

# SLAC: Linux Advanced Networking

# Überblick – Was erwartet Sie?

- Kurzvorstellung Vortrag und Referent
- Linux als Netzkomponente – Der Stand der Dinge. Was funktioniert sauber, was nicht?
- Vorstellen ausgewählter Kundenprojekte im Stil eines Kochrezepts sollen die Leistungsfähigkeit und Grenzen von Linux im Netzwerk zeigen.

# Kurzvorstellung Referent

- Name: Richard Müller
- Verantwortlich für Projektkoordination bei der Teamix GmbH in Nürnberg
- Einige Jahre Projekterfahrung in den Bereichen Networking mit Linux und OpenBSD

# Ziele dieses Vortrags

- Vermittlung von Praxiserfahrungen
- Aufzeigen des technisch sauber Machbaren.
- Ideen und Anregungen für den eigenen Adminalltag
- **Keine** Vermittlung von Basiswissen, aber Verweis an die richtigen Stellen zum Nachlesen.

# Linux als Netzkomponente

## (was jeder weiss)

- Router (statische Routingtabelle, pppoe, ...)
- VLAN-Tagging mit 802.1q
- WLAN Unterstützung (Client, AccessPoint)
- Bridgeing (aka "Linux als Switch")
- Firewalling (netfilter/IPTables)
- VPN-System (IPSec, OpenVPN)
- DNS / DHCP / Proxy - Server

# Linux als Netzkomponente

(was vielleicht nicht jeder weiss)

- Lernen von Routen durch Routingprotokolle (RIP, OSPF, BGP4)
- Advanced Routing mit mehreren Routingtabellen
- L4 Loadbalancing (IPVS)
- Bridged Firewalls (Netfilter, ebtables)
- VPNs: Redundante VPN-Strecken, L2-VPNs, Secure Tokens, ...
- Traffic Control (QoS)
- ... und noch viel mehr

# Was leider (noch) nicht geht

- Verwendung von spezialisierten Routing Engines (“Hardware Router”)
- High Performance Networking (>2.5GBit, >2Mpps)
- MPLS :-)

=> Dies sind Anforderungen von grossen Carrier-Backbones und nicht die von kleinen und mittleren Netzwerken und Internetübergängen.

# “Die Vorspeise”: Bündelung von günstigen DSL-Leitungen

## Beschreibung:

Ein Wohngebäude mit ca. 300 Bewohnern soll über mehrere günstige DSL-Anschlüsse ausreichend performant ans Internet angebunden werden.

Beeinträchtigungen durch Bandbreitenmissbrauch (DoS, Filesharing) sollen für normale Nutzer des Internetzugangs weitgehend vermieden werden.

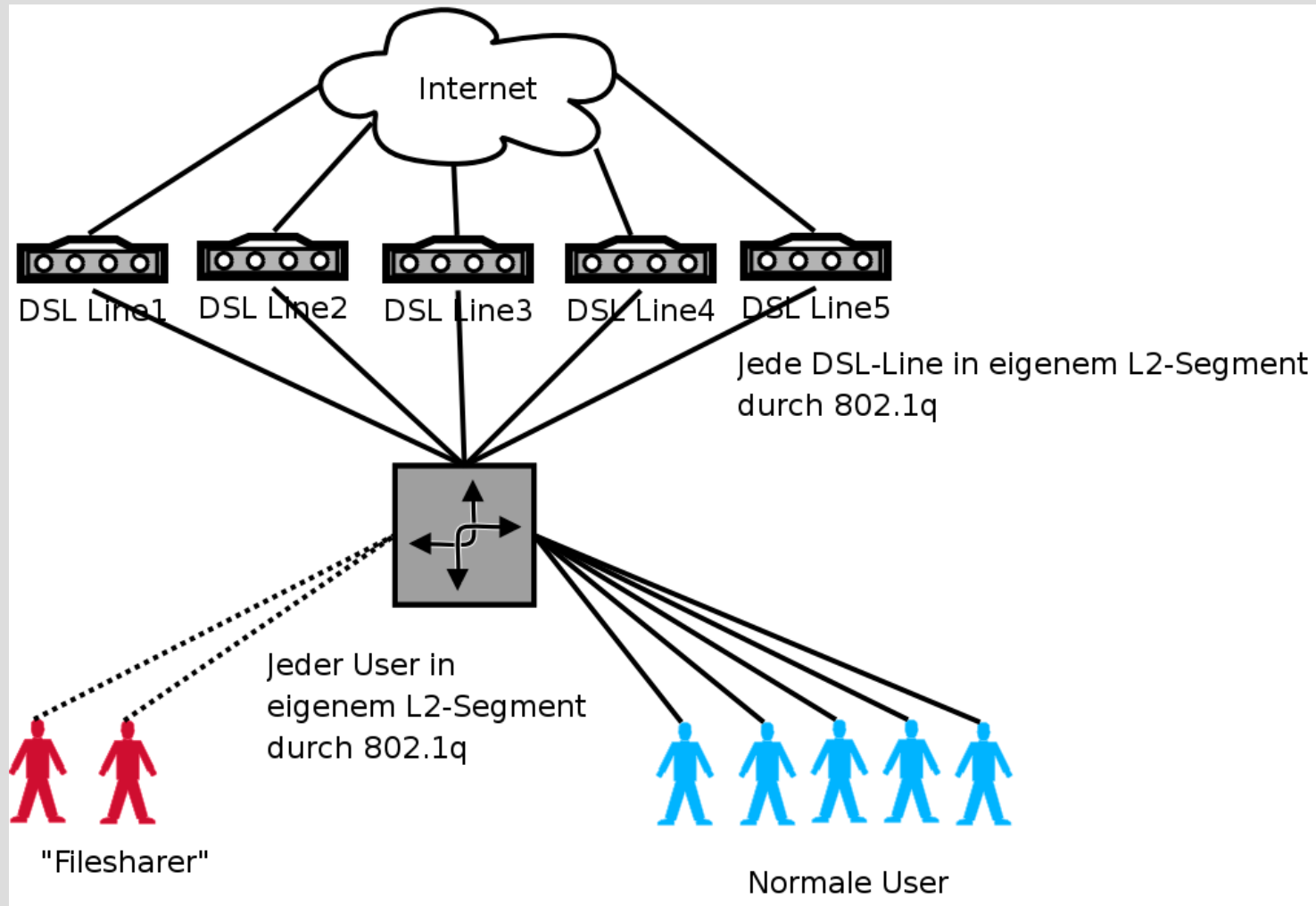
# Zutaten (Hardware)

