

# loaded

Sebastian Kraemer *kraemer@cs.uni-potsdam.de*

19. Juli 2004

## Zusammenfassung

In diesem Paper wird eine load-balancing Alternative für das Linux Betriebssystem vorgestellt, die anders als bereits vorhandene Lösungen (LVS) nicht im Kern sondern im Userspace läuft und als erste freie Implementierung für Linux auch das Balancing von IPv6 erlaubt.

## 1 Einleitung

Im Rahmen des IPv6 Labors, einer Veranstaltung zum Thema IPv6 an der *Universität Potsdam* befasste sich eine Gruppe von Studenten mit dem Thema *load balancing* in Zusammenhang mit dem IP Protokoll Version 6. Da die zu der Zeit verfügbaren Implementierungen keine Unterstützung für IPv6 anboten, entschloss ich mich kurzerhand einen eigenen Lastverteiler zu implementieren der auch IPv6 handhaben kann.

Es wird im folgenden kurz die bereits existierende Implementierung Linux Virtual Server (LVS) vorgestellt aus der sich eine Menge von Fähigkeiten ableiten wird, die ich in meiner eigenen Implementierung zu berücksichtigen hatte, um ein konkurrenzfähiges Programm zu erschaffen.

## 2 Begriffe

Um eventuelle Missverständnisse auszuschliessen werden hier kurz einige Begriffe erläutert die im weiteren noch auftauchen werden. Da der Leser eventuell etwas abweichende Vorstellungen von diesen Begriffen hat, und sich eventuell über deren Gebrauch hier wundern könnte, erschien mir eine kurze Erläuterung sinnvoll. Es wird jedoch grundlegendes Verständnis über den Sinn und die generelle Funktionsweise von Load Balancern vorausgesetzt.

- Virtuelle IP (VIP)

Eine IP Adresse die nicht wirklich an einen Rechner gebunden ist um diesen anzusprechen (zum Bsp per HTTP), sondern von einem Stück Software so verwaltet wird, das es so aussieht als sei sie einem Rechner fest zugewiesen. Diese

Software kann dann entscheiden was sie mit den Paketen die diese VIP tragen geschieht. Die Software könnte zum Beispiel die Adressen des Paketes basierend auf bestimmten Regeln umschreiben und die Pakete weiterleiten.

- **Load Balancer (Lastverteiler)**  
Eine Software die Pakete die für sie bestimmt sind nach einer bestimmten Strategie an andere Rechner verteilt. Eine Strategie könnte sein, die neu ankommenden Pakete dem Rechner mit der niedrigsten Last zuzuweisen. Lastverteiler sind oft so implementiert dass sie eine oder mehrere VIPs verwalten (siehe oben) und die Pakete die an diese VIP gerichtet sind an real existierende Rechner weiterleiten.
- **Backend**  
In diesem Zusammenhang Rechner die sich hinter einem Lastverteiler verbergen, also diejenigen Rechner an die der Lastverteiler die Pakete zum eigentlichen Dienst (zum Bsp. HTTP) weiterleitet.

### 3 Linux Virtual Server

Beim Linux Virtual Server handelt es sich um einen Kernelpatch, der bei einigen Distributionen wie zum Bsp. SuSE Linux bereits integriert ist. Ist dies nicht der Fall muss ein eigener Kernel mit entsprechendem LVS Patch neu kompiliert und installiert werden. Die Konfiguration des *LVS* geschieht im Userspace mittels des *ipvsadm* Befehls. Mittels der Kommandozeile kann der Administrator Virtuelle IPs verwalten und Strategien festlegen nach denen die Lastverteilung geschieht. So legt er zum Beispiel mittels

```
linux:~ # ipvsadm -A -t 1.2.3.4:80 -s rr
```

fest, dass eine neue VIP 1.2.3.4 zu existieren hat von der aus HTTP Pakete im Round Robin Verfahren zu verteilen sind. Die Rechner an die diese Verteilung gerichtet ist sind danach anzugeben:

```
linux:~ # ipvsadm -a -t 1.2.3.4:80 -r 192.168.0.1:80 -m
linux:~ # ipvsadm -a -t 1.2.3.4:80 -r 192.168.0.2:80 -m
```

usw. Hier werden die Rechner mit den IPs 192.168.0.1 und 192.168.0.2 als Backends festgelegt. *LVS* erlaubt mehrere Scheduling Strategien. Dies sind zur Zeit: Round Robin, Weighted Round Robin, Least Connection, Weighted Least-Connection, Locality-Based Least-Connection, Locality-Based Least-Connection with Replication, Destination Hashing und Source Hashing.

