voip - Prijedlozi #15206

miracleshaper - QoS

28.08.2008 17:13 - Ernad Husremović

Status:	Odbačeno	**Početak:**	28.08.2008
Prioritet:	Normalan	**Završetak:**	
Odgovorna osoba:	Ernad Husremović	**% završeno:**	0%
Kategorija:		**Procjena vremena:**	0.00 sat
Ciljna verzija:			

Opis

http://mshaper.florz.de/

Historija

#1 - 28.08.2008 17:15 - Ernad Husremović

- Odgovorna osoba postavljeno na Ernad Husremović

Miracleshaper

Contents

News

2008-07-08

```
Version 0.3 released. Does ECN marking on IPv6, plus includes prioritization of interactive SSH and TCP ACKs o
ver IPv6 in the default classifier, too, now (the latter probably doesn't work with out of the box OpenSSH, as
 it AFAIK doesn't do traffic class marking on IPv6). Also fixes a minor bug in the default classifier that cau
sed the maximum allowed downstream bandwidth to be increased rather than reduced in case of SSH traffic—makes
 you wonder whether that mechanism really is all that important ;-)
```

What is it?

The Miracleshaper is a traffic shaper and prioritizer (the latter being the primary purpose) designed specifically for DSLs, in particular the variety using PPPoE on top of G.922.1. A short overview of the distinctive features—most are explained in more detail below:

- Processes both up- and downstream.
- Accounts for the encapsulation overhead on DSLs.
- Achieves MTU granularity for IP packets bigger than the MTU.
- Does automatic configuration parameter switching based on the encountered traffic.
- Allows for joint shaping of multiple PPPoE sessions.
- Enables the use of multiple PPPoE sessions on a single DSL even when only one session per MAC address is allowed.
- Can do ECN marking.
- Works in userspace, interfaces to the kernel networking code through tap interfaces and a packet socket.
- Is configured in the sourcecode, so you should know a little C if you want to use it.
- Was written on i386 Linux, but should be easily portable to other architectures as well as operating systems.

Processes both Up- and Downstream

Well, it basically says it all. There is one queue per direction, and one rate limiter ("shaper") per queue that dequeues packets from its respective queue in priority order at the configured data rate.

In principle, the operation of up- and downstream is symmetric, but the configuration usually is not, as the goal of low latency for certain kinds of traffic can be reached by slightly different means in either direction, due to the fact that we are before the bottleneck in the upstream and behind it in the downstream. For details regarding the differences, and in particular on how the low latency on the downstream is achieved, see below.
Accounts for the Encapsulation Overhead on DSLs

So, why a shaper and prioritizer specifically for DSLs, you ask? Well, the usual PPPoE based DSL uses a stunning number of encapsulation layers:

The IP packet (which is what we actually want to transfer) is first encapsulated in PPP (adds 2 bytes), then that's wrapped inside PPPoE (another 6 bytes), which in turn is put into an Ethernet frame (14 bytes, or even 18 bytes if the Ethernet checksum trailer is also transferred over the DSL), after which a layer of RFC 2684 LLC encapsulation is applied (10 bytes), the result of which then is packaged using AAL5 into ATM frames (which adds another 8 bytes, then rounds up the size to the next multiple of 48 and finally adds another 5 bytes per 48 bytes of the result).

This not only means that quite a bit of the "raw bandwidth" of the line is consumed by protocol overhead, which most traffic shapers could be accommodated to by simply setting their shaping rate to the bandwidth actually available for payload, but also that the bandwidth available for payload heavily decreases when small packets are being transferred, without this being visible to the traffic shaper (it doesn't see any of the overhead that's being added between the DSL modem and the access concentrator). This is particularly bad, as those packets that need the lowest latency tend to be the smallest (like VoIP, SSH, or TCP ACKs). So if you have a VoIP call going on, for example, the available bandwidth decreases. If you have not taken that into account when setting the rate of the shaper, that will cause the queue in your DSL modem to fill up and thus the latency to increase. If you have taken that into account, on the other hand, you would have only relatively little bandwidth available for however-big packets you want to transfer even when there is no VoIP call going on.

