Hochschule
Augsburg University of
Applied Sciences

Diplomarbeit

Studienrichtung Wirtschaftsinformatik

Kinh Luan Nguyen

Hochverfügbare Firewallsysteme mit OpenBSD

Verfasser der Diplomarbeit:
Kinh Luan Nguyen
Mendelssohnstr.2
86368 Gersthofen
Telefon: +49 821 5892882
E-Mail: luan@luan.de

Fakultät für Informatik
Telefon: +49 821 5586-3450
Fax: +49 821 5586-3499

Erstprüfer: Prof. Dr. Hubert Högl
Zweitprüfer: Prof. Dr. Wolfgang Klüver
Abgabe der Arbeit: 30.09.2009

Hochschule Augsburg
University of Applied Sciences
Baumgartnerstraße 16
D 86161 Augsburg

Telefon +49 821 5586-0
Fax +49 821 5586-3222
http://www.hs-augsburg.de
poststelle@hs-augsburg.de

Kurzfassung

Die Diplomarbeit beschreibt die Möglichkeit mit Hilfe von OpenBSD, einem freiem Betriebssystem, ein funktionales und hochverfügbares Firewallsystem aufzubauen.
Hierzu wird eine detaillierte Beschreibung der Funktionen und Merkmale der OpenBSD Firewall Software geliefert, die dann als Theorie zur Installation eines Firewallsystems auf zwei embedded-PCs des schweizer Herstellers PCEngines vom Typ ALIX 2D13 dient. Das so entwickelte Firewallsystem bietet die Basis für vielfältige Anwendungsszenarien und das ganze bei minimalen Kosten. Zum besseren Verständnis wurden außerdem zwei Videos erstellt, die zum einen die Installation vorführen und zum anderen eine Hochverfügbarkeitspräsentation geben.

Die fertige Installation steht als Dateisystem-Abbild auf der Begleit-CD zur Verfügung und kann in Verbindung mit der Diplomarbeit als Grundlage für weitere Projekte genutzt werden.

Danksagung

Diese Diplomarbeit stellt den Abschluß einer langjährigen Ausbildung dar. Den größten Dank gilt meiner Familie, die mich immer unterstützte.

Speziell folgenden Personen möchte ich einen ganz besonderen Dank aussprechen:

- C. Mühlbach - meine bessere Hälfte
- M. Steinhauser - meinem „Kumpel“
- G. Klingbeil - meinem Kameraden
- Prof. Dr. Hubert Högl für die Bereitschaft als Prüfer dieser Arbeit und für die Offenheit bei der Themenauswahl
- Prof. Dr. Wolfgang Klüver für die Funktion des Zweitprüfers

Ein großes Danke auch an alle, die während des Studiums behilflich waren und mit denen ich so einiges gemeinsam vollbracht habe.

Inhaltsverzeichnis

Kurzfassung i

Danksagung ii

Inhaltsverzeichnis iii

1. Einleitung 1

2. OpenBSD 3
2.1. Geschichte von OpenBSD . . . 4
2.2. Eigenschaften von OpenBSD . . . 4
2.3. Einsatz und Organisation von OpenBSD . . . 10
2.4. Installation von OpenBSD . . . 11

3. Firewall 13
3.1. Definition Firewall . . . 13
3.2. Firewall Einführung . . . 16
3.3. NAT - Network Address Translation . . . 20

4. Das Firewallsubsystem pf von OpenBSD 23
4.1. Geschichte von pf . . . 24
4.2. Grundlagen von pf . . . 25
4.2.1. Funktionen von pf . . . 26
4.2.2. Steuerung von pf . . . 33
4.3. Syntax und Konfiguration von pf . . . 34
4.3.1. Listen und Makros . . . 35
4.3.2. Tabellen . . . 36
4.3.3. Anker - anchors . . . 38
4.3.4. Filterregeln . . . 40
4.3.5. NAT Konfiguration . . . 44

4.3.6. Weiterleitungen . . . 46
4.3.7. Optionen . . . 46
4.3.8. scrub . . . 49
4.3.9. Priorisierung mit ALTQ . . . 51
4.3.10. Address Pools . . . 53

5. Beispielanwendungen von pf **55**
5.1. pf als einfacher NAT-Router . . . 55
5.2. pf als sicherer WLAN-Accesspoint . . . 57
5.3. pf als load-balancer . . . 62

6. Hochverfügbarkeitsyssteme **66**
6.1. Allgemeines . . . 66
6.2. Hochverfügbarkeit mit OpenBSD . . . 67
6.2.1. CARP . . . 68
6.2.2. pfsync . . . 69

7. HA Firewallsystem mit ALIX Plattform **71**
7.1. die ALIX Plattform . . . 71
7.2. Installation OpenBSD auf ALIX . . . 74
7.3. HA System mit CARP und pfsync . . . 76

8. Fazit **83**

A. Abkürzungsverzeichnis **iv**

B. Abbildungsverzeichnis **v**

C. Literaturverzeichnis **vi**

D. BSD-Lizenz **vii**

E. Eidesstattliche Erklärung **viii**

1. Einleitung

In den letzten 15 Jahren hat das Medium Internet die Welt vollkommen verändert. Das Internet ist überall präsent und kaum eine Firma oder ein Haushalt ist nicht im „Netz der Netze“. Die elektronische Kommunikation, die Informationsgewinnung und auch die soziale Plattform sind wichtige Aufgaben des Internets geworden. Und das sowohl im privaten als auch in geschäftlichem Bereich.

Doch leider gibt es auch Schattenseiten durch das Internet. Durch die globale Vernetzung müssen sich die Teilnehmer auch Gedanken über den Schutz des eigenen Systems bzw. des eigenen Netzwerks machen, denn die Gefahren durch Software (Schädlinge, Viren und Trojaner) oder auch Sabotage durch Hacker nehmen tagtäglich zu. Das Hauptmittel zum Schutz vor derartigen Gefahren stellen sogenannte Firewallsysteme dar. OpenBSD bietet aufgrund seiner Eigenschaften eine hervorragende Basis zum Einrichten von Firewallsystemen an.

In der Arbeit wird OpenBSD und seine Möglichkeiten im Bereich Firewallsysteme untersucht, eine detaillierte Einführung und Dokumentation der Firewallkomponente gegeben und anschließend wird die Theorie zum Aufbau einer redundanten Firewallsystems zum besseren Verständnis angewandt.

Beginnend mit einer Einführung in das Betriebssystem OpenBSD geht es weiter in die Grundlagen der Firewalltechnologien. Konkret wird es dann in einem eigenen Kapitel, wenn OpenBSDs Packetfilter pf beschrieben wird, der die Basis des Firewallcodes von OpenBSD bildet. Danach folgt eine Einführung in das Thema Hochverfügbarkeit. Nach

Behandlung der Theorie werden anhand von Beispielen die einfache Anwendung des Packetfilters in typischen Szenarien dargestellt.

Anschließend wird anhand eines konkreten Hardwareprojekts das gesammelte Wissen in eine Implementierung einer komplett redundanten und hochverfügbaren Firewall umgesetzt. Als Hardwaregrundlage dienen hier zwei embedded-PCs des schweizer Herstellers PCEngines vom Typ ALIX.

2. OpenBSD

Abbildung 2.1.: Das OpenBSD Logo mit seinem Maskottchen Puffy

> Linux people do what they do because they hate Microsoft. We do what we do because we love Unix.
>
> *(Theo de Raadt)*

2.1. Geschichte von OpenBSD

Die Geschichte von OpenBSD beginnt im Jahr 1995, als Theo de Raadt, ein Mitbegründer und Hauptentwickler des freien Betriebssystem NetBSD gebeten wurde, seine Mitgliedschaft im Projekt-Team zu beenden und zeitgleich sein Zugang zum Quellcodearchiv gesperrt wurde. Die genauen Hintergründe dieses Vorgangs sind bis heute umstritten und es herrschen unterschiedliche Aussagen darüber. So schreibt beispielsweise Peter Wayner[1] , dass de Raadt auf der NetBSD Mailing-Liste falschen Umgang mit den anderen Teilnehmern gezeigt und öfters sich im Ton vergriffen hätte.

Teo de Raadt verließ daraufhin das NetBSD Projekt und gründete OpenBSD mit dem Ziel, ein sicheres und elegant konstruiertes Betriebssystem zu erschaffen und alle Ideen zu implemntieren, für die das NetBSD Projekt keinen Anklang fand. Mitte 2007 war es dann soweit und OpenBSD wurde der Öffentlichkeit zugänglich gemacht. Somit stellt OpenBSD neben NetBSD und FreeBSD das dritte freie Unix-ähnliche Betriebssystem dar, dass seine Wurzeln im BSD 4.4 besitzt.

2.2. Eigenschaften von OpenBSD

OpenBSD ist ein Unix-ähnliches Betriebssystem, dass in der *BSD* Version 4.4 seinen Ursprung hat. OpenBSD steht unter der *BSD-Lizenz*[2] jedem frei zur Verfügung. Die *BSD-Lizenz* stellt eine der liberalsten Lizenzen im Bereich der freien Lizenzen dar. Im Wesentlichen erlaubt die Lizenz jedermann (privat und kommerziell) die Software frei zu benutzen, zu modifizieren und zu verbreiten. Einzige Bedingung ist, dass der ursprüngliche Copyright Vermerk nicht entfernt werden darf. Diese Freiheit wird vor allem von Unternehmen geschätzt, da sie damit das Recht haben, eigene Produkte auf Basis von OpenBSD zu entwickeln, ohne dass sie die Modifikationen als Quellcode offenlegen müs-

[1] vgl. [Way00] Kapitel 18.3
[2] vgl. Anhang C

sen. Damit ist die BSD-Lizenz deutlich liberaler wie die GPL-Lizenz, die immer eine Offenlegung des Quellcodes verlangt.

OpenBSD besitzt die typischen Eigenschaften eines echten Unix Systems, wie etwa Mehrbenutzerfähigkeit, preemptivem Multitasking und starke Netzwerkfunktionen. Im Gegensatz zu Linux, dass eigentlich nur ein Kernel ist und erst durch die Paketierung der verschiedenen Distributionshersteller zu einem vollen Betriebssystem wird, handelt es sich bei einem BSD-System immer um ein komplettes Betriebssystem, dass in einem Quelltextarchiv gemeinsam entwickelt wird. So kommt das Betriebssystem neben dem Kernel mit einem Satz von Netzwerkdiensten und Anwendungsprogrammen aus einer Hand. Dadurch ist die Kompatibilität für Entwickler von Software stets gegeben und sie müssen ihre Programme nicht an mehrere Varianten des eigentlich selben Betriebssystems wie bei Linux testen und anpassen.

Als freies Betriebssystem trifft man bei OpenBSD auf viele bekannte freie Software, wie zum Beispiel den Mailserver Sendmail, den Webserver Apache und den DNS-Server bind. Der große Unterschied ist jedoch, dass die verwendeten Versionen stark modifiziert und erweitert wurden, um einerseits dem Sicherheitsanspruch von OpenBSD gerecht zu werden und andererseits die nur in OpenBSD vorhandenen Sicherheitsfunktionen auch auszunutzen.

OpenBSD zeichnet sich durch folgende besondere Eigenschaften aus:

1. **vollkommen freie Software**
   nur Software und Code, die wirklich frei sind werden benutzt und in das Projekt aufgenommen.
2. **kompromisslose Stellung gegenüber Software-Lizenzen**
   es werden keinerlei Lizenzen akzeptiert, die irgendwelche Einschränkungen der Freiheit beinhalten. Insbesondere kommerzielle Lizenzen und die berühmten *NDAs* sind ausgeschlossen.
3. **hohe Qualität der Dokumentation**

sämtliche Dokumentation, wie zum Beispiel die *man-pages* müssen akribisch korrekt, aktuell und umfangreich sein.

4. **Sicherheit**
   stellt den Hauptfokus von OpenBSD dar. Folgende Eigenschaften weisen OpenBSD in Bezug auf Sicherheit aus:[3]

   a) das ständige Bestreben das sicherste Betriebssystem der Welt zu schaffen

   b) full-disclosure: Eines Sicherheitsphilosophie, die sich dadurch auszeichnet, dass ein wirklich sicheres System eine dauerhafte und komplette Offenlegung des Quellcodes auf allen Ebenen (kernel und userland) garantiert und Details über Sicherheitslücken jedem unverzüglich offenlegt. Verschleierungen von gefundenen Sicherheitslücken, wie oft üblich bei Software, sind somit ausgeschlossen.

      Vorteile dieser Philosphie sind:

      i. es ermöglicht einer deutlich größeren Anzahl von Individuen die gefundenen Lücken bzw. die Fehler anzuschauen und folglich auch zu beheben

      ii. Programmierer und Ingenieure können aus den gefundenen Fehlern mehr lernen

      iii. Hersteller werden gezwungen schneller Fehlerkorrekturen zu veröffentlichen

      Einen Nachteil gibt es aber auch: Zeitgleich zur Bekanntgabe der Sicherheitslücke können auch böswillige Individuen diese ausnutzen.

   c) kontinuierlicher Security-Audit Prozess seit 1996. Der gesamte Quelltext des Betriebssystems wird fortlaufend immer wieder nach möglichen Programmierfehlern und vorhandenen Sicherheitslücken überprüft. Dies geschieht immer parallel zur Entwicklung und wird von einem eigenen Security Auditing Team

[3] vgl. [Opeb]

durchgeführt, von denen einige Mitglieder Sicherheitsexperten bei der renommierten Firma *Network Associates* arbeiten.

Eine weitere Eigenschaft des Prozesses ist die proaktive Sicherheitsarbeit. Während des Audits werden oft Fehler gefunden und korrigert, die zu dem Zeitpunkt noch keine Sicherheitslücke darstellen, aber später in den einschlägigen Sicherheit-Mailinglisten wie zum Beispiel *bugtraq* [4], als kritische und kompromittierbare Fehler in anderen Betriebssystemen veröffentlicht werden.

Als ergänzende positive Konsequenz dieser Arbeit werden immer wieder neue Techniken entworfen, um die gängigsten Sicherheitslücken von Anfang an zu vermeiden. Einige Beispiele hierfür sind:

i. Austausch von Standard C-Funktionen durch eigene sicherere Varianten. *strlcpy() and strlcat()* sind berühmte Kandidaten.

ii. neue Speicherschutzmechanismen wie *W^X*[5] oder zufallsgesteuerte *malloc()* und *mmap()* Funktionen. Diese Mechanismen schützen vor den oft auftretenden *buffer overflow* und vor allem *stack overflow* Angriffen. Letzterer Angriff wird wirksam verhindert, denn wenn der *stack* nicht direkt ausführbar ist, kann eingeschleuster Schadcode auch nicht direkt ausgeführt werden. Die Programme terminieren einfach in solchen Fällen.

d) integrierte Kryptographie Funktionen[6]: OpenBSD beinhaltet standardmäßig starke Kryptographiefunktionen. Angefangen von *IPSec* über *OpenSSH* bis hin zu sicheren *Hash* Funktionen und eigenem Zufallszahlengenerator. Beinahe alles, was sequentielle Zahlenfolgen beinhaltet, wird bei OpenBSD durch starke Zufallszahlen ersetzt. Beispielsweise Speicherzuweisungen oder Netzwerkpaketierungen. Volle VPN-Unterstützung, sowohl als Server wie als Client sind standardmäßig dabei. Darüber hinaus wird Kryptographie Hardware,

[4] vgl.[sec]
[5] eine Speicherschutz Policy, bei der jede Speicherseite in einem Prozessraum entweder beschreibbar oder ausführbar ist
[6] vgl.[Opea]

die hauptsächlich zur Beschleunigung der Kryptographie-Chiffren dient sehr gut unterstützt.

e) Sicherheit von Anfang an: Standardmäßig laufen nur die nötigsten Dienste nach einer OpenBSD Installation. Nur was explizit verwendet wird, soll der System-Administrator auch starten und freigeben. Dabei lernt der unerfahrene Administrator auch gleich das System besser kennen und immer auch das Thema Sicherheit bei Entscheidungen miteinzubeziehen.
Diese *policy*, bei der von Anfang an nur die wenigsten Dienste gestartet werden, findet auch immer mehr bei anderen Betriebssystemen Verwendung.

5. **Qualität des Codes**
Laut Wikipedia[7] zeichnet sich Quellcodequalität durch folgende Eigenschaften aus:

a) Lesbarkeit

b) Einfachheit bei der Wartung, Fehlerbehebung, Fehlerbeseitigung, Veränderung, Portabiliät und beim Testen

c) geringe Komplexität

d) geringem Resourcenbedarf, insbesondere im Prozessor- und Speicherverbrauch

e) Anzahl der Compiler- und Linkewarnungen

f) solide Parameterüberprüfungen und Fehlerbehandlung

Das sind Punkte, auf die der Projektleiter von OpenBSD, Theo de Raadt, achtet und deren Einhaltung er von allen beteiligten Entwicklern erwartet. Ergänzend dazu stellt auch die saubere Dokumentation des Quellcodes eine Selbstverständlichkeit dar.

Kommentare wie dieser von Manolis Tzanidakis[8]:

[7] vgl. [Wikb]
[8] vgl. [Tza]

> *„OpenBSD's clean code and design also provide rock-solid stability. I used to have lots of problems and crashes with new versions of Linux (kernel 2.6.x) and FreeBSD (versions 5.x and 6.x). The main focus of OpenBSD developers is to provide programs that work efficiently and thus prefer to improve their code rather than just add more new features and end up with a bloated and unstable product.“*

sind nicht selten.

Das Online Magazin OSNews vertritt mit folgender Abbildung das Denken der meisten OpenBSD Entwickler in Bezug auf „code quality“:

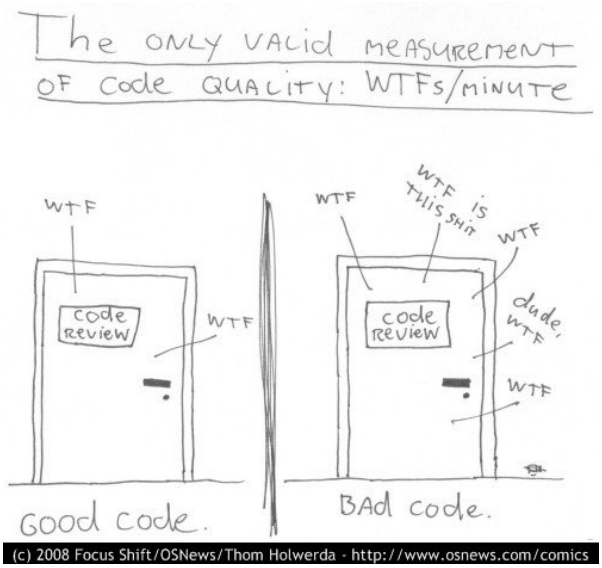

Abbildung 2.2.: Software quality measurement

6. **hohe Portabiliät**
   OpenBSD lässt sich mit voller Unterstützung[9] auf 16 Plattformen betreiben. Und dies sowohl im 32-bit als auch 64-bit Modus. Faktisch sind es noch ein paar weitere, aber diese sind dann entweder experimentiell oder werden nicht mehr vom Projekt direkt aufgrund Personalmangels unterstützt.