Es fallen also positiv auf: die relativ einfache Konfiguration und die grosse Vielfalt an Scheduling Strategien. Negativ zu nennen sind der Aufwand zum neuen Erstellen eines Kernels (falls nötig), die fehlende IPv6 Unterstützung und die eventuellen Risiken<sup>1</sup> die Implementierungen im Kern immer mit sich bringen.

<sup>1</sup>Bei fehlerhaften Implementierungen kommt es mit Sicherheit zu einem Kernel Ooops oder dergleichen, während bei einer Userspace Implementierung maximal der Daemon beendet wird

## 4 netfilter

Seit dem Kern 2.4 benutzt Linux *netfilter* um IP Pakete anhand bestimmter Regeln zu filtern. Der Befehl zum Erstellen und Verwalten dieser Regeln nennt sich *iptables*. Die Möglichkeiten die der Kern mit *netfilter* bietet sind sehr umfangreich. Pakete können unter anderem akzeptiert, verworfen, beantwortet oder umgeschrieben werden. Die Regeln nach denen dies geschieht werden mittels sogenannter Targets festgelegt. Ein für mich sehr interessantes Target ist QUEUE. Pakete, die an dieses Target gerichtet sind, werden vom Kernel an ein Userspace-Programm weitergeleitet welches die Pakete entgegennehmen, beliebig modifizieren, und dann dem Kernel zurückgeben kann. Eine Neuinstallation/Konfiguration des Kernels ist hierzu in der Regel nicht notwendig. Das Modul *ip\_queue* sollte bereits vorhanden sein:

```
linux:~ # modprobe ip_queue
linux:~ # cat /proc/net/ip_queue
Peer PID           : 0
Copy mode          : 0
Copy range         : 0
Queue length       : 0
Queue max. length : 1024
linux:~ #
```

Ist dieses Modul geladen, ist das QUEUE Target verfügbar. Das *iptables* sourcepaket enthält eine Bibliothek mit der man auf die Pakete die an das QUEUE Target gerichtet sind zugreifen kann. Diese nennt sich *libipq* und bot sich zusammen mit dem QUEUE Target förmlich an, bei einem Load Balancer mitzuwirken.

Im obigen Beispiel ist der Load Balancer nicht aktiv, was an der Peer PID von 0 zu erkennen ist. Auf einem laufenden System könnte die Ausgabe wie folgt aussehen:

```
linux:~ # cat /proc/net/ip_queue
Peer PID           : 1328
Copy mode          : 2
Copy range         : 65535
Queue length       : 0
Queue max. length : 1024
```

Der *loaded* Daemon hat hier die PID 1328 und kann Pakete bis zu einer Länge von 65535 Byte entgegennehmen. Dies entspricht der maximalen Länge eines IP Paketes. Existiert die Datei */proc/net/ip\_queue* nicht, so ist das QUEUE Target nicht aktiv und das dazugehörige Modul muss noch geladen werden.

## 5 loaded

Die grundlegende Idee die hinter dem Userspace Lastverteiler steckt, ist das QUEUE Target von *netfilter*. Pakete die an eine VIP<sup>2</sup> gerichtet sind, werden an das QUEUE Target und damit an den Daemon weitergeleitet, der anhand seiner Konfiguration feststellt wie die Adressen umzuschreiben und die Pakete weiterzuleiten sind. Effektiv sind also nur folgende Schritte zum Balancing notwendig, wenn man auf das QUEUE Target aufsetzt:

