HA-fähige Firewall mit OpenBSD/PF (Packet Filter)

Wehrhaft abtauchen

Hochverfügbare Firewalls gelten als hohe Kunst im Netzwerkschutz. Dank Clustering bleibt das Netz online, selbst wenn eine Firewall ausfällt. OpenBSD/PF (Maskottchen: Kugelfisch) besitzen gegenüber dem bekannteren Duo Linux/Netfilter einige Vorzüge. Der Artikel erklärt das Setup – für Linuxer verständlich. Stephan A. Rickauer

Ein guter Ruf eilt OpenBSD [1] voraus: Es gilt als äußerst sicher und empfiehlt sich für Firewalls als interessante Alternative zu Linux. Für hohe Sicherheitsanforderungen bringt es viele Techniken mit, von zufällig gewählten Speicheradressen über zufällige Prozess-IDs (Abbildung 1) und zufällige TCP-Sequenznummern bis zu ausgefeiltem Buffer-Overflow-Schutz. Auch die Philosophie passt: Ein schlankes System mit aufwändigen Code-Audits, um Sicherheitslücken aufzuspüren. Das Team um Theo de Raadt will erklärterweise „das sicherste Betriebssystem“ [2] entwickeln.

Das Leben danach

Nach der Installation (Kasten „OpenBSD-Installation“) ist OpenBSD ähnlich wie Linux zu administrieren: Admin-Benutzer anlegen, SSH-Daemon konfigurieren, nicht benötigte Dienste abschalten, fehlende nachinstallieren und konfigurieren. Die im Kasten erwähnte Manpage »man 8 afterboot« hilft weiter.

OpenBSD kennt kein Runlevel-Konzept (Ausnahme: Single User Mode). Entsprechend einfach ist die Struktur der Start-Konfigurationsdateien (»man rc«). Die Einstellungen sind verteil auf »/etc/rc.conf« (nicht ändern), »/etc/rc.conf.local« (eigene Konfiguration) und »/etc/rc.local« (eigene Startanweisungen).
Die Firewalls in diesem Artikel verzichten auf den Internet-Superserver Inetd, der lokale Sendmail-Daemon startet ohne Flags und die PF-Firewall mit einer eigenen Regeldatei. Dazu sind folgende Einträge in »/etc/rc.conf.local« nötig:

```
inetd=NO
sendmail_flags=NO
pf=YES
pf_rules=/etc/pf/firewall-pf.conf
```

Zwei weitere Files enthalten Kommandos für den Shutdown (»/etc/rc.shutdown«) und solche, die vor dem Wechsel des Securelevel (»/etc/rc.securelevel«) laufen. Die vier Sicherheitsebenen beschränken die Funktionsweise des Systems zunehmend, so verbietet Securelevel 2 zum Beispiel jede Änderung von Firewallregeln, ohne einen Neustart auszuführen.
Wer weitere Dienste installieren will, beispielsweise SNMP, kompiliert entweder einen so genannten Port (»man 7 ports«) oder installiert mit »pkg_add« ein Binary-Paket. Beides geht leicht von der Hand und klappt auch via Internet. Die OpenBSD-Community empfiehlt das Paketsystem.

Gepflegter Purismus

Bei der Konfiguration des Netzwerks nach der Erstinstallationsroutine fällt der von OpenBSD gepflegte Purismus auf. Für jedes Netzwerk-Interface ist eine eigene Datei zuständig. Ihr Name beginnt mit »hostname« (mit diesem String, nicht mit dem Namen des Hosts) und sie trägt die Interface-Bezeichnung als Suffix, beispielsweise »/etc/hostname.fxp0«. Ihr Inhalt entspricht den typischen »ifconfig«-Parametern: Art der Schnittstelle, IP-Adresse, Subnetz- und gegebenenfalls Broadcast-Adresse sowie weitere Optionen:

```
inet 10.0.0.1 255.0.0.0 NONE
```

Die Bootskripte lesen den Inhalt aus und konfigurieren das Interface passend. Nachträglich geht das auch mit einem Aufruf von »sh /etc/netstart fxp0«. Der Hostname des Systems steht in »/etc/myname«, sein Standardgateway in »/etc/mygate«. Die nachfolgend beschriebenen virtuellen CARP-Schnittstellen für das Failover-Setup werden auf gleiche Weise konfiguriert.
Das von Linux bekannte Sysctl dient auch auf OpenBSD der Kernelkonfiguration zur Laufzeit. Die Parameter stehen in »/etc/sysctl.conf«. Dort finden sich

auch auskommentierte Zeilen für das Aktivieren von IPv4- und IPv6-Forwarding sowie Einstellungen, die das CARP-System braucht:

```
net.inet.carp.allow=1
net.inet.carp.preempt=1
net.inet.carp.log=0
net.inet.carp.arpbalance=0
```

