Schizophrenic HTTP Server

Alan Barrett <barrett@ee.und.ac.za>

Abstract

Advertisers often commission contractors to operate HTTP servers on their behalf, to provide a presence on the World-Wide Web, and the contractors often wish to use a single host to serve multiple advertisers. This paper discusses methods of hiding that implementation detail from WWW clients, and reasons for wanting to do so. Several techniques are considered, and sample implementations of the techniques are presented.

1 Introduction

The World-Wide Web (WWW) allows advertisers (in the broad sense of entities with information that they wish to publish) to provide information in a form that is readily accessible to Internet users throughout the world. Recently, advertisers in developed countries where Internet connectivity is widespread have come to see establishing a WWW presence as an important way of publishing their information, even for advertisers that did not previously have any Internet connectivity. Operating a WWW server and producing documents in the necessary electronic formats are specialised tasks, and servers that carry popular information may need to be located on high-performance hosts with high-capacity network connections. Because of these considerations, many advertisers choose to contract out their WWW server operations.

Contractors who provide WWW service for several advertisers may wish to reduce resource utilisation by placing servers for more than one advertiser on a single physical host. The desire to keep this implementation detail hidden, and to provide the illusion of multiple independent servers despite the fact that they are all running on the same host, imposes special requirements on the server implementation.

We begin by briefly examining the HTTP protocol used between WWW clients and servers and considering the goal of running multiple near-independent servers on a single host in the light of the constraints imposed by the protocol. Next, we consider implementation issues, both in general terms and related to typical Unix hosts. This is followed by a description of some actual implementations of these ideas. Finally, we discuss the ancillary issue of techniques for assigning multiple IP addresses to a single host.

2 The World-Wide Web and the HTTP protocol

The World-Wide Web can be thought of a set of documents distributed throughout the global Internet, and interconnected by hypertext links. Various protocols allow a client (typically a program under control of a human user) to obtain documents from a server (typically a program running on a distant host). The primary protocol used for transferring hypertext documents is Hypertext Transfer Protocol (HTTP) [1] but other protocols, such as FTP and Gopher, are also sometimes considered part of the WWW. Hypertext documents are written in Hypertext Markup Language (HTML) and can contain references to other hypertext documents as well as to non-hypertext information, such as binary or plain text files, images and sounds.

Hypertext links use references called Uniform Resource Locators (URLs) [2], which contain information about the protocol to be used to obtain the referenced information, the name of the host on which the relevant server resides, and server-dependent information (called an *url-path* by RFC 1738 [2]). For example, when a WWW client wants to fetch the document named by the URL[1] `http://www.advertiser.domain/collection/document.html`, it will open a connection to the server `www.advertiser.domain` using the HTTP protocol, and will ask the server for the document using the *url-path* `collection/document.html`. HTTP URLs can also contain an optional port number, which can be used to differentiate between multiple servers on a single host.

The HTTP protocol is used to send the document name (*url-path*) from the client to the server, and then to send the document itself from the server to the client. The server host name and port number are known by the client, but they are not transmitted from the client to the server [3]; the server is expected to be able to locate the document without needing to be told that information.

[1] URLs tend to be rather long. In this document, it has sometimes been necessary to break lines in the middle of URLs, host names, or file names. Where that has been done, hyphens have *not* been inserted.

3 Multiple near-independent HTTP servers on a single host

An advertiser often wants the contractor to provide the illusion that the advertiser has its own HTTP server, with the server's domain name being associated with the advertiser rather than the contractor. Although this may be ascribed partly to vanity on the part of the advertiser, it does make it easier for the advertiser to switch to a different contractor or to begin running the server itself in the future, without URLs that refer to the advertiser's data needing to be changed when the server is moved.

An HTTP client and server can use the server host name (or IP address), the port number, and the *url-path* to distinguish between different documents. A contractor who uses the same physical host to operate servers for several advertisers would need to ensure that at least one of those portions of the URL can be used to distinguish between different advertisers.