1. Entgegennehmen des Paketes  
Der Daemon nimmt ein Paket entgegen sobald es eintrifft. Dafür steht durch die *libipq* eine Funktion zur Verfügung die in einem langsamen Systemaufruf solange blockiert bis ein Paket eingetroffen ist.
2. Richtungsfeststellung  
Der Daemon untersucht nun ob das just entgegen genommene Paket an die VIP gerichtet ist (also von aussen kommt) oder bereits ein Antwortpaket aus dem Backend-Netz ist. In letzterem Fall ist lediglich die Absenderadresse des Pakets durch die VIP zu ersetzen und das Paket an den Kernel zurückzugeben.
3. Verteilungsentscheidung  
Ist das Paket an die VIP gerichtet und wurde bereits ein Paket von derselben Absenderadresse einmal verarbeitet, steht das Backend das dieses Paket zu verarbeiten hat bereits fest: es muss sich um dasselbe Backend handeln um die Konsistenz zum Bsp. einer TCP Verbindung sicherzustellen. Pakete die zu einer TCP Verbindung gehören dürfen auch nur von einem Backend behandelt werden. Diese Zuordnung merkt sich der Daemon mittels einer Tabelle im Speicher. Sobald er feststellt dass ein Paket vorliegt von einer Adresse die er noch nie gesehen hat, entscheidet er mittels des (konfigurierbaren) Scheduling an welches Backend dieses Paket zu gehen hat, trägt die neue Zieladresse in das Paket ein und vermerkt das entsprechende Backend in der Tabelle um folgende Pakete richtig zu verarbeiten.
4. Abgabe  
Sind alle Adressen umgeschrieben und die Prüfsummen der Header neu berechnet worden, wird das Paket an den Kernel zurückgegeben der anhand seiner Routingtabelle entscheidet auf welchem Interface das Paket hinauszusenden ist.

Loaded (wie sich der Daemon nennt) verarbeitet dabei jeglichen IPv4 und IPv6 Verkehr (also sowohl UDP, ICMP und TCP). Es können aber auch selektiv zum Beispiel nur HTTP oder nur DNS Verkehr verarbeitet werden. Die Entscheidung hierfür wird mittels *iptables* gefällt, mit dem der Administrator festlegt welche Pakete an das QUEUE Target (und damit an den Daemon) zu richten sind:

---

<sup>2</sup>Welche durch den `-d` switch in `iptables` festgelegt ist

```
linux:~ # iptables -t nat -A PREROUTING -p tcp -d 1.2.3.4 -j QUEUE
```

Leitet zum Beispiel sämtlichen TCP Verkehr der an die Adresse 1.2.3.4 gehen soll an den Daemon weiter. Es handelt sich bei 1.2.3.4 also um eine VIP. Es können hier alle Feinheiten von *netfilter* genutzt werden die es so gibt, also das Eingrenzen auf spezielle Ports oder Adressbereiche. Da *netfilter* und das QUEUE Target auch für IPv6 zur Verfügung stehen, funktioniert das ganze analog mit IPv6, lediglich der Daemon muss IPv6 Pakete anders handhaben als IPv4 Pakete (die Adressen im Header sind anders umzuschreiben).

## 5.1 Architektur

*loaded* besteht aus 4 Komponenten deren Zusammensetzung in folgender Grafik verdeutlicht ist. Das Parsen der Konfigurationsdatei scheint mir zu unwichtig um es genauer zu erklären. Das Packet-handling läuft zusammen mit dem Scheduler in einem eigenen Thread. Dieser erledigt die 4 Arbeitsschritte die bereits ausführlicher beschrieben wurden. In einem anderen Thread versucht *loaded* durch ansprechen eines (konfigurierbaren) Dienstes herauszufinden, ob die Backends noch aktiv sind oder neue hinzugekommen sind. So hat der Scheduler stets eine aktuelle Liste aller verfügbaren Backend Knoten und kann nicht antwortende Backends ignorieren oder neu aufgetauchte in seine Scheduling-Liste mit aufnehmen.

Als Scheduling Strategien stehen momentan Round Robin und Weighted Round Robin zur Verfügung.

## 5.2 Konfiguration

Die Konfiguration geschieht wie unter UNIX üblich mittels Textfiles. Per default wird die Konfigurationsdatei `loaded.config` im aktuellen Verzeichnis geladen. Im IPv6 Fall heisst diese Datei `loaded.config6`. Im Anhang befindet sich ein HOWTO, in der das Aufsetzen und die Konfiguration von *loaded* ausführlich beschrieben ist.

## 5.3 Vor- und Nachteile

Die Implementierung eines Load Balancers im Userspace bringt natürlich den Nachteil mit sich, dass jedes eintreffende Paket aus dem Kernspace in den Userspace und zurück kopiert werden muss. *LVS*, als Kernelpatch konzipiert, erspart sich diese Arbeit natürlich und gewinnt dadurch an Effizienz. Erstaunlicherweise jedoch ist der Zeitverlust durch das Kopieren der Pakete in den Messungen (siehe weiter unten) gar nicht merklich hervorgetreten. Auch die Anbindung nach 'ausen' mittels eines GBit Netzes brachte keine Ergebnisse aus denen ein klarer Zeitverlust abzulesen war. Eventuell lag es an den verwendeten Messstrategien. Die Messdaten könnten im Rahmen eines wissenschaftlichen Projektes um Latenzzeiten, Anzahl existierender Verbindungen usw. erweitert werden.