So, the obvious solution employed by the Miracleshaper is that it computes how much bandwidth a given packet does actually consume on the DSL, including all the overhead, and does its shaping based on that.
Achieves MTU Granularity for IP Packets Bigger than the MTU

As the Miracleshaper processes link-layer frames rather than network-layer packets, it in particular sees IP packets when they are already fragmented. This means that if someone, for some reason, decides to send an IP packet that's larger than the PPP interface's MTU, the Miracleshaper can let slip through some high-priority packet in between the fragments of the oversized packet, thus guaranteeing that the line cannot be blocked by low-priority traffic for longer than one MTU worth of data takes to transfer. This in particular is a potential problem with dsl_qos_queue, which does the prioritization at the network layer.
Does Automatic Configuration Parameter Switching Based on the Encountered Traffic

An additional feature that allows for better utilisation of the downstream bandwidth is automatic parameter switching upon encountering certain kinds of traffic. For example, the Miracleshaper can automatically reduce the downstream shaping bandwidth and maximum queue size when encountering upstream VoIP traffic, and also automatically switch back to full bandwidth and maximum queue size when no such traffic has been seen for a certain amount of time. Why this is particularly useful for the downstream is explained below.
Allows for Joint Shaping of Multiple PPPoE Sessions

In contrast to shaping done separately on each PPP interface, the Miracleshaper also can prioritize, for example, VoIP packets on any of a number of sessions over the packets from an ISO download on any other session.
Enables the Use of Multiple PPPoE Sessions on a Single DSL even when only One Session per MAC Address is Allowed

The Miracleshaper can create any number of tap interfaces with a different MAC address each that can be used for establishing multiple PPPoE sessions from different MAC addresses on lines that allow only one session per MAC address, such as T-Com's T-DSL, without the need for any patches for RP-PPPoE, and even the kernel PPPoE implementation can be used with sufficiently recent kernels (2.6.21-rc3 or newer).

Can Do ECN Marking

In case the queue overflows, the Miracleshaper can ECN-mark packets instead of dropping them in case an ECN-capable transport is indicated by the packet.

Was Written on i386 Linux, but Should be Easily Portable to other Architectures as well as Operating Systems

Basically, if the operating system provides tap interfaces and packet sockets, porting should be easy. The program uses the RDTSC instruction as its timing source, which would need to be replaced by its equivalent on other architectures—or possibly a call to gettimeofday() in case no such instruction exists.

How does it work?

The Upstream

In the upstream, things are simple and reliable: You have to send packets no faster than the modem will transmit them, so the modem's queue will stay empty. You then can select the packet with the highest priority from the local queue whenever the line is free for the next packet and it will get transmitted by the modem without any additional queueing delay.

The Downstream

In the downstream, things are a bit different, obviously: If someone decides to flood you with low-priority packets faster than your DSL is able to move them to your end of the line, where the Miracleshaper does its job, the queue at the far end is going to fill up and to thus cause any high-priority packets to be delayed as well. With most providers, there is nothing you can do about that. So, the only hope is that we might be able to somehow influence well-meaning senders to send traffic in such a way as to avoid the far-end queue filling up.

Assuming that most bulk traffic that could cause the far-end queue to fill up is TCP traffic, the Miracleshaper is based on how this can be done in case of TCP—but due to similar operating principles, this should work with other bulk-transfer protocols, too. Now, in case of TCP, there are two basic ways to influence the sender's transmission behaviour that don't require any state of each connection to be kept:

The first is that we can delay acknowledgements, which signals to the sender that it's approaching the route's bandwidth and thus causes it to reduce the rate at which it's increasing its transmission speed. Instead of actually delaying the acknowledgements, we just as well can delay the data fragments that are going to cause the acknowledgements to be sent in the first place. This basically means nothing else than that we are limiting the bandwidth—the TCP sender will automatically adapt its sending speed to the bandwidth limit that causes the data and thus the acknowledgements to

be delayed. As the average queue size at the far end of the DSL clearly correlates with the ratio between its filling and its emptying speed, we can reduce the average queue size by simply limiting the bandwidth.

