeleGate is written in the C language and the current version (4.0.7) consists of about 67,000 lines of source code. It runs over various platforms and has been distributed and used widely.

4.1 Platforms

With very active feedback/assistance from users belonging to the DeleGate mailing list (*delegate@etl.go.jp* with over 950 members currently), the machine dependence in the program has been almost eliminated. Thus it can be compiled and run 'as is' on many UNIX platforms, including Solaris, SunOS, DEC-OSF1, IBM-AIX, HP-UX, IRIS, NeXTSTEP, Linux, FreeBSD, BSD/OS, BOW(BSD on Windows) and so on.

The porting of it to WindowsNT and Windows95 is nearly completed now and it has been provided for practical use. To absorb the architectural differences between UNIX and Windows, mainly in the process generation model and file system functionality, about 1,300 lines of codes were added to cope with the Windows system.

4.2 Distribution

Since June, 1994, DeleGate has been distributed from our anonymous FTP server at: (URL:ftp://ftp.etl.go.jp/pub/DeleGate/). So far, the DeleGate distribution file has been downloaded 49,000 times from more than 2,200 sites (domains), of which 1,500 sites are from Japan, and 700 are from 48 other countries. It has been also distributed in several CD-ROM collections of free software, as supplements bundled with magazines and other publications.

A recent observation on our WWW server (URL:http://www.etl.go.jp/), which is accessed from about 3,000 sites every week (of which about one third is from overseas) shows that about one third of users are using some HTTP proxies, and about 29% of proxy users have adopted DeleGate. The approximate shares for other proxies are: 43% for CERN-HTTPD, 29% for Squid, and 16% for Netscape-Proxy (The sum of the shares exceeds 100% because some sites are using several proxies).

4.3 History

The development of DeleGate started in March, 1994. About that time Gopher appeared as a new application for retrieving networked information on the Internet. The World Wide Web then followed, and very quickly almost totally overwhelmed the Internet.

These applications, enthusiastically accepted by the Internet community, and sometimes dubbed as "killer applications", revealed weaknesses of the Internet and problems brought on by the very wide interconnectivity of the Internet. Typical and serious requirements for such applications were: crossing firewalls transparently without breaking the security, making do with the narrow band width of the networks, and coping with diverse character codings of text data to be exchanged among different platforms. To cope with the firewall and narrow band width problem, a caching proxy feature was added to the new version of CERN-HTTPD, released in the spring of 1994. CERN-HTTPD was the first and the most widely used WWW server at the time.

At almost the same time, independently of such developments in HTTP, in March of 1994, the author started creating a caching proxy for Gopher. This proxy, named DeleGate, was immediately extended to support NNTP, Telnet, FTP, HTTP and so on, and it was then released to the public in June, 1994. In the beginning, its usage spread as a proxy server which converts three types of character codes in Japanese documents. Such conversion was indispensable in displaying diversely encoded Japanese texts with the incompletely Japanized WWW browsers of the time. They were mostly based on NCSA Mosaic, which could deal with only one of the codings. This problem was resolved as browsers were improved to support all Japanese codings properly, especially by Netscape Navigator, which appeared in the spring of 1995.

Having fulfilled this initial role as a "character code conversion proxy", the development of DeleGate has continued and begun to be utilized as the author originally envisioned, that is, as a general purpose caching proxy which can relay various protocols solving a variety of problems.

5 Conclusion

DeleGate has been developed to be a general purpose application gateway which is applicable to a variety of situations and application protocols, and be adaptable for a variety of platforms. After three years of development, DeleGate has gained wide acceptance among the Internet users and is being used daily by many users for many different practical purposes. While the implementation of DeleGate is not yet perfect, it has satisfied requirements in practical usage, solving a variety of problems appeared one after another which have provided many useful hints for further improvement. The author is not yet sure where is the goal of the application level gateway, but he is sure that there are more horizons of proxies not explored yet.

References

1. Chapman, D., B. and Zwicky, E., D.: Building Internet Firewalls, O'Reilly & Associates, Inc. (1995)
2. Avolio, F., M. and Ranum, M., J.: A Network Perimeter with Secure External Access, Proceedings of the Internet Society Symposium on Network and Distributed System Security (1994)
3. Luotenen, A. and Altis, K.: World-Wide Web Proxies, The First International WWW Conference (1994)
4. Squid Internet Object Cache, http://squid.nlanr.net/Squid/.
5. Thau, R.: Design considerations for the Apache Server API, The Fifth International WWW Conference (1996).
6. Glassman, S.: A Caching Relay for the World Wide Web, The First International WWW Conferences (1994)
7. Abrams, M., Standridge, C. R., Abdulla, G., Williams, S., and Fox, E. A.: Caching Proxies: Limitations and Potentials, The Fourth International WWW Conference (1995)
8. Berners-Lee, T., Fielding, R., and Frystyk, H.: Hypertext Transfer Protocol – HTTP/1.0, RFC1945 (1996)
9. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1, RFC 2068 (1997)
10. Berners-Lee, T., Masinter, L. and McCahill, M.: Uniform Resource Locators, RFC1738 (1994)
11. Berners-Lee, T. and Connolly, D.: Hyper Text Markup Language – 2.0, RFC1866 (1995)
12. Freed, N. and Borenstein, N.: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, RFC2045 (1996)
13. Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., Jones, L.: SOCKS Protocol Version 5, RFC1928 (1996)
14. Postel, J. and Reynolds, J.: File Transfer Protocol, RFC959 (1985)
15. Padmanabhan, V. N., and Mogul, J. C.: Improving HTTP Latency, The Second International WWW Conference (1994)
16. Nielsen, H. F., et al.: Network Performance Effects of HTTP/1.1, CSS1, and PNG. WWW Technical Report, NOTE-pipelining-970214 (1997)
17. Sato, Y.: Development of a Protocol Mediation System DeleGate, Electrotechnical Laboratory Technical Report, TR-94-17 (1994)
18. Sato, Y.: Features of Multipurpose Proxy Server DeleGate, INTERFACE, Vol.21, No.9 (1995) 130–146
19. Sato, Y.: Fundamental Technologies and Applications of Internet Firewalls – Construction of DeleGate, Computer Software, Vol.14, No.1 (1997) 55–63