Caching Technical White Paper

The NetEqualizer Caching Option (NCO) is an embedded caching capability that runs directly on the NetEqualizer 3000 and 4000 series. When NCO is activated, a squid proxy server and web cache daemon are loaded, along with proprietary NetEqualizer Caching configuration and integration software and a new internal solid-state drive (SSD). In this white paper, we discuss the inner workings of NCO and the results we have seen to-date.

How does the NetEqualizer Caching Option work?

The NetEqualizer Caching Option is integrated with Equalizing, providing a comprehensive bandwidth management strategy. Traffic can be accessed from cache or accessed from the Internet and equalized, as needed. ***NCO caches all port 80 traffic file sizes from 2MB to 40MB, including YouTube videos.*** Any type of static content that is frequently accessed will benefit from caching.

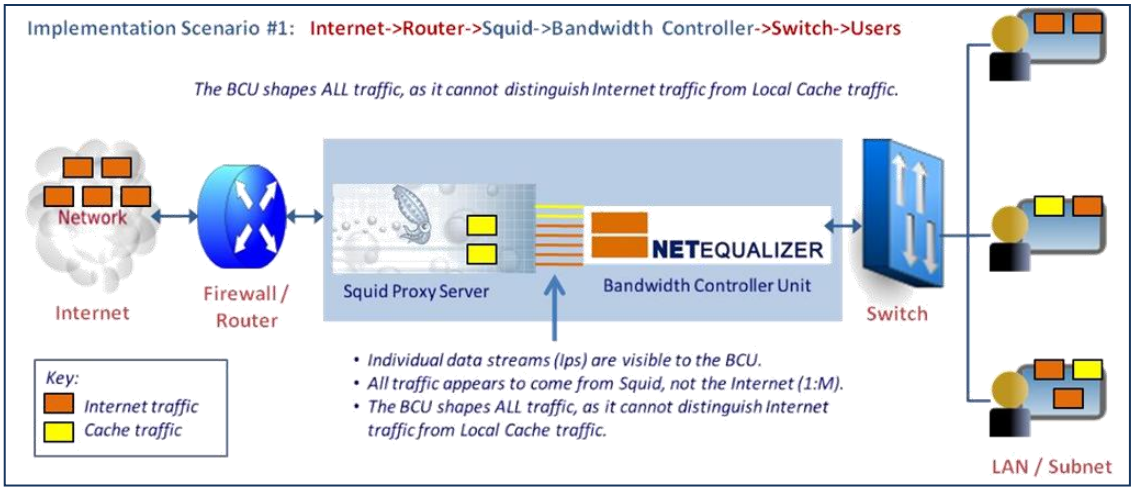
NetEqualizers are 1U rack-mountable units, shaping bandwidth levels from 4Mbps - 5Gbps bi-directionally. NCO is available on NE3000 and NE4000 series. See Data Sheets for detailed specifications.

We have built-in the capability to cache YouTube videos, which is not available in off-the-shelf Squid caching. With our NetEqualizer 5.0 release, we've created an integration with NetEqualizer and co-resident Squid such that bandwidth control and caching can work seamlessly. This required some creative routing logic and actual bug fixes to the bridging and routing in the Linux kernel. We also had to develop a communication module between the NetEqualizer and the Squid proxy server so the NetEqualizer gets advance notice when data is originating locally from cache and not the Internet. The NetEqualizer then appropriately acts on only traffic originating from the Internet.

Why did we need to do all this? Adding a caching proxy server to your network, in line with your bandwidth controller, can get tricky. Let's walk through two implementation designs to see what the challenges are with each:

Implementation Scenario #1: Placing the Squid Proxy Server BEFORE the Bandwidth Controller

To use a Squid proxy server, your network administrator must put hooks in your router so that all Web requests go the Squid proxy server before heading out to the Internet. Sometimes the Squid proxy server will have a local copy of the requested page, but most of the time it will not. When a local copy is not present, it sends your request on to the Internet to get the page (for example the Yahoo! home page) on your behalf. The Squid proxy server will then update a local copy of the page in its cache (storage area) while simultaneously sending the results back to you, the original requesting user. If you make a subsequent request to the same page, Squid will quickly check it to see if the content has been updated since it stored it away the first time, and if it can, it will send you a local copy. If it detects that the local copy is no longer valid (the content has changed), then it will go back out to the Internet and get a new copy.