- Zwei VLAN-fähige Switches (802.1q)
  - DSL-Fanout
  - Teilnehmer-Fanout (evtl. LRE, DSLAMs, etc.)
- Linux-System mit mindestens zwei NICs
- ADSL-Anschlüsse nach Belieben
  - hier 5 Stück der Sorte 6Mbit/512kbit

# Zutaten (Software)

- Linux Advanced Routing Features
  - ab Kernel 2.2
  - Kommandozeilentool `ip`
  - Hier: Lastverteilung per source-based Routing
- Linux Traffic Control Features
  - ab Kernel 2.2, besser Kernel 2.6 mit `htb`
  - Kommandozeilentool `tc`
  - Alternativ `wondershaper` (aka `tc` aus der Tüte)
- `nprobe` und `flow-tools` zur Protokollierung

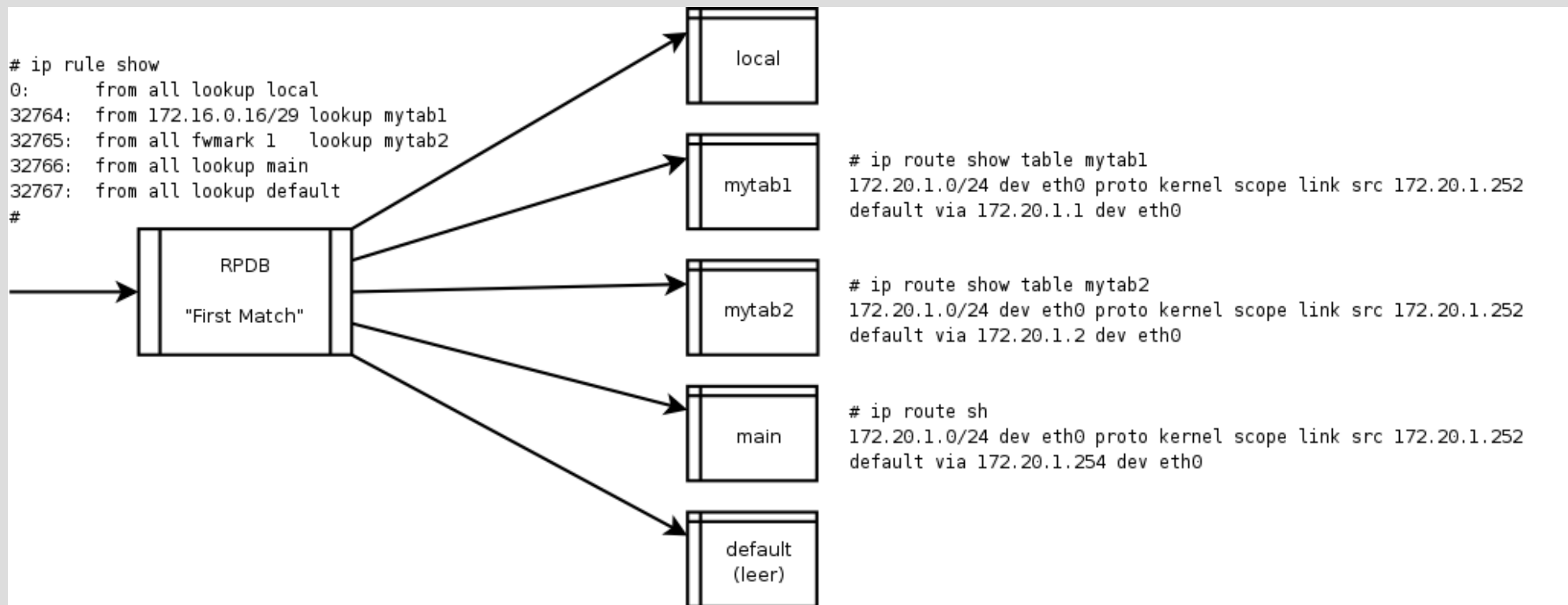
# Netzübersicht



# Exkurs: ip in 5 Minuten

- ip ist als Ersatz für die Kommandos `ifconfig`, `arp` und `route` gedacht.
- Syntax vollständig in BNF dokumentiert und sehr CISCO-lastig.
- Neue Features:
  - Verwendung von mehreren Routingtabellen durch die Vorschaltung einer Routing Policy Database (rdb)
  - Überwachen von Routingtabellen
  - Direktes Fragen nach Routingentscheidungen
  - u.v.m.

# Exkurs: Routing mit RPDB



# Vorgehen – Anbindung der Teilnehmer

- Jeder Teilnehmer bekommt ein eigenes LAN-Segment (VLAN und /28 Subnetz)
  - Damit ist IP-Adressenklaue ausgeschlossen
  - IP-Adressen werden komfortabel per DHCP verteilt
- Die Teilnehmernetze werden statisch per RPDB auf die DSL-Interfaces verteilt.
- Auf den DSL-Interfaces wird TrafficShaping (hier: wonder shaper mit HTB) aktiviert.