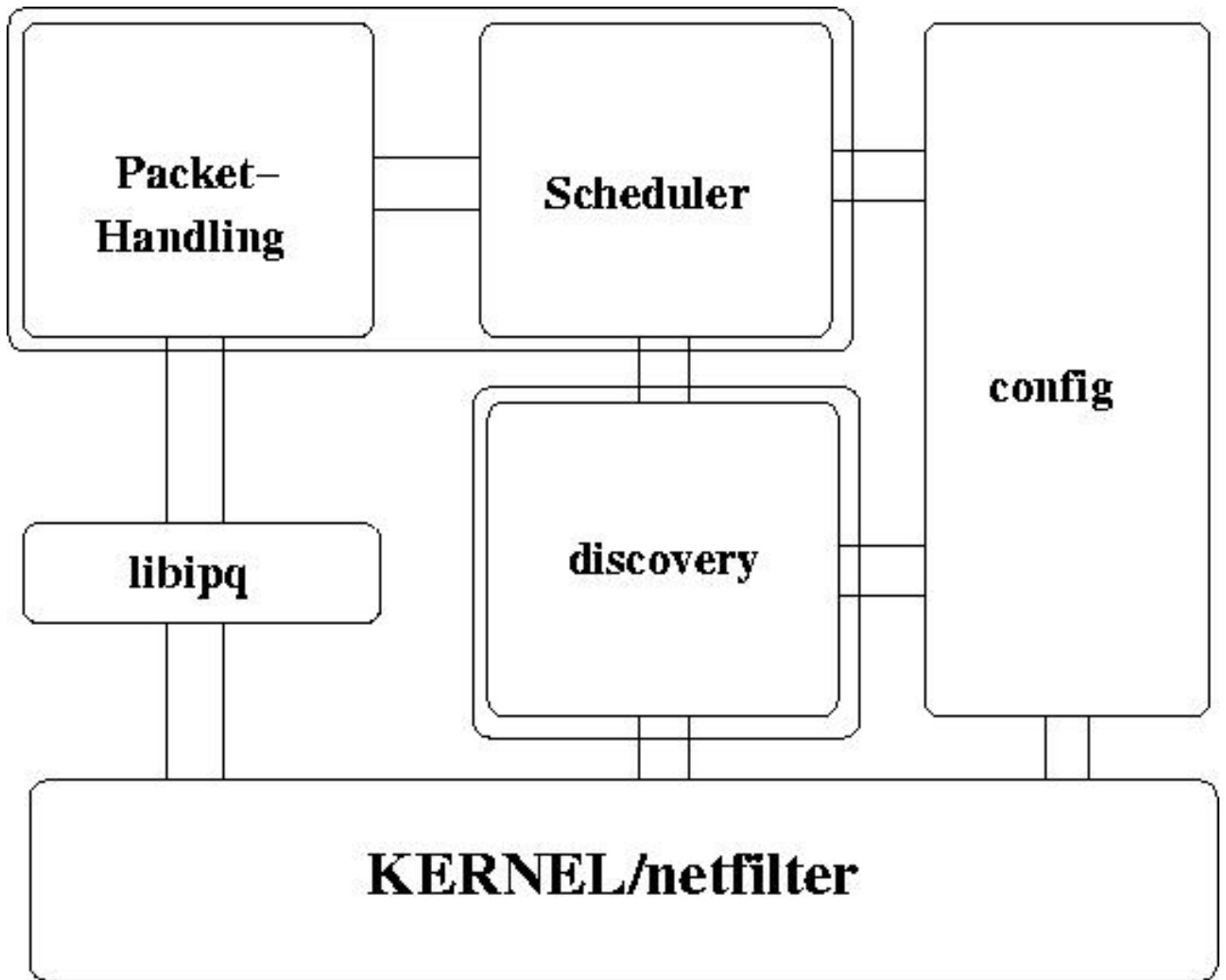


Abbildung 1: *loaded* Architektur.

Der Linux Kern unterstützt seit einiger Zeit Zero-Copy Mechanismen mit denen mittels `mmap()` direkt auf im Kernel befindliche Pakete zugegriffen werden kann ohne diese in andere Puffer kopieren zu müssen. Eventuell bietet sich an *libipq* in dieser Hinsicht zu erweitern um einen Nutzen daraus zu ziehen. Eine Userspace Implementierung hat dann im Gegensatz zu einer Implementierung im Kernel keinen Nachteil mehr (obwohl wie gesagt bei den Messungen auch so schon kein nennenswerter Zeitverlust auftrat).