Now, if you add a bandwidth controller to the mix, things get interesting quickly. In the case of the NetEqualizer, it decides when to invoke fairness based on the congestion level of the Internet trunk. However, with the bandwidth controller unit (BCU) on the private side of the Squid server, *the actual Internet traffic cannot be distinguished from local cache traffic*. The setup looks like this: Internet->Router->***Squid->Bandwidth Controller Unit***->Users, depicted in the diagram at the above.

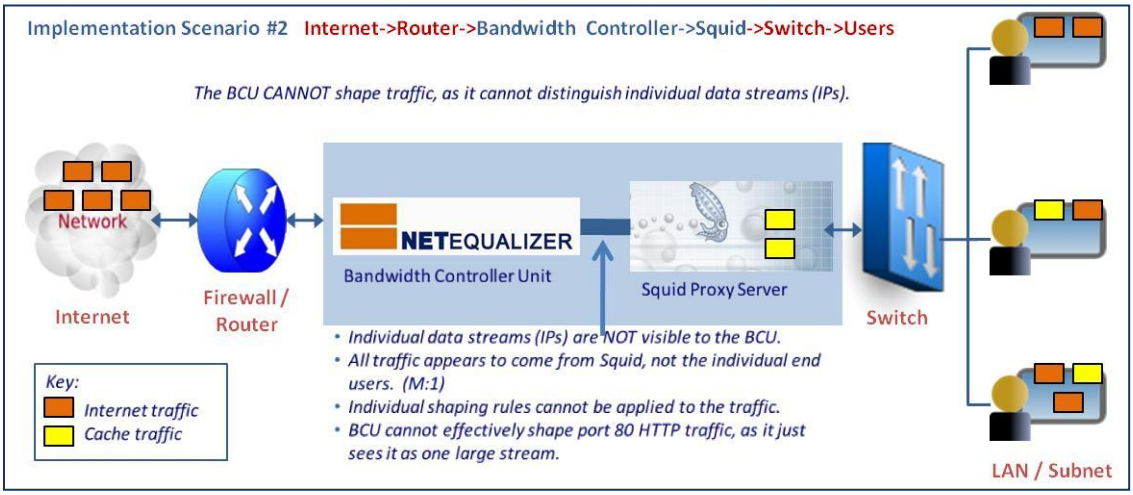
APconnections, Inc. // 303.997.1300 // www.netequalizer.com

The BCU in this example won't know what is coming from cache and what is coming from the Internet. Why? Because the data coming from the Squid cache comes over the same path as the new Internet data. ***The BCU will erroneously think all the traffic is coming from the Internet and will shape Local Cache traffic as well as Internet traffic, thus defeating the higher speeds provided by the cache.***

Implementation Scenario #2: Placing the Squid Proxy Server AFTER the Bandwidth Controller