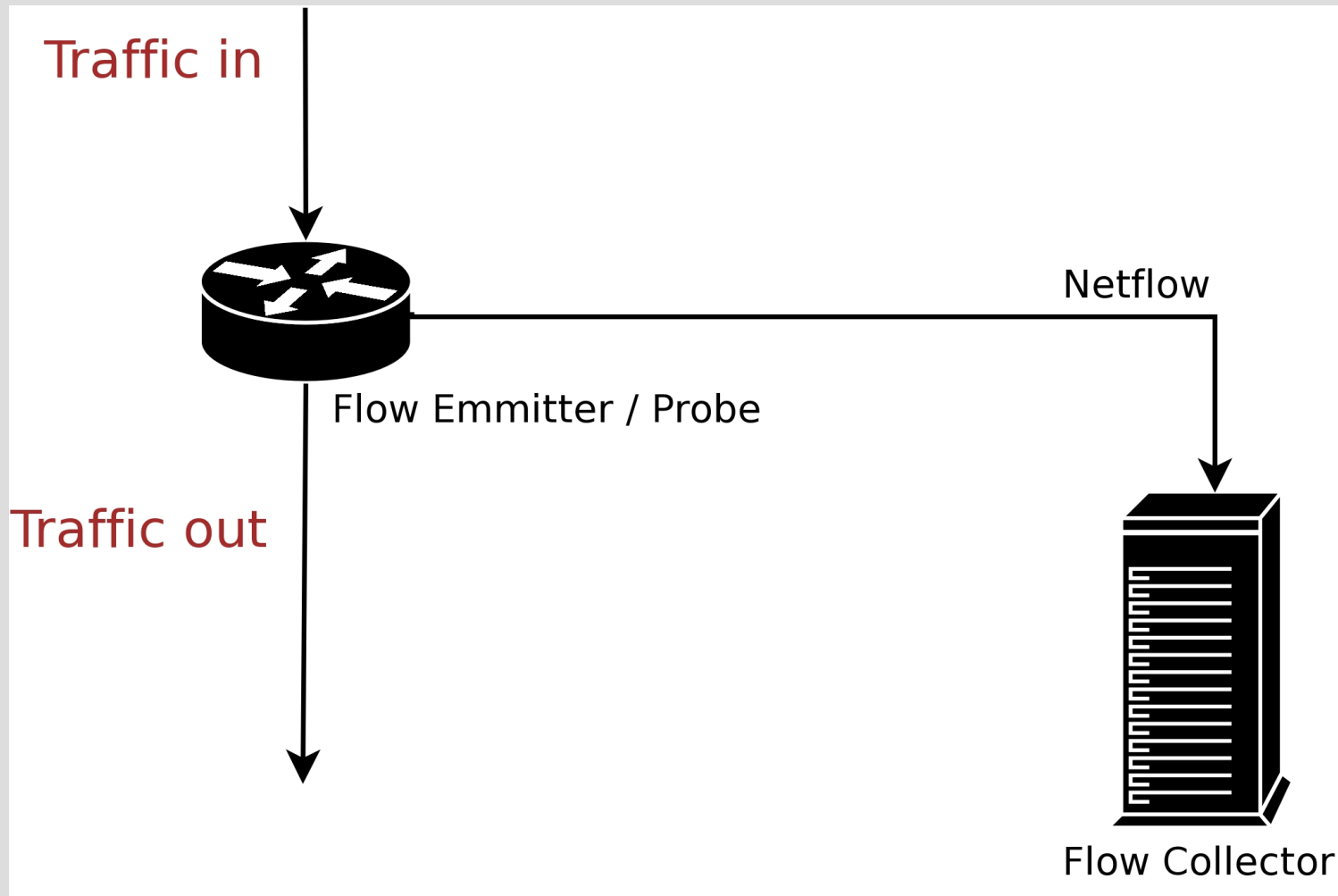
<http://www.lartc.org/>



# Vorgehen – Powersauger bremsen

- Aufgrund der netflow-Daten eines Monats wird festgelegt wer (welches Subnetz) ein Powersauger ist.
- Als Massregelung wird dieses Subnetz für den nächsten Monat mittels Traffic Controll (wondershaper auf Teilnehmerinterface) sehr stark beschnitten (Upstream).

# Exkurs: Netflows



# Screenshots - RPDB

```
# ip rule show
0:      from all lookup local
32597:  from 172.23.58.160/29 lookup 100
32598:  from 172.23.58.152/29 lookup 101
32599:  from 172.23.58.144/29 lookup 102
32600:  from 172.23.58.136/29 lookup 103
32601:  from 172.23.58.128/29 lookup 104
32602:  from 172.23.58.120/29 lookup 105
32603:  from 172.23.58.112/29 lookup 100
[...]
32762:  from 172.16.0.40/29 lookup 104
32763:  from 172.16.0.32/29 lookup 104
32764:  from 172.16.0.16/29 lookup 105
32765:  from 172.16.0.8/29 lookup 105
32766:  from all lookup main
32767:  from all lookup default
```



# Screenshots – Eine Routingtabelle

```
# ip route show table 100
[...]
```

172.16.1.120/29	dev eth1.616	proto kernel	scope link	src 172.16.1.121
172.16.1.112/29	dev eth1.615	proto kernel	scope link	src 172.16.1.113
172.16.1.104/29	dev eth1.614	proto kernel	scope link	src 172.16.1.105
172.16.1.96/29	dev eth1.613	proto kernel	scope link	src 172.16.1.97
172.16.1.88/29	dev eth1.612	proto kernel	scope link	src 172.16.1.89
172.16.1.80/29	dev eth1.611	proto kernel	scope link	src 172.16.1.81
172.16.1.72/29	dev eth1.610	proto kernel	scope link	src 172.16.1.73
172.16.1.64/29	dev eth1.609	proto kernel	scope link	src 172.16.1.65
10.0.4.0/24	dev eth0.104	proto kernel	scope link	src 10.0.4.1
10.0.5.0/24	dev eth0.105	proto kernel	scope link	src 10.0.5.1
213.139.154.0/24	dev eth2	proto kernel	scope link	src 213.139.154.67
172.23.58.0/24	dev eth3	proto kernel	scope link	src 172.23.58.254
10.0.0.0/24	dev eth0.100	proto kernel	scope link	src 10.0.0.1
10.0.1.0/24	dev eth0.101	proto kernel	scope link	src 10.0.1.1
10.0.2.0/24	dev eth0.102	proto kernel	scope link	src 10.0.2.1
10.0.3.0/24	dev eth0.103	proto kernel	scope link	src 10.0.3.1
10.1.0.0/24	dev eth0.200	proto kernel	scope link	src 10.1.0.1

```
default via 10.0.0.254 dev eth0.100
```