Das Common Address Redundancy Protocol (CARP, [5]) ist eine freie Implementierung des von Patenten belasteten VRRP (Virtual Router Redundancy Protocol). CARP eignet sich für Cluster mit zwei oder mehr Knoten. Die portierbare UCARP-Variante (Unix CARP, [6]) läuft auch auf Linux-Systemen.

CARP wurde im Rahmen des OpenBSD-Projekts neu entwickelt und erstmals in OpenBSD 3.5 integriert. Es weist einem Cluster von mindestens zwei Maschinen des gleichen Subnetzes zusätzlich zu einer virtuellen IP- auch eine virtuelle MAC-Adresse zu (Abbildung 3). Die Nodes tauschen ihre CARP-Informationen kryptographisch signiert aus.

Ein Cluster-Node verhält sich als Master, der zweite als Backup. Der Vorteil: Im Gegensatz zu einer Linux/Heartbeat-Lösung muss beim Ausfall des CARP-Masters das Ersatzsystem keine geänderte IP/MAC-Adressbeziehung im Netz propagieren. Das Backupsystem übernimmt die Kombination aus MAC und IP vom Master. Ein weiterer praktischer Nebeneffekt der virtuellen MAC-Adresse ist die Möglichkeit, ARP- und IP-Pakete an beiden Nodes anzunehmen, die sich gegen-

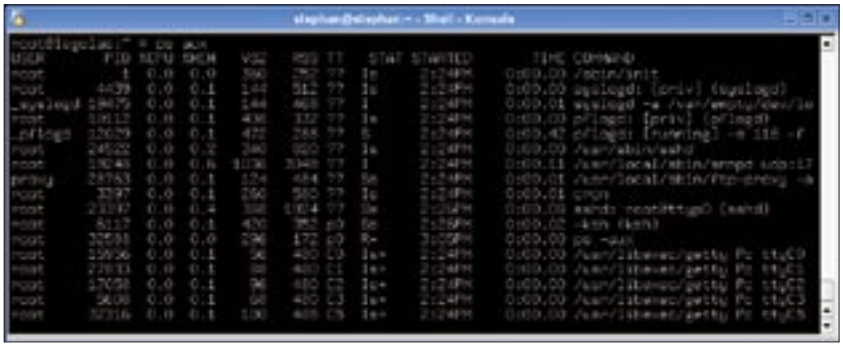

Abbildung 1: Die OpenBSD-Firewall benutzt zufällige Prozess-IDs. Daher sind bereits auf einem frisch gestarteten System IDs mit stark unterschiedlichen Werten anzutreffen.

OpenBSD-Installation

Das Projekt verkauft eigene CD-Sets; OpenBSD lässt sich aber auch via FTP und HTTP installieren. Der angehende OpenBSD'ler lädt ein Bootimage von einem FTP-Mirror und brennt es auf CD. Das Aufspielen des Systems verläuft reibungslos, wenn er die FAQ zu Rate zieht [3]. Dem BSD-Purismus entsprechend stehen weder grafische Installer noch aufwändige Wizards bereit. Trotzdem (oder vielleicht gerade deswegen) ist der Vorgang konsistent und logisch aufgebaut.

Linuxer stutzen höchstens bei der fremden Definition des Begriffs Partition, die sich als Teil eines Slice entpuppt. Ein Slice ist wiederum das, was in Linux- und Windows-Kreisen als Partition gilt. Die Hilfeseite [3] behebt auch diese Verwirrung. Für OpenBSD empfiehlt es sich stärker als für Linux, vor der Installation zu klären, ob das System mit der eigene Hardware zurechtkommt.

Ungewohnte Bezeichnungen

Während der Installation taucht die nächste sonderbare Benennung auf: Die Namen der Netzwerkschnittstellen hängen vom jeweiligen Treiber ab, »fxp0« etwa steht für eine 100-MBit-Karte von Intel, »xl0« für ein 3Com-Interface. Anschließend geht es nach der Auswahl des Installationsmediums oder Online-Mirrors zur Wahl der so genannten Filesets. Die Testmaschinen für diesen Artikel haben alle Default-Pakete außer »games« installiert.

