

QBox: A QoS Appliance for LAN/WAN Links using Linux

Ron Senykoff

10/23/2005

Abstract

Internet access has become increasingly important in the daily operations of both individuals and businesses. As internet dependency has increased the availability of faster connections has also increased. However, newly emerging technologies such as Voice over IP and videoconferencing are placing additional demands on the already scarce resource of bandwidth. The distinction here is not so much about additional bandwidth, but about how the bandwidth is utilized. Traditional internet usage such as web browsing or email downloads operate asynchronously in such a way that a small, variable wait time does not necessarily adversely effect the user experience.

Enter the newer technologies and tolerances become increasingly strict. In VoIP applications, for example, a delay of greater than ~300ms starts to make the conversation unusable. With QBox we use traffic classification combined with queuing disciplines to prioritize and shape the traffic. By classifying traffic into appropriate queues we can guarantee certain levels of service for certain applications while maximizing the utilization of bandwidth. Existing methods for doing QoS are either very expensive, or very complicated. QBox targets creating a low-cost solution that requires only the most basic networking skills for installation. By selecting an integrated motherboard and using Compact Flash for storage, moving parts are eliminated while ease of installation and reliability are maximized.

1 Introduction

1.1 Interactive applications need more than bandwidth

A very common misconception is that performance of internet applications is most closely tied with bandwidth. The more bandwidth available, the faster it will be. Practically, this is only part of the relationship. Interactive applications such as SSH, Voice over IP (VoIP), and videoconferencing range from requiring very little bandwidth in the case of SSH, to requiring massive amounts of bandwidth (typically > 512 Kbps) for videoconferencing. What they have in common is a need for low and predictable latency, and reliability.

1.2 Current implementations are either complex or expensive

With QBox we decided to focus on providing a simple appliance that can provide traffic shaping capabilities. Solutions geared towards large corporate environments exist, but these are priced at a level that may prevent many smaller organizations from access. Free solutions also exist based on open-source tools. These can be very complex to implement, requiring that the user write custom scripts to shape traffic. The prebuilt solutions currently are tied into other products, such as m0n0wall, which provide all other sorts of packages of which QoS is only a piece. Thus the implementer is faced with implementing undesirable or already implemented services just to gain access to the QoS capabilities.

1.3 QBox provides affordable, simple access to Quality of Service without requiring a change in network topology

In this paper we describe how QBox can be used to implement QoS within most environments by simply dropping it in place and doing some simple configuration. With QBox only basic networking skills and an understanding of the usage on your network are required. Unlike other open-source solutions, QBox does not require advanced knowledge of queuing disciplines or programming / scripting languages.

1.4 How the rest of the paper is structured

The rest of this paper first discusses related work in Section 2, and then describes our implementation in Section 3. Section 4 describes how we evaluated our system and presents the results. Section 5 presents our conclusions and describes future work.

2 Related Work

2.1 Commercial Traffic Shaping Solutions

Currently there are several commercial solutions available for providing traffic shaping on LAN to WAN links. These products are relatively expensive, yet provide some very robust functionality and the ability to shape extremely high volumes of traffic. We reviewed the functionality of several commercial products to see what

2.1.1 Packeteer Packetshaper

Packeteer [1] is one of the larger providers of traffic shaping appliances. Their products are mainly priced on throughput, with additional features becoming available on their higher-end appliances. Packeteer's Packetshaper can handle rates as high as 500Mbps while offering compression and centralized management of multiple appliances. An appliance with this functionality starts at \$34,000 USD. In addition the Packetshaper provides reporting on types of traffic and the traffic's usage, and adaptive classification of traffic.

2.1.2 Emerging Technologies ET/R1800G

Emerging Technologies [2] is a newer provider of network appliances. They build their appliances from PC parts and base the software on FreeBSD. Traffic prioritization, bandwidth management, compression, and statistics gathering are all supported. The products are designed to be able to handle a lot of traffic, and do provide some nice interfaces. Entry level price is about \$5995.

2.2 Open Source Traffic Shaping Solutions

There are several projects within the open source community that provide traffic shaping capability. We reviewed both Linux and FreeBSD based solutions to see if an existing project could be used for QBox. Of the solutions we found, we could not find an easily deployable application that was specific to traffic shaping. Some people have put major effort into queuing projects, however this has been at the kernel level to provide the functionality of queuing itself. For Linux the Traffic Control package [5] has now been included in the kernel source tree. On FreeBSD the ALTQ project [6] has now also been merged into the kernel.