# Screenshots – Traffic Control DSL-Interfaces

```
# /sbin/wondershaper eth0.100 5900 490
# /sbin/wondershaper eth0.100

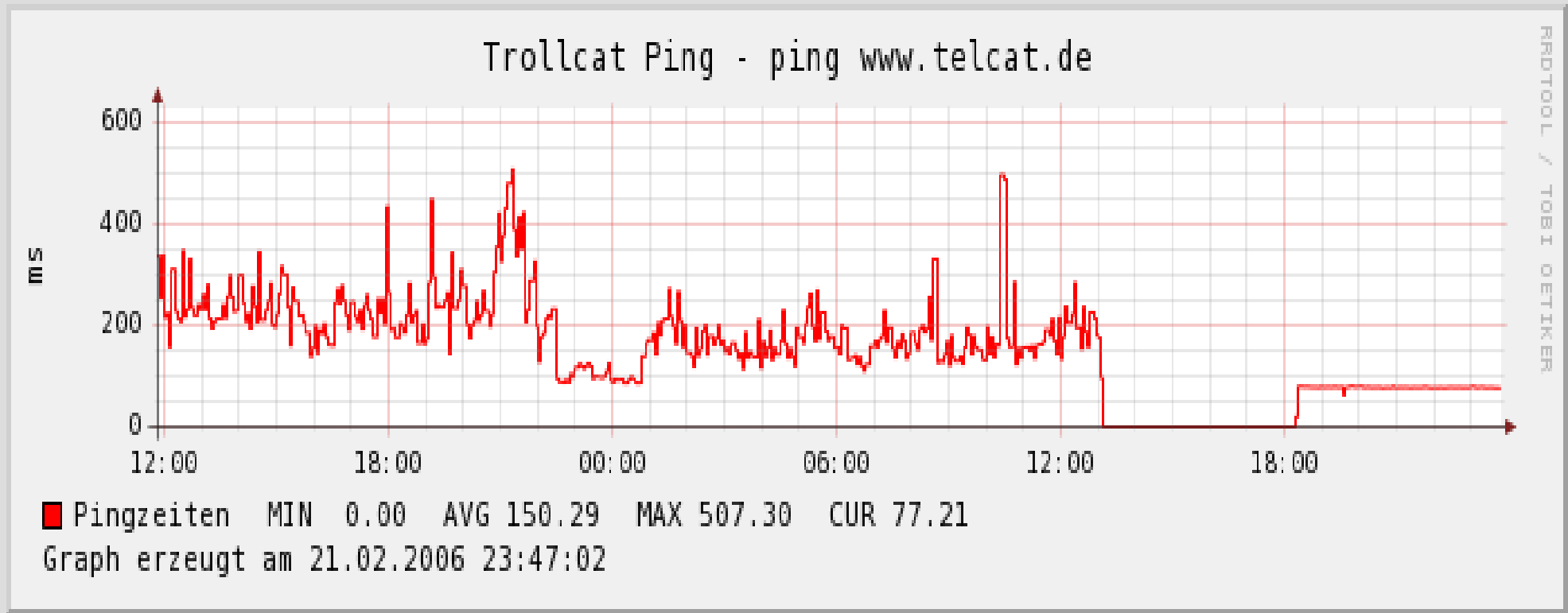
qdisc htb 1: r2q 10 default 20 direct_packets_stat 0
  Sent 195913540563 bytes 767577149 pkts (dropped 173268, overlimits 152716616)
qdisc sfq 10: parent 1:10 limit 128p quantum 1514b perturb 10sec
  Sent 23966233338 bytes 404128661 pkts (dropped 0, overlimits 0)
qdisc sfq 20: parent 1:20 limit 128p quantum 1514b perturb 10sec
  Sent 171947307225 bytes 363448488 pkts (dropped 173268, overlimits 0)
qdisc ingress ffff: -----
  Sent 623786014399 bytes 831763405 pkts (dropped 1386, overlimits 0)
class htb 1:1 root rate 490000bit ceil 490000bit burst 6Kb cburst 1660b
  Sent 195937130649 bytes 767577143 pkts (dropped 0, overlimits 0)
  rate 23240bit 32pps
  lended: 0 borrowed: 0 giants: 0
  tokens: 101916 ctokens: 26967

class htb 1:10 parent 1:1 leaf 10: prio 1 rate 490000bit ceil 490000bit burst 6Kb cburst 1660b
  Sent 23966233338 bytes 404128661 pkts (dropped 0, overlimits 0)
  rate 12208bit 27pps
  lended: 404128661 borrowed: 0 giants: 0
  tokens: 101916 ctokens: 26967

class htb 1:20 parent 1:1 leaf 20: prio 2 rate 441000bit ceil 441000bit burst 6Kb cburst 1654b
  Sent 171947307225 bytes 363448488 pkts (dropped 173268, overlimits 0)
  rate 11072bit 5pps
  lended: 363448482 borrowed: 0 giants: 0
  tokens: 113239 ctokens: 29852
```



# Screenshots – Alte Lösung vs. neue Lösung



# Screenshot – Statistiken mit Netflows

IP-Adressen	Volumen in MB	Volumen in Byte
172.16.0.11	11759MB	12330515558
172.16.0.115	885MB	928050742
172.16.0.123	81016MB	84951573636
172.16.0.131	4975MB	5217009525
172.16.0.146	1150MB	1205764234
172.16.0.154	21014MB	22035087111
172.16.0.162	27205MB	28526947572
[...]		
172.23.58.1	8782MB	9208398372
172.23.58.113	805MB	844611020
192.168.255.40	601MB	630015170
-----+-----+-----		
Statistiken		
-----+-----+-----		
GESAMT GERUNDET	1010628MB	1059720212247
GESAMT ABSOLUT	1016265MB	1065630800647
ABWEICHUNG	0.55%	

# Hauptgericht: Redundant angebundener Loadbalancer

## **Beschreibung:**

Ein renommierter Internetserviceanbieter benötigt Komponenten für einen vollständig redundanten Internetzugang und Loadbalancing von Webapplikationen.

Aus Budgetgründen muss eine kostenbewusste aber trotzdem sehr zuverlässige Lösung entworfen werden.

# Zutaten (Hardware)

- 2 VLAN-fähige Switches (802.1q)
  - Je ein Switch für jeweils die Hälfte der Webserver und jeweils eine Internetanbindung
- 2 Linux-Systeme im Cluster mit mindestens vier NICs:
  - IP-Upstream Provider 1
  - IP-Upstream Provider 2
  - LAN-Segment für Webserver
  - Heartbeat zwischen beiden Loadbalancern bzw. Routern