The contractor could choose the technically simple solution of assigning different port numbers for each advertiser. For example, `http://www.contractor.domain:8000/` and `http://www.contractor.domain:8001/` might refer to two servers operated on the same host on behalf of different advertisers. However, this leaves advertisers who are allocated a non-standard port number at a disadvantage, both because it makes it more difficult for humans to guess or remember the required URL, and because it may require a different port number to be assigned at a later date if the advertiser's information is moved to a host where the allocated port number is already in use for a different purpose.

Using the *url-path* to distinguish between advertisers is also technically simple. For example, `http://www.contractor.domain/advertiser1/document.html` and `http://www.contractor.domain/advertiser2/document.html` might refer to information provided on behalf of different advertisers. Here, users who wish to guess the URL would need to know the correct contractor name, and if the advertiser switches to a new contractor then obsolete URLs referring to the old contractor may present a problem.

Using only the host name to distinguish between advertisers is the most desirable solution from the advertisers' point of view. For example, `http://www.advertiser1.domain/document.html` and `http://www.advertiser2.domain/document.html` might refer to information provided on behalf of different advertisers, and the client need not know that the same physical host serves both advertisers. For this to work, the server host needs to know which host name is used in any HTTP request from a client, but that may be difficult because the HTTP protocol does not usually pass the host name from the client to the server. Methods of implementing a server to cope with this problem will be considered below.

Using both the host name and the *url-path* to distinguish between advertisers is also possible. For example, `http://www.advertiser1.domain/advertiser1/document.html` and `http://www.advertiser2.domain/advertiser2/document.html` might refer to information provided on behalf of different advertisers. Any implementation technique that can cope with the case where only the host name, or only the *url-path*, is used to distinguish between advertisers will also be able to cope with the case where both are used.

4 General implementation considerations

We now consider implementation techniques that will allow the host name (or both the host name and the host-specific *url-path*) to be used to distinguish between multiple near-independent servers on a single physical host. This can be divided into techniques that use a single IP address and techniques that use multiple IP addresses.

4.1 Using a single IP address

If a contractor's HTTP server host has only one IP address, then the domain name system (DNS) [4] can be configured to map multiple names to that IP address, using a different name for each advertiser associated with the server host. For example, the DNS could contain information like this[2]:

```
www.contractor.domain.
     A      192.0.2.10
www.advertiser1.domain.
    CNAME  www.contractor.domain.
www.advertiser2.domain.
    CNAME  www.contractor.domain.
10.2.0.192.in-addr.arpa.
    PTR    www.contractor.domain.
```

Here, the advertiser-related domain names are simply aliases for the domain name of the server host operated by the contractor. A client wishing to contact one of the advertisers' HTTP servers would instead contact the contractor's server. In this situation, the server would have no way of knowing which host name the client used, because the host name is not used in either the HTTP protocol or the underlying TCP or IP protocols. The IP address is available to the server, but all

[2] Each pair of lines represents a single DNS RR

advertisers use the same IP address, so that is not useful for differentiating between advertisers. Thus, when the server has only one IP address, the *url-path* would have to be used to differentiate between advertisers who are associated with the same server host, regardless of whether or not multiple host names are also used.

Using both the host name and the *url-path* to differentiate between advertisers seems to satisfy an advertiser's wish for the URL to incorporate their own host name rather the contractor's host name, with all the advantages that that has for the advertiser's vanity and for the ease of moving the advertiser's information to a different host. However, the fact that the URLs `http://www.contractor.domain/advertiser1/document.html` and `http://www.advertiser2.domain/advertiser1/document.html` are alternative names for the document that one might prefer to call `http://www.advertiser1.domain/advertiser1/document.html` has the disadvantage that the non-preferred names may be used inadvertently in references to the documents, and this would make it more difficult for the server to be moved at a future time.