[9] Stand August 2009

2.3. Einsatz und Organisation von OpenBSD

OpenBSD ist ein freies Softwareprojekt, das von Theo de Raadt (dem Gründer) geleitet wird. OpenBSD wird durch die liberale BSD-Lizenz weltweit privat sowie kommerziell eingesetzt. Genaue Zahlen zur Verbreitung existieren nicht, aber die BSD Certification Group hat 2005 eine Umfrage gestartet, aus der man den Anteil aller BSD Benutzer, die auch OpenBSD benutzen erkennen kann.

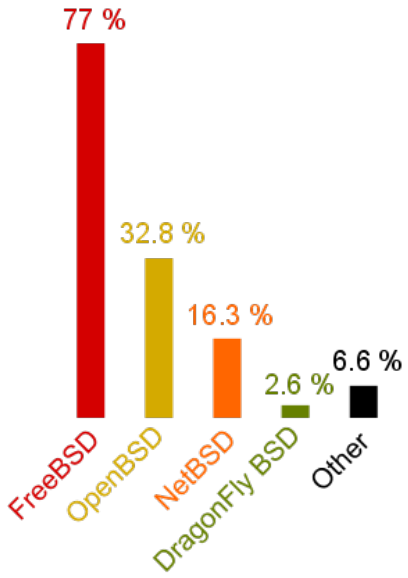

Abbildung 2.3.: OpenBSD Benutzeranteil

Vorrangig wird OpenBSD aufgrund seiner Eigenschaften in sämtlichen Bereichen eingesetzt, in denen es auf Sicherheit und Stabilität ankommt, sowohl auf kleinen embedded Plattformen, wie auch auf ganz großen Serverfarmen. Einer der beliebtesten Einsatzzwecke von OpenBSD ist der Einsatz als Firewall und Gateway (Proxy, Mailgateway, Load-Balancer). Aufgrund seiner guten Dokumentation ist es auch ein beliebtes Betriebssystem von Unix-Entwicklern.

Entwickelt wird OpenBSD von ca. 120 Entwicklern weltweit. Darunter viele Sicherheitsexperten und System-Ingenieure. Entwickler bei OpenBSD wird man nur durch persönliche Auswahl durch den Projektleiter. Zu den Kriterien zählen nicht nur die fachliche Eignung, sondern auch eine gewisse Leidenschaft für das Projekt und dessen straffe Führung.

Das Projekt finanziert sich hauptsächlich aus Spenden und Verkauf von CD-Sets, die die OpenBSD Software auf startbaren Medien enthält. Seit Juli 2007 gibt es auch eine OpenBSD Foundation, deren Hauptaufgaben sind, Spenden (kommerzielle) zu organisieren und die Entwicklung der OpenBSD Projekte zu unterstützen. Zu den OpenBSD Projekten gehören außer OpenBSD selber noch:

- OpenSSH: dem wohl bekanntesten telnet Ersatz
- OpenNTPD: einer wirklich freien NTP Zeitserver Alternative
- OpenCVS: einer freien und sicheren CVS Alternative
- OpenBGPD: eine freie Implementierung des BGP Protokolls zur Kommunikation von dynamischen Routern

2.4. Installation von OpenBSD

OpenBSD ist kostenlos von zahlreichen Spiegelservern weltweit erhältlich. Eine Liste aller aktuellen Spiegelserver erhält man unter `http://www.openbsd.org/ftp.html`. Man kann OpenBSD via Netzwerk (ftp und http) direkt von einem der Spiegelserver installieren oder man lädt sich die *install.iso* herunter und beginnt mit der Basisinstallation des Betriebssystems via CD. Dieses CD-Abbild gibt es erst seit wenigen Versionen von OpenBSD. Vorher war man gezwungen ein CD-Set zu kaufen, wenn man ein bootbares Installationsmedium mit dem Basissystem benötigte. Doch die Kritik, dass dies nicht einem freiem Betriebssystem entsprach war so groß, dass das Projekt beschloss, auch dies zu ändern.

Die Installation selbst ist sehr einfach gehalten. Man muss lediglich das nach dem Start der Installation ausgeführte Install-Skript durchlaufen. Sollten hierbei Schwierigkeiten auftreten, dann hilft wie in den meisten Fällen ein Blick in die sehr gute Dokumentation des Betriebssystems. In diesem Fall speziell die *OpenBSD FAQ*, die man unter `http://www.openbsd.org/faq.html` findet. Die *OpenBSD FAQ* ist in den letzten Jahren vom Umfang so gewachsen, so dass sie einem OpenBSD Benutzer immer als erste Anlaufstelle dienen sollte. Dort werden auch zahlreiche Ratschläge für Umsteiger von anderen Betriebssystemen, einschließlich *Linux* gegeben. Und wie bei allen Dokumentationen bei OpenBSD wird sie ständig auf den aktuellen Stand gehalten.

3. Firewall

In diesem Kapitel werden allgemeine Grundbegriffe zum Thema Firewalls behandelt, die zum weiteren Verständnis der Diplomarbeit nötig sind. Angefangen über grundlegende Definitionen bis hin zu den gängigsten Konzepten geht es in den Netzwerkbereich mit einer Sammlung an Techniken, die für den Einsatz von Firewalls unabdingbar sind.

3.1. Definition Firewall

Eine allgemeine Definition nach ELLERMANN[1]:

> *„Ein Firewall ist eine Schwelle zwischen zwei Netzen, die überwunden werden muss, um Systeme im jeweils anderen Netz zu erreichen. Durch technische und administrative Massnahmen wird dafür gesorgt, dass jede Kommunikation zwischen den beiden Netzen über die Firewall geführt werden muss. Auf der Firewall sorgen Zugriffskontrolle und Auditing dafür, dass das Prinzip der geringsten Berechtigung durchgesetzt wird und potentielle Angriffe schnellstmöglich erkannt werden.“*

Diese Schwelle/Firewall ist Bestandteil eines Sicherheitssystems und besteht in der Regel aus einer Kombination von soft- und hardware-technischen Komponenten. Durch diese

[1] vgl. [Ell94] S.10

Kombination entsteht erst das eigentliche Firewallsystem. Eine oder mehrere Firewallsysteme bilden dann die Hauptbestandteile im technischen Bereich eines Sicherheitssystems. Weitere Bestandteile können so einfache Dinge wie sichere Kennwörter und Zugangskontrollen sein.

Der Schutz zwischen zwei Netzwerken ist die Hauptaufgabe einer Firewall. Im allgemeinen wird damit ein externes, meist das Internet, und ein internes Netzwerk gemeint. Ein internes Netzwerk kann ein privates Heimnetzwerk oder auch ein Firmennetzwerk sein. In diesem einfachen Beispiel übernimmt die Firewall auch parallel die Rolle des Gateways. Es handelt sich um ein Singlebox-Firewallsystem.

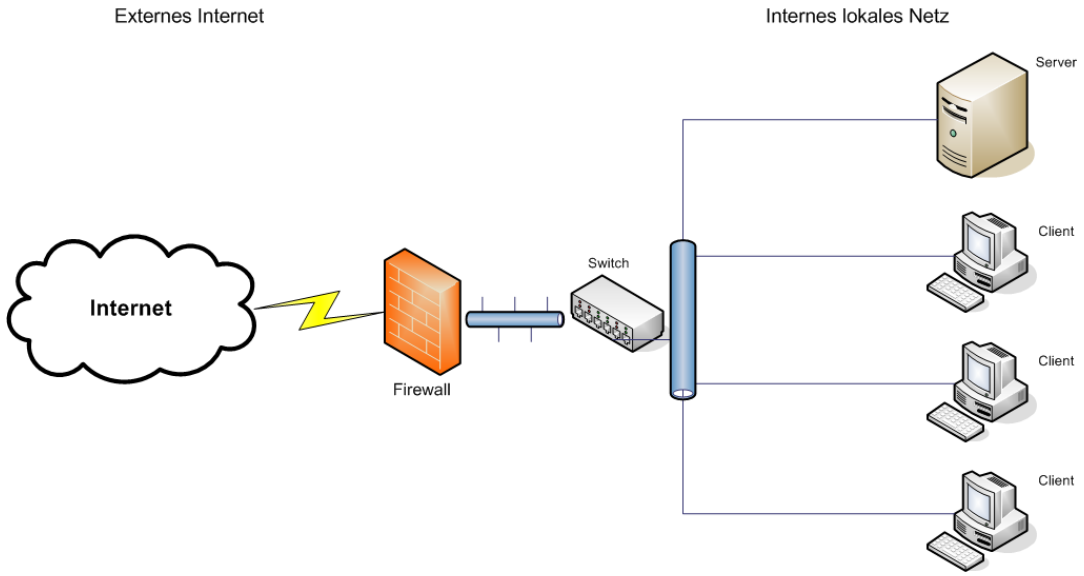

Abbildung 3.1.: einfache Firewall Topologie

Ein internes Netzwerk kann auch aus mehreren Teilnetzwerken oder aber auch aus mehreren eigenständigen Netzwerken bestehen. Teilnetzwerke (Subnetze) werden gerne zur Trennung von Abteilungen in Firmen genutzt. Eigenständige Netzwerke können bei Verwendung von Drahtlos-Netzwerken (WLAN) vorkommen. Auch wenn im lokalen Netzwerk Internetdienste angeboten werden, die direkten Zugriff aus dem Internet erfordern, wird das lokale Netzwerk um ein separates Netzwerk ergänzt. Dieses eigenständige Netz-

werk wird dann als DMZ bezeichnet. Den Rechnern in der DMZ werden durch die Firewall direkte Zugriffe auf bestimmte Dienste erlaubt und somit können diese Rechner Dienste im Internet direkt anbieten.

Beliebte Einsatzzwecke sind zum Beispiel der Betrieb eigener Web- und Mailserver oder bei Einsatz einer mehrstufigen Firewall-Architektur der Betrieb eines Proxy-Servers. Auch hier hat die Firewall die Funktion des Gateways zum Internet.

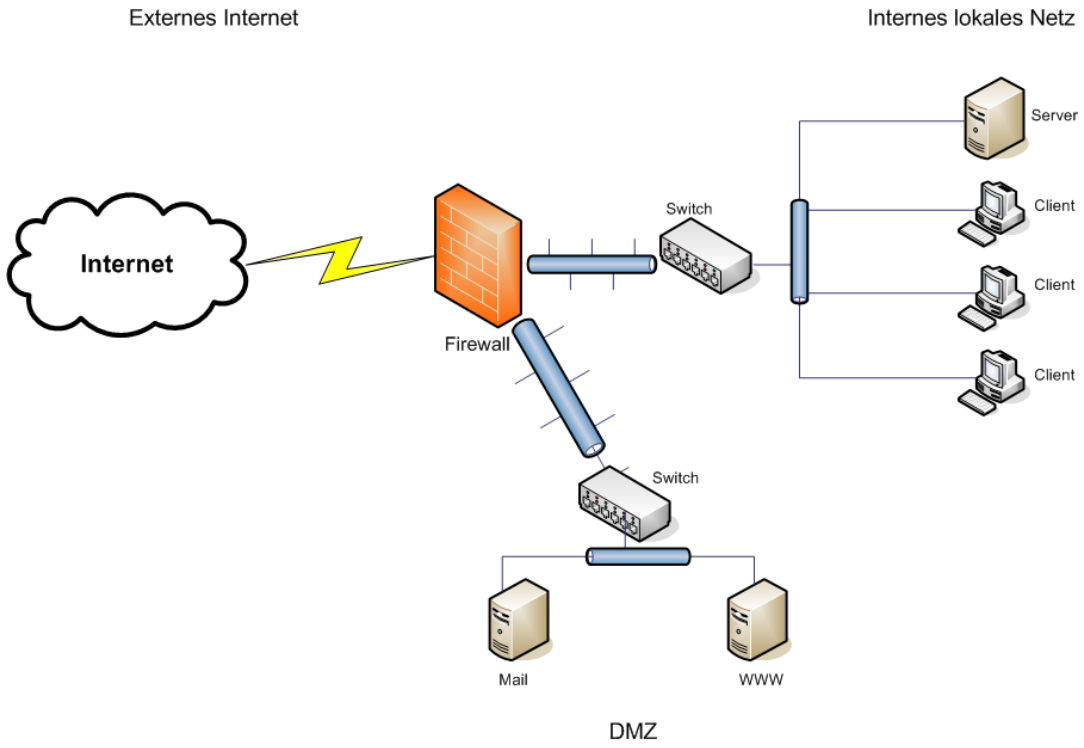

Abbildung 3.2.: Firewall mit einer DMZ

Allgemein ist eine Firewall bei der Anzahl ihrer intern zu schützenden Netzwerke nur durch ihre internen Netzwerkschnittstellen beschränkt. Beim Betrieb mit DMZ sind demzufolge neben einer externen Schnittstelle mindestens zwei interne Schnitstellen erforderlich.

3.2. Firewall Einführung

Nach OPPLIGER[2] unterscheidet man zwei Firewall-Grundkonzepte, Paketfilter und Anwendungs-Gateways, die auf vielfältige Art und Weise miteinander kombiniert werden können. Neben diesen Grundkonzepten gibt es andere Architekturen, etwa die Stateful-Inspection-Firewall, die den Grundgedanken des Paketfilters mit Funktionsweisen eines Anwendungs-Gateways kombiniert.

Ein Firewallsystem soll die Schutzfunktion des internen Netzwerks übernehmen. Da die Firewall an sich das Verbindungsstück zum anderen (meist externen) Netzwerk ist, fungiert sie auch als Gateway. Je nachdem wie das Firewallsystem mit Komponenten ausgestattet wird, werden verschiedene Typen realisiert.

Mögliche Komponenten eines Firewallsystems und der daraus resultierenden Firewall-Typen inklusive Vor- und Nachteile sind:

1. **Paketfilter**
   Ein *Paketfilter* untersucht alle ein- und ausgehenden TCP/IP Pakete anhand vordefinierter Regeln auf Passierbarkeit. Hierzu werden die Pakete in den OSI-Schichten 3 und 4 der TCP/IP Protokollfamilie überprüft. Bei der Überprüfung werden in der Regel Absender- und Empfängeradresse, sowie Absender- und Zielport überprüft. Moderne Paketfilter wie der von OpenBSD beherrschen darüber hinaus noch andere Prüfungskriterien. Nach diesen Kriterien kann dann der Zugriff auf die internen Netzwerke explizit anhand der Firewall-Regeln festgelegt werden. Das Ergebnis der Überprüfung ist dann entweder eine Abweisung, eine Verwerfung oder eine Erlaubnis zu passieren.

   Eine Erweiterung stellen die *stateful-packet-inspections Paketfilter* (dynamische, zustandsgesteuerte Filter) dar, bei denen nicht nur laufende Verbindungen überwacht, sondern auch in Abhängigkeit von bestimmten Protokolleigenschaften neue Filterregeln generiert und Ports temporär geöffnet bzw. nach Verbindungsende

[2] vgl. [Opp97]

wieder geschlossen werden. Insbesondere bei UDP-Paketen wäre ansonsten eine Überwachung nicht möglich, da diese zustandslos sind. Technisch werden für die durchgereichten Verbindungen in einer Tabelle *states* generiert um bei Bedarf dieses *states* als Informationsquelle auszulesen und zur weiteren Bearbeitung zu nutzen. *states* sind Zustandsbeschreibungen, die verschiedene Parameter enthalten wie Quell- und Zieladresse, Quell- und Zielport, sowie das verwendete Protokoll. Die Anzahl möglicher *state* Einträge hängt primär vom Hauptspeicher ab. Im Allgemeinen sind Paketfilter, die *stateful* sind deutlich flexibler in der Handhabung, einfacher zu warten und meist auch performanter wie Paketfilter ohne diese Eigenschaft. Zudem bietet die Existenz der *state-Tabelle* Möglichkeiten, weitere Schutzmaßnahmen zu integrieren.

Vorteile von Paketfilter:

- flexible Konfigurationen möglich, vor allem bei Einsatz von stateful-packet-inspections Paketfilter.
- leicht erweiterbar
- effektiv
- transparent für den Benutzer

Nachteile von Paketfilter:

- Konfiguration der Filterregeln erfordert hohe Kompetenz
- Fehler im Paketfilter stellen größere Gefahr dar, als Fehler bei anderen Firewall-Komponenten
- Durch Überprüfung auf TCP/IP Ebene nur IP-bezogene Regeln, keine benutzerbezogene Filterung möglich

2. **Application gateway**
   Bei einem *application level gateway* findet eine Überprüfung oder eine Filterung

nur auf Schicht 7 im OSI-Modell statt. Von der Funktionalität her stellen die application gateways einen Proxy-Server da, der auf einzelne bestimmte Anwendungen ausgerichtet ist. Das bedeutet, sie übernehmen die Vermittlerrolle in einer Verbindung und geben vor, der eigentliche Absender der Anfrage zu sein. Daher wird dieser Typ auch *proxy-firewall* genannt. Bei kritischen Protokollen wie ftp spielen sie eine wichtige Rolle. Zusätzlich können application gateways auch Authentifizierungsfunktionen anbieten. Vorteile von application level gateways:

- leicht einzurichten
- sehr effizient
- Möglichkeit auf Benutzerebene Zugriff zu beschränken

Nachteile von application gateways:

- Schutz auf Anwendungsebene begrenzt
- erfordert abgesicherten Rechner zum Betrieb („bastion host“)[3]
- kaum ausbaufähig, es sei denn in Kombination mit einem *Paketfilter*

Welcher Typ von Firewall eingesetzt wird, hängt primär davon ab, was man erreichen will. Ist ein einfacher, solider Schutz gefragt, dann wird man in der Regel einen *Paketfilter* bevorzugen. Soll es hingegen möglichst einfach und schnell realisiert werden, dann greift man zu einem *application level gateway*.

