Fast Web browsing with a caching proxy

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

Section 1. About this tutorial

Should I take this tutorial?

This tutorial will show you how to compile, install, and configure oops, an open source, high-performance, multi-threaded Web proxy under Linux. Caching Web proxies are especially useful for accelerating Web browsing performance while at the same time conserving your network bandwidth. If this sounds like something you'd like to do, then read on. This tutorial is for you!

IBM offers a commercial caching proxy server. The IBM WebSphere Edge Server provides *reverse proxy caching*, which improves Web server response time, and *forward proxy caching*, which reduces network bandwidth requirements. To learn more, visit the WebSphere Edge Server home page (http://www-4.ibm.com/software/webservers/edgeserver/).

About the author

For technical questions about the content of this tutorial, contact the author, Daniel Robbins, at *drobbins@gentoo.org* .

Residing in Albuquerque, New Mexico, Daniel Robbins is the President/CEO of *Gentoo Technologies, Inc.* , the creator of **Gentoo Linux**, an advanced Linux for the PC, and the **Portage** system, a next-generation ports system for Linux. He has also served as a contributing author for the Macmillan books *Caldera OpenLinux Unleashed*, *SuSE Linux Unleashed*, and *Samba Unleashed*. Daniel has been involved with computers in some fashion since the second grade, when he was first exposed to the Logo programming language as well as a potentially dangerous dose of Pac Man. This probably explains why he has since served as a Lead Graphic Artist at **SONY Electronic Publishing/Psygnosis**. Daniel enjoys spending time with his wife, Mary, and his new baby daughter, Hadassah.

Section 2. Introducing oops

What is a Web proxy server?

Literally, a "proxy" is a person who performs a task on behalf of another, acting as a substitute. As you might guess, a proxy server is a program that performs a particular task on behalf of its client, usually a task that it could have performed itself if necessary. And, in the world of the Web, a Web proxy server is a program that acts as a messenger, accepting HTTP requests from Web browsers like Netscape, Mozilla, Internet Explorer, or Konqueror. Then, the Web proxy fetches the appropriate data from the Web and hands it to the client. By using a Web proxy, a Web client can access the Internet without actually establishing a direct connection with any machines outside the local network; the Web proxy handles all outside Web communication.

What's the point of a Web proxy?

Web proxies have many uses. One of the most common is to provide Internet access to a LAN that rests behind a firewall. The proxy is able to bypass the firewall, establishing Internet connections on behalf of the Web clients inside the LAN. Thus, a LAN can be totally cut off from the Internet, yet the proxy acts as a "bridge" and allows Web browsing to continue to function. In addition, the Web proxy can also require the LAN users to authenticate with itself (normally with a username and password) before agreeing to provide service.

What's the point, continued

Nearly every modern Web browser can be configured to use a Web proxy rather than make direct connections to the Internet. Normally, configuration involves specifying the IP address and port of the proxy server. Then, the Web browser forwards its requests to this particular host rather than attempting to contact the remote site by itself.

What's a caching Web proxy?

In addition to providing basic HTTP proxying functionality, many Web proxies also maintain a local cache of recently-accessed Web pages and images. When a page or image is requested that's already in the proxy's cache, the proxy hands the cached copy back to the client rather than downloading it directly from the Web site again. This does two things: first, it generally speeds up Web page loads for your LAN. Secondly, it can reduce the bandwidth used for Web browsing by reducing redundant page and image downloads.

What's a caching Web proxy, continued

While most Web browsers have their own built-in cache, it's typically quite small -- normally not much bigger than 10 MB. In comparison, caching Web proxies allow for much larger disk-based caches -- typically several hundred MB in size, and often in the multi-GB range. And unlike a simple browser proxy, a caching Web proxy will accelerate Web performance and reduce network bandwidth for an entire group of users; this is something that the relatively wimpy browser cache can't do.

What's a transparent proxy?

OK, we've covered Web proxies and caching Web proxies, but what is a transparent Web proxy? Quite simply, it's a Web proxy configured in a special way so that the Web clients (browsers) *can't even tell* that their HTTP requests are traveling through a proxy. As far as they're concerned, they think that they're communicating with the actual remote site directly. Transparent proxies are especially cool because when they're set up correctly, proxying will "just work", with no Web browser configuration required. Not only that, but transparent Web proxies are much harder to bypass when set up correctly.

What is oops?

Oops is a very high quality, speedy, and open source caching proxy server, available at *http://zipper.paco.net/~igor/oops.eng/* . In this tutorial, I'll show you how to compile, install, and configure oops under Linux.