Ein komplettes Basissystems liegt nach etwa einer Viertelstunde auf der Platte. Am Ende bestätigt ein motivierendes „Congratulations!" den Erfolg. Nett ist die Möglichkeit, bereits vor dem abschließend fälligen Reboot grundlegende Dinge zu konfigurieren: »/mnt/usr/sbin/chroot /mnt« versetzt den Benutzer in dasselbe Dateisystem wie nach dem Neustart.

Man, Mann

Linuxern ist das Prinzip der Manpages vertraut. OpenBSD dokumentiert darin aber deutlich mehr, selbst Treiber und Kernelfunktionen. Vollständige Anleitungen und FAQs finden sich in Manpages. Ein einfaches »man fxp« informiert über den FXP-Netzwerkkartentreiber. Einen Überblick über die C-Bibliotheken gibt »man 3 intro« und »man help« erleichtert den Einstieg in die OpenBSD-Welt. Ein erstmals gebootetes Systems begrüßt seinen Anwender mit dem Hinweis auf »man afterboot(8)«. Dort stehen alle Aufgaben, die nun fällig sind.

Die OpenBSD-Community veröffentlicht halbjährlich (Mai und November) so genannte »-release«-Fassungen. In der Zwischenzeit stellt das Team Patches als Source-Diff bereit. Es ist einfacher, dem »-stable«-Zweig per CVS zu folgen als die Patches manuell einzuspielen. Die Weiterentwicklung beschränkt sich auf den »-current«-Branch, vergleichbar dem »-unstable«-Zweig von Debian.

Beim Wechsel von »-release« nach »-stable« ist es nötig, das gesamte Userland und den Kernel neu zu kompilieren. Was für Admins von Binary-basierten Linux-Boxen wie ein Alptraum klingt, klappt dank guter Dokumentation [4] erstaunlich problemlos. Via anonymem CVS den gewünschten Stable-Branch runterladen:

```
cd /usr
CVSROOT=anoncvs@anoncvs.example.org:/cvs
export CVSROOT
cvs -d$CVSROOT checkout -rOPENBSD_3_7 -P src
```

Dann den Kernel neu kompilieren (OpenBSD rät ausdrücklich davon ab, den Kernel selbst zu konfigurieren):

```
cd /usr/src/sys/arch/i386/conf
config GENERIC
cd ../compile/GENERIC
make clean && make depend && make
make install
```

Schließlich Neustart des System (wichtig) und Kompilieren des Userland:

```
rm -rf /usr/obj/*
cd /usr/src
make obj
cd /usr/src/etc
env DESTDIR=/ make distrib-dirs
cd /usr/src
make build
```

Wer mehrere Maschinen auf Stable wechseln oder updaten will, kann eine eigene Release erzeugen und die neuen Binaries auf alle Maschinen kopieren.

Bis OpenBSD 3.7 dient die C-Shell (»csh«) als Standard, auf neueren Systemen übernimmt die Korn-Shell (»ksh«) diese Rolle. Die KSH war auch früher schon mit der Grundinstallation verfügbar. Sie bietet allen Komfort, den Linuxer mit der Bash genießen: Kommandozeilen-Komplettierung, History und vieles mehr.

Abbildung 2: Das Cross-Plattform-Werkzeug Firewall-Builder unterstützt alle PF-Optionen, die das Normalisieren des Netzwerkverkehrs steuern.

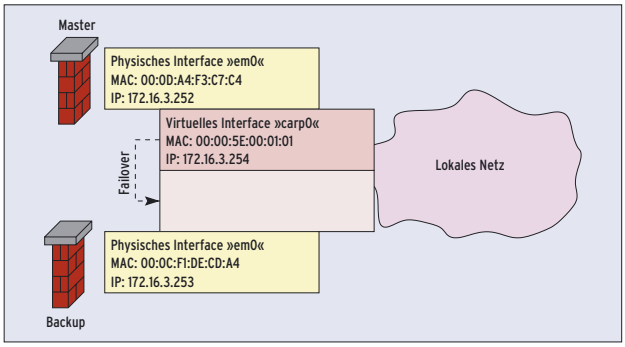

Abbildung 3: CARP stellt für den Cluster nicht nur eine virtuelle IP, sondern auch eine virtuelle MAC-Adresse zur Verfügung. Nach dem Failover reagiert das Backupsystem auf diese IP und MAC.

seitig sichern. Das führt zu transparentem ARP-Loadbalancing.