Anwender, die Wert auf Sicherheit legen, verwenden Kombinationen von beiden Typen. Oft ist dies auch unumgänglich, da insbesondere bei größeren Netzwerken mehrere Bereiche anders gesichert werden müssen. Man spricht dann vom *mehrstufigen Aufbau* eines Firewallsystems. Wieviele Stufen eingerichtet werden, hängt vom den Schutzanforderungen, der Infrakstruktur, dem geschätzten Verwaltungsaufwand und den Kosten ab. Eine andere Bezeichnung für den mehrstufigen Aufbau ist ein *multibox Firewallsystem*.

[3] Rechner, der nur das nötigste an Diensten besitzt, um möglichst sicher betrieben werden zu können

Schematische Darstellung eines mehrstufigen Aufbaus, wie sie auch vom Bundesministerium für Sicherheit in der Informationstechnik[4] empfohlen wird:

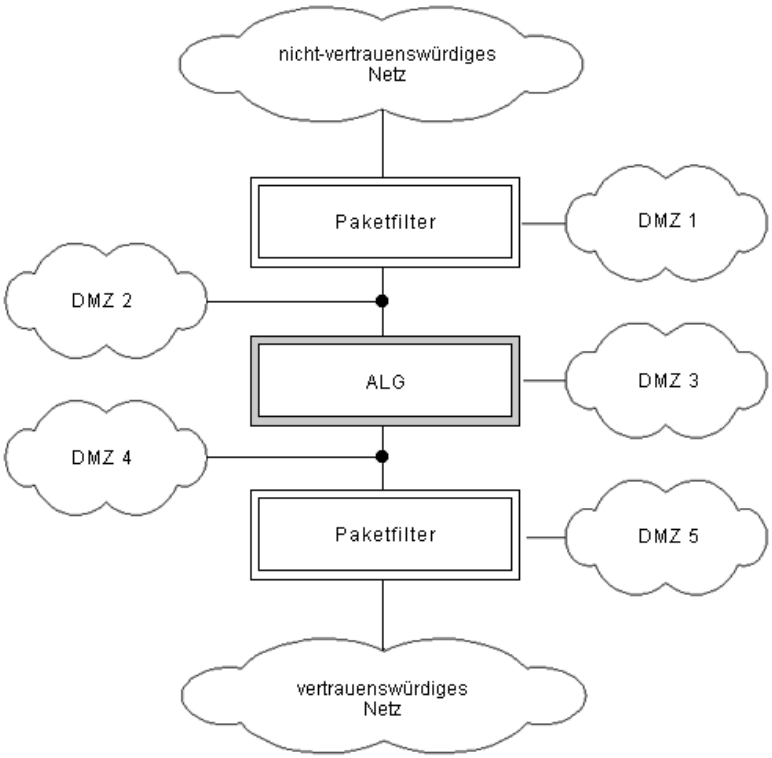

Abbildung 3.3.: Mehrstufiger Aufbau, bestehend aus einer Hintereinanderschaltung von Paketfilter, Application-Level-Gateway und einem weiteren Paketfilter (DMZ).

[4] vgl. [BSI] Abb.2

3.3. NAT - Network Address Translation

Eine weitere wichtige Funktion von gängigen Firewallsystemen ist das Verbergen des dahinter liegenden Netzwerks. Diese Funktion wird am einfachsten mit dem NAT-Verfahren[5] erreicht. Das NAT-Verfahren zeichnet sich im wesentlichen dadurch aus, dass die im lokalem Netz vergebenen nicht-öffentlichen IP-Adressen[6] durch den Router (hier die Firewall, die als Gateway arbeitet) umgeschrieben werden, damit diese auch routingfähig sind.

NAT ist ein Verfahren, dass historisch im Zuge der Publikmachung des Internets entstanden ist. Ursprünglich war das Internet ein Netzwerk, dass für das Militär entwickelt und später auch von Forschungseinrichtungen genutzt wurde. Die Anzahl der benötigten Internet Adressen (IP-Adressen) war demzufolge gering und es herrschte kein Bedarf, an der Spezifikation von IPv4[7] etwas zu verändern. Dies änderte sich rasch, als das Internet der breiten Masse zugänglich gemacht wurde. Internetprovider benötigten auf einmal ganze IP-Netzwerkbereiche, damit sie ihre Kunden vernetzen konnten. Auch der enorme Boom des Internets im privaten Bereich erforderte Handlungsbedarf. Zwar wurde relativ frühzeitig mit der Arbeit an dem Design von dem 128-bittigem IPv6[8] begonnen, dem offiziellem Nachfolger des 32-bittigem IPv4, um der wachsenden Nachfrage nach IP-Adressen gerecht zu werden, doch es musste temporär eine Zwischenlösung her. Dies war die Geburtstunde von NAT.

Die Funktionsprinzip von NAT ist, dass interne private Adressen auf eine externe öffentliche IP-Adresse umgesetzt werden. Dies geschieht direkt im IP-Header und wird vom Router bzw. der Firewall gehandhabt. Zur späteren Zuordnung geschieht dies intern durch Anlegen von Tabellen, deren Spalten die Adressumsetzungen abspeichern. Da dies in beiden Richtungen geschehen kann, unterscheidet man auch zwei Hauptvarianten von NAT:

[5] vgl. RFC 1631[IET94]
[6] Adressen aus dem privaten Adressbereich. Vgl. RFC 1918[IET96]
[7] vgl. [IET81]
[8] vgl. RFC 2460[IET98]

1. *source-NAT*
   Bei dieser Variante werden die internen IP-Adressen nach außen hin „verborgen“. Das ist auch die am häufigsten verwendete Variante und alle modernen Paketfilter beherrschen sie. Andere Bezeichnungen für *source-nat* sind *IP-Masquerading* und *outbound NAT*.

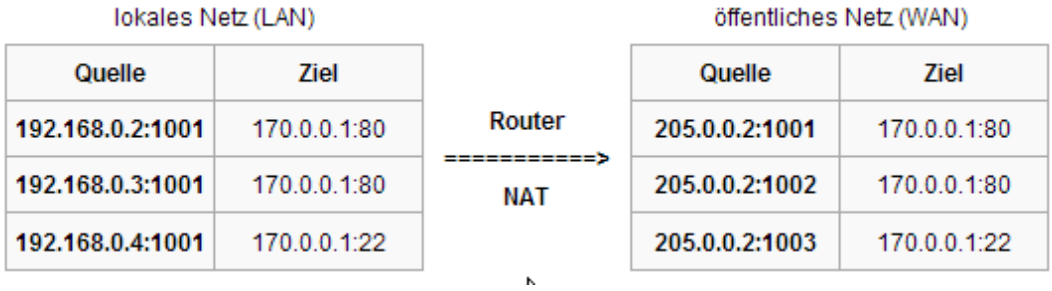

Abbildung 3.4.: Funktionsweise source-NAT

2. *destination-NAT*
   Bei dieser Variante werden verschiedene Anfragen an bestimmte Ports von außen gezielt intern an einen Rechner weitergeleitet. Diese Variante findet man oft vor, wenn Server im lokalen Netzwerk Dienste öffentlich anbieten sollen. Andere Bezeichnung hierfür lautet *inbound NAT*.

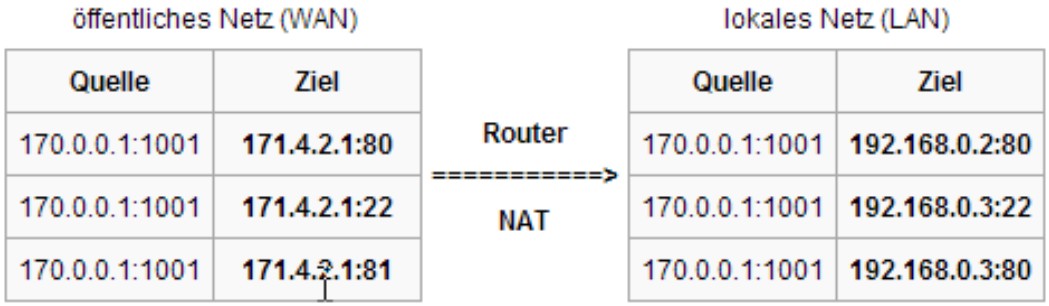

Abbildung 3.5.: Funktionsweise destination-NAT

Außer den zwei Hauptvarianten gibt es noch das ***1:1 NAT***, dass nur fortgeschrittene Firewallsysteme beherrschen. Genauer gesagt deren Paketfilterkomponente. Dabei wird die Adressumsetzung bidrektional in beiden Richtungen angewendet, so dass die Möglichkeit besteht öffentliche IP-Adressen direkt mit im internen Netzwerk stehenden Rechnern zu nutzen, obowhl sie mit einer privaten IP-Adresse konfiguriert werden. Die 1:1 Umsetzung der zu ansprechenden öffentlichen IP-Adresse wird von dem Firewallcode übernommen. Zum Betrieb muss die Firewall einen *arp proxy* bereitstellen.

Die Funktion der Nutzung eines ganzen Netzwerks hinter einer einzigen öffentlichen IP-Adresse hilft zwar, die Knappheit der Adressen zu verzögern, aber leider nicht zu beseitigen. Zudem bringt das NAT-Verfahren durch die Adressumschreibungen im IP-Header auch einige Komplikationen in Verbindung mit anderen etablierten Netzwerkprotokollen wie *IPSec* und *ftp* mit sich[9].

Das 15 Jahre nach Erfindung von NAT es immer noch eine bedeutende Rolle in der Vernetzung heutiger Netzwerke spielt, ist der nach wie vor sich verzögernden Einführung von IPv6 zu verdanken. Darum ist es wichtig, dass man als Firewall-Administrator sich mit dieser Technologie auskennt, die wenn es nach ihren Erfindern gehen würde, schon längst nicht mehr verwendet werden dürfte.

[9] vgl. RFC 3027[IET01b]

4. Das Firewallsubsystem pf von OpenBSD

compared to working with iptables,
PF is like this haiku:
A breath of fresh air,
floating on white rose petals,
eating strawberries.
Now I´m getting carried away:
Hartmeier codes now,
Henning knows why it fails,
fails only for n00b.
Tables load my lists,
tarpit for the asshole spammer
death to his mail store.
CARP due to Cisco,
redundant blessed packets,
licensed free to me.

Jason Dixon

Dieses Kapitel beschreibt und dokumentiert ausführlich *pf*, den OpenBSD Paketfilter, der als Softwaregrundlage zum Aufbau einer Firewall unter OpenBSD dient. Alle Angaben beziehen sich auf die Version 4.5 des OpenBSD Betriebssystems.

4.1. Geschichte von pf

OpenBSD als *fork*[1] von NetBSD hat lange Zeit *IPfilter* vom Australier Darren Reed als Paketfilter Subsystem eingesetzt. Dies änderte sich, als Darren Reed im Jahr 2001 bekannt gab, dass seine Software zwar unter der freien BSD-Lizenz stehe[2], er die Bedingungen aber anders intrepretiere. Dies wurde unter den OpenBSD Entwicklern, allen voran dem Projektleiter Theo de Raadt genauer diskutiert. Es stellte sich heraus, dass die Lizenz von *IPFilter* größtenteils wortwörtlich der BSD-Lizenz entsprach. Darren Reed gab jedoch bekannt, dass jegliche Veränderung am Code seine Genehmigung erfordere. Darauf wollte sich vor allem das OpenBSD Team nicht einlassen, da es nicht deren Philosophie entsprach und entfernte *IPFilter* kurzerhand aus dem Quelltextarchiv.

Etwa ein paar Monate vorher begann in der Schweiz ein Entwickler mit Namen Daniel Hartmeier seine ersten Programmierversuche am OpenBSD Kernel. Resultat war eine Vorabversion eines Paketfilters. Dieser wurde, weil ihm kein besserer Name einfiel, kurzerhand *pf* (Akronym für Paketfilter) genannt. Weitere OpenBSD Entwickler nahmen sich dem Quellcode vor und es entstand der offizielle *IPFilter* Nachfolger für das OpenBSD Projekt mit Einführung ab Version 3.0 des Betriebssystems.

pf hat sich seitdem nicht nur rasant entwickelt, sondern wurde auch auf den anderen freien BSD Betriebssystemen wie FreeBSD und NetBSD portiert und zum Standard erklärt. Allerdings ist natürlich die Variante von OpenBSD als Original und Entwicklerversion, immer bei den Funktionen mindestens einen Schritt voraus. Eine Portierung auf *Linux*, wie sie bei *IPFilter* existiert, gibt es jedoch nicht und ist auch nicht geplant.

[1] Bezeichnung für eine Abspaltung eines Projekts im Open Source Bereich
[2] vgl. [Wei]

4.2. Grundlagen von pf

pf gehört zur Klasse der *stateful* Paketfilter. Das bedeutet *pf* untersucht auch den Zustand der durchgereichten Pakete, um dynamisch agieren zu können. Wie zum Beispiel temporär ports zu öffnen etc.. *pf* ist fest in den OpenBSD Kernel integriert und arbeitet direkt in der Netzwerkschicht der TCP/IP Protokollfamilie.

Folgende Abbildung zeigt den internen Ablauf, wenn ein Paket im System bearbeitet wird:

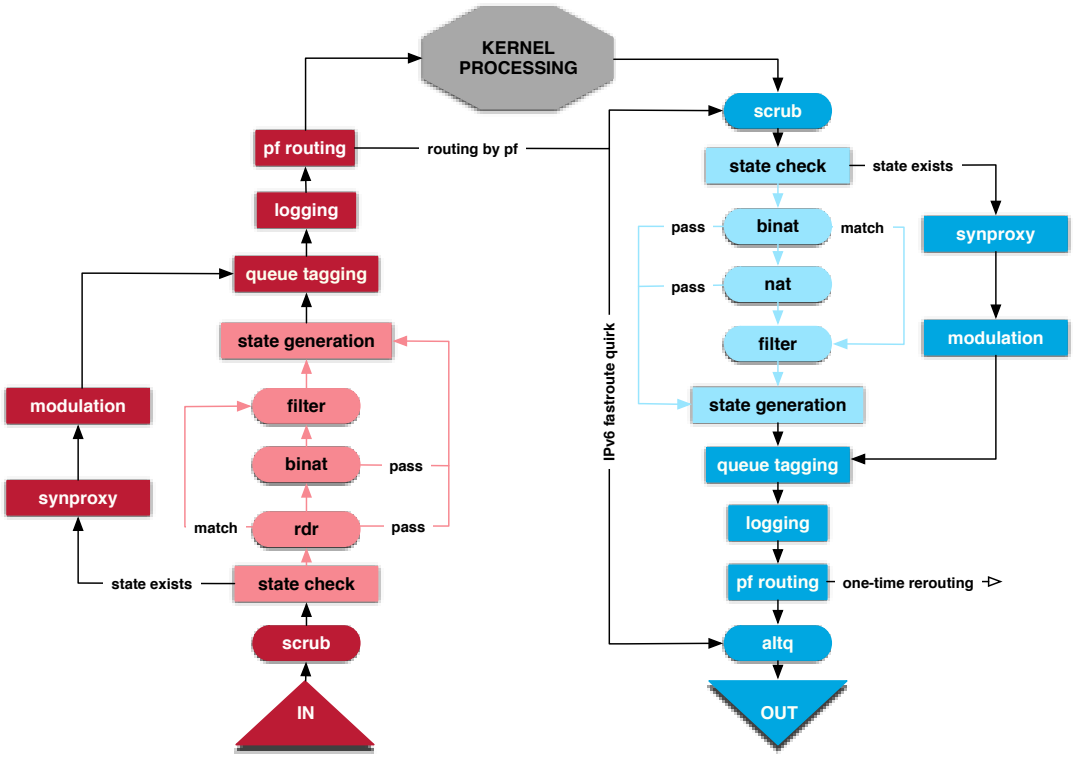

Abbildung 4.1.: Packetfluss von pf

pf, das ursprünglich nur als Ersatz für einen Paketfilter gedacht, ist in den letzten Jahren der Entwicklung zu einem ausgereiftem und funktionsstarken Firewallsystem herangereift. Unter Experten gilt *pf* als einer der mächtigsten und funktionalsten Firewallsyste-

me überhaupt, deren Benutzung sehr leicht zu erlernen ist, dank ausgeklügelter Syntax. So verwenden zum Beispiel renommierte Sicherheitsunternehmen wie die *Gesellschaft für Netzwerk- und Unix-Administration mbH (GeNUA)* OpenBSD und *pf* als Grundlage ihrer Firewallsysteme, die weltweit den höchsten Sicherheitsstandards genügen und unter anderem auch vom BSI dafür zertifiziert wurden[3].

4.2.1. Funktionen von pf

pf bietet folgende Funktionen entweder direkt oder in Kombination mit anderen OpenBSD Diensten an:

Paketfilterung	Paketweiterleitung	Paketuntersuchung
Paketpriorisierung	Paketnormalisierung	Paketverfolgung
automat. Optimierungen	Bandbreitensteuerung	Betriebssystemerkennung
Anti-Spoofing	NAT	IPv6 Unterstützung
failover mit CARP	anchors	state-Synchronisation mit pfsync
Redundanz mit CARP	AAA mit authpf	Lastverteilung mit relayd

Tabelle 4.1.: pf-Funktionsübersicht

Außer den Grundfunktionen eines modernen Paketfilters (siehe Zeile 1 der Tabelle), bietet pf noch eine ganze Menge mehr an technisch anspruchsvollen Funktionen.

Paketpriorisierung: Das QoS System ALTQ für BSD Unixsysteme wurde in OpenBSD komplett mit dem eigenem pf code verschmolzen. Somit bietet pf als einziger Paketfilter eine voll integrierte Bandbreitenkontrolle an.

Alle Datenpakete, die von einem Client Rechner gesendet werden, gelangen in eine Warteschlange auf dem Gateway. Das *scheduler* auf dem Gateway/Router entscheidet dann über die Reihenfolge der Abarbeitung der einzelnen Datenpakete und auch darüber, welche Warteschlange als nächstes bearbeitet wird. Demzufolge

[3] vgl. [GeN]

kann bei hoher Netzwerklast es Verzögerungen in der Paketdatenverarbeitung geben. Dies ist insbesondere dann der Fall, wenn von den Clients mehrere Programme parallel und häufig Netzdaten gesendet und empfangen werden.

Als Beispiel kann man sich einen großen Upload von einem Client vorstellen, der die ganze zur Verfügung stehende Bandbreite (=Internetanbindung)ausnutzt. Parallel dazu will der Client über SSH einen Rechner fernwarten. Die Folge ist, dass die SSH Sitzung unerträglich langsam arbeitet, da mit erheblichen Verzögerungen im Ansprechverhalten der Befehlseingaben zu rechnen ist. Im Idealfall sollte der Gateway mit dem installiertern Paketfilter differenzieren können, welche Paket zeitkritischer sind und welche auch kleinere Verzögerungen vertragen. In dem Beispiel macht es in der Praxis weniger aus, wenn die Uploadgeschwindigkeit kurzzeitig einbricht, aber eine verzögerte oder gar abgebrochene Verbindung bei SSH sollte besser vermieden werden.