4.2 Using multiple IP addresses

If a contractor's HTTP server host has multiple IP addresses, each IP address could be associated with a different domain name, and thus with a different advertiser. For example, the DNS could contain information like this[3]:

```
www.contractor.domain.
    A       192.0.2.10
www.advertiser1.domain.
    A       192.0.2.11
www.advertiser2.domain.
    A       192.0.2.12
10.2.0.192.in-addr.arpa.
    PTR     www.contractor.domain.
11.2.0.192.in-addr.arpa.
    PTR     www.advertiser1.domain.
12.2.0.192.in-addr.arpa.
    PTR     www.advertiser2.domain.
```

Here, the fact that the different domain names refer to the same physical host is entirely hidden from the clients. A client wishing to contact one of the advertisers' HTTP servers would use the unique address allocated by the contractor to that advertiser, and the server on the contractor's host could use the IP address to determine which advertiser the client wished to contact. In this situation, the advertiser's name does not need to be encoded in the *url-path*, and a single physical host can provide the illusion to the HTTP clients that there are several separate virtual hosts, each running an independent HTTP server.

[3] Each pair of lines represents a single DNS RR

5 Implementation on Unix-like systems with multiple IP addresses

This section considers techniques for implementing the above ideas on a host with multiple IP addresses and a Unix-like operating system with a BSD sockets programming interface.

After a TCP socket has been created, the *bind* system call is used to set the local IP address and TCP port number on which connections will be accepted. The IP address can be specified as a single address or as a reserved value that will match any address. If a single IP address is specified, then the socket will accept connections only on that address, and not on other addresses that are associated with the same physical host. After a socket has accepted a connection from a client, the *getsockname* system call can be used to find the local IP address involved. Either or both of the *bind* and *getsockname* system calls could be used as part of an implementation that wishes to behave differently for connections to different IP addresses.

HTTP servers are typically run in one of two modes: Either as a long-running daemon that continuously listens for and services client connections, or as relatively short-lived servers invoked from a long-running daemon (typically the *inetd* process) to service a single client connection on each invocation. When servers are run from *inetd*, it's fairly common for a wrapper program to be interposed between *inetd* and the server proper, for the purpose of performing additional authorisation or logging.

In addition to the resources actually needed to answer requests from clients, each server process will consume additional resources during initialisation. A long-running daemon is initialised only once, while short-lived servers invoked from *inetd* have to be initialised every time a client connects to the host. Offsetting that advantage, however, a long-running daemon uses some resources while it is idle between connections. A high frequency of client connections and a high complexity of server initialisation tends to indicate that a long-running daemon would be preferrable, while a low frequence of client connections and a low complexity of server initialisation tends to indicate that a short-lived server invoked by *inetd* would be preferable.

Taking the above considerations into account, the following four strategies seem reasonable:

- *inetd* invokes server

  If the server is invoked by *inetd*, then a separate instance of the server will be started for each client connection, regardless of the local IP address involved. The server would be passed a connected socket as its standard input, and could use the *getsockname* system call to find the local server IP address. It would then have to use the IP address to determine which advertiser's information was involved, and adjust its behaviour accordingly.

- *inetd* invokes wrapper that runs server

  If *inetd* invokes a wrapper program that in turn runs the server, then the wrapper could use the *getsockname* system call to determine which local IP address is involved. The wrapper program could invoke the server with different command line arguments depending on which IP address was used. A different set of server configuration parameters or files could be used for each local IP address, and this might be accomplished without the server software itself needing to be modified, provided only that the server software allows command line arguments to be used to control its configuration.

- Server runs as a daemon bound to a single IP address

  If the server is a long-running daemon process, and uses the *bind* system call to restrict the local IP addresses on which it will accept connections, then the server can assume that every connection which it accepts is associated with the same advertiser. The host could run several HTTP servers in this way, with each server bound to a different local address, and with each server configured to handle information associated with a different advertiser.

- Server runs as a daemon accepting connections on any IP address

  If the server is a long-running daemon process, and uses the *bind* system call to specify that it will accept connections on all local IP addresses, then it could use the *getsockname* system call after each connection is accepted, to determine the local IP address to which the connection was made. For each connection, it would then have to use the IP address to determine which advertiser's information was involved.

One of the above strategies, using a wrapper between *inetd* and the server, could be implemented without modifying either the *inetd* program or the server, but might require a specially written wrapper. The other three strategies mentioned above all require modifications to server software if the server was not originally designed for these uses.

6 Sample server implementations

We now look at sample implementations using the techniques identified in the previous section.