Um einen weiteren Single Point of Failure (SPoF) zu vermeiden, arbeiten HA-Firewalls oft an getrennten Switches (Abbildung 4, LAN- und DMZ-Anbindungen), die via Spanning Tree Protocol (STP) verbunden sind. Diese Technik kann mit CARP aber Probleme bereiten, nach dem Wechsel von Backup auf Master und umgekehrt vergehen dann etliche Sekunden Umschaltzeit.

Die Lösung: Alle Switch-Ports, an denen die Firewalls angeschlossen sind, müssen sich im so genannten Forwarding State befinden. Je nach Switch lautet das Kommando dafür »port fast« oder »fast link«, die Hersteller konnten sich auf keinen einheitlichen Namen einigen.

Im Beispielnetz soll eine HA-Firewall drei Segmente bedienen: LAN, DMZ und Internet-Uplink (siehe Tabelle 1 und Abbildung 4). Die Konfiguration der CARP-Schnittstellen gleicht der einer gewöhnlichen Netzwerkkarte. Mit Ausnahme des »advskew«-Werts sind die Parameter auf beiden Firewalls identisch:

```
echo "inet 172.16.3.254 255.255.254.0 ➥
  172.16.3.255 advskew 50 vhid 1 pass ➥
  hs3el9ja+e3 carpdev em0" > ➥
  /etc/hostname.carp0
sh /etc/netstart carp0
```

Advskew gibt die Wertigkeit eines Systems gegenüber dem anderen an. Je höher dessen Wert, desto niedriger ist seine Priorität. Der Backup-Node braucht also einen höheren Advskew als der Master. Hinter »vhid« verbirgt sich die Virtual Host ID, mit ihr lassen sich Nodes und ihre CARP-Interfaces einander zuordnen. Der »pass«-Wert authentifiziert die zwischen den Nodes ausgetauschten CARP-Nachrichten, »carpdevice« nennt das physikalische Interface, das dem virtuellen zugrunde liegt.

Nach diesen Schritten ist CARP für ein Segment bereits fertig konfiguriert. Das System sollte auf der virtuellen IP 172.16.3.254 Pings beantworten. Ein »ifconfig carp0« zeigt den Status »MASTER« auf einem Knoten und »BACKUP« auf dem zweiten (siehe Listings 1a und 1b).

Listing 1a: Erste Firewall

```
01 # ifconfig carp0
02 carp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
03         carp: MASTER carpdev em0 vhid 1 advbase 1 advskew 50
04         inet 172.16.3.254 netmask 0xfffffe00 broadcast 172.16.3.255
```

Listing 1b: Zweite Firewall

```
01 # ifconfig carp0
02 carp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
03         carp: BACKUP carpdev em0 vhid 1 advbase 1 advskew 100
04         inet 172.16.3.254 netmask 0xfffffe00 broadcast 172.16.3.255
```

Listing 1c: Firewall 2 übernimmt

```
01 # ifconfig carp0
02 carp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
03         carp: MASTER carpdev em0 vhid 1 advbase 1 advskew 100
04         inet 172.16.3.254 netmask 0xfffffe00 broadcast 172.16.3.255
```

Listing 2: PF-Regeln

```
01 # Lasse allen Traffic aus 192.168.0.0/24 auf 192.168.0.1 inklusive Rückverbindung zu
02 pass in on em0 from 192.168.0.0/24 to 192.168.0.1 keep state label "test"
03 #
04 # Erlaube SMTP von 172.16.2.254 nach 130.60.1.11 und logge die Verbindung
05 pass in log inet proto tcp from 172.16.2.254 port >= 1024 to 130.60.1.11 port 25 keep state
```

Simulanten

Um einen Ausfall der ersten Firewall zu simulieren, genügt es, deren CARP-Interface zu entfernen: »ifconfig carp0 down && ifconfig carp0 destroy«. Die virtuelle IP bleibt trotzdem pingbar, weil Firewall 2 von Backup auf Master wechselt (Listing 1c, Zeile 3). Die Umschaltzeit beträgt weniger als eine Sekunde – Heartbeat braucht dafür mit etwa zwölf Sekunden deutlich länger.

Die anderen CARP-Devices beider Firewalls werden analog konfiguriert. Wichtig ist der oben erwähnte Sysctl-Parameter »net.inet.carp.preempt«. Er stellt sicher, dass beim Ausfall eines einzigen Interface alle anderen den Status ebenfalls wechseln.