Ursprünglich begründet liegt dieser Umstand daran, dass die SSH Pakete aufgrund ihres späteren Einreihens erst nach den Uploadpaketen verarbeitet werden. In modernen Systemen ist das durch mehrere Warteschlangen, die zudem auch noch parallel abgearbeitet werden durch das Multitasking des Kernels nicht ganz so auffallend. Sind jedoch die Warteschlangengrößen nicht ausreichend dimensioniert für alle Pakete, dann kommt es zur Verwerfung der Paket, was in Beispielfall zum Abbruch der SSH Verbindung führt.

Standardmäßig wird bei OpenBSD ein *fifo scheduler* eingesetzt, der sich unter anderem um die Verwaltung der Warteschlangen kümmert. Diese Art der Verarbeitung nach dem *fifo* Prinzip ist nicht sehr vorteilhaft, wenn man OpenBSD als performantes Firewallsystem einsetzen will. Darum haben die Entwickler zusätzlich zwei weitere *scheduler* implementiert:

- *Classed-Based Queuing* (CBQ)
- *Priority Queuing* (PRIQ)

Beim *CBQ*[4] wird durch einen Algorithmus die gesamte zur Verfügung Netzwerkbandbreite auf mehrere Warteschlangen oder Klassen aufgeteilt. Klassen können aus einer Reihe von Parametern gebildet werden. Beliebte Parameter sind Priorität, Netzwerkschnittstellen oder auch Applikationen. Die Warteschlangen werden in einer Hierarchiestruktur angelegt. Die oberste Ebene ist die Warteschlange an der Wurzel (*„root queue“*). Dieser obersten Warteschlange wird die gesamte zur Verfügung stehende Bandbreite zugewiesen. Unterhalb dieser können Kinder und Kindeskinder Warteschlangen definiert werden, jeweils mit eigenen Bandbreitengrenzen, aber in der Summe nur maximal so viel, wie die Wurzel vorgibt.

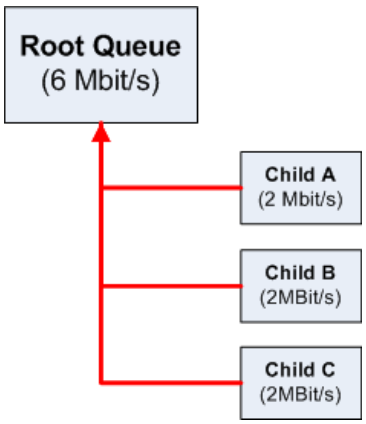

Abbildung 4.2.: einfache CBQ-Hierarchie

Eine genauere Klassifizierung erreicht man, in dem man die Hierarchiebenen erhöht. So kann man dann auch nach Benutzer oder nach Applikation priorisieren bzw. limitieren, was die Möglichkeiten der Anwendungen erhöht.

Durch Setzen des Parameters *borrows* ist es möglich, dass eine Warteschlange sich dynamisch zusätzliche Bandbreite ausleiht von anderen Warteschlangen, die diese gerade nicht nutzen. Die Reihenfolge der Abarbeitung dieser Warteschlangen geschieht abwechselnd auf oberster Ebene gleichmäßig verteilt, nach dem bekanntem *round robin* Verfahren. Innerhalb einer Unterebene durch Vergabe eines Priori-

[4] vgl. [Floa]

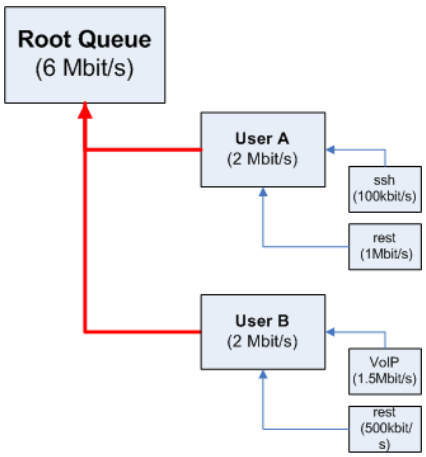

Abbildung 4.3.: erweiterte CBQ-Hierarchie

tätsparameters. Bedingung hierfür ist, dass die Warteschlangen den selben Vater besitzen.

Das *PRIQ* basiert darauf, dass einer Netzwerkschnittstelle mehrere Warteschlangen mit vorgebener Priorität zugewiesen werden. Diejenige Warteschlange mit der der höheren Priorität wird dann immer vor einer Warteschlange mit niedrigerer Priorität abgearbeitet. Ist die Warteschlange leer, dann wird mit der nächstniedrigeren fortgesetzt. Sollten zwei oder mehrere Warteschlangen diesselbe Priorität zugewiesen bekommen haben, dann wird auch hier wieder das bereits erwähnte *round robin Verfahren* angewendet.

Die Reihenfolge der Abarbeitung der Warteschlangen geschieht streng nach der Prioritätsnummernvergabe. Innerhalb der Warteschlangen kommt das konventionelle *fifo Verfahren* zur Anwendung.

Die Struktur des *PRIQ* Verfahrens ist im Gegensatz zum *CBQ* auf eine Ebene beschränkt, da außer der Prioritätsnummer keine weiteren Parameter zur Priorisierung berücksichtigt werden. Demzufolge ist es auch nicht möglich, die Priorisierung so fein wie beim *CBQ* zu gestalten.

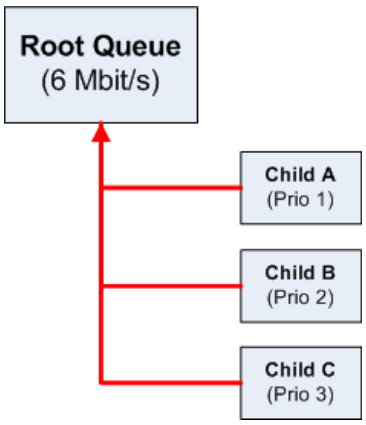

Abbildung 4.4.: PRIQ-Hierarchie

Zur präventiven Vermeidung von Staus in den Warteschlangen können zwei weitere Algorithmen verwendet werden: *Random Early Detection (RED)* und *Explicit Congestion Notification (ECN)*. Genauere Informationen über die Funktionsweisen der Algorithmen findet man bei Sally Floyd[5] und im RFC 3168[6].

Paketnormalisierung: Die Normalisierung von Netzwerkpaketen[7] ist eine Funktion, die *pf* einzigartig macht. Wenn eine Firewall als gateway benutzt wird, dann findet grundsätzlich eine Paketmanipulation statt. Bekanntestes Beispiel ist bei Einsatz von NAT. Diese Paketmanipulation erschwert es bestimmten Applikationen, insbesondere den *IDS-Systemen* präzise zu arbeiten und das genaue Ziel der Pakete zu identifizieren. Ein weiterer Fall liegt vor, wenn ein Angreifer seine IP-Pakete verfälscht und der reguläre Paketfilter keinen Misbrauch erkennen kann und somit Zugang gewährt. Die Umsetzung der Normalisierungsfunktion bei OpenBSD haben die Entwickler ***scrubbing*** getauft. Im einzelnen wird die Normalisierung erreicht, in dem alle ankommenden Pakete behandelt werden, um Zweideutigkeiten zu vermeiden. Fragmentierte Pakete werden wieder zusammengesetzt und ungültige TCP/IP Flagkombinationen bereinigt.

[5] vgl . [Flob]
[6] vgl. [IET01a]
[7] vgl. [HPK]

scrubbing hilft den Schutz zu verbessern und einige Fehler im Netzwerkstack von manchen Betriebssystemen auszugleichen, die dadurch verwundbar werden.

Betriebssystemerkennung: *pf* besitzt die Fähigkeit bei ankommenden Daten eine Analyse des verwendeten Betriebssystems durchzuführen. Dies geschieht anhand der Untersuchung von Merkmalen im *TCP SYN Paket* und einem Vergleich mit einer lokalen Betriebssystemliste. Mit dieser Funktion ist es möglich, Firewallregeln nicht nur auf Benutzer oder Applikationen, sondern auch auf das verwendete Betriebssystem wirken zu lassen.

Nachteile hierbei sind, dass die Pakete relativ einfach zu fälschen sind und es nur bei TCP Paketen funktioniert. Zudem kann bereits ein Update des Betriebssystems eine Erkennung unmöglich machen, da sich hierdurch die Kennzeichnung im SYN Paket geändert hat.

IPv6 Unterstützung: OpenBSD beherrscht von Haus aus komplett IPv6. Auch *pf* hier von Anfang an auf IPv6 Unterstützung entworfen worden, so dass einem Einsatz nach globaler Einführung von IPv6 nichts im Wege steht. Im Gegenteil, man kann sicher sein, dass die Firewall kompatibel und ausgereift ist in Verbindung mit den neuen Adressen.

AAA Unterstützung: In Verbindung mit der *authpf shell* wird *pf* zum Authentifizierungsgateway. Technisch ist *authpf* eine *shell*, die gestartet wird, wenn sich ein Benutzer via SSH auf der Firewall authorisiert hat. Dann wird ein vordefinierter Satz an Firewallregeln der Firewall hinzugefügt und bei Beendigung der Sitzung mitsamt aller verwendeten Verbindungen beendet. Auf diese Weise kann man verschiedenen Benutzern oder Benutzergruppen unterschiedlichen Zugriff auf das Netzwerk geben. Mögliche Anwendungsszenarien sind die Verwendung zur Anmeldung an ein Drahtlosnetzwerk oder für Außendienstmitarbeiter, die vom Hotel aus mal schnell sich ins Firmennetzwerk einwählen wollen. Zudem werden Benutzeran- und abmeldungen mit genauem Zeitstempel protokolliert.

Lastverteilung: *pf* bringt von Haus aus Funktionen zur Lastverteilung (*„load balancing“*) mit. Bei *pf* wird dies realisiert durch das Konzept von ***address pools***. Man definiert einen pool von mehreren Adressen, Rechnern oder Diensten und fügt die Anweisungen für das Load-Balancing hinzu. Dabei existieren 4 Methoden, wie so ein Adressen-Pool verwendet werden kann (*bitmask, random, source-hash und round-robin*), was auch 4 unterschiedlich Methoden für die konkrete Lastverteilung bedeuten. Zudem kann die Lastverteilung in beiden Richtungen, ein- und ausgehend, sowie mit NAT umgehen.

In Kombination mit dem *relayd* Dienst, *CARP* und *pfsync*[8] lassen sich sehr sichere Hochverfügbarkeitscluster aufbauen.

Redundanz: Redundanz ist Vorraussetzung für Hochverfügbarkeit. Redundanz wird durch die OpenBSD eigene Entwicklungen ***CARP*** und ***pfsync*** bereitgestellt. CARP, ein Protokoll, das den Aufbau von Redundanzverbänden ermöglicht, entstand aus einer patentrechtlichen Geschichte und stellte bei Vorstellung ein Meilenstein in der freien Softwareszene dar.

pfsync ist eine augeklügelte Protokollerweiterung von *pf*, die es ermöglicht, die Zustandstabellen des Paketfilters, sprich die Zustände aller derzeit in Bearbeitung befindlichen Verbindungen, mit anderen *pf* Firewalls vom selben Redundanzverbund synchronisieren zu können.

Details zu *CARP* und *pf* folgen in Kapitel 6.2.

Anti-Spoofing: *pf* besitzt einige interne Funktionen, die vor einigen bekannten Eindringsmethoden schützt. Zu diesen zählen beispielsweise Schutz vor TCP SYN-Flood, IP-Spoofing und uRPF Attacken.

[8] siehe nächsten Punkt „Redundanz“

4.2.2. Steuerung von pf

In der Regel wird das *pf* Subsystem beim Systemstart durch das Systemstartskript */etc/rc* permanent gestartet. Dazu muss der Systemverwalter in der Konfigurationsdatei */etc/rc.local.conf* die Startoption setzen:

```
$ sudo echo "PF=yes >> /etc/rc.conf.local"
```

pf liest dann aus der Konfigurationsdatei */etc/pf.conf* die zu benutzenden Regeldefinitionen aus. Es ist auch möglich durch Setzen der Variable *pf_rules* in */etc/rc.local.conf* einen eigenen Pfad zu einer Konfigurationsdatei zu definieren. Jedoch wird empfohlen, die Standarddatei */etc/pf.conf* zu benutzen, da diese besser in das Gesamtsystem integriert ist. Dies trifft beispielsweise auf die täglichen Wartungsskripte zu, die automatisch von der Firewall-Konfiguration ein Backup in */var/backup* anlegen.

Zur Steuerung von *pf* dient der Befehl ***pfctl***. Tabelle 4.2 stellt die wichtigsten *pfctl* Parameter vor[9].

-a	lade zusätzlichen *anchor*
-d	deaktiviert die Firewall
-e	aktiviert die Firewall
-f	gibt den Pfad zur Konfigurationsdatei an
-g	aktiviere Ausgabe (nützlich zum debuggen)
-h	zeige Hilfe
-F	setzt die Firewallregeln zurück (efordert Angabe der Regelsätze)
-k	löscht alle state-Datensätze eines Rechners
-n	Regeln nicht anwenden, sondern nur auf Syntax überprüfen
-s	listet Informationen über den Firewallzustand auf (erfordert Angabe von weiterem Parameter, welcher definiert, was aufgelistet werden soll)
-v	schaltet verbose-mode ein
-z	lösche Regelstatistiken

Tabelle 4.2.: pfctl-Parameterübersicht

[9] vgl. manual page von pfctl

4.3. Syntax und Konfiguration von pf

Beim Design von *pf* wurde viel Wert daraufgelegt, dass die Syntax einfach verständlich und logisch aufgebaut ist. Das Resultat ist eine Syntax die aussieht wie knappe englische Sätze. Auf Abkürzungen wird weitestgehend verzichtet. Eine einfaches Regelpaar, das sämtlichen eingehenden Datenverkehr verbietet und sämtlichen ausgehenden erlaubt, würde so aussehen[10]:

```
block in all    # nichts reinlassen
pass out all keep state # alles rauslassen (inkl. state)
```

Zum Vergleich hierzu das gleiche in *iptables* Syntax von Linux:

```
iptables -A INPUT -j REJECT
iptables -A OUTPUT -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
```

Eine *pf* Konfigurationsdatei besteht aus folgenden Komponenten, die je nach Bedarf kombiniert und eingesetzt werden:

- Listen und Makros
- Tabellen
- Optionen
- Queuing
- Übersetzung
- Filterregeln

[10] die Schlüsselwörter *keep state* können weggelassen werden, weil dies ab OpenBSD Version 4.1 standardmäßig an ist

4.3.1. Listen und Makros

Eine Liste dient der Zusammenfassung von mehreren ähnlichen Werten. Werte können Protokolle, Adressen, Rechner, Ports und Dienste sein. Listen erleichtern das Strukturieren des Regelwerks und das vereinfachte Verfassen von Regeln nach dem Minimalprinzip. Anstelle eine Regel pro Wert zu definieren, genügt es eine Liste mit ähnlichen Werten aufzustellen und eine einzige Regel für die Liste zu definieren. Folglich können komplexere Regeln und Regelwerke verkürzt und somit auch allgemein lesbarer und verständlicher dargestellt werden. Zudem erleichtert es auch die Wartung und Optimierung des Regelwerks.

Zum Definieren einer Liste genügt es die Werte in geschweiften Klammern {} zu setzen:

```
{ 192.168.1.1, 192.168.32.3, 192.168.11.1 }
{ ssh, smtp, domain, www, pop3s }
{ pc1, pc2 pc3 }
```

Die obigen Beispiellisten werden intern von *pf* wieder in ihre Elemente n zerlegt und es werden bei Einsatz der Listen n-Regeln erstellt. Die folgende Regel

```
block in em0 from { 192.168.1.1, 192.168.32.3, 192.168.11.1 } to any
```

wird für die Anwendung in diese drei Regeln umgewandelt:

```
block in em0 from 192.168.1.1 to any
block in em0 from 192.168.32.3 to any
block in em0 from 192.168.11.1 to any
```

In einer Regel dürfen auch mehrere Listen vorkommen und das Komma zur Trennung der Elemente ist optional. Auch ein Verschachteln unterschiedlicher Listen ist erlaubt. Listen sind nicht nur auf Filterregeln beschränkt, sondern sie können auch in Weiterleitungsregeln eingesetzt werden.

Makros sind benutzerdefinierte Variabeln, die IP-Adressen, Ports, Rechnernamen, Schnittstellennamen und auch Listen beinhalten können. Mit Makros lassen sich syntaktisch

korrekte Regeln einfacher und übersichtlicher entwerfen. Makrobezeichnung müssen mit einem Buchstaben beginnen und dürfen auch Zahlen beinhalten. Lediglich die Benutzung der reservierten Wörter ist nicht gestattet. Referenziert werden Makros durch das *$-Zeichen.*

```
externe_if = "fxp0" # Intel Netzwerkkarte
webserver = "{ 192.168.1.1, 192.168.1.2, 192.168.1.3 }"
tcp_dienste = "{ ssh, smtp, domain, www, pop3s}"

pass in on $ext_if proto tcp from any to $webserver port $tcp_dienste
```

Das obige Beispiel definiert die drei Makros externe_if (externe Netzwerkkarte), webserver (Liste aus drei Webservern) und TCP_Dienste (ports). Bei letzterem wurde für die ports, die Namensgebung der Dienste verwendet. Das System kennt alle ports und Namen, die in */etc/services* definiert sind. Die Filterregel besagt, dass alle TCP-Pakete (von allen Quellen und allen Quellports), die an der externen Netzwerkschnittstelle fxp0 ankommen mit den fünf Zielports passieren dürfen.

4.3.2. Tabellen

Mit Tabellen können IP-Adressen und ganze IP-Adressenbereiche (in *CIDR* Notation) definiert werden. Dabei spielt es keine Rolle, ob es sich um IPv4 oder IPv6 Adressen handelt. Ausnahmen sind hier die 0/0 Adresse und der 0.0.0.0/0 Bereich[11]. Diese können aber durch ein Makro ersetzt werden.

Der entscheidende Vorteil in der Verwendung von Tabellen im Vergleich zu Listen, liegt an der Geschwindigkeit und am Resourcenverbrauch. Tabellen sind deutlich performanter auszulesen wie Listen. Desweiteren sind sie effizienter im Prozessor- und Speicherverbrauch.