Ein eindeutiger Vorteil von Implementierungen im Userspace (neben der Einfachheit) liegt in der Stabilität. Der Daemon kann in einem chroot laufen (eventuell auch als nobody User) und stellt dann selbst im Falle eines Programmfehlers kein Sicherheitsrisiko dar.

## 5.4 Probleme

Eventuell auftretende 'Probleme' könnten DNS Anfragen aus dem internen Backend Netz nach aussen sein die der Balancer fälschlicherweise als Antwortpakete deuten könnte. Demzufolge sollte mittels *iptables* festgelegt werden das UDP Pakete mit Zielport 53 aus dem Backend Netz nicht vom Balancer zu behandeln sind sondern über das MASQUERADING Target laufen sollen:

```
linux:~ # iptables -t nat -A POSTROUTING -o eth1 -p udp \\  
--dport 53 -j MASQUERADE
```

Alternativ könnte der Dienst, der balanciert wird, so konfiguriert werden DNS reverse lookups zu vermeiden und stattdessen IP Adressen zu loggen. Eine nicht unübliche Konfiguration, da dies auf stark belasteten Servern ohnehin effizienter ist.

Beim Lastverteilen von IPv6 ist zu beachten, dass die Adressauflösung für Schicht 2 nicht mehr mittels ARP sondern mittels ICMPv6 funktioniert. Es ist also davon abzuwachen bei IPv6 ICMP Verkehr zu balancieren da dies mit den *neighbor discovery* Nachrichten von ICMPv6 kollidieren könnte.

Auf der Firewall die sich vor jedem Firmennetz (und damit auch vor Netzen in denen Lastverteiler aktiv sind) befinden sollte, sind natürlich 'komische' Pakete herauszufiltern. Dies schliesst das reassembling von IP-Fragmenten ein und ist sowohl vom Sicherheitsstandpunkt als auch von der Effizienz her anzuraten.

## 5.5 Messungen

Loaded wurde einem Stresstest unterzogen um festzustellen ob es sich für den Einsatz in der realen Welt eignet. Das Setup für die Messungen war wie folgt:

- Übertragung von 488 MB Daten per **wget** (HTTP) im 100 MBit LAN per IPv4.
- „Stress-Test“ durch Anbindung des Balancers an ein GigaBit-Netz.

Clients	Client 1	Client 2	Client 3	Client 4	Mittelwert
1	48,5s				48,5s
2	57s	78s			67s
3	112s	116s	123s		117s
4	145s	187s	166s	158s	164s

Abbildung 2: Übertragungszeiten für 488MB.

- Balancer: Linux PC, P3 700 MHz
- Backends:
  - 1x Linux PC, Dual P3 800 MHz
  - 2x Sun Blade 150

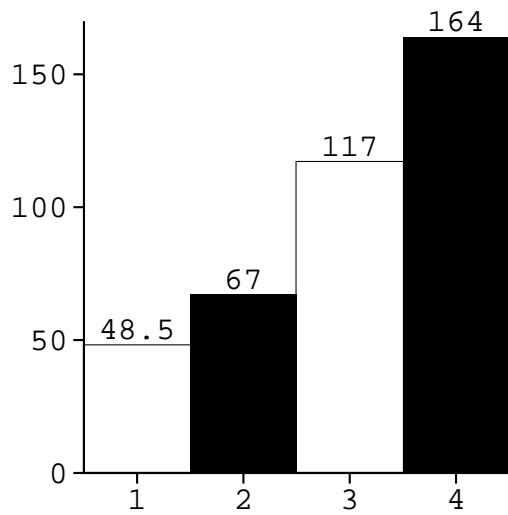


Abbildung 3: Durchschnittliche Übertragungszeit für 488MB.

Clients	IPv4
1	<b>47s / 10,39 MB/s</b>
2	68s / 14,35 MB/s
3	119.66s / 12,23 MB/s
4	<b>160.25s / 12,18 MB/s</b>

Abbildung 4: Durchschnittliche Übertragungszeit für 488MB.