Currently, the most popular open source, high-performance Web proxy cache is called Squid, and is available from *http://www.squid-cache.org* . Squid is an adequate Web cache, but I personally prefer the lesser-known oops,

---

Advantages of oops

I like oops because it's open source and very well designed. oops is multi-threaded, which improves performance on single-processor machines and allows for dramatic performance increases on multi-processor systems. In addition, oops performs time-consuming tasks (such as cache integrity checks) in the background so that they don't interfere with its Web proxying abilities, which is something that Squid doesn't do.

Oops also supports a number of advanced features including bandwidth control, connection limiting, regular expression-based redirection, and other goodies. In this tutorial, I'll familiarize you with only those features that are required to get oops to run in transparent mode; after oops is up and running, you can begin exploring its more advanced features on your own.

Our plan

For this tutorial, our goal is to set up oops under Linux, and then optionally configure transparent mode using iptables. Transparent configuration is especially useful if your Linux box is providing Internet access for your LAN (acting as a gateway), because then all Web browsers on your LAN will automatically use oops as a proxy. All that's required is that your LAN machines are configured to use the Linux machine as the default gateway. If this isn't how your LAN is configured, you can still have all your LAN clients use oops, but you may need to configure each Web browser to use the proxy server running at your Linux box on port 3128. Either that, or use an alternate method that I'll point out later.

Section 3. Downloading and installing oops

Downloading sources

To install oops, head over to *http://zipper.paco.net/~igor/oops.eng/* , click on the download link, and download the most recent stable version of oops available (currently *the oops-1.5.6.tar.gz file* .) Once downloaded, extract the tarball by typing:

```
# tar xzvf oops-1.5.6.tar.gz
```

Security considerations

Whenever you're running a network service, it's always a good idea to configure it so that it doesn't run as "root" if at all possible. When a network service is running as a non-root user and a security hole is exploited, it's likely that the reduced privileges of the non-root account will prevent the security hole from becoming a major threat.

Of course, oops is a very well designed, security-conscious program; however, the creator of oops decided to add this feature as an additional security measure. Here's how it works: oops starts with root permissions, but then "drops" to a non-privileged account almost immediately.

Security considerations, continued

On my system, I have a special "squid" (non-login) user and group that oops uses. Oops runs under this UID, providing an additional layer of security. In addition, I have the oops /var/log and /var/run directories set up with special permissions so that only root and the members of the "squid" group can read the files within. We'll take a look at how to do this in a bit, but for now, you'll need to decide what UID you'd like oops to use.

Oops should use a UID of a non-login account; several good choices are "daemon", "squid" or "oops". If necessary, create a matching group with the same name (making sure that the UID's default group points to this group), and you're ready to roll. From this point forward, you should replace any references I make to the "squid" user/group with the name of the group that you've chosen to use. I'll highlight squid in red as a reminder.

Compiling oops

OK; time to get this thing compiled. First, enter the oops source directory:

```
# cd oops-1.5.6
```

Then, run the configure script with the following options; "\" is a line continuation character and can be omitted if you enter all the configuration options on one line:

```
# ./configure --prefix=/usr --libdir=/usr/lib/oops --enable-oops-user=squid \
--sysconfdir=/etc/oops --sbindir=/usr/sbin --localstatedir=/var/run/oops
```

After configure completes, type:

```
# make
```

The source will start compiling.

A possible patch

When I compiled oops on my system, I had to make a small patch to the sources to get them to compile correctly. If you get a compile error related to the strerror_r() function, perform this simple patch:

```
# cd src
# cp config.h.in config.h.in.orig
# sed -e '/STRERROR_R/d' config.h.in.orig &gt; config.h.in
```

After patching, restart compilation by typing:

```
# cd ..
# make
```

Installing oops, part 1

Now, we're ready to install oops. This part is a bit tricky, since the Makefile is geared for Solaris rather than Linux systems. Go ahead and type:

```
# make install
```

The "make install" process will get most things set up correctly, but now, a bit of additional tweaking is required. Here's what you need to do:

```
# rm -rf /usr/oops
# rm -rf /usr/lib/oops/modules
```

This will remove two extraneous directories created by "make install".