Tabellen können als Quell- oder Ziel- Adresse in Filterregeln, *scrub*-Regeln, Übersetzungsregeln wie NAT und in Umleitungsregeln eingesetzt werden. In NAT Regeln auch

[11] alle Adressen und Adressbereiche

als Übersetzungsadresse und in Umleitungsregeln auch als Umleitungsadresse. Eine weitere mögliche Verwendung von Tabellen ist der Einsatz als Ziel-Adresse bei *routing*-Regeln. Durch Voranstellung eines Ausrufezeichens (!) vor Adressen und Adressgruppen ist eine Negierung möglich.

Tabellennamen werden zwischen Spitzklammern (< >) geschrieben. Referenzierung erfolgt wie bei den Listen und Makros über das $-Zeichen. Definition der Tabellen werden in der in der Konfigurationsdatei mit der *table* Direktive eingeleitet. Zusätzlich können die Attribute *const* und *persist* gesetzt werden. Mit *const* wird die Tabelle als fix und nicht mehr manipulierbar definiert. Somit ist eine Änderung des Inhalts zur Laufzeit ausgeschlossen (siehe nächster Abschnitt). Ohne das Attribut *persist* wird eine Tabelle automatisch vom Kernel aus dem Speicher entfernt, wenn keine Regeln mehr auf die Tabelle referenzieren.

Tabellen lassen sich direkt während der Laufzeit mit Hilfe von *pfctl* manipulieren. So kann man schnell zu Testzwecken Änderungen am Regelwerk vornehmen, ohne dass man die Konfigurationsdatei hierfür bearbeiten muss. Mit *pftctl* ist es auch möglich den Tabelleninhalt im Hauptspeicher auszulesen. Damit kann man zur Laufzeit des ganzen Systems gezielt Werte einfacher aus umfangreichen Tabellen verändern. Voraussetzung hierbei ist, dass die Tabelle nicht explizit als unveränderbar definiert wurde (siehe weiter oben).

Beispiel um schnell eine neue IP-Adresse in die *blacklist* eines SPAM-Abwehrsystems aufzunehmen:

```
pfctl -t blacklist -Tadd 137.250.1.254
```

Falls die Tabelle noch nicht existiert, wird sie automatisch angelegt. Den Inhalt einer Tabelle kann man mit

```
pfctl -t blacklist -Tshow 137.250.1.254
```

ausgeben.

Eine Anzeige aller derzeit verfügbaren Tabellen erhält man durch

```
pfctl -s Tables
```

4.3.3. Anker - anchors

Neben der Konfigurationsdatei */etc/pf.conf* mit ihren Definitionen erlaubt *pf* den Einsatz von sogenannten *anchors* (Anker). Anker bieten zwei Vorteile für die Konfiguration der Firewall:

1. Erleichterung der Strukturierung längerer Konfigurationen, indem externe Konfigurationsdateien eingebunden werden können. Während Tabellen und Listen lediglich ähnlich Werte zusammenfassen, können *anchors* komplett eigene Regelsätze enthalten. Diese Möglichkeit erlaubt es verschiedene Regelsätze auszuprobieren, ohne dass die Basiskonfiguration in der Hauptdatei verändert werden muss. So kann eine modulare Trennung der Regelsätze erreicht werden.
2. Möglichkeit während der Laufzeit von *pf* komplexere Änderungen an der Konfiguration mit *pfctl* zu machen ohne, dass man das ganze Firewallsystem neustarten muss.

Um *anchors* für Filterregeln zu definieren verwendet man das Schlüsselwort *anchor* in der Konfigurationsdatei mit einem einer Variable, die als Namen des Ankers verwendet wird. Zur Aktivierung dient das Schlüsselwort *load*. Für NAT-Regelsätze stehen die Schlüsselwörter *nat-anchor* und *binat-anchor* zur Verfügung. Bei Weiterleitungsregelsätze das Schlüsselwort *rdr-anchor*.

Eim Auslagern eines Regelsatzes für die E-Mail Behandlung mit Hilfe eines Ankers kann sieht folgendermaßen aus:

```
anchor mail
load anchor mail from /usr/local/etc/pf_anchors/mail_rules
```

Zusätzlich erlauben *anchors* eine abstrakte Formulierung der Konfiguration. So kann man durch Einsatz von *anchors* Einfügemarken für externe Regelsätze festlegen, die aber erst bei Bedarf zur Laufzeit mit eigentlichen Regeln befüllt werden. Am einfachsten ist dabei die Regeln in einer normalen Textdatei zu definieren und dann mit *pfctl* dem Anker zu zuweisen.

Bei Annahme, dass in der */etc/pf.conf* ein Anker mit dem Namen „interneMitarbeiter“ definiert wurde, wendet man *pftctl* folgendermaßen in Verbindung mit der Textdatei „regeln-interneMitarbeiter“ an:

```
pfctl -a interne-Mitarbeiter -f /usr/local/etc/anchors/regeln-interneMitarbeiter
```

Eine weitere Möglichkeit besteht darin, einen Anker mit dem Asteriskzeichen (*) zu definieren. So werden dann alle Dateien, die im angegebenen Pfad liegen alphabetisch geladen und eingebunden.

```
anchor "spam/*"
```

Dieser Syntax ist gerade bei Verwendung in Verbindung mit *authpf* zur Benutzerauthentifizierung sehr nützlich.

Auch der Einsatz von *anchors* zur Modularisierung direkt in der Hauptkonfigurationsdatei kann Vorteile bringen. Dazu wird ein Anker oder eine Verschachtelung von Ankern mitsamt seinem Regelsatz als Block in der */etc/pf.conf* definiert.

```
anchor "external" on webserver {
            block
            anchor out {
            pass proto tcp from any to port { 80 }
            }
            pass in proto tcp to any port 22
            }
```

Das obige Beispiel definiert einen Anker, der auf einem Webserver vollen Zugriff über Port 80 (*http*) gewährt, aber direkt nur über Port 22 (*ssh* für Fernwartung).

4.3.4. Filterregeln

Filterregeln stellen die Kernkomponente einer Firewallkonfiguration dar. Mehrere Filterregeln werden als Regelsatz bezeichnet. Filterregeln bestimmen ob Datenpaket durchgelassen oder abgelehnt werden. Gefiltert wird direkt auf den Schichten 3 (IPv4 und IPv6) und auf Schicht 4 (TCP,UDP,ICMP und ICMPv6) anhand den Header-Feldern (Kriterien) des jeweiligen Protokolls. Die meistgenutzten Kriterien sind:

- Quelladressen
- Zieladressen
- Zielport
- das Protokoll an sich

Filterregeln definieren neben den Kriterien, auf die ein Paket zutreffen muss auch die resultierende Handlung (passieren oder ablehnen), die bei Übereinstimmung durchgeführt wird. Filterregeln werden in sequentieller Reihenfolge von oben nach unten durchgearbeitet. Es werden immer alle Regeln überprüft bis es zu einer Handlung kommt. Ausnahme bildet hier der Einsatz des Schlüsselwortes ***quick*** in einer Regel. Dann wird die weitere Abarbeitung übersprungen und die Handlung wird sofort ausgeführt. Ansonsten ist die Handlung der letzten zutreffenden Regel diejenige, die ausgeführt wird.

Die Syntax von Filterregeln lässt sich folgendermaßen darstellen:

```
action [direction] [log] [quick] [on interface] [af] [proto protocol] \
       [from src_addr [port src_port]] [to dst_addr [port dst_port]]  \
       [flags tcp_flags] [state]
```

action die Aktion, die bei Übereinstimmung ausgeführt wird. Es sind folgende Schlüsselwörter möglich: *block* zum Ablehnen und *pass* zum Durchlassen. Bei Durchlassung wird das Paket an den Kernel für die weitere Verarbeitung durchgereicht und bei Ablehnung greift die gesetzte Ablehnungspolicy in der Konfigurationsdatei[12].

[12] vgl. Abschnitt Optionen

direction die Richtung des Paketflusses. Entweder *in* für reinwärts oder *out* für rauswärts.

log ob das zu verarbeitende Paket protokolliert werden soll. Bei zustandsbezogenen Regeln (*stateful*) wird nur das Paket protokolliert, das den *state* aufgebaut hat. Mit Angabe von *log (all)* werden alle Pakete protokolliert.

quick bei Angabe des Schlüsselwortes *quick* wird die Regel als letzte zutreffende Regel interpretiert und deren Handlung (*action*) ausgeführt wird.

interface gibt die zu untersuchende Netzwerkschnittstelle an. Dabei kann es sich auch um eine Schnittstellengruppe handeln, die zuvor durch die Netzwerkkonfiguration angelegt worden ist, wie zum Beispiel bei *carp* Gruppen.

af die Adressfamilie des Pakets. *inet* für IPv4 und *inet6* für IPv6. *pf* kann in der Regel anhand der Quell- und Zieladressen die richtige Familie erkennen.

protocol das verwendete Schicht 4 Protokoll (tcp,udp,icmp,icmpv6), ein gültiges Protokoll aus */etc/protocols*, eine Protokollnummer zwischen 0 und 255 oder einen Satz von Protokollen aus einer vordefinierten Liste.

src-address, dst-address die Quell- und Zieladresse(n) aus dem IP-Header. Bei der Angabe können verschiedene Notationen und Schlüsselwörter benutzt werden:

- einzelne IPv4- oder IPv6-Adresse
- CIDR-Netzwerkblock
- ein *fqdn*[13], der beim Laden des Regelsatzes via DNS aufgelöst und dann eingesetzt wird
- den Namen der Netzwerkschnittstelle. Dann werden alle Adressen, die der Schnitstelle zugewiesen wurden eingesetzt

[13]fully qualified domain name: kompletter Domainname inkl. Hostangabe wie www.hs-augsburg.de

- den Namen der Netzwerkschnitstelle gefolgt von einer Netzmaske. Dann werden alle Adressen der Schnittstelle mit der Netzmaske kombiniert um einen CIDR-Netzblock zu schaffen, der dann eingesetzt wird.
- den Namen der Netzwerkschnittstelle in Klammern. Das weist *pf* an, sämtliche der Schnittstelle zugeordneten Adressen dynamisch zu aktualisieren, falls sich diese ändern. Verwendung beispielsweise bei dynamisch zugewiesenen Adressen (*DHCP*).
- eine Liste oder Tabelle
- sämtliche oben genannten Varianten als Negation unter Verwendung des Ausrufezeichens (!)
- das Schlüsselwort ***any*** für alle Adressen
- das Schlüsselwort ***all*** das eine Kurzschreibweise von *from any to any* darstellt.

src-port, dst-port der benutzte Quell- und Zielport. Angaben wie folgt:

- als gültige Nummer zwischen 0 und 65535
- als gültiger Dienstname aus */etc/services*
- als Menge von ports anhand einer Liste
- als ein Bereich mit mathematischer Notation:
  - ungleich (!=)
  - kleiner als oder größer als (< >)
  - kleiner gleich oder größer gleich (<= >=)
  - als Bereich (><) und invertierter Bereich (<>). Dies sind binäre Operatoren und benötigen zwei Argumente zur Eingrenzung, fügen die Argumente aber nicht in den Bereich ein.

– inklusiver Bereich (:), ebenfalls ein binärer Operator, der aber die Argumente in den Bereich einfügt.

tcp_flags gibt die TCP-flags an, die im TCP-Header der zu untersuchenden Pakete enthalten sein müssen. Standard ist hier S/SA, was *pf* mitteilt, dass nur auf die S- und A- (SYN und ACK) Flags geachtet werden und zutreffen soll, wenn nur das SYN-Flag aktiviert ist

state gibt an, ob Statusinformation (*states*) zur Verbindung generiert werden sollen und auf welche Weise.

- *keep state* normale states erzeugen (Standardwert)
- *modulate state* starke zufallsgenerierte states erzeugen
- *synproxy state* Benutzung des SYN-Proxy um alle eingehenden Pakete erst über den Proxy weiterzuleiten, anstatt direkt auf die Ziel bzw. Quellsysteme. Schützt vor gefälschten SYN-Pakete und beinhaltet *keep state* und *synproxy state*.

Anzumerken ist, dass seit OpenBSD Version 4.1 Verbindungen standardmäßig immer *stateful* erzeugt werden. Das bedeutet, die Angabe von *keep state* in den Filterregeln kann entfallen.

Außerdem beherrscht *pf* auch für **UDP-Verbindungen *states***. Dies ist insofern speziell, weil UDP ein zustandsloses Protokoll ist und eigentlich dazu nicht in der Lage ist. Technisch realisiert wird dies, indem für UDP-Verbindungen beim Aufbau regulär ein Eintrag in der state-Tabelle vorgenommen wird (wie bei TCP) und gleichzeitig eine Ablaufzeit (*timeout*) für diese Verbindung gesetzt wird. So lange die Zeit noch nicht abgelaufen ist, werden alle UDP-Pakete, die zu dieser Verbindung gehören passieren können.

4.3.5. NAT Konfiguration

Ein elementarer Bestandteil von Firewalls, die auch Gateway-Funktionen übernehmen, ist die Konfiguration des NAT-Verhaltens. *pf* bietet dazu folgende Syntax:

Die Syntax von Filterregeln lässt sich folgendermaßen darstellen:

```
nat [pass [log] on interface [af] from src_addr [port src_port] to \
    dst_addr [port dst_port] -> ext_addr [pool_type] [static-port]
```

nat das Schlüsselwort, das NAT aktiviert.

pass Pakete werden durch diese Option von den Filterregeln ignoriert

log Protokollierung des ersten NAT-behandelten Pakets. Mit *log (all)* werden alle Pakete protokolliert.

interface das Interface auf dem NAT eingesetzt werden soll

af die zu verwendende Adressen-Familie. Entweder *inet* für IPv4 oder *inet6* für IPv6 Pakete.

src-address die interne Quelladresse des Pakets, das übersetzt werden soll. Notation analog zu den Paketfilterregeln, allerdings ohne das Schlüsselwort *all*.

src-port der zu benutzende Quellport. Notation ananlog zu den Filterregeln.

dst-address die Zieladresse, der zu übersetzenden Adressen. Notation genau wie bei *src-address*

dst-port der Zielport. Notation analog zu *src-port*.

ext_address die extern zu verwendende Ip-Adresse, auf die alle internen Adressen übersetzt werden. Notation analog zu den Paketfilterregeln.

pool_type bestimmt den *address pool*, der für das NAT verwendet wird

static_port *pf* soll den Quellport der TCP- und UDP-Pakete nicht verändern

Bei Benutzung von automatisch vergebenen Adressen kann wie bei den Filterregeln der Einsatz von Klammern um die Adressen oder den Schnittstellennamen gearbeitet werden. Für das spezielle 1:1 NAT kann das Schlüsselwort *binat* anstelle von *nat* verwendet werden.

Beispiel für einfaches NAT, bei dem ein ganzes internes Netzwerk auf eine öffentliche IP übersetzt wird:

```
nat on fxp0 from 192.168.192.0/24 to any -> 88.198.32.222
```

Eine Verwendung von 1:1 NAT bei der ein interner SVN-Server auf eine öffnetliche IP-Adresse reagieren soll sieht so aus:

```
svn_int = "192.168.192.100"
svn_ext = "88.198.32.223"

binat on fxp0 from $svn_serv_int to any -> $svn_serv_ext
```

Eine Möglichkeit mit Hilfe von *pfctl* einer Liste der aktiven NAT-Verbindungen anzuzeigen sieht so aus (inkl. möglicher Ausgabe):

```
pfctl -s state

fxp0 TCP 192.168.0.5:1333 -> 88.198.215.222:41436 -> 65.42.33.245:22 TIME_WAIT: ↩
    TIME_WAIT
fxp0 UDP 192.168.0.5:2491 -> 88.198.215.222:60527 -> 65.42.33.245:53   MULTIPLE: ↩
    SINGLE
```

Die Ausgabe oben gibt zwei aktive NAT-Verbindungen aus. Eine SSH Verbindung vom Rechner 192.168.0.5 auf Rechner 65.42.33.245. Die IP-Adresse 88.198.221.222 gehört dem *pf* gesteuerten Gateway. Die zweite Verbindung analog dazu, wobei es sich um eine DNS-Abfrage (UDP Protokoll!) handelt.

4.3.6. Weiterleitungen

Bei Einsatz von NAT und dem Wunsch nach Zugriff auf bestimmte Dienste im internen Netzwerk kommen Weiterleitungen zur Anwendung. Weiterleitungen können auch Umleitungsfunktionen darstellen, die beispielsweise bei Einsatz von lastverteilenden Gateways erfoderlich sind. Weiterleitungen werden bei *pf* mit dem Schlüsselwort *rdr* als Abkürzung für redirect definiert.

Die Syntax von Weiterleitungsregeln lässt sich folgendermaßen darstellen:

```
rdr [pass [log] on interface [af] from src_addr [port src_port] to \
    dst_addr [port dst_port] -> ext_addr [pool_type] [static-port]
```

Eine einfache Weiterleitung aller externen Anfragen auf den TCP-Port 80 (http) wird im folgenden Beispiel an einen internen Webserver gezeigt:

```
www_int = "192.168.192.100"
rdr on tl0 proto tcp from any to any port 80 -> $www_int
```

Ganze Bereiche lassen sich mit *Anfang:Ende* definieren. Dabei ist auch ein Einsatz vom Metazeichen * erlaubt.

Wichtig hierbei ist, dass sämtliche Weiterleitungspakete auch nach den Weiterleitungsregeln die Filterregeln durchlaufen. Nur wenn dort eine Übereinstimmung stattfindet, wird es zu einer Weiterleitung kommen. Dies kann man mit dem Schlüsselwort *pass* umgehen.