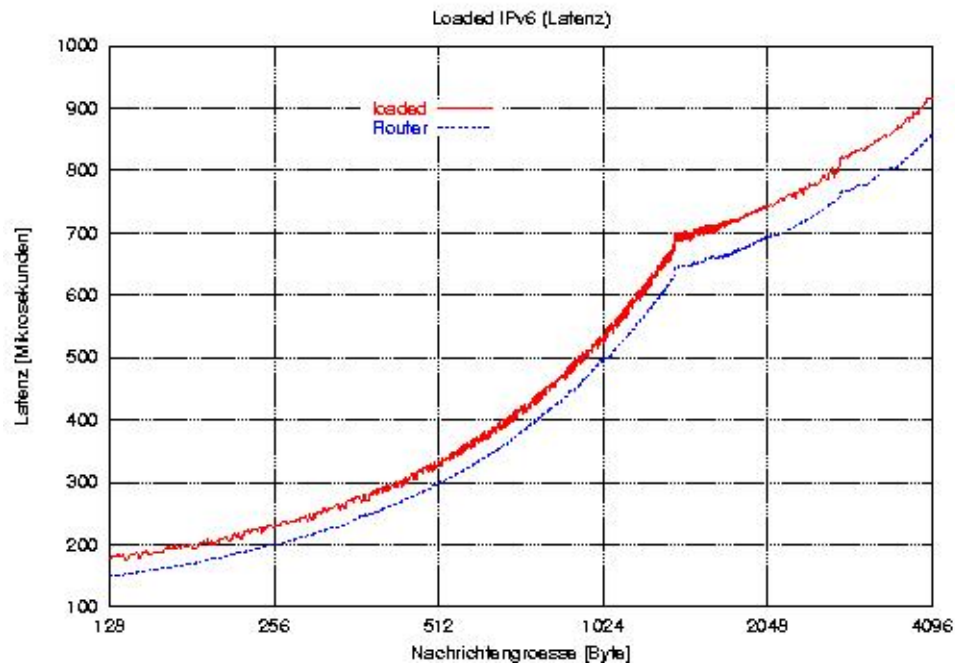


Abbildung 5: IPv6 Latenzzeiten

Desweiteren wurden von Sven Friedrich mittels eines Universitatseigenen Sockping Programms Latenz und Bandbreitenmessungen durchgefuhrt. Jeweils einmal ohne Lastverteiler, einmal mit *loaded* als Lastverteiler und einmal mit *LVS*.