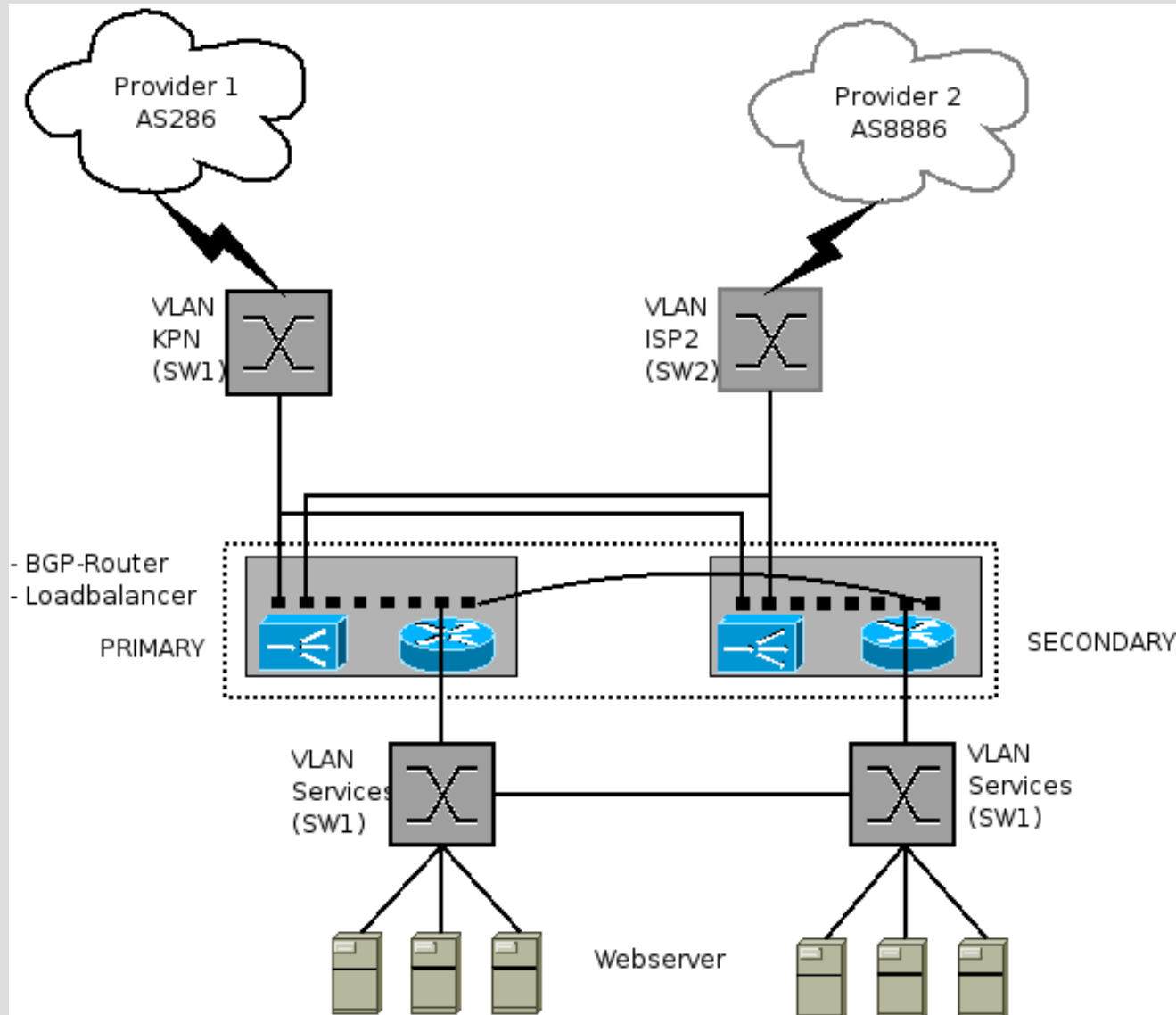
# Zutaten (Software)

- Linux IPVS (TCP/UDP Loadbalancer)
  - ab Kernel 2.6 fester Bestandteil
  - Kommandozeilentool `ipvsadm` bzw. `ldirectord` als Dienstemonitor
- Routing Daemon `quagga`
  - für externe Anbindung mit BGP4
- `heatbeat` zum Clustern gegen Hardwarefehler

# Zutaten (Sonstiges)

- Eigener providerunabhängiger IP-Addressbereich (PI-Space)
- Eigene AS-Nummer
- Zwei voneinander unabhängige Internetzugänge

# Netzübersicht

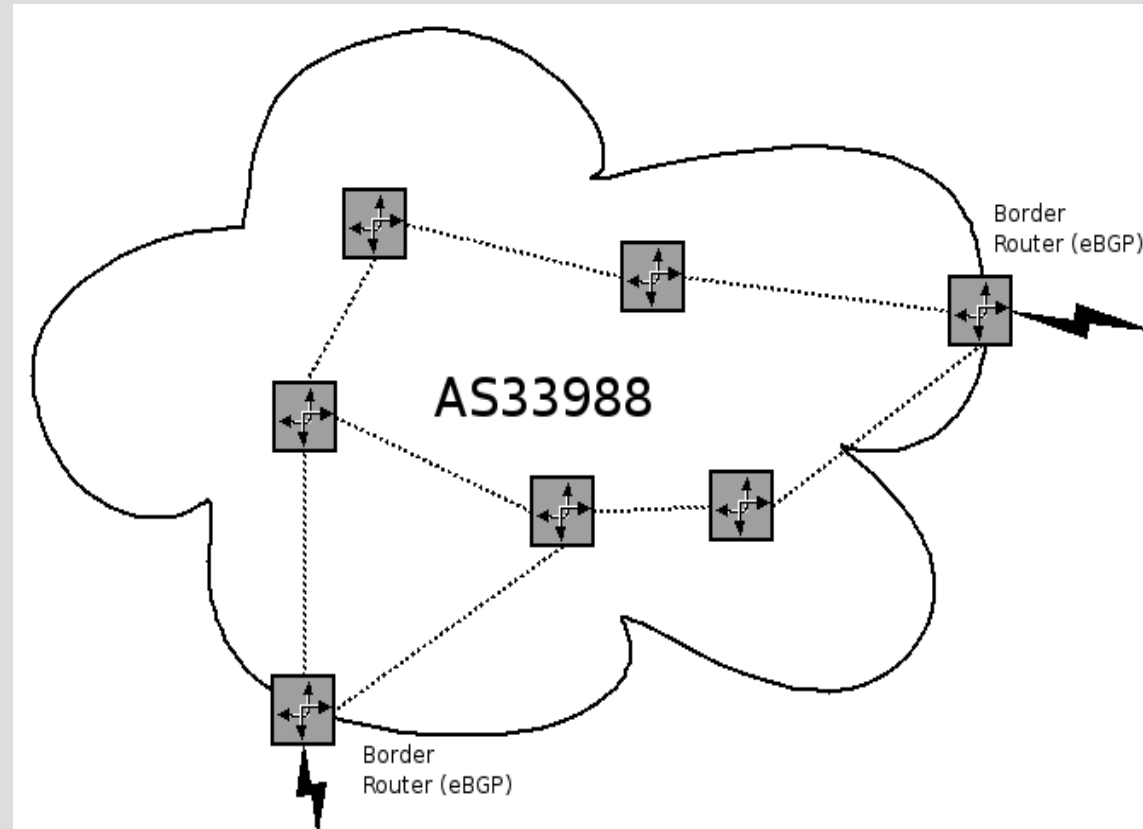


# Exkurs: Providerunabhängig ins Internet

- Wenn man sich selbst ans Internet verbinden will braucht man zuerst eigene IP-Adressen (PI-Space) und eine Autonome System Nummer (ASN)
- Die eigenen IP-Adressen werden dann unterhalb der ASN bekannt gegeben (announced).  
Dies geschieht durch BGP-Sessions die man zu den Routern der Provider aufbaut an die man angeschlossen ist.