2.2.1 Wondershaper

Bert Hubert [8] wrote a very popular shell script for performing traffic shaping on Linux called the "Wondershaper." This script is geared mostly for home users on xDSL or cable modem lines, but the script can also be applied to corporate environments. Wondershaper makes the calls to the 'tc' utility to generate several queues and prioritize interactive traffic versus bulk and web browsing. Hubert takes into account upstream queuing by ISPs and throttles back the overall connection to unload upstream queues and gain control with the Wondershaper. While the Wondershaper script provides a great basis for traffic shaping, it requires that the implementor already have a Linux box up and running, and that they know enough about integrating Linux into a network to create a bridge or router with Linux on which to place the Wondershaper.

2.2.2 m0n0wall

"m0n0wall is a project aimed at creating a complete, embedded firewall software package that, when used together with an embedded PC, provides all the important features of commercial firewall boxes (including ease of use) at a fraction of the price (free software)." [9] m0n0wall is based on FreeBSD and hardware support is geared towards embedded platforms from Soekris Engineering [10] and PC Engines [11]. The use of embedded platforms is very

appealing for the QBox project, and the ability to install directly to Compact Flash and boot a system is a trait we decided we wanted to implement for QBox.

2.2.3 LEAF - Linux Embedded Application Firewall

The LEAF project provides “A secure, feature-rich, customizable embedded Linux network appliance for use in a variety of network topologies.”[7] One of LEAF’s main goals is to maintain as small a footprint (amount of space required for installation) as possible. With the addition of the uClibc project for a compiler geared towards embedded systems, the Bering uClibc branch has come to be the most maintained branch of LEAF.[3, 4] Bering uClibc is based on the Linux 2.4 kernel thus capable of including Traffic Control, and by default comes with Shorewall firewall[12], SSH, and routing related packages all which can be booted from a floppy. The modularity of LEAF packages is inherited from its Linux Router Project roots[13], and there is thus a well-defined format for adding / removing / creating packages for LEAF.

3 Implementation

3.1 Building the Box

3.1.1 OS Selection

We decided that LEAF would be the best platform from which to base our solution. Several factors were critical in the decision process:

Modularity LEAF is based upon Linux Router Project modules. The well-defined structure for bringing these modules withing the package management framework of LEAF would save us a lot of time. A LEAF module automatically integrates with both the web interface and the console interface. Also, the modularity would allow us to easily remove packages (such as Shorewall) that were not needed. m0n0wall, while a fantastic project, is more of an all-in-one solution which appeared would be more difficult to customize.

Size Since this was to be an embedded solution, small size was a must. LEAF offered the smallest base distribution which was small enough to fit on a floppy. While our plan was to use Compact Flash to hold the configuration, the smaller the CF card the cheaper. m0n0wall requires at least 8MB of Compact Flash, with more required as you add additional packages. This is still a very small size and quite a feat, so in reality both m0n0wall and LEAF were equally capable in the size category.

3.1.2 Hardware Selection

Now that we had chosen our software, we needed to pick the hardware for the project. We compared 2 boards and while both appeared to be excellent candidates, we chose the board with overall lower cost.

The Mikrotik Routerboard series offers an excellent option for embedding Linux. We looked at the Routerboard 200 which has 2 NICs and a serial port for configuration. Mikrotik even maintains a LEAF distribution for the Routerboard on their site, ready for download. The Routerboard 200 uses SODIMM laptop type RAM so we would need to purchase RAM in addition to the board, case, CF card, and power supply.

PC Engines offers their Linux-capable WRAP board, which includes 128MB of on-board RAM for \$120 USD versus the \$200 price tag of the Routerboard. We would just need to buy a case (\$15 anodized aluminum), power supply (\$25), and CF card (\$30) to round out the selection. While they do not maintain a LEAF distribution like Mikrotik, our plan was to customize LEAF anyways so in the end it would not have much impact.