PF statt IPtables

Der Paketfilter PF [7] ist das Pendant zu Linux' Netfilter und seit Version 3.0 fester Bestandteil von OpenBSD. Er ersetzt den Vorläufer IPfilter (Ipf), der wegen seiner Lizenzprobleme aus dem OpenBSD-Projekt entfernt wurde. PF gilt als ausgereift und reich an ausgefeilten Features, die unter Linux ihresgleichen suchen (und gelegentlich in zusätzlichen Tools und Kernelpatches auch finden).

PF beherrscht mit der Scrubbing-Technik das Normalisieren des Netzverkehrs, integriertes Traffic Shaping (Packet Queueing and Priorization) und Anchors

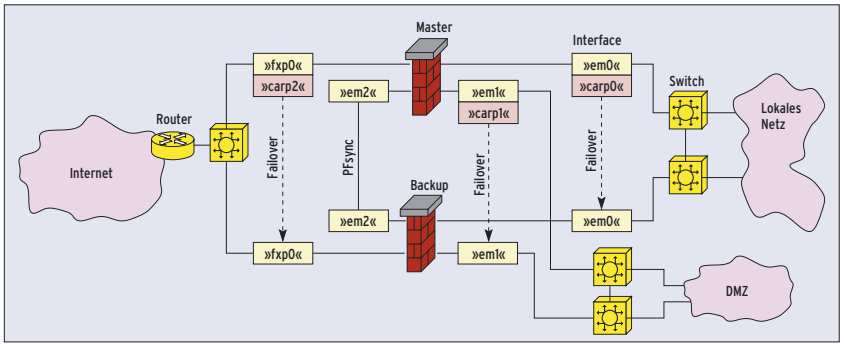

Abbildung 4: Master und Backup synchronisieren sich über PFsync, für diese wichtige Aufgabe verwenden sie eine eigene Netzwerkkarte (»em2«). LAN und DMZ sind über je zwei Switches an die CARP-isierten Interfaces (»em0« und »em1«) angeschlossen. Beim Ausfall eines Master-Interface schalten alle CARPs auf Backup um.

für das dynamische Ändern von Regeln. Listen, Makros und Tabellen machen die ohnehin einfache Syntax der PF-Regelsätze noch lesbarer (Listing 2).

Zudem kennt PF alle NAT-Varianten, Stateful Filtering (auch für Stateless UDP) sowie State-Synchronisation mit PFsync, mit dessen Hilfe ein Firewall-Takeover via CARP vollständig transparent erfolgt: Alle Tabellen, die den Status der aktuellen Verbindungen vorhalten, synchronisiert PFsync fortlaufend zur Backup-Firewall. Das Netfilter-Äquivalent »ct_sync« für Linux befindet sich dagegen noch in der Entwicklung.

Bequemer mit FW-Builder

Wer FW-Builder ([8] und der Artikel in diesem Heft) zur Firewall-Administration nutzt, wechselt besonders leicht von Netfilter zu PF oder umgekehrt. Das Programm schreibt Regelsätze für beide Firewalls aus einer identischen Policy, die Policy-Compiler machen's möglich. Theoretisch reicht es, in den FW-Builder-Settings von IPtables auf PF zu wechseln und die Interfacenamen anzupassen. Probleme gibt es nur mit Regeln, die an mehrere Interfaces gebunden sind. FW-Builder kreiert für Netfilter automatisch »INPUT«-, »OUTPUT«- und »FORWARD«-Chains. Da Letztere bei PF fehlen, dürfen Regeln, die für mehrere Interfaces gelten, nur in der globalen Policy stehen.

Leider unterstützt der PF-Policy-Compiler von FW-Builder zurzeit weder MAC-Adressen-Matching noch Traffic Shaping oder Anchors. Mit etwas Scripting-Kunst und der Hilfe des Prolog-Dialogs gelingt es aber, Anchors manuell einzufügen. Scrubbing unterstützt FW-Builder bereits (Abbildung 2). Sehr praktisch ist, dass das Tool die PF-Regeln in Form von Tables generiert, was die Größe der Regelsatzdatei im Vergleich zu IPtables glatt halbiert. Der volle Funktionsumfang von PF erschließt sich nur mit manuellem Regeldesign. Beim Einstieg und Verständnis der Syntax hilft FW-Builder aber allemal – es generiert übersichtliche und sogar kommentierte Regelsätze.