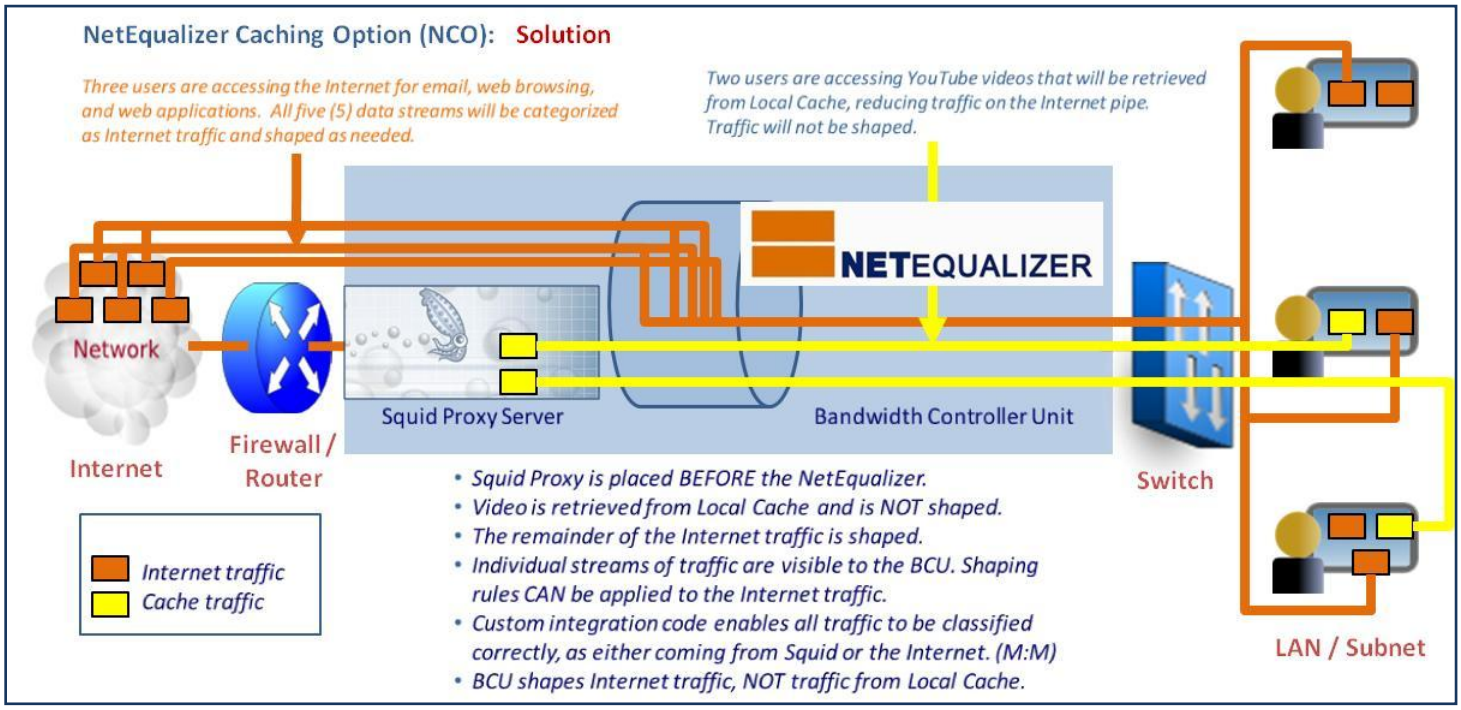
In this situation, the obvious solution would be to switch the position of the BCU to a setup like this: Internet->Router->***Bandwidth Controller Unit->Squid***->Users, depicted in the diagram below.

This configuration would be fine except that now all the port 80 HTTP traffic (cached or not) will appear like it is coming from the Squid proxy server and your BCU will not be able to shape individual traffic, nor apply individual rules, such as putting rate limits on individual users.



NetEqualizer Caching Option (NCO): Solution

The final technical solution addresses all of the issues mentioned in the implementation options above, namely:
1) we correctly identify and separate out Local Cache traffic from Internet traffic, so that we can shape Internet traffic and NOT Local Cache, and 2) we maintain visibility to individual data streams (IPs) so that bandwidth shaping rules can be applied to individual traffic. This is shown in the diagram below. For details around the Key Challenges we encountered, refer to Appendix A. To read more about what we tuned in Squid, refer to Appendix B.



Why Not Just Cache and Forget About Bandwidth Control?

At this point, you may be wondering if Squid caching is so great, why not just dump the BCU and be done with the complexity of trying to run both? Well, while the Squid server alone will do a fine job of accelerating the access times of large files such as video when they can be fetched from cache, ***a common misconception is that there is a big relief on your Internet pipe with the caching server***. This has not been the case in our real world installations.

The fallacy for caching as panacea for "all things congested" assumes that demand and overall usage is static, which is unrealistic. The cache is of finite size and users will generally start watching more YouTube videos when they see improvements in speed and quality (prior to Squid caching, they might have given up because of slowness), including videos that are not in cache. So, the Squid server will have to fetch new content all the time, using additional bandwidth and quickly negating any improvements.

Additionally, not all Internet traffic is of a static nature. Many traffic types, such as emails, web browsing, web chat, and web applications are not good candidates for caching due to: 1) their small size, and 2) their content changes frequently (so cache would never be used - the caching server would be going to the Internet for a fresh copy most of the time).

If you had a congested Internet pipe before caching, you will likely still have one afterward, leading to slow access for many e-mail, web chat and other non-cacheable content. The solution is to include a bandwidth controller in conjunction with your caching server. This is what NetEqualizer 5.0 and above now offers.

The fallacy for caching as a panacea for "all things congested" assumes that demand and overall usage is static…if you had a congested Internet pipe before caching, you will likely still have one afterward…

The solution is to include a bandwidth controller as well…

Art Reisman
Founder & CTO
AP Connections, Inc.

What are the Results (Prove it!)?