3.1.3 Getting LEAF to boot on the WRAP board

Getting LEAF onto the CF card such that it would boot the WRAP board proved to be more of a challenge than we originally expected. We soon found that repeatedly swapping the CF card in and out of our card reader and onto the board became very time-consuming. For this reason a LEAF group maintains a network boot-able (using PXE) kernel that loads up serial port support, support for many popular NICs, and some very convenient applications to grab files,

mainly wget and smbmount. After a lot of fiddling around we were able to get a gentoo box serving as a TFTP server for the WRAP board.

The installation of the network boot environment is beyond the scope of this article, and further information can be found at <http://leaf.sourceforge.net/doc/guide/bucu-ide.html#id2799843>. The WRAP board does not contain a keyboard controller, and the image provided for the TFTP server does not handle this gracefully. But after watching 'keyboard error' scroll for 20 seconds on boot, it will continue. The main requirement is that you must go into the WRAP board's BIOS (by hitting 's' during the memory check) to enable network booting.

Once we got the WRAP board to network boot, we were able to use smbmount to mount a local windows share and copy the base LEAF files over to our mounted CF card (/dev/hda1). Using syslinux we made the already FAT formatted CF card boot-able. We configured LEAF to load the natsemi.o module for the network card chips and rebooted. Still no luck. We then discovered that the WRAP board BIOS redirects VGA output to the serial console. Some syslinux documentation warned that this could cause problems and provided a workaround. After adding 'CONSOLE 0' to syslinux.cfg we rebooted and were able to successfully get to a login prompt. By default there is no root password on LEAF.

3.2 Customizing LEAF

Finally we were into an environment that we could build upon. Keep in mind that the default LEAF install fits on a floppy disk, so there is a lot of bare-bones functionality. To help with editing files, we added elvis.lrp, a vi-like editor, and ncurses.lrp. We configured dropbear (the ssh server) so that we could stop working from the serial console. We added the bridge.lrp package, and changed the network configuration so that the WRAP board would come up as a bridge rather than a router. We had to remove shorewall.lrp as we were not interested in firewalling, and shorewall would not work in conjunction with our bridge anyways.

3.2.1 Adding the traffic control modules

Once we had LEAF coming up as a bridge, we were ready to create our custom .lrp package. We found that we needed to add the traffic control modules for the LEAF kernel, as they are not compiled in by default. We added the modules sch_sfq, sch_htb, and cls_u32. We originally worked with sch_cbq instead of sch_htb, but the implementation of it on LEAF was lacking some capabilities probably due to the 2.4 version of the kernel. Hierarchical Token Bucket (HTB) is easier to configure than Class Based Queueing (CBQ qdisc) as CBQ has many parameters that have poor documentation if any.

3.2.2 Write the back-end shell scripts and the user-modifiable shell scripts

Now we had a bridge with traffic shaping capability. What we needed to do was then create a lrp package for our QBox functionality. There is currently a traffic control package for LEAF, but we found it rather complicated for your basic user to set up. One of the goals of this project was to create an appliance that was simple to configure, so we decided we should write a custom interface and queuing scheme.

We didn't want the end user needing to look at large scripts and find variables to adjust. Thus we decided to break out the variables into separate files. There is a separate file for each 'band' of traffic for port classification and IP classification. So if you want to add port 4569 (Asterisk VoIP port) to the interactive traffic class, you edit highpriports.sh through the web interface and add 4569 into the list. Same approach for IPs. Maybe you want to classify a whole server as bulk traffic (your email server, for example). Or possibly you want to provide high priority to an application server that is on the WAN (Citrix). All of this becomes relatively straightforward. One just needs to think in terms of ports and IPs only, and most scenarios can be solved for classification.