Die Regeln müssen noch auf die Firewalls gelangen und dort aktiv werden. Im Gegensatz zu IPtables übergibt das »pfctl«-Werkzeug nicht jede Regel einzeln an den Kernel, es erwartet vielmehr eine vollständige Konfigurationsdatei. FW-Builder erstellt zu ihrem Laden ein eigenes kleines Skript.

Ein Blick in die Pfctl-Manpage offenbart eine Fülle von Optionen. Die Angabe von »-e« schaltet PF ein, »-d« deaktiviert es wieder. Dabei gehen States, Tables und Chains nicht verloren – sehr praktisch fürs Testen. Ein weiteres Schmankerl ist »pftop«. Das Tool per »pkg_add« nachzuinstallieren lohnt: Es zeigt alle denkbaren Daten von PF zur Laufzeit in Form des bekannten »top«-Programms an.

FTP nur per Proxy

Das altbekannte Problem „FTP via Firewall und dann auch noch NAT" ([9], [10]) macht OpenBSD heute noch zu schaffen. Leider ist kein Modul verfügbar, das dem »ip_nat_ftp« von Netfilter entspräche. Ohne FTP-Proxy klappt weder aktives noch passives FTP. Der mitgeliefert Proxy ist in seiner Funktion allerdings beschränkt und arbeitet nur, wenn entweder ausgehender Traffic auf allen Ports erlaubt ist oder das so genannte Policy Based Tagging von PF zum Einsatz kommt.

Restriktivere Umgebungen, die den ausgehenden Verkehr einschränken, sind mit dem FTP-Proxy von Camiel Dobbelaar [11] deutlich besser aufbauen. Er ist einfach zu konfigurieren, versteht sich prächtig mit PF und ermöglicht dadurch problemloses Active/Passive-FTP. ▶

Tabelle 1: Interfaces der Firewalls

System	LAN	DMZ	Internet	PFsync
Master-Firewall	172.16.3.252 (em0)	130.60.230.186 (em1)	10.0.0.1 (fxp0)	192.168.91.252 (em2)
Backup-Firewall	172.16.3.253 (em0)	130.60.230.187 (em1)	10.0.0.2 (fxp0)	192.168.91.253 (em2)
Virtueller Cluster	172.16.3.254 (carp0)	130.60.230.188 (carp1)	130.60.5.218 (carp2)	nicht vorhanden

Tabelle 2: Messergebnisse

Testart	Ohne Firewall	OpenBSD (B_1) ohne PF	OpenBSD (B_1) mit PF	OpenBSD (B_1) mit PF und Scrub	Linux (B_2) ohne Netfilter	Linux (B_2) mit Netfilter
TCP_STREAM	694 MBit/s	487 MBit/s	467 MBit/s	467 MBit/s	499 MBit/s	500 MBit/s
TCP_MAERTS	807 MBit/s	783 MBit/s	735 MBit/s	718 MBit/s	783 MBit/s	775 MBit/s
TCP_RR	1998 Transakt./s	1991 Transakt./s	1971 Transakt./s	2003 Transakt./s	1377 Transakt./s	1212 Transakt./s
TCP_CC	1020 Transakt./s	995 Transakt./s	824 Transakt./s	780 Transakt./s	689 Transakt./s	686 Transakt./s
TCP_CRR	1143 Transakt./s	574 Transakt./s	545 Transakt./s	523 Transakt./s	534 Transakt./s	521 Transakt./s
UDP_RR	2075 Transakt./s	2013 Transakt./s	2004 Transakt./s	2015 Transakt./s	1282 Transakt./s	1357 Transakt./s

Für standhafte Firewalls gehört es sich, nur explizit erlaubten Verkehr durchzulassen. Sie müssen zusätzlich zur normalen Policy das IP-Protokoll 112 erlauben, um nicht versehentlich CARP-Nachrichten auszufiltern, die die Nodes untereinander austauschen. Gleiches gilt für den PFsync-Verkehr, der beide Firewalls auf dem aktuellen State-Table-Stand hält. Hier müssen Pakete des IP-Protokolls 240 von und zu den noch zu definierenden PFsync-Devices erlaubt sein. Die PFsync-Schnittstelle – wie könnte es anders sein – wird in der Datei »/etc/hostname.pfsync0« konfiguriert:

```
up syncdev em2
```

Nach dem Aufruf »sh /etc/netstart pfsync0« ist das Sync-Interface aktiv. Das Beispiel verwendet ein dediziertes Interface ausschließlich für den PFsync-Traffic. Wer mit Netzwerkkarten nicht klotzen will und lieber eine in Servern ohnehin meist überflüssige USB-Schnittstelle benutzen möchte, wirft einen kurzen Blick auf die durch »apropos usb|grep -i ether« generierte Liste und lässt seiner Phantasie freien Lauf.