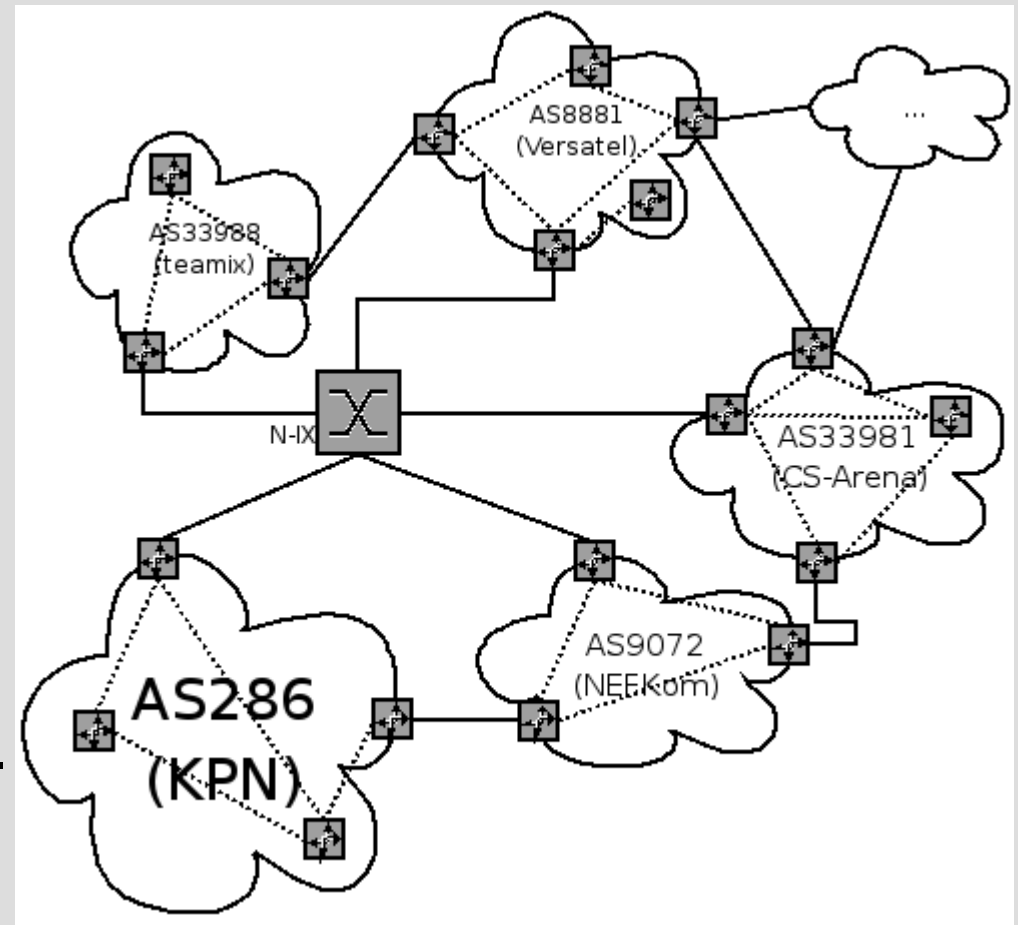
# Exkurs: Blick in ein Autonomes System

- Interior Gateway Protocol (IGP)
  - OSPF
  - RIP
  - IS/IS
  - iBGP
- Exterior Gateway Protocol (EGP)
  - eBGP (v4)



# Exkurs: Das Internet als “wilder AS-Haufen”

- Peering
  - nur Prefixes aus dem benachbarten AS austauschen
- Transit
  - Alle Prefixes weitergeben, die man kennt
- Internet Exchange
  - Öffentlicher Anschlusspunkt
- Private Interconnect
  - Dedizierte Verbindung von 2 AS



# Exkurs: Dynamisches Routing mit Linux und quagga

- Fork von GNU Zebra (Routing Daemon)
- Comand Line Interface erinnert stark an Cisco-IOS
- Architektur:
  - zebra core daemon “zebra” (Schnittstelle zum Kernel und Forwarding Information Base - FIB)
  - Zerv-Clients – Implemetierung der einzelnen Routing Protokolle
    - ospfd (OSPFv2 für IPv4)
    - ripd (RIPv1 und v2)
    - ospf6d (OSPFv3 für IPv6)
    - ripngd (RIPv3 für IPv6)
    - bgpd (BGP4+ - iBGP und eBGP)

# Exkurs: Dynamisches Routing mit Linux und quagga

- Das CLI eines jeder Daemons (zebra, bgpd, usw.) wird über einen eigenen TCP-Port angesprochen (Beispiel: `telnet localhost bgpd`)
- Jeder Daemon hat seine eigene Konfigurationsdatei (Oft: `/etc/quagga/* .conf` )

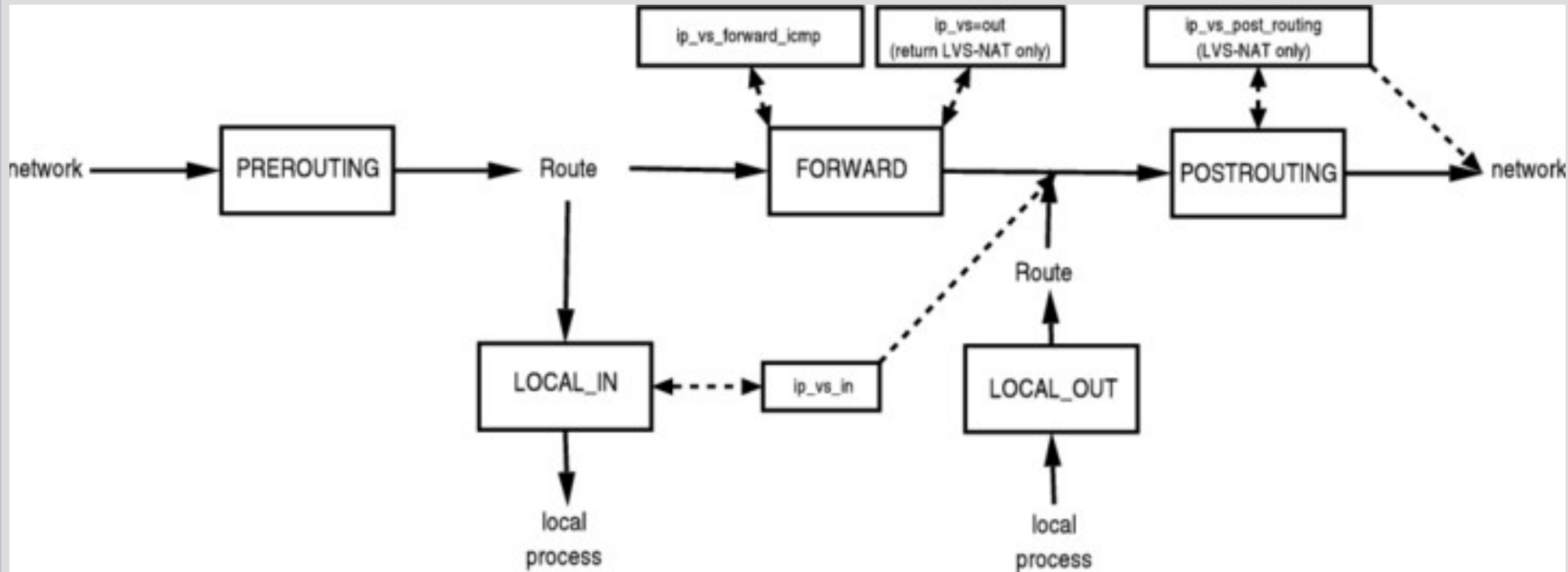
# Beispiel: Aufbau einer eBGP-Session mit quagga

```
hostname SLAC06-bgpd
password SLAC06rules
enable password SLAC06rulez
log stdout
!
router bgp 35117
  bgp router-id 134.222.105.20
  network 83.137.184.0/22
  neighbor 134.222.105.17 remote-as 286
  neighbor 134.222.105.17 description KPN
  neighbor 134.222.105.17 filter-list 2 out
!
ip as-path access-list 1 deny .*
ip as-path access-list 2 permit ^$
ip as-path access-list 2 deny .*
!
```