3.2.3 Create QBox LRP package

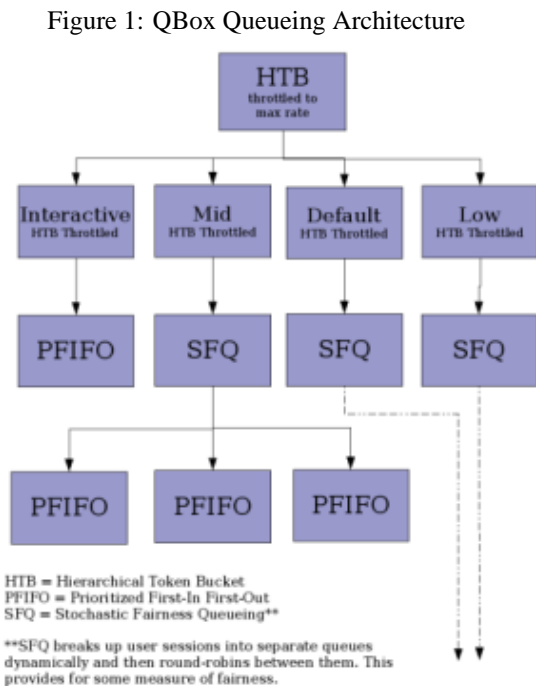
Once we had all our scripts built up, including our custom init script, it was time to build the .lrp package. A .lrp package is really just a zipped tar file. You create a directory tree on it for the custom files you want inflated into the RAMdisk at boot. In addition, lrp packages require several additional files. One lists all the config files that a user may edit. Another lists which files get backed up to disk (remember, everything is in RAM here), and another contains help documentation. We created the file and added it to the leaf.cfg file of packages to load, and we were done. We now had a custom LEAF build, that boots up as a bridge and sets up the queues and traffic classification parameters

specified by the user. A user simply needs a WRAP board and a CF card with at least 2.5 MB of RAM (still quite small). The user downloads our distribution and copies all files over to the FAT formatted CF card. Once copied they execute syslinux against the disk which makes it boot-able. By default QBox comes up as 192.168.1.1, so a laptop or computer can be connected and login via web or ssh to change the IP and tweak the QoS settings.

3.3 How QBox Works

By basing QBox off the existing Bering uClibc branch of the LEAF open source project we could focus on the functionality needed for traffic shaping. Because traffic shaping happens at the kernel level, it simply needs to be configured and then it will continue to run. No user-space applications or services need to be executing for it to work.

3.3.1 Queue Architecture



A goal of the QBox project was to allow someone to configure QoS and queueing without knowing much if anything about queueing. Thus we needed to come up with a general queueing architecture that could be applied to most situations.

We decided on a 4 queue approach, composed of Interactive, Mid, Default, and Low priority classes of traffic. By defining them in this manner for the end user, they need to simply think of what traffic do they want at different levels. Interactive should be obvious, being VoIP or maybe Citrix traffic. Mid is for user applications like web traffic, that we want to guarantee some level of service, but also fairness. Low is for intentionally classifying traffic to a small band that will not impact other applications much if they need bandwidth. However, since QoS is dynamic, the low class can take as much as it needs so long as any other higher priority application doesn't want more bandwidth. Default is used to provide fairness to all other apps, and prevent them from being impacted by low. The default band is not configurable, except for its speed.

The user can define ports and IPs for the classes of traffic, as well as the rates for each queue. These rates are used to make sure that no one queue takes over too

much bandwidth. If a queue fills up, it will start dropping packets, which in the case of TCP packets, will throttle the other end. UDP applications will not alter their behavior based on dropped packets directly, as there is no inherent ACK in UDP. However, most current WAN applications that use UDP are interactive in nature, which gives the end application control of how to handle a lost packet rather than letting the communication layer interfere. This is why one would want interactive traffic to have enough allocated bandwidth to never drop a packet. With this assumption in mind we do not apply fairness to the interactive queue, but simple Prioritized First-In First-Out. The prioritization is based on the TCP flags. All other classes have fairness applied as they are assumed to try and consume more bandwidth than is available.

3.3.2 Transparent Bridging

We configured QBox to function as an Ethernet Bridge. This essentially turns the QBox into an in-line filter. It should be placed between the LAN and the WAN such that all LAN-WAN traffic passes through the QBox. Because it is a bridge, it can simply be put in place without changing any routing for the network.

3.3.3 Bash Scripts

All of the setup of the queues and classification parameters is accomplished with a set of Bash shell scripts. A custom init script for QBox was created that executes after the bridging is set up. The init script calls the QBox setup script

located in /sbin. The setup script looks in /etc/qbox for all of the user-modifiable config scripts and uses them to define its variables. It then loops through the different parameters for each class and builds of the classifications and associates them with the appropriate queues. Because the setup script is executed by the init script, the LRP interface provides for reloading the configuration. Thus QBox can be adjusted on-the-fly without requiring any restarts of any services. We are simply changing kernel parameters. Thus queue changes can be made while applications are running, without the user knowing.

4 Evaluation

4.1 How we tested QBox

2 criteria were used to evaluate the effectiveness of QBox. Bandwidth utilization of various classes of traffic, and packet loss. QBox was tested on both a T-1 business level connection of 1.5Mbps down / up and a home ADSL 6Mbps Down / 768Kbps up.

Testing was performed by placing VoIP calls (interactive traffic) while running a Speakeasy.net speedtest from a local server in Chicago. We would try listening to music on hold with VoIP while running the speed test. The music on hold would generate approximately 80Kbps of traffic, and the speed test would soak up the rest of the connection.

4.2 Performance

QBox performed exactly as expected. When we allocated greater than 80Kbps for our interactive band, the Speakeasy test would not impact our call. However, we found that we needed to throttle the overall connection to about 80% of our actual specified rate. Once we throttled the whole connection back we became in control of most queuing in the connection and could properly shape the traffic. We were unable to measure maximum throughput of QBox, except to verify that it could handle a 6Mbps link with very low CPU utilization. We found the behavior to be very impressive, as QBox was able to clean up a VoIP call that would have otherwise been unusable. Without any queuing in place, while running a Speakeasy bandwidth test all call attempts suffered tremendously. Once we turned on the appropriate queuing (port based classification into the interactive class), the calls changed to be perfectly clear. SSH sessions were also tested and responsiveness remained excellent.

4.3 Cost

The hardware for QBox, including the Compact Flash card can be purchased for less than \$200 USD. When we compare this with commercial products that at the \$3000 USD price range will only route 2Mbps of traffic, the price difference is dramatic. At <\$200 USD QoS becomes a viable option for many SOHO environments.

4.4 Ease of Use

Through the use of the base LEAF build we were able to integrate QBox into a straightforward web interface, and also an SSH available interface. A user simply needs to modify some parameters via a web browser and restart the QBox service and the queuing has been altered.

4.5 Comments

QBox turned out to be very successful at using Linux to create a bandwidth shaping appliance. PC Engines says that the WRAP board can route approximately 50Mbps of traffic. If we could shape that amount with this board the cost savings increase even more, as all of the commercial products charge more as you increase bandwidth. The commercial products do however provide some very rich functionality that QBox does not, and thus QBox is not going to be for everyone.

5 Conclusions and Future Work

5.1 Interactive applications need traffic shaping

We were able to demonstrate that without traffic shaping, applications such as VoIP suffer horribly when other network traffic competes for the use of the bandwidth.

5.2 Traffic Shaping on Linux provided reliability to interactive applications

By using an embedded platform and Linux, we were able to provide an easily configurable QoS appliance at a fraction of the cost of commercially available solutions.

5.3 QBox is a viable low-cost alternative to commercial QoS appliances

A combination of Open-Source technologies with embedded hardware has enabled us to build a low cost solution to what was previously only available to large corporations. As the use of interactive applications grows, need for QoS will grow. QBox can help to fill this need.

5.4 Next steps

Currently, while greatly simplified, QBox still requires that users edit files. We would like to build a custom interface for QBox as opposed to the standard LRP interface. This would allow for more guided user input and input validation. Testing could be greatly improved through the setup of some tools to generate different traffic situations. We would like to add support for additional boards like those made by Soekris or Mikrotik. However, with the additional hardware comes additional support and more room for the project to get difficult to maintain.

References

- [1] Packeteer <http://www.packeteer.com>
- [2] Emerging Technologies ET/R1800G Network Appliance http://www.etinc.com/product_info.php?products_id=88860
- [3] uClibc Compiler for Embedded Environments <http://www.uclibc.org/>
- [4] Bering uClibc LEAF Distribution <http://leaf.sourceforge.net/bering-uclibc/>
- [5] Linux Advanced Routing and Traffic Control <http://lartc.org/>
- [6] FreeBSD's altQ Project <http://www.rofug.ro/projects/freebsd-altq/>
- [7] LEAF - Linux Embedded Application Firewall <http://leaf.sourceforge.net/>
- [8] Burt Hubert's Wondershaper [tp://ds9a.nl/](http://ds9a.nl/)
- [9] m0n0wall <http://m0n0.ch/wall/>
- [10] Soekris Engineering <http://www.soekris.com/>
- [11] PC Engines <http://www.pcengines.ch/>
- [12] Shorewall Firewall <http://www.shorewall.net/>
- [13] Linux Router Project <http://freshmeat.net/projects/linuxrouterproject/>