Tolles Trio: CARP, PF, PFsync

CARP, PF und PFsync sind damit konfiguriert und der Firewall-Cluster ist einsatzbereit. Fehlt nur noch das Logging. PF-Logeinträge gehen nicht durch den Syslogd, vielmehr ist das Spezialdevice »pflog0« zuständig. Es schreibt ebenfalls Dateien nach »/var/log/«, allerdings im Tcpdump-Format. Diese Logfiles und das Log-Device selbst lassen sich per Tcpdump inspizieren:

```
tcpdump -n -e -ttt -i pflog0
tcpdump -n -e -ttt -r /var/log/pflog
tcpdump -n -e -ttt -i pflog0 port 80 and ➥
  host 192.168.1.3
```

Wer lieber Syslog und Ascii-Logdateien verwendet, findet im PF-Handbuch, Kapitel „Aufzeichnen“, eine gute Anleitung zur automatisierten Konvertierung.

Immer da

Hochverfügbare Firewallsysteme sind mit OpenBSD und PF leicht zu realisieren. Dank der Secure-by-Default-Philosophie ist OpenBSD bereits nach der Instal-

Belastungstest

Firewalls sollen den Internetzugang sichern und unerwünschte Pakete blocken, ohne den erlaubten Verkehr zu bremsen. Performance-Werte sind in der Praxis daher ein wichtiges Entscheidungskriterium. Wie sich OpenBSD/PF gegen Linux/Netfilter schlägt, hat der Autor mit dem Netperf-Benchmark [12] gemessen.

Die Testsysteme

Als Firewall diente ein Intel P4, 2,67 GHz (533 FSB, 512 KByte Cache) mit 512 MByte DDR-Speicher, U-ATA-Festplatte (7200 RPM) mit 80 GByte sowie zwei Ethernetkarten Intel PRO/1000MT DP (82546EB). Es liefen abwechselnd OpenBSD 3.7-Stable (System B_1) und Suse Linux 9.0 (B_2), je auf einer eigenen Festplatte gleicher Bauart installiert. Als Testclient (System A) kam ein Pentium M (1,6 GHz) zum Einsatz, mit 1 GByte Speicher, Intel 82540EP Gigabit-Ethernet und Suse Linux 9.3. Die Gegenstelle (Testserver C) war bis auf die Speicherausstattung (1 GByte DDR) baugleich mit der Firewall, lief aber mit Suse Linux 9.1. Die Verbindung zwischen A und B sowie B und C stellten zwei Dell Powerconnect 5212 GBit-Switches her; nur in einem Testlauf waren A und C direkt verbunden. Die Verbindungskabel waren Kat.7 S/FTP.

Um einen Eindruck vom Verhalten der getesteten Firewallsysteme in realistischem sowie in idealisiertem Umfeld zu erhalten, nutzte der Tester die sechs Netperf-Varianten [13] TCP_STREAM, TCP_MAERTS, TCP_RR, TCP_CC, UDP_RR und TCP_CRR:

- Stream: Upload eines großen Blocks von Daten (Bulk Upload), gemessen in MBit pro Sekunde.
- Maerts: Stream rückwärts gelesen, also Bulk Download in MBit pro Sekunde.
- RR: Misst die Anzahl der Requests und Responses (je 1 Byte groß) pro Sekunde.
- CC: Ermittelt, wie viele Verbindungen (Connect/Close) der Client pro Sekunde auf- und abbauen kann (ohne Datenaustausch).
- CRR: Connect, Request und Response. Gibt pro Sekunde an, wie viele Verbindungsaufbauten inklusive eines Request (128 Byte) und einer Response (16 KByte) gelingen.

Das Endergebnis in Tabelle 2 ist der Mittelwert aus je drei Testläufen mit 60 Sekunden Dauer. Dieser Ablauf fand für sechs Setups statt:

- A und C direkt verbunden
- Verbindung über Firewall B_1 ohne PF
- Firewall B_1 mit PF
- Firewall B_1 mit PF und Scrubbing
- Firewall B_2 ohne Netfilter
- Firewall B_2 mit Netfilter

Keines der involvierten Systeme war eigens auf Performance optimiert. Kerneloptionen, Send- und Receive-Puffer blieben bei den Standardwerten, um das Defaultverhalten der Firewalls zu ermitteln. Einzig der State-Memory-Pool in PF musste auf 1000 000 erhöht werden, damit der Test nicht als DOS-Attacke scheitert. Die Resultate sind – wie bei Benchmarks üblich – nur innerhalb des Setup gültig und nicht als generelle Aussage zu verstehen, dass eins der Systeme besser sei als das andere.