Since the release of YouTube caching support on our NetEqualizer bandwidth controller, we have been able to review several live systems in the field. Below we will go over the basic hit rate of YouTube videos and explain in detail how this effects the user experience. ***The analysis below is based on an actual snapshot from a mid-sized state university, using a 64 Gigabyte cache, and approximately 2000 students in residence***.

The Squid Proxy server provides a wide range of statistics. You can easily spend hours examining them and become exhausted with MSOS, an acronym for “meaningless stat overload syndrome”. To save you some time we are going to look at just one statistic from one report. From the Squid Statistics Tab on the NetEqualizer, we selected the **Cache Client List** option. This report shows individual Cache statistics for all clients on your network. At the very bottom is a summary report totaling all squid stats and hits for all clients.

TOTALS
ICP : 0 Queries, 0 Hits (0%)
HTTP: 21990877 Requests, 3812 Hits (0%)

At first glance it appears as if the ratio of actual cache hits, 3812, to HTTP requests, 21990877, is extremely low (.02%). As with all statistics the obvious conclusion can be misleading. Here is why:

*Our NCO caching results are based on a **real-world snapshot** from a mid-sized state university, using a 64 Gigabyte cache, and approximately 2000 students in residence.*

The NetEqualizer Cache is deliberately tuned to NOT cache HTTP requests smaller than 2 Megabytes.

This is done for a couple of reasons: 1) Generally, there is no advantage to caching small web pages, as they normally load up quickly on systems with NetEqualizer fairness in place, as they already have priority., and 2) With a few exceptions of popular web sites, small web hits are widely varied and fill up the cache – taking away space that we would like to use for our target content, Youtube Videos.

The Actual Data Cached is more meaningful than total Web Hits.

It is true that web sites today can often exceed a Megabyte. However, rarely does a web site of 2MB load up as a single hit. It is comprised of many sub-links, each of which generates a web hit in the summary statistics. A simple HTTP page typically triggers about 10 HTTP requests for perhaps 100KB of data total. A more complex page may generate 500KB. For example, when you go to the CNN home page there are quite a few small links, and each link increments the HTTP counter. On the other hand, a YouTube hit generates one hit for about 20MB of data. ***When we start to look at actual data cached instead of total Web Hits, the ratio of cached to not cached is quite different.***

Our cache set up is also designed to only cache Web pages from 2MB to 40MB, with an estimated average of 20MB. Using this average in this real-world study, when we looked at actual data cached (instead of hits) this gave us about 400GB of regular HTTP data, of which about 76GB came from the cache (19%). To be conservative, we rounded this down to 10%. ***Therefore, in our real-world study with the mid-sized state university, 10 percent of all HTTP data came from cache.*** This number is much more significant than the HTTP Web Hits statistic reveals.

Once the NetEqualizer Caching Option (NCO) was up and running, in our real-world study with a mid-sized state university we conservatively demonstrated savings of 10% HTTP traffic.

10% of all HTTP traffic now came from Local Cache.

Even more telling, is that effect these hits have on user experience.

YouTube streaming data, although not the majority of data on this customer system, is very time-sensitive, while at the same time being very bandwidth intensive. The subtle boost made possible by caching 10 percent of the data on this system has a discernible effect on the user experience. Think about it, if 10 percent of your experience on the Web is video, and you were resigned to it timing out and bogging down, you will notice the difference when those YouTube videos play through to completion, even if only half of them come from cache.

In summary, while Caching will not solve all your bandwidth congestion issues, it should be considered as part of an overall bandwidth management strategy, particularly in environments where YouTube traffic is a large percentage of your network traffic.

To Learn More…

We would be a happy to discuss the NetEqualizer Caching Option (NCO) with you, to help you determine if this is the right solution for your business.

Please email sales@apconnections.net or call us at 303.997.1300 x103 to schedule a discussion.

About APconnections, Inc.

APconnections is based in Lafayette, Colorado, USA. We develop cost-effective, easy-to-install and manage, traffic shaping appliances. Our NetEqualizer product family optimizes critical network bandwidth resources for any organization that purchases bandwidth in bulk and then redistributes or resells that bandwidth to disparate users with competing needs.

We released our first commercial offering in July 2003, and since then customers around the world have put our products into service. Our flexible and scalable solutions can be found at ISPs, WISPs, major universities, Fortune 500 companies, SOHOs and small businesses on six continents.