6.1 Wrapper between *inetd* and server

A line similar to the following in the `/etc/inetd.conf` file (the exact format is system dependent) will cause *inetd* to invoke the *tcpd* wrapper every time a client connects to the host's HTTP port[4]:

```
http stream tcp nowait root
    /usr/libexec/tcpd httpd
```

Version 7.0 of Wietse Venema's TCP wrapper program [5] can make decisions based on the result of the *getsockname* system call. If *tcpd* is compiled with its extended features enabled, then the following lines in the `/etc/hosts.allow` file can be used to invoke the actual HTTP server in a different way for each advertiser:

```
httpd@www.contractor.domain : ALL : \
    twist "/usr/libexec/httpd \
          -d /data/contractor"
httpd@www.advertiser1.domain : ALL : \
    twist "/usr/libexec/httpd \
          -d /data/advertiser1"
httpd@www.advertiser2.domain : ALL : \
    twist "/usr/libexec/httpd \
          -d /data/advertiser2"
```

The NCSA httpd server [6] uses the *-d* command line option to set the *ServerRoot* directory in which it expects to find other configuration files and the actual data to be made available to clients. If each advertiser has a separate *ServerRoot* directory, containing advertiser-specific configuration files, the host can behave differently depending on which advertiser the client wished to contact.

6.2 Server modified to use bind or getsockname

The NCSA httpd server [6] can be configured to operate as a long-running daemon (*standalone* mode) or as a short-lived server (*inetd* mode),

[4] This should be a single long line

by means of the *ServerType* option in the server configuration file.

Version 1.3 of the NCSA httpd server has been modified by this writer [7] to enable it to bind to a single IP address in standalone mode, and to modify its behaviour according to the result from a *getsockname* system call in either *inetd* mode or standalone mode. The following subsections describe the modifications in more detail.

6.2.1 Binding to a single IP address

The new *BindAddress* command in the server configuration file can be used to make the server use the *bind* system call in standalone mode to bind to a single IP address instead of accepting connections on any IP address. For example, the commands

```
BindAddress 192.0.2.11
```

or

```
BindAddress www.advertiser1.domain
```

could be used. If the address is specified as a domain name rather than as a numeric address, the domain name must map to exactly one IP address.

Using the BindAddress command in this way, a physical host can have several long-running HTTP daemons, each bound to a different IP address and each handling a different advertiser.

6.2.2 Using *getsockname*

The new *VirtualHost* section command in the server configuration file can be used to change the server behaviour according to the result of a *getsockname* system call, in either standalone mode or *inetd* mode. This allows the values of three key variables — the server's domain name, the electronic mail address of the server administrator, and the directory that contains the data to be served to clients — to be adjusted according to the local IP address.

For example, lines like the following in the server configuration file could specify the information associated with IP address 192.0.2.11:

```
<VirtualHost 192.0.2.11>
ServerName www.advertiser1.domain
ServerAdmin web@advertiser1.domain
DocumentRoot /data/advertiser1
</VirtualHost>
```

There can be several VirtualHost sections in the server configuration file, each associated with a different IP address, up to a maximum determined at compile time. A domain name can be used instead of a numeric IP address in the VirtualHost command, but then the domain name must map to exactly one IP address.

Using the VirtualHost feature, a physical host can use a single long-running HTTP daemon or can use *inetd* to invoke a separate short-lived HTTP server for each client request, and the server can modify its behavious according to the result from the *getsockname* system call, thus allowing multiple advertisers to be served.

When the VirtualHost option is used, an extra field containing the virtual server name is added to each record in the log that the server keeps of all client accesses. This is desirable because the other information in the log might not be sufficient to differentiate between similarly named documents associated with different virtual servers (that is, different advertisers).

7 Assigning multiple IP addresses to a host

A contractor who uses the techniques desribed above to run multiple near-independent HTTP servers on a single host may want to assign more IP addresses to the host than there are physical network interfaces. This section describes some techniques for assigning additional IP addresses to a host.