In der Praxis gleichauf

Stream- und Maerts-Test zeigen einen leichten Leistungsvorsprung für Linux/Netfilter (Abbildung 5, links). Auffällig ist, dass das OpenBSD/PF-Duo deutlich mehr Request/Response- und Connect/Close-Vorgänge abarbeitet (Tabelle 2, Zeilen TCP_RR, UDP_RR und TCP_CC). Diese Tests sind aber eher von theoretischem Interesse, da in der Praxis Client und Server nach einem erfolgreichen Verbindungsaufbau auch Daten austauschen. Im weit realistischeren CRR-Test liegen beide Systeme gleichauf, wie die rechte Grafik in Abbildung 5 belegt.

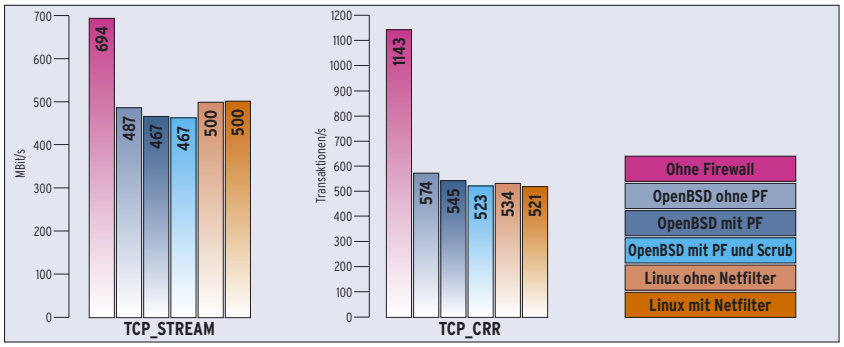

Abbildung 5: Links: Beim 60 Sekunden dauernden Netfilter-Test TCP_STREAM bremst OpenBSD/PF die Datenströme nur unwesentlich stärker als Linux/Netfilter. Rechts: 128 Byte geschickt und 16 KByte empfangen: OpenBSD/PF und Linux/Netfilter liegen beim realitätsnahen CRR-Test fast gleichauf.

lation ein enorm sicheres und gehärtetes System. Das puristische und konsequent durchdachte BSD-Design trägt nicht nur zur Sicherheit, sondern auch zur Benutzerfreundlichkeit bei. In puncto Performance und Stabilität steht OpenBSD nicht hinten an (siehe Kasten „Belastungstest“), lediglich die schlechtere Hardware-Unterstützung steht auf der Negativliste. *(fjl)*

Infos

[1] Homepage des OpenBSD-Projekts: [http://www.openbsd.org]
[2] Ziele des OpenBSD-Entwicklerteams: [http://www.openbsd.org/de/goals.html]
[3] Installationshilfe: [http://www.openbsd.org/faq/de/faq4.html]
[4] OpenBSD neu kompilieren: [http://www.openbsd.org/faq/de/faq5.html]
[5] CARP und PFsync: [http://www.countersiege.com/doc/pfsync-carp/]
[6] Portierbare CARP-Implementierung: [http://www.ucarp.org]
[7] PF: [http://www.openbsd.org/faq/pf/de/]
[8] Stephan A. Rickauer, „Maurermeister – Grafische Firewall-Administration mit FW-Builder 2.0“: LinuxUser 05/05, S. 72
[9] Frank Bernard, „FTP über Firewalls“: Linux-Magazin 06/02, S. 54
[10] Chris Grant, „FTP Reviewed“: [http://pintday.org/whitepapers/ftp-review.shtml]
[11] FTP-Proxy für OpenBSD: [http://www.openbsd.org/cgi-bin/cvsweb/src/usr.sbin/ftp-proxy/]
[12] Netperf: [http://www.netperf.org]
[13] Netperf-Handbuch: [http://www.netperf.org/netperf/NetperfTraining.html]

Der Autor

Stephan A. Rickauer ist Associate Member der Free Software Foundation und betreut freie Unix-Systeme am Institut für Neuroinformatik der ETH Zürich.