4.3.7. Optionen

Durch Optionsparameter in der Konfigurationsdatei */etc/pf.conf* kann man das Laufzeitverhalten von *pf* bestimmen. Folgende Optionen können gesetzt werden:

set block-policy *option* setzt das Standardverhalten der *block* Anweisung auf den Wert von *option*. Dieser kann *drop* für eine sofortige Verwerfung der Pakete oder *return* zur Übermittlung eines TCP-RST-Pakets bei geblockten TCP-Paketen und ein ICMP-UNREACHABLE für alle anderen Pakete sein. **Standardwert ist *drop*.**

set debug *option* setzt den *debug-level*. Mögliche Optionen sind *none* für keine debug Meldungen, *urgent* nur für ernste Fehler, *misc* für verschiedene Fehler in der state-Erzeugung und *loud* für Meldungen von typischen Ereignissen wie beispielsweise den Status einer Betriebssystemerkennung. **Standardwert ist *urgent*.**

set fingerprints *file* definiert den Pfad zur Datei, die die zu erkennenden Betriebssysteme als *fingerprint* enthält. **Standardwert ist */etc/pf.os***

set limit *option value* setzt verschiedene Begrenzungen für die Arbeit von *pf*. Mögliche Optionen und deren Standardwerte in Klammern dahinter:

frags maximale Anzahl der Einträge im gemeinsamen Speicher für die Verarbeitung der Paketnormalisierung. **(5000)**

src-nodes maximale Anzahl der Einträge im gemeinsamen Speicher, die für den Nachweis von Quell-Adressen verwendet werden. **(10000)**

states maximale Anzahl von *states* die gespeichert werden. **(10000)**

tables maximale Anzahl von Tabellen. **(1000)**

table-entries maximale Anzahl aller Einträge von allen Tabellen. **(200000)**[14]

set loginterface *interface* definiert die Netzwekschnittstelle, von der Statistiken erzeugt werden sollen. Es handelt sich hier um Volumenstatistiken sowie Verabeitungsstatistiken. Es liegt aus Geschwindigkeitsgründen eine Beschränkung auf **eine einzige** Schnitsttelle vor. **Standardwert ist *none*.**

[14] bei Systemen mit weniger als 100MB RAM, wird der Wert auf 100000 reduziert

set optimization *option* setzt die automatischen Optimierungsmaßnahmen von *pf*. Mögliche Optionen sind:

normal für die meisten Netzwerke

high-latency für Netzwerke mit hoher Latenz wie zum Beispiel Satelliten-Verbindungen oder Mobilfunkverbindungen.

aggressive verkürzte Verweildauer der state-Einträge in der state-Tabelle. Nur empfohlen für extrem ausgelastete Netzwerke, da die Gefahr besteht, dass temporär nicht genutzte Verbindungen gelöscht und somit getrennt werden. Reduziert deutlich den Speicherverbrauch einer ausgelasteten Firewall.

conservative lange Verweildauer der state-Einträge und somit werden auch nicht genutzte Verbindungen gehalten auf Kosten von Speicherverbrauch.

Standardwert ist *normal*.

set ruleset-optimization *option* steuert die Funktion des internen Regelsatzoptimierers. Mögliche Optionen:

none keine automatischen Optimierungen

basic aktiviert die folgenden Regelsatzoptimierungen:

1. Regelkopien werden entfernt
2. Redundante Regeln werden entfernt (Regeln, die durch andere Regeln komplett abgedeckt werden)
3. Zusammenfassung von Regeln in einer Tabelle, wenn vorteilhaft
4. Neusortierung der Regeln, um Geschwindigkeit zu erhöhen

profile Verwendung der aktuell geladenen Konfiguration, um die Reihenfolge der *quick* Operationen zu optimieren

Standardwert ist *basic*.

set skip on *interface* keinerlei Anwendung und Verarbeitung auf der genannten Schnittstelle. Nützlich bei *loopback-Schnittstellen*. Option kann mehrfach in der Konfiguration eingesetzt werden. **Standardmäßig nicht gesetzt.**

set state-policy *option* setzt das Verhalten der state-Interpretierung. Mögliche Optionen sind:

if-bound states sind gekoppelt mit der Schnittstelle, an der sie erstellt wurden. Wenn Pakete eintreffen, die übereinstimmend sind mit einem Eintrag in der state-Tabelle, aber nicht von der Schnitstelle kommen, wie im Eintrag aufgezeichnet, dann wird diese Übereinstimmung ignoriert. Sollten die Pakete auf keine Filterregel treffen, dann werden diese verworfen oder abgelehnt.

floating keine Überprüfung auf eine explizite Schnittstelle. Pakete dürfen von allen Schnittstellen kommen, so lange sie einen gültige state-Eintrag haben und die Richtung stimmt.

Standardwert ist *floating*.

set timeout *option value* setzt einige Zeitbeschränkungen. Mögliche Optionen mit ihren Standardwerten dahinter in Klammer:

interval Zeit zwischen Bereinigung von abgelaufenen states und Paketfragmenten. **(10s)**

frag Zeit bevor ein nicht-zusammengesetztes Paket abläuft. **(30s)**

src.track Zeit in der ein *source-tracking* Eintrag im Speicher gehalten wird nach Ablauf des letzten *states*. **(0)**

4.3.8. scrub

Das ***scrubbing*** wird mit dem Schlüsselwort *scrub* konfiguriert. Einfachste Variante ist:

```
scrub in all
```

Damit werden alle eingehenden Pakete normalisiert. Zusätzlich kann mit folgenden Parametern die Paketnormalisierung angepasst werden:

no-df entfernt das „nicht-fragmentieren-Bit“ vom IP-Header. Einige Betriebssysteme setzten dieses Bit automatisch in ihren IP-Paketen, welches zu keiner Ausführung von *scrub* führt. Besonders bei Nutzung von *nfs* ist dies der Fall.

random-id ersetzt das IP-identification-Feld von Paketen mit zufälligen Werten, um Betriebssysteme auszugleichen, die vorhersehbare Werte verwenden. Diese Option gilt nur für Pakete, die nach dem optionalen Zusammensetzen der Pakete nicht fragmentiert sind.

min-ttl *num* Setzen und Erzwingen einer minimalen Lebenszeit (*TTL*) von IP-Paketen in deren Header.

max-mss *num* Setzen und Erzwingen einer maximalen Segmentgröße (*MSS*) in TCP-Paketgrößen.

fragement reassemble Pufferung von fragmentierten eingehen Paketen und Zusammensetzung dieser, damit die Filterregeln nur komplette Pakete zur Bearbeitung bekommen. Vorteil liegt in der Selektion der zu bearbeitenden Pakete bei den Filterregeln. Nachteil ist der erhöhte Speicherverbrauch. Bedingung für Zusammenarbeit mit NAT.

fragment crop doppelte Fragmente werden ignoriert und überlappende abgeschnitten. Keine Pufferung, sondern direktes Durchlassen der Pakete.

fragment drop-ovl ähnlich *fragment crop*, außer dass auch doppelte und überlappende Pakete ignoriert (fallen gelassen) werden.

reassemble tcp normalisiert TCP-Verbindungen mit Beachtung ihrere *states*. Bei Anwendung darf keine Richtung angegeben werden, da diese bereits im *state* vorgegeben ist. Konkret werden zwei Maßnahmen getroffen:

- Verbot die TTL zu reduzieren

- Veränderung des Zeitstempels im TCP-Paket-Header mit einer Zufalsszahl.

4.3.9. Priorisierung mit ALTQ

Das Subsystem *ALTQ* zur Priorisierung von Paketen wurde vollständig in *pf* integriert, so dass die Anwendung keine extra Konfiguration extern erfordert. Zur Konfiguration dienen die Schlüsselwörter *altq on* und *queue*. *altq on* aktiviert die Priorisierung, bestimmt welcher Algorithmus verwendet wird und legt die Wurzelwarteschlange an. *queue* definiert die Eigenschaften der Kind-Warteschlangen.

Die allgemeine Syntax von *altq* lautet:

```
altq on interface scheduler bandwidth bw qlimit qlim \
     tbrsize size queue { queue_list }
```

interface die zu benutzende Netzwerkschnittstelle für die Priorisierung

scheduler den zu verwendenden Algorithmus bzw. *scheduler*. Mögliche Werte sind *cbq* und *priq*. Pro Schnittstelle kann nur ein Algorithmus aktiv sein.

bw die gesamte zur Verfügung stehende Bandbreite. Angaben der Werte können mit den Zusätzen b,Kb,Mb und Gb gemacht werden, die als bits, Kilobits, Megabits und Gigabits verstanden werden. Außerdem kann die Angabe auch relativ in Prozent der Schnittstellenbandbreite gemacht werden.

qlim optionale Bestimmung der maximalen Anzahl der Pakete, die in der Warteschlange gehalten werden soll. **Standardwert ist hier 50**.

size Größe des *token bucket regulators* in byte, der die Anzahl der Pakete limitiert, die ein Treiber aus einer Warteschlange auf einmal zur Verarbeitung herausziehen kann. Wenn der Wert nicht gesetzt wird, dann wird die gesamte Netzwerkschnittstellenbandbreite benutzt.

queue_list Menge an Kind-Warteschlangen, die unterhalb der Wurzel-Warteschlange erstellt werden

Folgendes Beispiel demonstriert eine CBQ-Aktivierung mit 10Mbit Bandbreite und drei Kind-Warteschlangen für drei gebräuchliche Dienste.

```
altq on fxp0 cbq bandwidth 10Mb queue { www, ssh, svn }
```

Zur detaillierten Konfiguration der Kind-Warteschlangen wird das Schlüsselwort *queue* verwendet. Die Syntax von *queue*:

```
queue name [on interface] bandwidth bw [priority pri] [qlimit qlim] \
      scheduler ( sched_options ) { queue_list }
```

name Name der Warteschlange (maximal 15 Zeichen lang)

interface optionale Angabe der Netzwerkschnittstelle die für die Warteschlange gültig ist. Wenn keine Angabe erfolgt, dann gilt sie für alle im System vorhandenen.

bw die Bandbreite, die der Warteschlange zugewiesen wird (nur gültig bei *CBQ*). Notation analog zu Schlüsselwort *altq*. Bei keiner Angabe wird die gesamte Bandbreite der Warteschlange auf der nächsthöheren Ebene genutzt.

pri die Priorität der Warteschlange. Bei *CBQ* von 0-7 und bei *PRIQ* von 0-15, wobei der Wert 0 die kleinste Priorität darstellt. Bei keiner Angabe wird 1 als Wert genommen.

qlim analog zum *qlim* Wert von *altq*

scheduler analog zum *scheduler* Wert von *altq* mit Bedingung das der Wert übereinstimmt mit der Warteschlange von der nächsthöheren Ebene.

sched_options optionale Zusatzoptionen um das Verhalten der *scheduler* zu bestimmen:

default definiert eine Standardwarteschlange, in der alle Pakete landen, die keiner anderen zugewiesen werden. Genau eine ist erforderlich.

red aktiviert *RED* in der Warteschlange

rio aktiviert *RED* in der Warteschlange im Ein- und Ausgangsmodus.In diesem Modus wird RED mehrere durchschnittliche Warteschlangenlängen verwalten und mehrere „treshold“-Werte, einen für jeden IP-QoS-Level.

ecn aktiviert *ECN* auf der Netzwerkschnittstelle. Beinhaltet *RED*

borrow bei Einsatz von *CBQ* kann durch diesen Parameter nicht Bandbreite von anderen Warteschlangen in der selben Ebene „ausgliehen“ werden

queue_list die Kind-Warteschlangen, die unter dieser Warteschlange erstellt werden sollen. Nur bei Einsatz von *CBQ*.

Um Datenverkehr einer Warteschlange zu zuweisen wird das *queue* Schlüsselwort in Verbindung mit Filterregeln eingesetzt.

Beispiel:

```
pass out on fxp0 from any to any port 80 queue www
```

4.3.10. Address Pools

address pools stellen ein syntaktisches Hilfsmittel für Weiterleitungsregeln und NAT-Regeln dar. Faktisch sind sie eine weitere Art der Ansammlung von IP-Adressen, deren Verwendung unter einer Gruppe von Benutzern bzw. Regeln geteilt wird. Eingesetzt kann ein *adress pool* bei Weiterleitungs- und Übersetzungsregeln. Es existieren 4 Methoden, wie ein solcher pool eingesetzt werden kann:

bitmask setzt den Netzwerk-Teil der Pool-Adresse über die Adresse, die modifiziert wird (Source-Adresse für NAT-Regeln, Ziel-Adresse für RDR-Regeln).

random zufällige Auswahl einer Adresse aus dem Pool

source-hash verwendet eine *hash* der Quelladresse, um eine Adresse aus dem Pool zu wählen. Dies stellt sicher, dass während der ganzen Sitzung immer wieder die gleich

Pool-Adresse verwendet wird. Wichtig vor allem bei sitzungsbasierten Anwendungen.

round-robin klassische Methode bei der alle Adressen im Pool der Reihe nach durchlaufen werden. Einzige Methode bei Definition des Pools als Tabelle.

round-robin stellt die Standardmethode dar. Außerdem wurde eine Erleichterung eingebaut für die zwei gebräuchlisten Methoden (*round-robin* und *random*). So muss man lediglich das Schlüsselwort *sticky-address* anfügen, damit *pf* bei der Auswahl immer die selbe Adresse aus dem Pool verwendet.

address pools spielen eine große Rolle, wenn man mit *pf* eine Lastverteilung oder Redundanz einrichten will. Im Abschnitt „Anwendung von pf“ wird darauf näher eingegangen.

5. Beispielanwendungen von pf

In diesem Kapitel werden einige gängige Anwendungen eines Firewallsystems mit *pf* beschrieben und wie man sie unter OpenBSD einrichtet und betreibt. Alle vorgestellten Anwendungen sollen nur als Grundlage dienen zum weiteren Ausbau. Die Anwendungen werden in der Art eines Rezepts beschrieben, das folgenden Aufbau hat:

- Lage
- Ziel
- Durchführung

Grundvoraussetzung zum Betrieb von *pf* zur Paketverarbeitung ist das Aktivieren von IP-Weiterleitungen im Kernel. Dies geschieht manuell während der Laufzeit mit dem Befehl:

```
sysctl net.inet.ip.forwarding=1
```

Dauerhaft eingerichtet wird der Kernelparamter in der Datei */etc/sysctl.conf*, damit auch bei Neustart die Weiterleitungen aktiviert bleiben.

5.1. pf als einfacher NAT-Router

Sicherlich eine oft verwendete Funktionalität eines Firewallsystems, gerade in kleineren Netzwerken ist der Einsatz als NAT-Router in einer singlebox-Umgebung.

Lage: ein einfaches lokales Netzwerk soll mit einem externen Netzwerk verbunden werden über einen gemeinsamen Zugang. Zur Verfügung steht ein Internetzugang über einen Internetprovider, der die IP-Adresse dynamisch vergibt.

Ziel: alle Rechner im lokalem Netz sollen uneingeschränkt das externe Netzwerk nutzen können. Ein Zugriff auf die internen Rechner im lokalem Netzwerk soll nur dann erlaubt sein, wenn die Verbindung vorher aus dem internem Netzwerk aufgebaut wurde. Die Konfiguration soll für die Client-Rechner möglichst einfach gestaltet werden, so dass keine großen Anpassungsarbeiten erforderlich sind.

Durchführung: Wir setzen OpenBSD mit *pf* eine Routerkonfiguration auf. Die Anforderung die Client-Anpassungen minimal zu halten, erfüllt man durch Einsatz eines DHCP-Servers auf dem Router. Dazu genügt eine Anpassung von **/etc/dhcpd.conf** nach eigenem Bedarf und zum Starten die Variable *dhcpd_flags* von *NO* auf „“ in der Systemstartdatei */etc/rc.conf.local* zu setzen. Für das Routing und das NAT wird eine einfache *pf* Konfiguration erstellt:

```
### Makros und Setup (OPTIONAL, aber schöner)
#die interne Netzschnittstelle (hier Intel NIC)
int_if = "fxp0"
#die externe Netzschnittstelle
ext_if = "fxp1"
#das lokale Netzwerk abgeleitet von int_if
lan_netz = $int_if:network

### NAT Regeln
# das NAT von lokal auf extern
# die Klammerung, weil ext_if dynamisch wechselnde IP bekommt
nat on $ext_if from $lan_netz to any -> ($ext_if)

### Fitlerregeln
# grundsätzlich alle Pakete verwerfen
block all
# loopback und lan_netz dürfen alles, aber mit states (default)
pass from {lo0, $lan_netz } to any
```

5.2. pf als sicherer WLAN-Accesspoint

Drahtlosnetzwerke sind heutzutage insbesondere bei mobilen Computersystemen Standard geworden. Zur sicheren Kommunikation wurden Protokolle wie WEP und WAP geschaffen. Die Sicherheitsmechanismen dieser Protokolle wird in Fachkreisen aber immer wieder kritisiert. Insbesondere das WEP-Protokoll wurde kurze Zeit nach Verbreitung ausgehebelt. Selbst beim aktuellen WPA2 Protoko wurden unlängst bewiesen, dass es verwundbar ist[1]. Während die eigentliche Datenübertragung bei bestehender Verbindung relativ sicher ist, liegen die größten Schwachstellen in Fehlern beim Entwurf der Authorisierung der Benutzer. OpenBSD hilft genau bei diesen Schwachstellen.

Lage: Ein Drahtlosnetzwerk soll das vorhandene Netzwerk sicher erweitern. Eine Firewall auf OpenBSD Basis ist bereits in Betrieb und die externe Anbindung ist das Internet über einen Provider. Interne und externe IP-Adressenvergabe ist dynamisch.

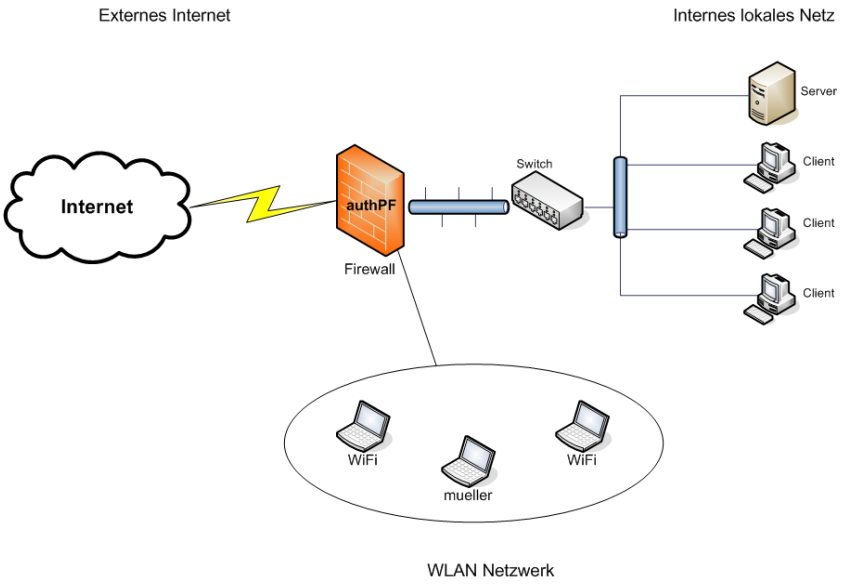

Abbildung 5.1.: Netzwerk mit WLAN Erweiterung

[1] vgl. [UK]

Ziel: Um dem Drahtlosnetzwerk beizutreten, müssen sich die Benutzer sicher authentifizieren an der Firewall. Bei valider Authentifizierung wird **benutzerbezogen** die Verwendung des restlichen Netzwerks ermöglicht. Damit kann man die Möglichkeiten der Zugriffe vom Drahtlosnetzwerke einzeln festlegen. Zudem soll jeder Authentifikation protokolliert werden.