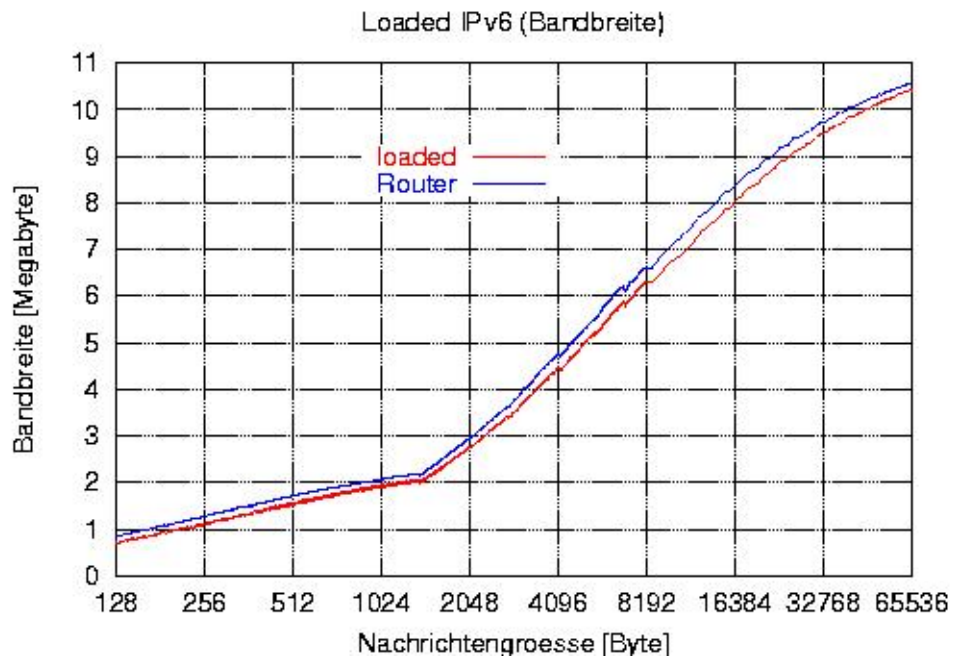


Abbildung 6: IPv6 Bandbreite

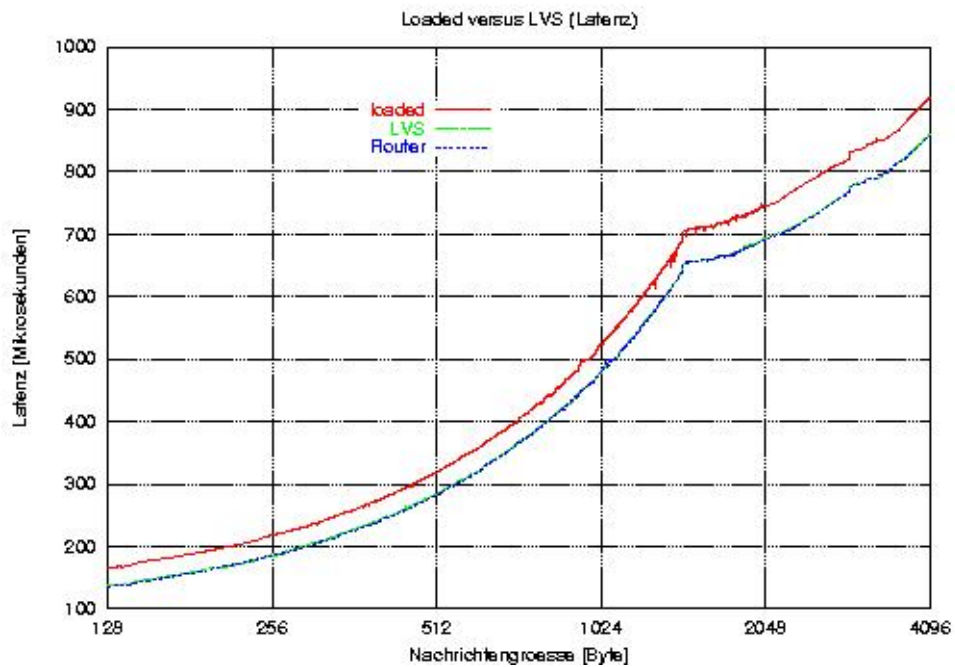


Abbildung 7: Latenzvergleich Router/loaded/LVS

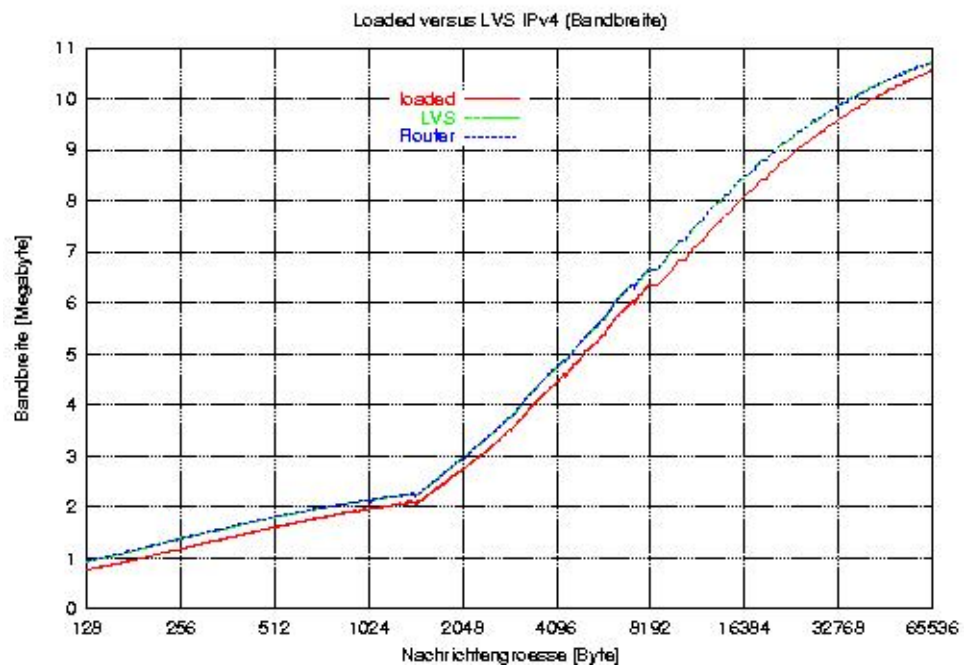


Abbildung 8: Bandbreitenvergleich Router/loaded/LVS

## Literatur

- [1] Linux: <http://www.linux.de/linux/>
- [2] LVS: <http://www.linuxvirtualserver.org>
- [3] Richard W. Stevens, *TCP/IP Illustrated Vol 1, Addison Wesley*
- [4] loaded: <http://sefault.net/research/networking>

## HOWTO

Das HOWTO ist in englischer Sprache. Aus sicher verständlichen Gründen (es liegt auch dem *loaded*-tarball bei) ist es besser die ausführliche Installationsanleitung in Englisch zu geben, da die Software ja nicht nur in Deutschland eingesetzt werden soll.