On operating systems derived from BSD Net2 or BSD 4.4, the *alias* option can be used with the *ifconfig* command to assign multiple IP addresses to a single interface. For example, the following commands assign a primary address and an alias to interface *ed0*:

```
ifconfig ed0 inet 192.0.2.10 \
    netmask 255.255.255.0
ifconfig ed0 inet alias 192.0.2.11 \
    netmask 255.255.255.0
```

On Solaris 2.3, the *ifconfig* command has an undocumented feature that allows an interface name to be followed by a colon and a number to assign additional addresses to the interface. For example, the following commands assign a primary address and an additional address to interface *le0*:

```
ifconfig le0 inet 192.0.2.10 \
    netmask 255.255.255.0
ifconfig le0:1 inet alias 192.0.2.11 \
    netmask 255.255.255.0
```

On most Unix-like systems, it should be possible to use the *ifconfig* command to assign addresses to any unused interfaces (such as interfaces associated with unused SLIP or PPP links). Many systems also have an *arp* command that can make the host respond to ARP requests for the additional addresses. In the absence of a way of making the host respond to ARP requests for the extra addresses, nearby routers might need to be specially configured to cope with the host's additional addresses.

On SunOS 4.1.3, SunOS 4.1.4 and HP-UX 9.05, and perhaps other systems, additional *vif* interfaces [8] can be added to the kernel, using code originally written by John Ioannidis. The *vif* interfaces could then be assigned IP addresses using the *ifconfig* command, and (if possible) the host could be made to respond to ARP requests for those addresses. For example, the following commands assign a primary address to the *le0* interface, assign an additional address to the *vif0* interface, and establishes an ARP table entry that associates the additional IP address with the host's ethernet address:

```
ifconfig le0 inet 192.0.2.10 \
    netmask 255.255.255.0
ifconfig vif0 inet 192.0.2.11
arp -s 192.0.2.11 0:80:3f:f5:b:b9 pub
```

8 Conclusion

It is possible, using multiple IP addresses on a single host, to allow a host to run several near-independent HTTP servers. This type of configuration is particularly interesting to contractors who operate servers on behalf of several advertisers, and is reasonably simple to implement. At the time of writing, the author believes that some tens of contractors use NCSA httpd with the VirtualHost modifications described here, and at least one of these supports approximately fifty virtual hosts on a single physical host.

References

[1] T. Berners-Lee, "Hypertext transfer protocol (HTTP)," Nov. 1993. `<URL:ftp://info.cern.ch/pub/www/doc/http-spec.txt.Z>`.

[2] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform resource locators (URL)," RFC 1738, Dec. 1994. `<URL:ftp://ds.internic.net/rfc/rfc1738.txt>`.

[3] T. Berners-Lee, "Uniform resource identifiers in WWW: A uniform syntax for the expression of names and addresses of objects on the network as used in the World-Wide Web," RFC 1630, June 1994. `<URL:ftp://ds.internic.net/rfc/rfc1630.txt>`.

[4] P. Mockapetris, "Domain names – concepts and facilities," RFC 1034, Nov. 1987. `<URL:ftp://ds.internic.net/rfc/rfc1034.txt>`.

[5] W. Venema, "TCP/IP daemon wrapper package," Jan. 1995. `<URL:ftp://ftp.win.tue.nl/pub/security/tcp_wrappers_7.0.tar.gz>`.

[6] NCSA, "httpd." `<URL:ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/>`.

[7] A. P. Barrett, "Modifications to NCSA httpd 1.3 for multiple servers on a multi-homed host," Nov. 1994. `<URL:ftp://ftp.ee.und.ac.za/pub/archiving/httpd-virtual-host.shar>`.

[8] J. Ioannidis, C. Smoko, and S. Haug, "vif-1.0 – Multiple IP addresses on the same interface," Jan. 1995. `<URL:ftp://ugle.unit.no/pub/unix/network/vif-1.0.tar.gz>`.

Author Information

Alan Barrett is a member of the teaching and research staff in the Department of Electronic Engineering at the University of Natal, Durban, South Africa, where he received the BScEng and MScEng degrees in 1985 and 1988 respectively. He is also a Director of Internet Africa, which is an Internet Service Provider based in South Africa.