Durchführung: Hardwareseitig wird die bestehende Firewall mit einer zusätzlichen Drahtlosnetzwerkschnittstelle erweitert. Zur Authentifizierung wird *authpf* eingesetzt im Verbund mit den vorhandenen Firewallregeln. Die Regeln werden erweitert, dass grundsätzlich jeglicher Zugriff aus dem Drahtlosnetzwerk, ohne vorherige Authentifizierung über das sichere SSH Protokoll, erst einmal mit Hinweis über eine Webseite abgelehnt wird .

authpf ist technisch eine shell, die aber lediglich zu AAA-Zwecken in Verbindung mit *pf* benutzt wird. Die Zusammenarbeit erfolgt dabei durch *anchors*. Neben einer allgemein gültigen Zugriffsregeldatei wird für jeden Benutzer, der Sonderzugriffsrechte besitzen soll ein eigener Regelsatz angelegt. Mit dem Systembefehl *adduser* werden die Benutzer auf der Firewall angelegt. Hierbei ist lediglich darauf zu achten, dass es Shell und *login class* der Wert *authpf* ausgeähnlt wird.

Damit *authpf* erfolgreich gestartet werden kann, müssen einige Konfigurationsdateien angelegt werden.

/etc/authpf/authpf.conf hier werden bei Bedarf die beiden Variablen **anchor** = *name* und **table** = *name* gesetzt. Standardmäßig lautet der Ankername *authpf* und die Tabelle *authpf_users*.

/etc/authpf/authpf.rules die Regeldatei, die den Regelsatz für den allgemeine Zugriff beinhaltet

/etc/authpf/users/BENUTZER/authpf.rules die benutzerspezifische Regeldatei

/etc/authpf/authpf.allow hier kommen alle Benutzernamen rein, die überhaupt berechtigt sind, dass *authpf* System zu benutzen und somit indirekt die Firewall

umzukonfigurieren

/etc/authpf/banned hier wird pro Benutzer eine leere Datei mit seiner Benutzerkennung angelegt, die besagt, dass der Benutzer aus dem System ausgeschlossen ist. Optional kann in der Datei der Grund des Auschlusses definiert werden, die dem jeweiligen Benutzer dann bei einem Versuch angezeigt wird.

Als nächstes wird eine einfache HTML-Seite für den Webserver angelegt, auf die alle nicht authentifizierten Benutzer umgeleitet werden und weitere Hinweise gegeben werden. Diese kopiert man dann nach */var/www/htdocs* in das Wurzelverzeichnis des Apache[2] Webservers. Zum Starten des Webservers modifiziert man in */etc/rc.conf.local* die Variable **httpd_flags** analog wie beim DHCP-Server weiter oben.

Die Konfiguration für alle Benutzer in **/etc/authpf/authpf.rules**:

```
### WLAN Netz config
#
# erlaubte Dienste vom WLAN ins Internet für alle
wifi_if = "ral0"
lan_netz = fxp0:network
wifi_out = "{ domain, auth, http, https }"
udp_out = "{ domain, ntp }"

# Zugang zum LAN wird abgelehnt
block in quick on $wifi_if proto { tcp, udp } from $user_ip to $lan_netz

# alles ins Internet erlaubt
pass in quick on $wifi_if proto { tcp, udp } from $user_ip to any port $udp_out
pass in quick on $wifi_if proto tcp from $user_ip to any port $wifi_out
```

Die Konfiguration für einen technischen Mitarbeiter mit Benutzerkennung „mueller“ in /etc/authpf/users/mueller, der zusätzlichen Zugriff auf das lokale Netzwerk bekommt:

```
### WLAN Netz config
#
```

[2] OpenBSD verwendet eine stark modifizierte Variante des Apache 1.3 Servers - getrimmt auf Sicherheit

```
# mehr erlaubte Dienste vom WLAN ausgehend für Mueller
wifi_out = "{ ssh, nntp, irc, domain, auth, http, https }"
udp_out = "{ domain, ntp }"

# alle Verbindungen auch ins LAN erlaubt
pass in quick on $wifi_if proto { tcp, udp } from $user_ip to any port $udp_out
pass in quick on $wifi_if proto tcp from $user_ip to any port $wifi_out
```

Die spezielle Variable *$user_ip* ist eine fest eingebautes Makro und beinhaltet immer die Quell-IP des Benutzers. Es gibt auch noch *$user_id*. Empfohlen wird aber die Tabelle *authpf_users* zu verwenden.

Zum Abschluss wird die vorhandene Hauptkonfigurationsdatei für den Betrieb mit dem neuen Drahtlosnetzwerk und *authpf* erweitert. Eine fertige *pf.conf* würde folgendermaßen aussehen:

```
### Makros und Setup (OPTIONAL, aber schöner)
#die interne Netzschnittstelle (hier Intel NIC)
int_if = "fxp0"
#die externe Netzschnittstelle
ext_if = "fxp1"
#die neue WLAN Schnittstelle (hier Ralink WLAN)
wifi_if = "ral0"

# der Webserver mit der Fehlerseite
auth_web = "localhost"

# Makro für DHCP
dhcp_dienste = "{ bootps, bootpc }"

# Tabelle der IP-Adressen der WLAN Benutzer
table <authpf_users> persist

#das lokale Netzwerk abgeleitet von int_if
lan_netz = $int_if:network
wifi_netz = $wifi_if:network
```

```
### Umleitungsregeln
#
rdr pass on $wifi_if proto tcp from ! <authpf_users> to any port http -> $auth_web

### NAT Regeln
# das NAT von lokal auf extern
# die Klammerung, weil ext_if dynamisch wechselnde IP bekommt
nat on $ext_if from $lan_netz to any -> ($ext_if)

# vom WLAN auf extern
nat on $ext_if from $wifi_netz to any -> ($ext_if)

### Filterregeln
#
# Normalisierung ist immer gut
scrub in all

# grundsätzlich alle Pakete verwerfen
block all

#authpf Regeln einbinden
anchor "authpf/*"

# unauthentifizierte WLAN Nutzer nur SSH erlaubt
pass in quick on $wifi_if inet proto { tcp, udp } from $wifi_netz to $wifi_if port ↩
    ssh

# loopback und lan_netz dürfen alles, aber mit states (default)
pass from {lo0, $lan_netz } to any
```

5.3. pf als load-balancer

pf kann hervorragend zum Aufbau eines *load balancers* dienen. Ein *load balancer* ist ein vorgeschalteter Server, der Daten auf einen Pool von Rechnern verteilt. Somit lassen sich Hochleistungssysteme aufbauen, die das Vielfache an Netzanfragen verkraften.Typisches Anwendungsszenario für *load balancer* sind hochfrequentierte Webseiten.

Lage: ein Webportal soll performanter werden. Der bisher eingesetzte Webserver verkraftet die Besucherzahlen nicht mehr.

Ziel: die Anzahl möglicher http-Anfragen für das Webportal soll deutlich gesteigert werden, um die wachsende Besucheranzahl zufrieden zu stellen. Möglichst stabile und kostengünstige Lösung gefordert.

Durchführung: Erweiterung des bisherigen einzigen Webservers zu einem Pool von mehreren Webservern, die durch einen vorgeschalteten *load-balancer* die Anfragen bearbeiten. Der *load-balancer* wird mit einem *pf* konfigurierten OpenBSD System realisiert.

Neben der richtigen Auswahl der verwendeten Hardware (CPU, Netzwerkkarten etc.) würde so eine Basiskonfiguration für die Lastverteilung bei Benutzung von vier Webservern folgendermaßen aussehen:

```
int_if = "fxp0"
ext_if = "fxp1"
www_ip = "88.32.213.1"

webserver = "{192.168.1.10, 192.168.1.11, 192.168.1.12, 192.168.1.123 }"

rdr on $ext_if from any to $www_ip port 80 -> $webserver round-robin
pass in on $ext_if from any to $www_ip port 80
```

Bei der Konfiguration werden im *round-robin* Verfahren die ankommenden Anfragen verteilt auf den vier Webservern. Ist die Webapplikation sitzungsabhängig, sollte man statt round-robin besser die *source-hash* Methode verwenden. Damit ist dann garantiert,

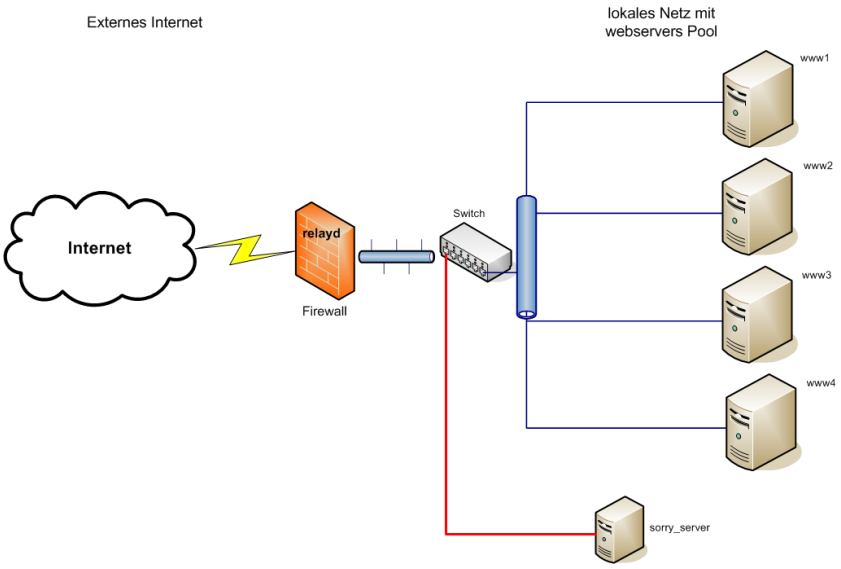

Abbildung 5.2.: Netzwerk mit load balancer

dass der selbe Benutzer während einer Sitzung auch immer auf den selben Webserver geleitet wird.

Einen Nachteil hat das System. Ein Ausfall eines Webservers aus dem Pool wir nicht mit dem *load balancer* kommuniziert und somit kann es zu Anfragen kommen, die mit einem Fehler beantwortet werden, weil der Server nicht mehr zur Verfügung steht.

Abhilfe schafft hier der Einsatz von ***relayd***, einem eigenen Dienst, der ursprünglich unter dem Namen *hostated* als Erweiterung für *pf* entwickelt wurde. *relayd* ist in der Lage auf den Schichten 3,4 und 7 Daten weiterzuleiten und zu verteilen. Gleichzeitig aber auch die Verfügbarkeit der Zielrechner zu überprüfen und bei Ausfall aus dem Pool zu entfernen. Somit ist *relayd* ein *load-balancer*, ein *application-level-gateway* und ein *transparenter proxy*.

Um *relayd* einzusetzen müssen wir unseren Regelsatz umdefinieren

```
int_if = "fxp0"
ext_if = "fxp1"
```

```
www_ip = "88.32.213.1"
webserver = "{ 192.168.1.10, 192.168.1.11, 192.168.1.12, 192.168.1.13 }"

# Anker für relayd
rdr-anchor "relayd/*"
anchor "relayd/*"

pass in on $ext_if from any to $www_ip port 80
```

Man entfernt also die Weiterleitungen in der Hauptkonfiguration. Diese Aufgabe übernimmt *relayd*, den man in der Datei */etc/relayd.conf* einstellt. Um die gleiche Funktionalität der Lastverteilung, aber mit den Funtkionen von *relayd* wie oben zu erreichen würde die Konfiguration so aussehen:

```
## Makro Definitionen
web1 = "192.168.1.10"
web2 = "192.168.1.11"
web3 = "192.168.1.12"
web4 = "192.168.1.13"
# offizielle externe IP
relayd_addr = "88.32.213.1"
relayd_port = "8080"
webservers_port = "80"
# derjenige Server, der als Ziel dient, wenn ALLE Poolserver off sind
sorry_server = "192.168.1.100"

# alle 5 sekunden Test auf Erreichbarkeit der Poolserver
interval 5
# globaler check timeout
timeouts 300
# protokolliere Statusnachrichten
log updates
# Definiton des pools und der Testseite, die aufgerufen werden soll
table <webservers> { $web1, $web2, $web3, $web4 }
# Definiton der Ausweichseite bei Totalausfall des Pools
table <sorry> disable { $sorry_server }
```

```
# eigentliche Weiterleitung
redirect "www" {
        listen on $relayd_addr port $relayd_port
# Markierung aller Pakete mit einem tag
tag REDIRECTED
#Weiterleitung an den Pool
forward to <webservers> port $webservers_port mode loadbalance check http "/test. ↩
    html" code 200
# bei Ausfall alles an den Ausfallserver
forward to <sorry> port $webservers_port check icmp
}
```

Ein solche Konfiguration stellt dann schon eine Art mini-Cluster dar und ist horizontal beliebig skalierbar. Bei Bedarf können einfach mehr Webserver installiert werden, um noch mehr Anfragen zu bewältigen.

6. Hochverfügbarkeitsyssteme

6.1. Allgemeines

Ausfälle in heutigen Netzwerken sind nicht nur gravierend für die Produktivität des eigenen Betriebs, sondern verursachen als Konsequenz meist immense Kosten. Bei einem Ausfall der Kommunikationstechniken wie beispielsweise E-Mail ist selbst ein kleines Unternehmen nicht mehr in der Lage sein Tagesgeschäft abzuwickeln. Ein Ausfall des gesamten Netzwerks ist gleichzusetzen mit einem Totalausfall der Firma.

Ein Firewallsystem spielt für die Verfügbarkeit des lokalen Netzwerks eine bedeutende Rolle. Schließlich bildet das System in den meisten Fällen den Ausgang aus dem Intranet, wodurch erst die Dienste des Internets genutzt werden können. Daher sollte man so gut wie technisch nur möglich, diese Komponente des Gesamtnetzwerkes gegen Ausfall absichern.

Hochverfügbarkeit beschreibt die Fähigkeit eines Systems, bei Ausfall den Betrieb weiterhin sicherzustellen. Dies kann durch verschiedene Maßnahmen zumindest teilweise gewährleistet werden, denn eine 100-prozentige Verfügbarkeit liegt im Bereich des Unmachbaren. Zu viele technische Faktoren spielen dabei eine Rolle. Eine allgemeine Liste der möglichen Eigenschaften eines Hochverfügbarkeitssystems wäre:

Qualität Die Qualität eines Systems kann sehr gut als Eigenschaft von Hochverfügbarkeit genommen werden. Sei es die Qualität der verwendeten Hardware und Software bis hin zur Qualität der eigenen Dokumentationen.

Sicherheit sichere Umgebungen erhöhen die Verfügbarkeit. Ein unsicheres System kann innerhalb von Minuten kompromittiert werden und im schlimmsten Fall auch lahmgelegt werdern.

Stabilität darunter versteht man in diesem Zusammenhang die Zeit, die ein System ohne größeren Eingriff unter allen vorstellbaren Lastbedingungen durchläuft. Zuverlässigkeit wäre ein anderer Ausdruck hierfür.

Redundanz hilft die Verfügbarkeitsrate von kritischen Systemen zu erhöhen, in dem identische Systeme als Reserve gehalten werden.

Eskalationsplan ebenso wichtig, wie die hohe Verfügbarkeit von Erstazteilen oder gar Ersatzsystemen ist ein Eskalationsplan, der im Falle eines Ausfalls möglichst genau definiert, was zu tun ist.

6.2. Hochverfügbarkeit mit OpenBSD

OpenBSD als Firewallsytem kann durch seine Eigenschaften und Software-Ausstattung sehr gut als Mittel für ein Hochverfügbarkeitsnetzwerk dienen. Das System mit seiner Unix-Basis gilt allgemein als sehr stabil. Der BSD-Netzwerkstack ist einer der ältesten und zuverlässigsten Netzwerkimplementierungen in einem Betriebssystem und wurde deshalb auch oft als Grundlage für neue Systeme benutzt. Auch die restlichen Kernkomponenten des Systems wie beispielsweise das Dateisystem ist seit Jahrzehnten bewährt und gilt als nahezu fehlerfrei. Unter anderem durch die komplett einsehbaren Quelltexte ist Sicherheit eines der herausragendsten Merkmale von OpenBSD und die Einstellung der Entwickler zu freier Software betonen dies.

Die Hauptkomponenten zum Einsatz eines hochverfügbaren Firewallsystems mit OpenBSD sind das *Common Redundancy Resoultion Protocol (CARP)* in Verbindung mit *pfsync*, einem Modul von OpenBSDs Paketfilter *pf*.

Es folgt eine theoretische Beschreibung der beiden Protokolle. Der praktische Einsatz und die Konfiguration wird in Kapitel 7 behandelt.

6.2.1. CARP

CARP wurde entwickelt als Ersatz für das in der freien Softwareszene verbreitete *Virtual Router Redundancy Protocol (VRRP)*. VRRP wurde von Ascend Communications, DEC, IBM, Microsoft und Nokia 1998 entwickelt, beinhaltet aber Patente von Cisco Systems. Das ist auch der Hauptgrund, wieso einerseits das Protokoll nach wie wor von der IETF nicht als Standard zertifiziert wurde, und andererseits die OpenBSD Entwickler CARP entwickelten.

CARP im Unterschied zu VRRP:

- echte freie Software ohne Patente
- sicherer, da die Kommunikation zwischen den Mitgliedern verschlüsselt wird
- beinhaltet *arpbalance*, eine Technik, die es mehreren Rechnern erlaubt dieselbe IP-Adresse zu benutzen
- arbeitet protokollunabhängig, somit auch für IPv6 nutzbar
- kann auch mit *load balancern* eingesetzt werden

CARP wurde entwickelt, um die Hochverfügbarkeit des Standardgateways in einem Netzwerk durch Redundanz zu gewährleisten. Technisch ist es ein Multicast-Protokoll, dass redundantes dynamisches Routing für Datenpakete auf dem Weg zum Gateway einführt. Realisiert wird dies, indem mehrere Router (gateways) zu einer logischen Gruppe zusammengefasst werden und im Netzwerk als ein Gerät ausgegeben werden. Dafür wird dem logisch gebildeten Router eine virtuelle MAC-Adresse und eine virtuelle IP-Adresse zugewiesen. Einer der Router innerhalb der Gruppe wird als sogenannter *Master* und die restlichen als *Slaves* definiert. Der *Master* bindet dann seine Netzwerkschnittstelle an die virtuelle MAC- und IP-Adresse und informiert gleichzeitig die anderen *Slaves*, dass

er jetzt die Führung übernimmt. *Slaves* bilden dann die potenziellen Ersatz-Router, die bei Ausfall des *Masters* dessen Rolle übernehmen können. Die Integrität und Prüfung unter den CARP-Rechnern erfolgt verschlüsselt durch sogenannte *advertisements*. In reglemäßigen Intervallen werden diese mit Hilfe des eigenen CARP-Protokolls (IP Protokoll 112)vom *Master* verschickt und die *Slaves* warten auf dieses Zeichen. Wenn dieses Zeichen nicht mehr empfangen wird, dann beginnen die *Slaves* mit dem Aussenden der *advertisements*. Ein simpler Algorithmus gewährt dabei, dass wenn mehrere Rechner gleichzeitig die Zeichen versenden, dennoch nur einer *Master* wird[1]. Die Frequenz, wie oft ein Zeichen versendet wird, kann bei der Konfiguration beeinflusst werden.