Installing oops, part 2

Now, we need to properly configure the /etc/oops, /var/log/oops, /var/lib/oops, and /var/run/oops directories. Only root, squid, and members of the squid group will be able to read the contents of /etc/oops, /var/log/oops, and /var/lib/oops; and only root and the squid user will be able to write into these directories:

```
# chmod o-rwx g=wxs /etc/oops
# chown -R squid.squid /etc/oops
# install -d -o squid -g squid -m0750 /var/log/oops
# chmod g+s /var/log/oops
# install -d -o squid -g squid -m0750 /var/lib/oops/storage
# install -d -o squid -g squid -m0750 /var/lib/oops/db
# install -d -o squid -g squid -m0755 /var/run/oops
# chmod g+s /var/run/oops
```

Section 4. Configuring oops

Basic oops configuration

Now that all the files and directories are in place, there's just one more step to complete before oops can start: the oops.cfg configuration file needs to be customized for your site. oops.cfg can be confusing to new users, so I've created *a minimal oops.cfg file* that you should copy to /etc/oops. But first, back up the original oops.cfg as oops.cfg.eg. The original contains many useful comments on oops' more advanced features, so you should keep it around for future reference.

---

nameserver, port, and userid

Now, I'll step you through my minimal oops.cfg file, and I'll explain how things work and what you'll need to change to get oops working under your environment. Let's take a look at the first three lines of oops.cfg:

```
nameserver                  192.168.1.1
http_port          3128
userid                           squid
```

On the first line, we specify a name server that oops will use to perform DNS lookups. If you need to, you can add multiple nameserver lines as necessary; oops will query each nameserver in a round-robin fashion.

Nameserver notes

It's important to specify nameservers in your oops.cfg, since doing so will allow oops to perform its own multi-threaded direct DNS queries. If you leave the nameserver parameters out, oops will fall back to using the gethostbyname_r() function to perform DNS lookups. This is a less-than-ideal solution, because gethostbyname_r() will only process one DNS lookup at a time, creating a bottleneck for highly multi-threaded programs like oops. The only downside to specifying nameservers in oops.cfg is that oops won't consult your /etc/hosts file during the name resolution process, instead performing a simple DNS query. Most of the time, this solution will work perfectly.

---

port and userid

On the second line, we set the port that oops will listen on: 3128 is standard for Web proxies. And, on the third line, we specify the user id that oops will use. Modify this line as necessary.

The logs

Next, we get to the log lines:

```
logfile                 /var/log/oops/oops.log { 3 1m }
accesslog          /var/log/oops/oops.access { 3 1m }
statistics          /var/log/oops/oops.stats
pidfile                 /var/run/oops/oops.pid
```

These lines shouldn't require any tweaking, but it's important to understand what they do. The first line configures oops to store its general log at /var/log/oops/oops.log, and to auto-rotate the log when it grows larger than one Megabyte ("1m"). Oops will use a 3-log rotation scheme.

The logs, continued

The next line specifies the location of the access log, which will log all HTTP requests; this file is configured to auto-rotate in the same manner as oops.log. Next, we specify a location for oops.stats. Unlike the other logfiles, oops.stats isn't periodically overwritten by oops (rather than having text appended to it like a normal log), so no auto-rotation is necessary. oops.stats contains a number of useful statistics, such as uptime, number of HTTP requests made since oops started, etc.

The pidfile

Finally, we specify a location for the pidfile, which will contain the numeric process id of the main oops process. In keeping with *FHS 2.1* (http://www.pathname.com/fhs/), we place this file in /var/run/oops.

Memory cache tuning parameters

Next, we have several cache tuning parameters:

```
mem_max                 64m
lo_mark                 32m
```

The first two parameters, mem_mark and lo_mark, tune the behavior of oops' in-memory cache. mem-mark sets a hard maximum limit of the size of the in-memory cache; if oops' in-memory cache grows beyond 64 MB in size, oops will start throwing objects away. Obviously, one should try to avoid ever triggering this behavior, and that's where lo_mark comes in. When oops' in-memory cache hits the lo_mark (in this case, 32 MB), oops will start moving in-memory objects to the disk cache.

Disk cache tuning parameters

Next, we have two disk cache tuning parameters:\

```
disk-low-free         3
disk-ok-free         5
```

disk-low-free tells oops to begin cleaning up the on-disk cache when it becomes more than 97% full, and disk-ok-free tells oops to continue cleaning until at least 5% of the total cache is made available for new objects. We haven't actually configured the size and location on oops' on-disk cache yet, but this will come a bit later in the file.