# Exkurs: Loadbalancing mit LVS

- Mit LVS hat man die Möglichkeit TCP und UDP-Sessions auf mehrere Server zu verteilen. (L4-Loadbalancing)
- Der Eintrag in die LVS-Kerneltabelle erfolgt durch das Kommandozeilentool `ipvsadm` oder durch weiter entwickelte Tools wie dem `ldirectord`
- Verteilungsmöglichkeiten (D)NAT, TUN und DR

# Exkurs: Loadbalancing mit LVS



**Linux Kernel Netfilter Hooks and LVS**

Horms <horms@verge.net.au>, v0.1.9-1, October 2003

<http://www.linuxvirtualserver.org/>



# Screenshots – Die LVS-Kerneltabelle

```
# ipvsadm -L
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port          Forward Weight ActiveConn InActConn
TCP  83.137.185.131:www wrd persistent 1200
  -> 83.137.186.101:www           Masq    1      20      79
  -> 83.137.186.41:www            Masq    1      21      57
  -> 83.137.186.26:www            Masq    1      21      37
  -> 83.137.186.116:www           Masq    1      49      68
  -> 83.137.186.56:www            Masq    1      41     155
  -> 83.137.186.71:www            Masq    1      33     126
  -> 83.137.186.86:www            Masq    1      30      74
[...]
```

# Screenshots – Idirectord.conf

```
# Global Directives
checktimeout=3
checkinterval=10
autoreload=yes
quiescent=no

# www.verivox.de - HTTP
virtual=83.137.185.131:80
    real=83.137.186.11:80 masq 1
    real=83.137.186.26:80 masq 1
    real=83.137.186.41:80 masq 1
    [...]
    persistent=1200
    service=http
    checktype=connect
    scheduler=wrr

[...]
```

# Exkurs: Hardwareredundanz mit heartbeat

- Heartbeat ermöglicht es einen oder mehrere Dienste mehreren Servern zuzuordnen, so dass im Desasterfall der Dienst automatisch auf einem anderen Server gestartet wird.
- Heartbeat ist für Router schnell und einfach konfiguriert.
- Die Konfiguration erfolgt über drei Dateien

<http://www.linux-ha.org/>



# Exkurs: Heartbeat - authkeys

- `/etc/heartbeat/authkeys` definiert ein `preshared secret` damit Clusternachrichten authentifiziert sind. Beispiel:

```
# cat /etc/heartbeat/authkeys  
auth 2  
2 sha1 meinkrasserpresharedsecret
```

# Exkurs: Heartbeat - ha.cf

- **Beherbergt alle Clusterrelevanten Optionen:**

```
keepalive 1           # jede Sekunde ein KEEP_ALIVE schicken
deadtime 120          # Failover nach 120 Sekunden
warntime 10           # Warnung im Log nach 10 Sekunden
initdead 180          # Beim Starten 180 Sekunden warten
auto_failback off    # Kein Automatischer Failback
bcast eth7            # Heartbeat Wege Definieren
ucast eth0 134.222.105.22
ucast eth2.120 83.137.184.3
ucast eth2.130 83.137.184.228
ucast eth2.140 83.137.184.244
ucast eth2.150 83.137.184.250
node PPRO-VVX-0002-PRI # Hostname node1
node PPRO-VVX-0002-SEC # Hostname node2
ping 83.137.184.10     # Ping Group 1
ping 134.222.105.17   # Ping Group 2
respawn hacluster /usr/lib/heartbeat/ipfail
apiauth ipfail uid=hacluster
```

# Exkurs: Heartbeat - haresources

- Beherbergt alle Dienste und DienstIPs:

```
PPRO-VVX-0002-PRI          IPaddr2::134.222.105.20/29 \
                           IPaddr2::83.137.184.251/29 \
                           IPaddr2::83.137.185.145/25/eth2.120 \
                           [...]
                           IPaddr2::83.137.185.254/25/eth2.120 \
                           ldirectord quagga
```

- Die entsprechenden Init-Scripts befinden sich unter  
`/etc/heartbeat/resource.d/`

# Häufige Fehler beim “Hauptgericht”

- `rp_filter` für die Upstream-Interfaces muss deaktiviert sein. (`/etc/sysctl.conf`)  
`net/ipv4/conf/eth0/rp_filter = 0`  
`net/ipv4/conf/eth1/rp_filter = 0`
- ARP-Anfragen sollen nur für lokal angeschlossenen IP-Netze beantwortet werden  
`net/ipv4/conf/all/arp_filter = 1`  
`net/ipv4/conf/all/arp_ignore = 1`  
`net/ipv4/conf/all/arp_announce = 1`

# Nachspeise: Redundante VPN-Stecken zur Standortvernetzung

## **Beschreibung:**

Eine bekannte Anwaltskanzlei mit Standorten über ganz Europa verstreut will mittels VPN das günstige Transportmedium Internet nutzen, hat aber gleichzeitig hohe Ansprüche an Verfügbarkeit und Ausfallsicherheit der VPN-Strecken.

Aus diesem Grund soll jeder Standort mit mind. 2 unterschiedlichen Internetanbindungen erschlossen sein.

# Weitere Projektanforderungen

- **keine** dynamischen Routingprotokolle wie RIP, OSPF oder gar BGP4
- Garantierte Umschaltzeiten bei Standleitungsausfall ( $\leq 20$  Sekunden)
- Eine Lastverteilung ist **nicht** erforderlich.