Step by Step guide for loaded installation (IPv4)  
 =====

### 1. Build -----

You can use the pre-build binary if you want. It should work.  
 If you want to build your own binary, install the libipq from the iptables sources. You have to build the libipq.a and copy it to /lib. Also copy the libipq subdir to /usr/include. Then you can "make" the loaded binary.

### 2. Ensure your kernel is ok -----

loaded requires netfilter and the QUEUE target. To look whether netfilter is properly installed on your system type

```
iptables -L
```

If you get a list like

```
Chain INPUT (policy ACCEPT)
target     prot opt source      destination
...
```

everything is fine. If not, install a netfilter-capable kernel or load the appropriate modules. To look whether the QUEUE target is supported at your system type

```
modprobe ip_queue
```

If the file "/proc/net/ip\_queue" appears in your proc filesystem then everything is ok, otherwise install a QUEUE capable kernel or appropriate modules.  
 Additionally the "ip", "route" and "ifconfig" programs must be installed (net-tools.rpm and iproute2.rpm are good choices).

### 3. Configuration -----

On the load-balancer edit loaded.conf. Replace the "VIP4" and the other variables with your own values. The comments should help you. The "RIP4" contains the network address of your backend nodes. 10.0.0.0/24 will work fine. The same for the "private\_GW" and the "server4" variables. Each value for a "server4" has to be assigned to a backend node. For example if you have

```
server4=10.0.0.1      # first IPv4 balanced server
server4=10.0.0.2      # second IPv4 balanced server
server4=10.0.0.3
```

you need 3 backends, and the corresponding IPs assigned to it. It will also work if you assign all the 3 IPs to one interface on one backend if you just want to test loaded and dont have 3 nodes handy. The default route on each backend node has to match with the "private\_GW" in loaded.conf. In our example, if you have

```
private_GW=10.0.0.254
```

the backend nodes default route can be set via:

```
route add default gw 10.0.0.254
```

on each node. You should use your distributions network-configuration tool to do this if you want to have this setup staying across reboots.

NOTE: On the backends you have to set the appropriate broadcast address ("broadcast" variable) if you want to use automatic node-discovery and failover. Also you have to ensure that they answer broadcast pings which is the default setup. You can enable it by hand (if it is disabled for some reason) with the

```
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

command. On the load balancer issue the command

```
ping -b <your_broad_cast_address> for example
ping -b 10.0.0.255
```

This shows you whether the backend nodes are replying. (You will see DUP packets for each additional node).

You have to tell loaded an interface where the virtual IP (VIP4) and the public gateway address (public\_GW) is assigned. This is done using the "public\_NIC" variable. Usually "eth0" will work. The traffic from the internet will arrive on this interface. The packets are translated and then written out using the "private\_NIC" interface. It can be the same interface as "public\_NIC" depending on your network topology. The "public\_GW" field might be uninteresting for your internet-router, but assign it in the config-file nevertheless. It can be used to set up routes for the virtual IP via this "public\_GW". But usually no special routes need to be set up on your internet router.

If you finished editing loaded.config and setting up the backend nodes, execute the loaded.config script at the load balancer and start the loaded Daemon afterwards. On a client try to ping the virtual IP. If you see replies coming back from the virtual IP, everything should be fine. You can call

```
tcpdump -i eth0 -n icmp
```

on the load balancer (where eth0 is your public and/or private NIC) to see the address translation at work. This will also help if the ping doesn't work. Once the ping works you can start setting up an webserver on the backend nodes and start browsing the VIP. Note that if you browse from one IP only, the traffic is balanced to one node only, too. If you browse from a different IP then a different node is choosing. Some services require DNS lookups for their logs (such as sshd) so you can enable a DNS masquerading rule in loaded.config which masquerades the lookups from within the backend-node net to outside. Then, you should probably only balance TCP and/or ICMP (at least non-DNS) traffic or the masquerading will clash with the balancing. This can be done by adding "-p tcp" etc to the QUEUE iptables rules.

If you set up a test-network with no DNS and incorrect /etc/hosts file so please don't blame loaded for the latency you get. :)

#### 4. Run

-----

You are done.