Miscellaneous settings

Next, we set a bunch of miscellaneous settings:

```
force_http11
force_completion         85
maxresident          1m
insert_x_forwarded_for          no
insert_via          no
always_check_freshness
```

"force_http11" tells oops to use HTTP/1.1 when connecting to Internet sites, even if the Web client uses HTTP/1.0. "force_completion" tells oops to continue downloading an object even if the user aborts the download, as long as at least 85% of the file has already been received. The idea behind this setting is to allow oops to expend a little additional bandwidth to add the object to its cache, since it might be requested at some point in the future.

Miscellaneous settings, continued

The "maxresident" sets a hard upper limit on the size of objects that oops will cache. In this case, oops will not cache a file that's bigger than 1 MB. The purpose of this parameter is to prevent larger objects from filling up oops' caches, thus forcing oops to throw away smaller objects that will likely be requested much more often than the larger files.

Then we turn the "insert_x_forwarded_for" and "insert_via" HTTP headers off. Normally, oops inserts two purely informational headers into every HTTP request. Since they're not needed, I turn them off -- saving a tiny bit of bandwidth and also making oops less "chatty". Finally, the "always_check_freshness" option instructs oops that whenever a cached page is requested, it should do a quick check to see whether this page is up-to-date before handing it back to the client. Without this option, oops will make certain assumptions about page freshness that may

not be correct, resulting in people getting old copies of certain sites. "always_check_freshness" gives us the Squid-like behavior that we want.

Limiting cacheable items

Next, we use the acl, stop_cache and stop_cache_acl parameters to avoid caching things that we shouldn't:

```
acl MYSITE          urlregex (www\.gentoo\.org|cvs\.gentoo\.org)
stop_cache_acl                  MYSITE
stop_cache          ?
stop_cache          cgi-bin
```

First, we use the acl parameter to create an new ACL (access control list) called "MYSITE". Think of an oops ACL as a pattern that we can refer to later in the file, allowing us to specify that some action should or should not be performed if the ACL happens to match. While oops uses ACLs quite extensively, this is the only one that appears in our sample configuration file. This particular ACL will match any URI containing "www.gentoo.org" and "cvs.gentoo.org".

Limiting cacheable items, continued

On the next line, we use the stop_cache_acl parameter to let oops know that any requests with a URI matching the MYSITE ACL should *not* be cached. This will effectively prevent any objects downloaded from www.gentoo.org and cvs.gentoo.org to be cached at all. Now, why would I want to do this? Well, oops happens to run on my server/gateway called "cvs.gentoo.org" and "www.gentoo.org". This machine also hosts the *http://www.gentoo.org* Web site using Apache.

Since oops and Apache are on the same machine, it would be a bit silly for oops to cache data from my site. Doing so wouldn't speed up Web browser requests and wouldn't reduce network bandwidth. Rather, it would just eat up valuable memory and disk-cache storage. You should consider using the acl and stop_cache_acl parameters to mask out sites that are hosted on the oops machine, or are reachable by oops over the LAN. There's no point in wasting valuable oops resources in these cases.

Next, we use two stop_cache parameters to tell oops to never cache any requests with a "?" or "cgi-bin" in the URI; this is a quick and easy way to avoid caching CGI data.

Groups

This next part of the configuration file is very important, and defines a group of hosts on our network that are allowed to use oops. Let's take a look:

```
group gentoo {

        networks         192.168.1/24 127.0.0/24;
        redir_mods         transparent;
        badports [0:79],110,138,139,513,[6000:6010] ;
        miss         allow;

        http {
                allow dstdomain * ;
        }
}
```

Groups, part 2

This particular group is called "gentoo" (feel free to change the name to something else), and includes all those hosts specified in the "networks" parameter -- everything with an IP address starting with 192.168.1 (my local LAN) as well as everything starting with 127.0.0 (localhost). If a machine has an IP address that is part of a network listed in the "networks" parameter, then it's considered part of the "gentoo" group. Of course, you'll need to make sure that the "networks" parameter includes the networks that should be allowed to use oops.

Groups, part 3

Next, the redir_mods line enables transparent caching functionality, and the badports parameter tells oops to reject requests that connect to certain remote ports (in this case, ports that are used for other network services and are very unlikely to serve any Web content). badports is followed with the "miss" parameter, which is very important, since it tells oops that if a machine in this particular group requests an object that doesn't happen to be in its memory or disk cache, then the object should be downloaded. Without this parameter, oops will only be able to serve already-cached objects to this group, which is almost certainly not what you want. And finally, the http parameter allows this group to request data from any ("*") remote host.