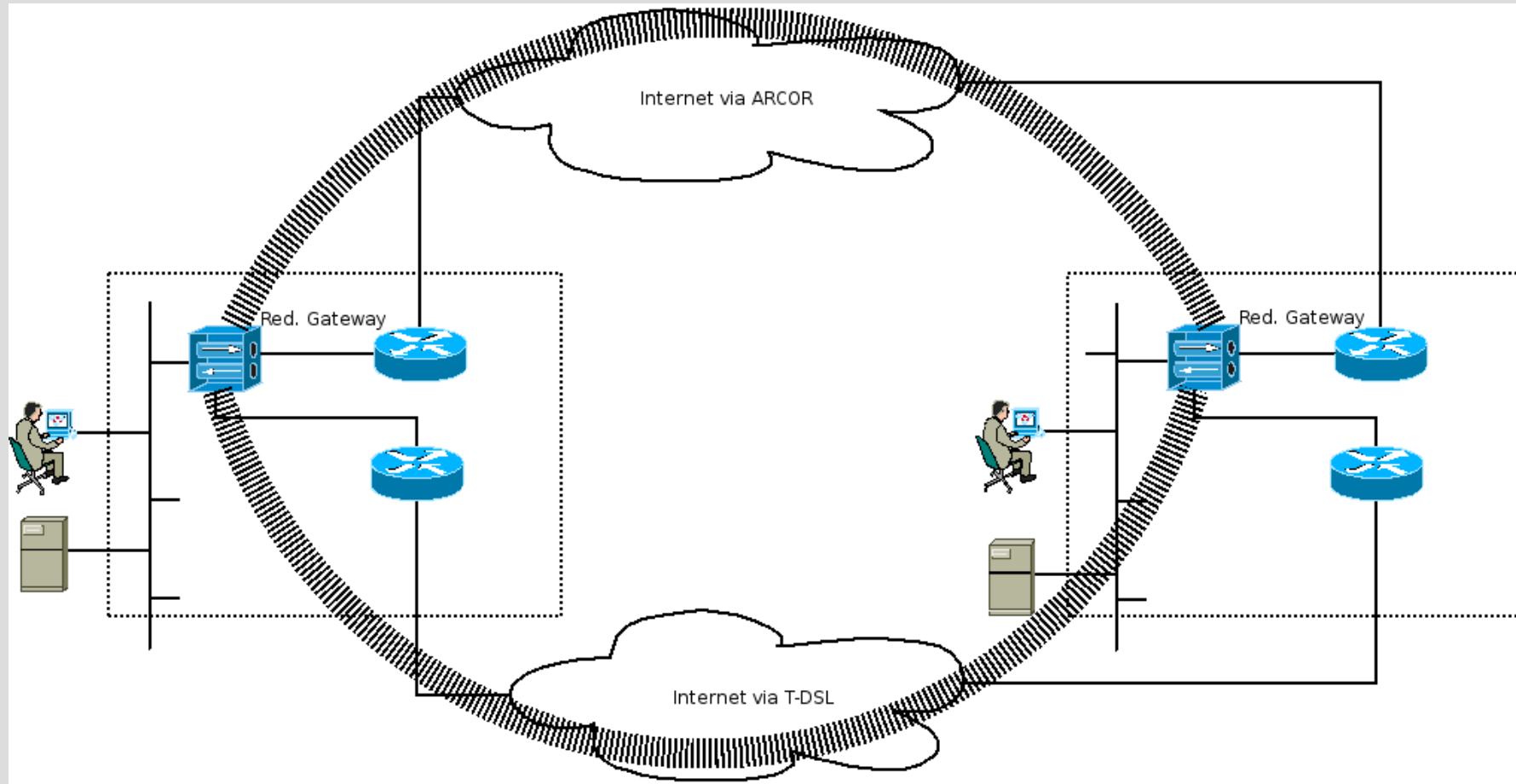
# Zutaten (Hardware)

- 2 Linux-Systeme (Cluster) pro Standort mit mindestens vier NICs
  - SDSL-Router Provider 1 (T-Online)
  - SDSL-Router Provider 2 (Arcor)
  - Lokales Standortnetz
  - Heartbeat zwischen beiden Routern

# Zutaten (Software)

- Linux Advanced Routing Features (siehe Vorspeise)
- heartbeat zum Clustern gegen Hardwarefehler (siehe Hauptgericht)
- OpenVPN
  - Freie und flexible VPN-Implementierung im Userspace

# Netzplan exemplarisch für 2 Standorte



# Vorgehen – VPN-Konnektivität

- Damit jeder Router, die anderen über die jeweils richtigen DSL-Strecken erreicht werden für die VPN-Endpunkte einfach Hostrouten (in der Tabelle `main`) gesetzt.
- Werden DSL-Zugänge mit dynamischen IP-Adressen verwendet, muss dies in den jeweiligen `ppp-scripten` geschehen und regelmässig geprüft werden (sehr aufwendig, da sich die Gegenstellen-IPs auch ändern können)

# Vorgehen – RPDB anpassen

- Nun wird die RPDB um zwei weitere Einträge (tdslvpn und arcovpn) erweitert:

```
ffm-pri:~# ip rule show
0:          from all lookup local
32763:     from all lookup tdslvpn
32764:     from all lookup arcovpn
32766:     from all lookup main
32767:     from all lookup default
```

# Vorgehen – Befüllen der Routingtabellen (I)

- Auszug aus der OpenVPN-Konfiguration:

```
[...]
```

```
ping 4
```

```
ping-restart 20
```

```
# Setting Routes
```

```
ipchange "/etc/openvpn/scripts/L3-BER-DUS start"
```

```
down "/etc/openvpn/scripts/L3-BER-DUS stop"
```

```
down-pre
```

- Wichtig ist hier vor allem die Option  
“down-pre”

# Vorgehen – Befüllen der Routingtabellen (II)

- Start/Stop-Script für die Routingeinträge:

```
#!/bin/bash
```

```
case "$1" in  
  start)
```

```
    ip route add 10.2.0.0/16 via $5 table tds1vpn  
    ip route flush cache
```

```
;;
```

```
stop)
```

```
    ip route del 10.2.0.0/16 via $5 table tds1vpn  
    ip route flush cache
```

```
;;
```

```
esac
```

```
exit 0
```

# In der Praxis auftretende Probleme bei der “Nachspeise”

- Bei der Verwendung von UDP als Encapsulierung, werden unidirektionale Fehler (z.B. routing-probleme) nicht bzw. falsch erkannt.
- Bei TCP als Encapsulierung werden zwar unidirektionale Fehler erkannt, es kann aber im Normalbetrieb zu Retransmit-Stürmen kommen (siehe Link unten).
- Der Failover findet nur bei sauberen Leitungsausfällen statt.

# Und nun...

..viel Spass beim richtigen Abendessen :)