Appendix A: Key Challenges

Key Technical Challenges

Our solution required us to overcome multiple hurdles. Below are listed the key challenges that we encountered :

1. **Getting Squid to Cache YouTube**
   There was the issue of just getting a standard Squid server to cache YouTube files. It seems that the URL tags on these files change with each access, like a counter, and a normal Squid server is fooled into believing the files have changed. By default, when a file changes, a caching server goes out and gets the new copy. In the case of YouTube files, the content is almost always static. However, the caching server thinks they are different when it sees the changing file names. Without modifications, the default Squid caching server will re-retrieve the YouTube file from the source and not the cache because the file names change.

2. **Moving to A More Recent Linux Kernel**
   We had to move to a newer Linux kernel to get a recent of version of Squid (2.7), in order to support the hooks for YouTube caching. A side effect was that the new kernel de-stabilized some of the timing mechanisms we use to implement bandwidth control. These subtle bugs were not easily reproduced with our standard load generation tools, so we had to create a new simulation lab capable of simulating thousands of users accessing the Internet and YouTube at the same time. Once we built this lab, we were able to re-create the timing issues in the Linux kernel and patch them.

3. **Setting up Firewall Redirect**
   It was necessary to set up a firewall re-direct (also on the NetEqualizer) for port 80 traffic back to the Squid server. This configuration, and the implementation of an extra bridge, were required to get everything working. The details of the routing within the NetEqualizer were customized so that we would be able to see the correct IP addresses of Internet sources and users when shaping. This addressed the issue with Implementation Scenario #2. As mentioned above, if you do not take care of this, all IPs (traffic) will appear as if they are coming from the Squid server.

4. **Sizing the Firewall Connection Tracking Table**
   The firewall has a table called ConnTrack (not be confused with NetEqualizer connection tracking, but similar). The connection tracking table on the firewall tends to fill up and crash the firewall, denying new requests for re-direction if you are not careful. If you just go out and make the connection table randomly enormous that can also cause your system to lock up. You must measure and size this table based on experimentation. This was another reason for us to build our simulation lab.

5. **Managing Number of Linux File Descriptors**
   There was also the issue of the Squid server using all available Linux file descriptors. Linux comes with a default limit for security reasons, and when the Squid server hit this limit (it does all kinds of file reading and writing keeping descriptors open), it locks up.

Squid Tuning Lessons Learned

Some of the Squid tuning changes we made to support YouTube caching were:

a. **Limiting the file size of a cached object to between 2 megabytes (MB) to 40 megabytes (MB)**
(i.e. minimum_object_size 2000000 bytes & maximum_object_size 40000000 bytes). We did this because if you allow smaller cached objects, they will rapidly fill up your cache, and there is little benefit to caching small pages.

b. **We turned off the Squid "keep reading" flag** (i.e. quick_abort_min 0 KB & quick_abort_max 0 KB)
This flag when set continues to read a file even if the user leaves the page. For example, when watching a video, if the user aborts on their browser, the Squid cache continues to read the file.

c. **We also explicitly told the Squid what DNS servers to use in its configuration file**. (i.e. dns_nameservers x.x.x.x)
There was some evidence that without this the Squid server may bog down, but we never confirmed it. However, it is cleaner to set these parameters and does not harm anything to do so.

d. **We learned that we had to be very careful to set the cache size not to exceed actual capacity.**
Squid is not smart enough to check your real capacity, so it will fill up your file system space if you let it, which in turn causes a crash. When testing with small RAM disks with less than 4GB of cache, we found that the Squid logs will also fill up your disk space and cause a lock up. The logs are refreshed once a day on a busy system. With a large amount of pages being accessed, the log will use close to 1 GB of data quite easily, and then to add insult to injury, the log back up program makes a back up. On a normal-sized caching system, with an SSD installed, there should be ample space for logs.

e. **We learned how to manage Squid's internal short-term buffer.**
Squid has a short-term buffer not related to caching. It is just a buffer where it stores data from the Internet before sending it to the client. Remember that all port 80 (HTTP) requests go through the Squid, cached or not, and if you attempt to control the speed of a transfer between Squid and the user, it does not mean that the Squid server slows the rate of the transfer coming from the Internet right away. With the BCU in-line, we want the sender on the Internet to back off right away if we decide to throttle the transfer. With the Squid buffer in-between the NetEqualizer and the sending host on the Internet, the sender would not respond to our deliberate throttling right away when the buffer was too large.