"world" group and storage

OK, here's the last part of the configuration file that we'll take a look at; the rest of it doesn't need any manual configuration, so we'll skip it:

```
group world {
        networks 0/0;
        badports [0:79],110,138,139,513,[6000:6010];
        http {
                deny dstdomain * ;
        }
}
```

```
storage {
        path /var/lib/oops/storage/oops_storage ;
        size 400m ;
}
```

The "world" group

As you can see, we create another group called "world" that matches every possible network address (0/0). This means that if a machine doesn't fall into the "gentoo" group, they'll be considered part of the "world" group and HTTP access will be denied ("deny dstdomain *") by default.

Cache storage

Finally, we define the location and size (400 MB) of the on-disk cache; you may want to reduce or increase the size of the on-disk cache according to your needs and your available storage. Also note that /var/lib/oops/storage/oops_storage will be created as a sparse file. This means that initially, the file will hardly take up any space on disk, but will grow as needed up to a maximum size specified here. OK, once you've finished, save and close the oops.cfg file, and we're finally done configuring oops!

Section 5. Running oops

Building storage

Now that oops is configured, we can instruct oops to create its on-disk cache. You only need to do this once. Type:

```
# oops -z -c /etc/oops/oops.cfg
```

The on-disk cache should be ready for use. At this point, you can start oops by typing:

```
# oopsctl start
```

Now, oops should be running, and you should be able to configure a Web browser on your LAN to use port 3128 of your Linux box as a Web proxy. Play around with oops for a bit and you should notice a nice speed improvement when viewing sites a second time.

Transparent mode

Now that oops is working as a normal proxy, you may want to get oops working in transparent mode. To do this, you'll need to meet several requirements:

1. Your Linux box must be the gateway for your LAN
2. Your Linux box must be using a 2.4+ kernel with netfilter enabled
3. You must have iptables installed

Note: if you need help meeting these requirements, you may want to take a look at *Rusty Russell's "unreliable" netfilter guides* or my *tutorial on Linux 2.4 stateful firewall design* , in which I take a closer look at getting netfilter up and running.

The iptables command

If you meet these requirements, then you can enable transparent mode by typing in the following command:

```
# iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 -j REDIRECT --to-port 3
```

This command will instruct all connections passing through your Linux gateway with a destination port of 80 to be automatically redirected to port 3128 of your Linux gateway, where oops will intercept and process them. Now, all the browsers on your LAN should start automatically using the proxy without any additional configuration. Note that if a browser is configured to use the oops proxy, it will still work.

Advanced routing setup

If your Linux box isn't your LAN's gateway, all is not lost. It's still possible to configure your network so that your Linux box can act as a transparent proxy. Fortunately, there is an excellent alternate method for doing this that's clearly described in the *Cookbook section* of the *Linux Advanced Routing HOWTO* .

Section 6. Resources and feedback

Resources

I hope you're enjoying your new oops high-performance multi-threaded caching Web proxy. Here are related resources that you may find helpful:

* Learn more about oops at *the official oops site* (http://zipper.paco.net/~igor/oops.eng/).
* Help yourself to my minimal *oops.cfg file* used in this tutorial.
* *The Linux Advanced Routing HOWTO* (http://www.linuxdoc.org/HOWTO/Adv-Routing-HOWTO.html) is an excellent resource for learning how to do neat things with the new 2.4 kernel netfilter functionality.
* Rusty Russell is the primary guy behind netfilter, and *his "unreliable" netfilter guides* (http://netfilter.samba.org/unreliable-guides/) are essential reading for those wanting to get the most out of netfilter.
* My *Linux 2.4 stateful firewall design tutorial* will show you how to use netfilter to create a powerful stateful firewall. Visit the *developerWorks* Linux zone (http://www-106.ibm.com/developerworks/linux/) for this and many more Linux tutorials, tools, and articles.
* Learn more about the "right" filesystem locations for everything by reading the *Linux FHS* (Filesystem Hierarchy Standard) (http://www.pathname.com/fhs/).
* Another open source caching proxy is the *Squid Web proxy* (http://www.squid-cache.org).
* A commercial caching proxy is the *IBM WebSphere Edge Server* (http://www-4.ibm.com/software/webservers/edgeserver/index.html). The Edge Server provides *reverse proxy caching*, which improves Web server response time, and *forward proxy caching*, which reduces network bandwidth requirements.

Your feedback

We look forward to getting your feedback on this tutorial. For technical questions about the content of this tutorial, you are welcome to contact the author, Daniel Robbins, at *drobbins@gentoo.org* .

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The Toot-O-Matic tool is a short Java program that uses XSLT stylesheets to convert the XML source into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML.