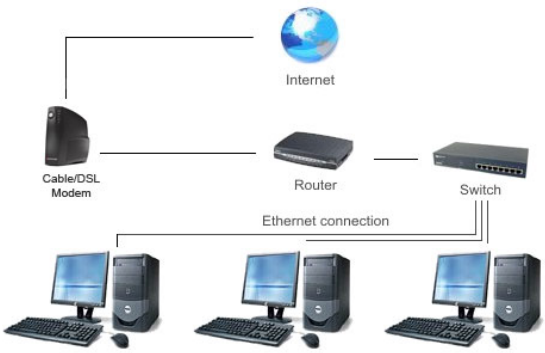
ANANDTECH
Your Source For Hardware Analysis and News

The Family Proxy

by **Christopher Rice** on 5/11/2010 2:45:00 AM
Posted in **Linux** , **Proxy**

Do you have a growing family at home slowly eating away at your bandwidth? Maybe you're a web surfing fanatic looking for a little more speed? If you answered yes to either, a caching proxy is for you. This simple addition to your home network can provide you with additional bandwidth by reducing common internet bandwidth usage. Normally these types of proxies are found in the commercial world, but they're just as useful at home.

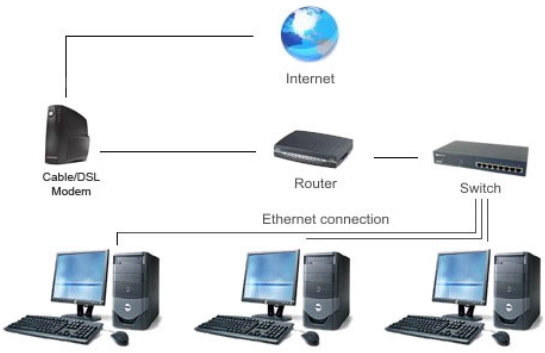


So what is a caching proxy server? The concept is pretty simple: when a request is made to a website, that content is then saved locally on the local caching proxy server. When another request for the same data is made by any machine on your network, that data is retrieved from your local proxy rather than the internet. The content can be anything from regular website content to a file you downloaded. For those with multiple computers in a single household, the bandwidth savings really adds up with patches and multi computer driver updates. If you've got some spare hardware available and are interested in boosting internet performance—and in getting your hands dirty with Linux—this article is for you.

Page 1

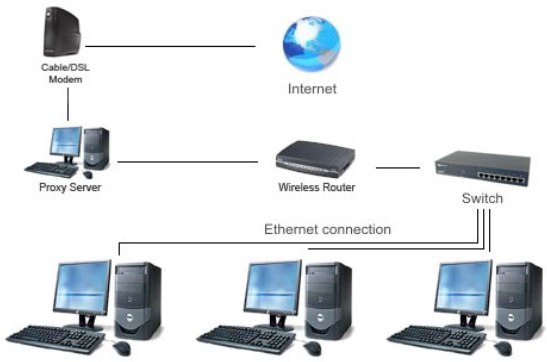
Introduction to Proxy Servers

Do you have a growing family at home slowly eating away at your bandwidth? Maybe you're a web surfing fanatic looking for a little more speed? If you answered yes to either, a caching proxy is for you. This simple addition to your home network can provide you with additional bandwidth by reducing common internet bandwidth usage. Normally these types of proxies are found in the commercial world, but they're just as useful at home. Below is an image of a traditional multi-computer home network.



Traditional Home Network

So what is a caching proxy server? The concept is pretty simple: when a request is made to a website, that content is then saved locally on the local caching proxy server. When another request for the same data is made by any machine on your network, that data is retrieved from your local proxy rather than the internet. The content can be anything from regular website content to a file you downloaded. For those with multiple computers in a single household, the bandwidth savings really adds up with patches and multi computer driver updates. The change to the network configuration is really quite small:



Home Network with Proxy Server

At this point many are likely asking how much this costs. If you read my previous article, you would know the answer right away: "It's free and it's on Linux". I suppose I need to preface that last comment with the qualification that you need some old "junky but functional" hardware lying around. There are many different Linux solutions we can deploy to achieve this goal. For this article I have chosen a solution of Arch Linux, Shorewall, and Squid.

We selected Arch Linux because it is a *rolling release* and has the latest and greatest packages. If you are not familiar with the phrase "rolling release", in Linux it indicated a distribution that keeps you up-to-date with the latest software updates via the package manager. You will never have to re-install or upgrade your server from one release version to the next with this style of distribution. The great part about a rolling release on a proxy/firewall setup is that once it's set up and working correctly, you will not have to go back and completely overhaul the server when a newer distribution update comes out.

Along with the different types of OS and application solutions, there are also multiple ways to set up a caching proxy. My preferred setup is a transparent caching proxy. A transparent proxy does not require you to make any additional changes to the client computers on your network. You utilize the proxy server as your home gateway, allowing the proxy server to automatically forward the ports to Squid. The second way to utilize Squid would be to set up your client machines to utilize the proxy server via the proxy settings in your browser. Although this may be the easiest way to set up a proxy server, it requires you to make changes for any machine that attaches to your network. The table below shows what I selected for my transparent caching proxy server.

Test Proxy System	
Component	**Description**
Processor	Intel Pentium 4 3.06GHz (3.06GHz, 130nm, 512K cache, Single-core + Hyper-Threading, 70W)
Memory	2x256MB PC800 RDRAM
Motherboard	Asus P4T
Hard Drives	120GB Western Digital SATA
Video Card	ATI Radeon 7000
Operating Systems	Arch Linux (32-bit)
Network Cards	Onboard Intel Gigabit PCI 100Mbit 3Com 3c905C-TX

I could have selected older equipment, but this is what I had laying around the house. As seen in the table,

one of the hardware requirements for a transparent proxy is to have two network cards or a dual port network card. We recommend against using wireless for either of the connections to the proxy server, and a Gigabit Ethernet connection from the proxy to the rest of the network is ideal. (The connection to your broadband link can be 100Mbit without imposing any bottleneck.) Another quick suggestion: If you download a fair amount of files, it may be a wise idea to utilize at least a 120GB HDD. The idea is that the more space you have, the longer you can keep your files stored on your proxy server. With storage being so cheap, you could easily add a 500GB or larger drive for under $100.

Now that we have our hardware and a good idea what we want to set up, it's time to get installing. I'll try to keep this portion simple and to the point, although if you have questions later feel free to post a comment.

Page 2

Proxy Server How To

Start by installing Arch Linux (or your chosen distribution) onto the hardware you selected. If you are in need of a little assistance with the installation, I recommend using this wiki guide and then set up yaourt. Once you have completed your standard Linux installation you need to ensure your network is configured properly. In the case of my transparent proxy, I plugged one network port directly into my cable router and allowed it to grab and IP address via DHCP. The second adapter is then given an IP address of your choice (I chose 10.4.20.1; other common IP addresses would be 192.168.x.x).

At this point you will want to test your network configuration. Start with trying to get out to the internet. If this works, plug your secondary network adapter into whatever switch/router you have available. Take your desktop or laptop that's plugged into the same switch and assign it an IP address in your 10.4.20.x range. (For DHCP setups, see below.) You should now be able to ping your new proxy server (10.4.20.1) from your desktop/laptop. As a quick note for the users who only have a wireless cable modem, it is okay to have both interfaces of your proxy server and desktop plugged into the same cable modem hub.

Now that we have the configuration of the network cards complete, we just need to do a quick installation and configuration of Shorewall/Squid. That may sound like a daunting task to the Linux initiate, but this is actually very simple. First go ahead and install both Squid and Shorewall. Arch has both readily available in the package repository (from a command prompt: **yaourt –S shorewall squid**). If you are not utilizing Arch, you can download the packages manually from www.shorewall.net and www.squid-cache.org.

Whether you installed Arch Linux or another distribution as your base OS, Shorewall has one simple command to get it set up: **cp /usr/share/shorewall/Samples/two-interfaces/* /etc/shorewall**. (This copies the base two-NIC example to your live Shorewall directory, which saves a lot of manual work.) Make a quick edit to /etc/shorewall/shorewall.conf and change the Startup_Enabled to yes and you now have a functioning Shorewall. The only thing you need to do for Shorewall at this point is add the following rule into the /etc/shorewall/rules file: **REDIRECT loc 3128 tcp www**. Start Shorewall by typing: **shorewall start** from the command line, and add it to your boot process by putting **shorewall** into the DAEMONS section of **/etc/rc.conf**.

Now that Shorewall is fully functional and configured, we need to configure Squid. I found a short wiki guide that will assist with the initial set up of Squid. Once you have completed the configuration in the wiki guide, you need to pay close attention to a few configuration settings located in /etc/squid/squid.conf. The **cache_memline** should be set to **half of your installed ram** on your proxy server. In my case I have 512MB of total memory so I configured cache_mem to 256. The other setting that you need to pay attention to is **maximum_object_size**. This setting is the maximum file size your proxy will retain. I set my maximum size to 2048MB in order to retain everything up to a CD ISO. Be cautious of using 2048 if you have anything less than a 120gb drive as your storage space could be gone in the matter of a few days. To get the caching proxy in place and running, the most important line to add is **http_port 3128 transparent**. The key here is the addition of "transparent", which turns squid into a caching proxy that won't require any additional configuration on your client PCs.

If you followed all of the directions correctly, you're now ready to configure all the machines on your network with a 10.4.20.x IP address with the gateway set as 10.4.20.1. Don't forget to configure your DNS as well (in **/etc/resolve.conf**). Now that you have everything fired up give your new proxy a spin around the internet. If you would like to do a good test, download a decent size file (i.e. larger than 1MB). Once the

download is complete, you should be able to download it again a second time and get LAN speeds on the download. If you have multiple computers, use another machine on your network and attempt to download the same file and you should again see LAN download speeds.

Proxy Server with DHCP

Although I wanted to keep this short and to the point, a common question inevitably comes up: what if you still want to use DHCP? There are a few ways to tackle this issue. If you're lucky enough to have a router/cable modem that will allow you to change what IP addresses it assigns to the network, simply change it over to your new 10.4.20.x subnet and have it assign the gateway of 10.4.20.1. If this is not the case, you will need to disable DHCP on your router and install the DHCP server package (in Arch: **pacman –S dhcp**). The configuration can be a bit of a hassle, so here's my **/etc/dhcpd.conf**.

Start the DHCP service on your proxy (**/etc/rc.d/dhcpd start**) and test DHCP on your desktop/laptop. Assuming all goes well, add **dhcpd** to your DAEMONS in **/etc/rc.conf**. If you happen to reboot your Linux box, after a minute or so your proxy should be back up and running.

Page 3

Linux Neophyte Troubleshooting (by Jarred)

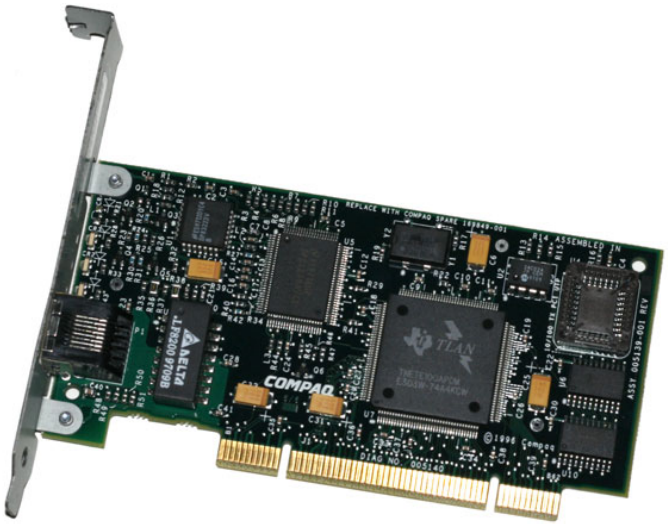
I have to give Chris credit: he knows a lot about Linux. In fact, I'm pretty sure he's forgotten more about the subject than I have yet learned! However, being "new" (relatively) to Linux allows me to provide some insight that he may have glossed over. If you're an experienced Linux user, nothing I say here is likely to help out, but for the rest of you I thought before posting this article I'd take a stab at setting up my own proxy server. The "simple" process ended up taking a couple days of on-and-off troubleshooting to get everything working properly. What follows is a brief summary of the things I learned/experienced during my Linux proxy crash course.

First, let's start with the hardware. I had a mini-ITX motherboard and parts available, which would have been perfect! Sadly, the board only has a single network adapter and an x16 PCI-E slot, so I looked elsewhere. I ended up piecing together a system from spare parts.

Jarred's Test System	
Component	**Description**
Processor	Intel Core 2 Quad Q6600 (2.40GHz, 65nm, 2x4MB cache, quad-core, 1066FSB, 105W)
Memory	2x2048MB DDR2-800 RAM
Motherboard	ASRock Conroe1333-eSATA2
Hard Drives	300GB Maxtor SATA
Video Card	NVIDIA GeForce 7600GT
Operating Systems	Arch Linux (64-bit)
Network Cards	Onboard NVIDIA Gigabit (NForce) PCI TRENDnet Gigabit (Realtek 8169)

Obviously, my spare hardware is a bit more potent than what Chris had lying around, and frankly it's complete overkill for this sort of box. On the bright side, it runs 64-bit Linux quite well, and the NVIDIA GPU makes it gaming capable (if you're not too demanding). Getting Arch installed was the easy part, though; configuring things properly took quite a bit more effort.

I followed the directions and... nothing worked. Ugh. Now, I have to put a disclaimer here: I initially used an old Compaq PCI NIC as my secondary network adapter... and I discovered it was non-functional after a while spent troubleshooting. Or at least, it didn't work with Linux and caused the PC to lock up when I tried to load the driver. Good times! So make sure your hardware works properly in advance and you'll save yourself a headache or two. I picked up the TRENDnet Gigabit NIC at a local shop for just $20 and it installed without a hitch.



Old hardware isn't a problem with Linux; broken on the other hand…

As far as configuring Linux, the wikis Chris linked were generally helpful, though they're more detailed than most people will want/need. The "Arch Way" essentially boils down to giving you a fishing pole and some bait and trying to teach you to fish rather than providing you with a nice salmon dinner. Arch has benefits, and you *will learn* something about Linux (whether you want to or not), but if you're a newbie plan on spending a fair amount of time reading wikis and searching for solutions as you come to grips with the OS.

After Arch was running and I discovered my Compaq NIC was dead, installing the second NIC required a bit of unexpected work. Since it wasn't present during the OS install, the drivers weren't loaded by default. Using **lspci**, I was able to find my new NIC, determined it was a Realtek 8169 chipset, and a short Google later I found the necessary driver: **modprobe r8169**. After spending some time reading about ifconfig and trying a few settings, I got the NIC installed and (apparently) functional, so now it was time to get squid and shorewall configured. (Note that this would have likely been unnecessary had the NIC been present during the Arch install.)

While Chris likes the 10.4.20.x network, I prefer the customary 192.168.x.x. Chris listed a global DNS name server of 216.242.0.2, which will work fine (a name server from CiberLynx), but I put in the name servers from my ISP (Comcast). I grabbed this information from the /etc/resolv.conf file, placed there by DHCP from the cable modem. I also wanted to use DHCP as much as possible. The result is that I have my onboard NIC plugged into my cable modem, and the TRENDnet NIC connected to my wireless router. I set a static IP of 192.168.1.1 for the TRENDnet NIC, with DHCP providing IPs from 192.168.1.5 through 192.168.1.250. Really, though, I only need one for the wireless router, which then provides its own DHCP for a different subnet: 192.168.10.x. The good thing about this setup is that I never had to touch the configuration on my wireless router, which has been working fine. I just unplugged it from the cable modem and connected it to the Linux box.

Configuring shorewall was simple, but I ended up not getting network access from my Linux box. That was a "works as intended" feature, but I wanted to surf from the Linux box as well. I had to add **ACCEPT $FW net tcp www** to **/etc/shorewall/rules** file to get my local networking back, and I added a line to allow FTP to work as well. Getting squid to work wasn't a problem… after figuring out that Chris forgot the "transparent" option for the http_port setting. I created the directory **/home/squidcache** for the proxy (**mkdir /home/squidcache** then **chmod 777 /home/squidcache**), just because I liked having the cache as a root folder. With everything finally configured properly, I did some testing and found everything worked about as expected. Great! I also installed X Windows, the NVIDIA driver, and the KDE desktop manager as per the Beginner's Guide Wiki—useful for editing multiple text files, surfing the web for configuration information, etc. Then I decided to reboot the Linux box to make sure it was truly working without a hitch.

After the reboot, sadly to say I was back to nothing working… locally or via the proxy. Some poking around (using **dmesg** and **ifconfig**) eventually led me to the discovery that my NICs had swapped names after the reboot, so the NForce NIC was now eth1 and TRENDnet was eth0. One suggestion I found said that if I put the drivers for my NICs into the MODULES section of rc.conf, I could specify the order. That didn't work,

unfortunately, but another option involved creating a file called **/etc/udev/rules.d/10-network.rules** with two lines to name my NICs. (Get your MAC Address via **dmesg|grep [network module]** or **udevadm info -a -p /sys/class/net/[Device: eth0/eth1/wlan0/etc.]**.) So I added:

```
SUBSYSTEM=="net", ATTR{address}=="[NVIDIA NForce MAC]", NAME="eth0"
SUBSYSTEM=="net", ATTR{address}=="[TRENDnet MAC]", NAME="eth1"
```

At this point, everything worked properly, but I did run into a few minor quirks over the next day or so of testing. One problem was that Futuremark's Peacekeeper benchmark stopped working. Troubleshooting by Chris ended up showing that there was a problem with the header being sent from the Futuremark server (Message: "Invalid chunk header" in **/var/log/squid/cache.log**). Telling squid not to cache that IP/server didn't help, as the malformed header problem persisted, but we were able to work around the issue by modifying the shorewall rules. Now the redirect line reads: **REDIRECT loc 3128 tcp www - !service.futuremark.com**—in other words, redirect all web traffic except for service.futuremark.com through the proxy.

Wrapping things up, here are the final configuration files that I modified for my particular setup. Providing these files almost certainly goes against the Arch Way, but hopefully having a sample configuration can help a few of you out.

/etc/dhcpd.conf: Put your own ISP name servers in here (from /etc/resolv.conf).
/etc/rc.conf: Specify your network setup, server name, and startup daemons.
/etc/shorewall/rules: The necessary redirect for web traffic to work with your proxy.
/etc/shorewall/shorewall.conf: Only changed the one line to STARTUP_ENABLED=Yes.
/etc/squid/squid.conf: Huge file full of proxy options; here's the short version without comment lines.

Update: *It seems my proxy was throttling performance when using "diskd" for the cache directory; changing it to aufs has fixed the situation. With diskd, I experienced intermittent bursts of Ethernet transfer rates, with other transfers limited to <500KB/s. We're not sure why this happened, but you may want to check your network transfer rates with* ***iptraf*** *(****pacman -S iptraf****, then run it and choose the "S" option to view real-time network transfers).*

So what are the benefits to running the proxy cache? If you run multiple machines (I've got more than a dozen at present, with systems constantly arriving and leaving), the proxy cache means things like Windows Updates won't have to go to the web every time and download several hundred megabytes of data. That same benefit is potentially available for other services (i.e. FTP), and in an ideal world I'd be able to cache the various Steam updates. Sadly, Valve doesn't appear to like that, so all of my systems need to go out to the Valve servers to update. Except, you can manually copy your steamapps folder from one system to another and avoid the downloads. But I digress. The squid proxy can also provide a host of other capabilities, from anti-virus support to web filtering and even limiting access to certain times of the day.

The bottom line is that if you have an old system lying around—certainly my quad-core proxy is overkill, and even a Pentium 4 is more than you actually need—you can definitely benefit. A small ITX box or perhaps even an Atom nettop would be perfect for this sort of thing, but most of those lack the requisite dual NICs. You could try a PCIe NIC with mini-ITX, though it's questionable whether the x1 cards will function properly in a mini-ITX board with a single x16 slot intended for graphics use. Barring that, a uATX setup would work fine. Our only recommendation is that you ***consider the cost of electricity compared with the hardware***. Sure, Linux will run fine on "free" old hardware, but a proxy server will generally need to be up and running 24/7, so you don't want to have a box sucking down 100W (or more) if you can avoid it.