What that does not help with, though, is the burstiness of the traffic: The average bandwidth will be the same, whether packets arrive at the queue one at a time, every milliseond, or a thousand in one big burst, every second. But the burst obviously causes the queue to fill up further than the packet every millisecond. That is where the second option comes in: We can drop packets, which, by the way of missing acknowledgements, signals to the sender that there is a congestion and thus causes it to back off in order to try to dissolve the congestion. The effect of this backoff on transmission speed is intended to be temporary only, if any—the purpose is to reduce the number of packets in-flight in the network so as to not overload the queues within the network and to thus avoid packet drops. So it is basically intended to reduce the number of packets in-flight to the minimum that is necessary in order to still maintain the maximum possible transmission speed given the latency of the route. The optimum in this respect obviously is reached when the fragments are being sent equally-spaced over time—or in other words: When the traffic as un-bursty as possible. Which luckily coincides with what we want. The easiest way to achieve this is to limit the queue size and drop any packets that don't fit into the queue. That way, given that the input rate to the queue is sufficiently larger than the output rate, any bursts (much) larger than the queue size cause packets to be dropped and thus the sender to back off. That is why the Miracleshaper reduces the shaping bandwidth as well as the maximum queue size in the downstream when it encounters high-priority packets.

The same effect as by dropping a packet can be achieved by marking packets using early congestion notification (ECN) without wasting bandwidth for dropped packets, as the sender is required by the RFC to behave as if a packet had been dropped when it sees ECN marked packets, which is why the Miracleshaper is capable of ECN marking. Of course, this only works with flows where the IP header indicates an ECN-capable transport, which in turn requires ECN to be enabled on both endpoints of the flow.

Experience

I am able to make decent VoIP calls while downloading at 100 KiB/s and at the same time uploading at about 15 KiB/s on my 2 Mbps/192 kbps DSL with interleaving activated (RTT to the access concentrator is 45 ms). I have not tested this with many parallel TCP connections, like with peer to peer file sharing, and I doubt that it works well, but I'd be interested in hearing from anyone who has tried. Also, when no VoIP call is going on, I can do a download at 230 KiB/s in parallel with an upload at 20 KiB/s.

Installation and Configuration

All options are compile-time options at the moment, which is why I don't offer any pre-compiled binaries. The major reason for this is that I would have to make up some configuration syntax that allows for the specification of packet filters for the classification of packets—and then even implement that. For the upstream, a simple solution would be to make the kernel packet marker (former nfmark/fwmark) available through the tap interface, then one could simply use iptables to do the classification. But that unfortunately does not solve the problem for the downstream …

So, you will have to edit the config.h for general configuration (documentation can be found in the file itself) and the classifiers.c for the two packet classifier functions for up- and downstream, respectively, as well as for the so-called profiles that implement the automatic configuration parameter switching. There are numerous functions available from pkt.c and pkt.h for the dissection of packets—and it shouldn't be difficult to add new ones as needed. My own classifier functions are included as an example.

You probably should use the Miracleshaper with a kernel with HZ==1000. Of course, you need tun/tap device support (CONFIG_TUN) and packet socket support (CONFIG_PACKET) in the kernel.

Once your configuration is complete, a simple make should build the mshaper binary for you—that single binary is all you really need. If you want to have a Debian package (which also includes an init script), you can use dpkg-buildpackage instead. By default, the program will daemonize when started, use the -d option if you want to prevent that.

The Miracleshaper creates persistent tap interfaces, so you can kill and restart the program without anyone noticing (except for a few lost packets, maybe). Should you ever want to get rid of any of the persistent tap interfaces, the -r command line option might be useful. It allows you to specify one interface that is to be removed.

#2 - 28.08.2008 17:16 - Ernad Husremović

- Fajl mshaper_0.3.tar.gz dodano

#3 - 03.09.2008 11:02 - Ernad Husremović

- Naslov promijenjeno iz miracleshaper u miracleshaper - QoS

#4 - 30.05.2010 11:07 - Ernad Husremović

- Status promijenjeno iz Dodijeljeno u Odbačeno

Fajlovi

mshaper_0.3.tar.gz	20 KB	28.08.2008	Ernad Husremović