CARP ist nach mehrjähriger Entwicklung flexibler einsetzbar, wie ursprünglich geplant (gateway-failover). So existieren Installationen, bei denen CARP generell für alles eingesetzt wird, was kritisch ist. Gleich ob DNS-, Mail- oder Webserversysteme[2] in kleinen oder großen Netzwerken.

6.2.2. pfsync

pfsync ist ein einfaches Multicast Protokoll (IP Protokoll 240), das dazu dient, die von *pf* generierten Verbindungszustände (*states*) mit einem anderen Firewallsystem zu synchronisieren. Dies erlaubt in Verbindung mit CARP den Aufbau von hochverfügbaren Firewallsystemen. Technisch wird das durch eine dedizierte Netzwerkschnittstelle realisiert, die das Protokoll in Verbindung mit anderen Firewallsystemen einsetzt. Über das Protokoll werden Nachrichten übertragen, die zur Synchronisierung dienen. Bei diesen Nachrichten handelt es sich um Einträge (*insertions*), Aktualisierungen (*updates*) und Löschungen (*deletions*) von *states*. Parallel zum Senden dieser Nachrichten wird über das Protokoll auch an der Netzwerkschnittstelle gelauscht, ob andere Firewallsystem etwas senden.

[1] vgl. [McB]
[2] vgl. [Bec]

Die Kombination von CARP und pfsync erlaubt es, hochverfügbare Firewallsysteme einzurichten, die bei Ausfall eines Knoten transparent weiterarbeiten.

7. HA Firewallsystem mit ALIX Plattform

In diesem Kapitel wird ein funktionales HA-Firewallsystem mit Hilfe von OpenBSD und der Embedded-PC Plattform ALIX vom schweizer Hersteller PCEngines aufgebaut. Das System soll primär zur Demonstration der Hochverfügbarkeitsfunktionen von *pf* unter Laborbedingungen dienen, aber auch ein Einsatz in der Praxis ist möglich.

7.1. die ALIX Plattform

Die schweizer PCEngines GmbH entwickelt und vertreibt seit Jahren embedded PC Platinen, die international durch ihre Funktionalität und aufgrund ihres exzellentem Preis-Leistungs-Verhältnisses sehr beliebt sind. Gerade im Sicherheitsbereich, der Heimat von Firewallsystemen, reichen diese kleinen Rechnersysteme oft aus, um funktionale Systeme zu entwickeln und auch später einzusetzen. Die Systeme laufen völlig geräuschlos, da durch Einsatz von stromsparenden Komponenten keine aktive Kühlung notwendig ist. Eine maximale Leistungsaufnahme von unter 10Watt und mehrere Netzwerkschnittstellen sind ideale Voraussetzungen, mehrere solcher Platinen für ein HA-System zu verwenden.

Das für die Experimente eingesetzte System besteht aus 2 Platinen vom Typ ALIX 2D13. Sie besitzen im Wesentlichen[1] folgende Ausstattung:

[1] volle Ausstattung siehe http://www.pcengines.ch/alix2d13.htm besucht am 17.08.2009

- CPU AMD Geode LX800 mit 500MHz Takfrequenz
- 256 MB DDR SDRAM
- 3 FastEthernet Netzwerkschnittstellen (VIA)
- CompactFlash Sockel
- miniPCI Erweiterungsslot, USB, IDE, RS-232

Abbildung 7.1.: ALIX 2D13 Innenansicht

Die beiden Platinen werden zum optimalen Betrieb in herstellereigenen Gehäusen verwendet. Als Datenträger empfiehlt es sich Produktivbetrieb unbedingt auf CompactFlash Karten zu achten, die den Titel *„industrial grade“* tragen. Diese CF-Karten arbeiten mit höherwertigeren Flashspeicherbausteinen im Vergleich zu gewöhnlichen CF-Karten, belohnen aber dafür mit längerer Lebenszeit und vor allem höheren Schreibzy-

klen. Nachteilig ist allerdings, der deutlich höhere Preis. Gerade wegen den niedrigen Schreibzyklen versagt eine gewöhnliche CF-Karte innerhalb von Wochen im Dauerbetrieb. Eine Alternative zu den *industrial grade* Karten sind Minifestplatten, auch *microdrives* genannt. Diese Art von Datenträger wurde einst erfunden, als die CF-Karten noch um ein vielfaches teurer waren und keine großen Kapazitäten vorweisen konnten. Dank dem Auslaufstatus der *microdrives* kann man diese heutzutage für wenig Geld erwerben, doch Vorsicht, nicht jedes *microdrive* lässt sich als Festplatte betreiben. Der Grund hierfür liegt darin, dass sie ursprünglich nur für den OEM-Markt etnwickelt wurden zur Integration in Multimediageräten wie MP3-Abspieler etc. und deshalb oft durch ihre eigene Steuersoftware (firmware) auf bestimmte Modelle dieser Gerätereihen beschränkt sind. Eine Nutzung als allgemeiner Datenträger kann daher nicht garantiert werden, was oft schon daran scheitert, dass sie nicht mal vom CF-Sockel erkannt werden.

Für die Experimente werden *microdrives* vom Hersteller Seagate, Typ ST-1 mit einer Kapazität von 2,5Gb eingesetzt.

Abbildung 7.2.: zwei ALIX 2D13 in ihren Gehäusen

7.2. Installation OpenBSD auf ALIX

Da die ALIX-Plattform eine sogenannte *headless platform*, dass heisst es steht kein Bildschirmanschluß zur Verfügung. Das ist bei embedded Plattformen üblich, da Grafikchipsätze verhältnismäßig zu viel Strom verbrauchen würden. Eine Systemausgabe erhält man über den integrierten RS-232 seriellen Anschluss mit Hilfe eines Nullmodemkabels. Folglich gibt es zwei Möglichkeiten das Betriebssystem zu installieren:

- Installation an einem Desktop-PC mit Hilfe eines CF-Kartenleser
- Installation über das Netzwerk, da die Geräte *PXE* bootfähig sind

Die Vorbereitung für eine Installation über das Netzwerk wird hier schrittweise beschrieben:

1. Gerät mit Nullmodemkabel mit stationärem Rechner
2. Vorbereitung des Stationären Rechners: In einer virtuellen Umgebung wird ein Basissystem von OpenBSD gemäß der Installationsanleitung[2] installiert. In dieser Umgebung wird ein DHCP-Server vorbereitet mit folgender Konfiguration in */etc/dhcpd.conf*

```
shared-network ALIX {
            option  domain-name "alix.local";
            # ARCOR DSL DNS Server
            option  domain-name-servers 195.50.140.252, 195.50.140.114;

            subnet 192.168.111.0 netmask 255.255.255.0 {
                    option routers 192.168.111.254;
                    filename "pxeboot";
                    range 192.168.111.55 192.168.111.58;
                    default-lease-time 86400;
                    max-lease-time 90000;
            }
}
```

[2] vgl. [Proa]

Zur Aktivierung des tftp-Servers, der die Startdateien im Netzwerk bereitstellt, wird in der Datei */etc/inetd.conf* das Kommentarzeichen vor dem tftp Eintrag entfernt. Es wird ein leeres Verzeichnis */tftpboot* angelegt und die benötigten Dateien für den Systemstart (*pxeboot, bsd.rd, /etc/boot/conf*) werden von der Installations-CD dorthin kopiert. Um die serielle Bildschirmausgabe zu erhalten, wird in der *boot.conf* Datei folgender Eintrag gemacht:

```
# COM1 mit 38400 Baud
stty com0 38400
set tty com0
```

Die Werte sollten analog zu den Werten des *ALIX tinybios* gesetzt werden, sonst erkennt man später auf der Ausgabe nichts.

3. mit Hilfe eines Terminalprogramms und eingeschaltetem ALIX System noch einmal die Einstellungen des *tinyBios* der Plattform kontrollieren und gegebenenfalls ändern.

4. auf der OpenBSD Maschine den dhcpd server mit

```
dhcpd &
```

starten und dann im Terminalprogramm verfolgen, wie der Installationskernel startet und die reguläre Installation beginnt.

Auf der Begleit-CD ist ein Videotutorial für die eigentliche Installation enthalten.

7.3. HA System mit CARP und pfsync

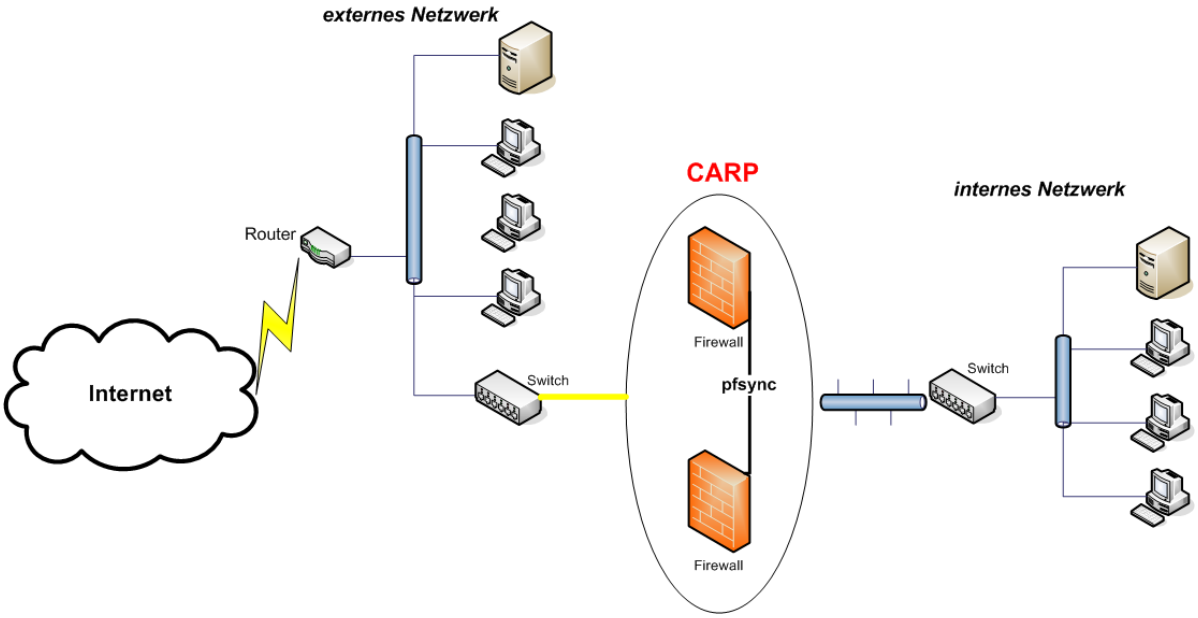

Abbildung 7.3.: ALIX Firewallsystem mit CARP und pfsync

Zwei ALIX Knoten werden als Firewallsystem im CARP Verbund betrieben. Dazu wurde beide Rechner identisch installiert mit Ausnahme der Netzwerkkonfiguration. Jeder Rechner besitzt drei Netzwerkschnittstellen (LAN 1-3), was für den Einsatz als redundantes Firewallsystem prädestiniert ist. Das interne Netzwerk befindet sich hinter einem externen Netzwerk und besitzt keinen direkten Internetzugang. LAN1 (vr0) ist mit dem

Schnitstelle	MASTER	SLAVE
vr0	192.168.111.40	192.168.111.50
vr1	192.168.100.40	192.168.100.50
vr2	192.168.200.40	192.168.200.50
carp0	192.168.111.88	
carp1	192.168.100.88	
pfsync0	up sync if vr2	

Tabelle 7.1.: CARP Konfiguration

externen Netzwerk verbunden. LAN2 (vr1) ist die Verbindung zum internen Netzwerk. LAN3 (vr2) ist das dedizierte Netzwerk für den *pfsync* Betrieb.

Als Vorarbeit für den Betrieb des Systems werden folgende Kernelparameter in */etc/-sysctl.conf* aktiviert:

```
net.inet.ip.forwarding=1
net.inet.carp.preempt=1
```

Die Konfiguration der Netzwerkschnittstellen erfolgt je in einer Datei ***/etc/hostname.schnittstellename***. Für LAN1 beispielsweise:

```
# /etc/hostname.vr0
inet 192.168.111.40 255.255.255.0 NONE
```

Für LAN2 und LAN3 analog. Die CARP-Schnittstellen werden um CARP Parameter ergänzt und zwar unterschiedlich auf dem MASTER und SLAVE Knoten.

```
# /etc/hostname.carp0
# externe CARP Gruppe 1 mit Passwort fhacarp
inet 192.168.111.88 255.255.255.0 192.168.111.255 vhid 1 pass fhacarp


# /etc/hostname.carp1
# interne CARP Gruppe 2 mit Passwort carpfha
inet 192.168.100.88 255.255.255.0 192.168.100.255 vhid 2 pass carpfha
```

Der Unterschied zwischen interner und externer Gruppe liegt an der Position im Gesamtnetzwerk. Intern ist die virtuelle IP-Adresse des Gateways für das interne Netzwerk. Extern bildet die IP-Adresse, unter der man den Rechnerverbund ansprechen kann.

CARP Konfiguration für den SLAVE Knoten:

```
# /etc/hostname.carp0
# externe CARP Gruppe 1 mit Passwort fhacarp
inet 192.168.111.88 255.255.255.0 192.168.111.255 vhid 1 advskew 100 pass fhacarp
```

```
# /etc/hostname.carp1
# interne CARP Gruppe 2 mit Passwort carpfha
inet 192.168.100.88 255.255.255.0 192.168.100.255 vhid 2 advskew 100 pass carpfha
```

Hier sind zusätzlich die *adskew* Parameter gesetzt, die das Sendeintervallverhalten der *advertisings* bestimmen. Je niedriger der Wert, desto öfter wird gesendet. Um sicher zu gehen, dass der MASTER immer wieder MASTER wird, selbst nach einem Ausfall, wird daher der Wert hier erhöht.

Zum Abschluß wird eine einfache *pf* Konfiguration auf beiden Knoten eingerichtet und *pf* mit **pf=YES** in */etc/rc.local.conf* dauerhaft aktiviert:

```
int_if="vr1"
ext_if="vr0"

localnet=$int_if:network

tcp_services = "{ ssh, http }"
udp_services = "{ domain }"

nat on $ext_if from $localnet to any -> $ext_if

block all
# Pflichtregeln für CARP und pfsync
pass quick on { vr2 } proto pfsync keep state (no-sync)
pass on { vr0 vr1 } proto carp keep state

pass on { vr0 vr1 } proto icmp
pass on { vr0 vr1 } proto tcp from any to any port $tcp_services
pass on { vr0 vr1 } proto udp from any to any port $udp_services

# Spezialregel für traceroute
pass on { vr0 vr1 } inet proto udp from any to any port 33433 >< 33626
```

Nach einem Neustart wird überprüft, ob die Schnittstellen alle angelegt wurden. Dazu

reicht es **ifconfig** einzugeben. Wichtig sind hier vor allem die CARP Schnittstellen. Auf dem MASTER muss es so aussehen:

```
carp0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:00:5e:00:01:01
        priority: 0
        carp: MASTER carpdev vr0 vhid 1 advbase 1 advskew 0
        groups: carp
        inet 192.168.111.88 netmask 0xffffff00 broadcast 192.168.111.255
        inet6 fe80::200:5eff:fe00:101%carp0 prefixlen 64 scopeid 0x6
carp1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:00:5e:00:01:02
        priority: 0
        carp: MASTER carpdev vr1 vhid 2 advbase 1 advskew 0
        groups: carp
        inet 192.168.100.88 netmask 0xffffff00 broadcast 192.168.100.255
        inet6 fe80::200:5eff:fe00:102%carp1 prefixlen 64 scopeid 0x7
```

Am Schlüsselwort „MASTER“ erkennt man, dass CARP aktiv ist und der Rechner Masterstatus angenommen hat. Die Gegenkontrolle auf dem SLAVE muss dann entsprechend so aussehen:

```
carp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:00:5e:00:01:01
        priority: 0
        carp: BACKUP carpdev vr0 vhid 1 advbase 1 advskew 100
        groups: carp
        inet 192.168.111.88 netmask 0xffffff00 broadcast 192.168.111.255
        inet6 fe80::200:5eff:fe00:101%carp0 prefixlen 64 scopeid 0x6
carp1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:00:5e:00:01:02
        priority: 0
        carp: BACKUP carpdev vr1 vhid 2 advbase 1 advskew 100
        groups: carp
        inet 192.168.100.88 netmask 0xffffff00 broadcast 192.168.100.255
        inet6 fe80::200:5eff:fe00:102%carp1 prefixlen 64 scopeid 0x7
```

Hier steht anstelle von „MASTER“ das Schlüsselwort „BACKUP“, was bestätigt, dass der SLAVE Rechner in Bereitschaft steht.

Ein erster Test aus dem externen Netz kann durch das *ping* Werkzeug erfolgen, dass auf jedem Betriebssystem vorhanden ist, um Netzverkverbindungen zu testen.

```
# ping auf die externe CARP_vIP von einem WinXP-Client

C:\>ping 192.168.111.88

Ping wird ausgeführt für 192.168.111.88 mit 32 Bytes Daten:

Antwort von 192.168.111.88: Bytes=32 Zeit<1ms TTL=255
Antwort von 192.168.111.88: Bytes=32 Zeit<1ms TTL=255
.
.
```

Das Gleiche sollte auch vom internen Netz auf die interne virtuelle IP-Adresse 192.168.100.88 erfolgreich verlaufen.

Ob pfsync korrekt läuft, kann man nach einem Vergleich der state-Tabellen von MASTER und SLAVE erkennen.

```
root@master:~# pfctl -s states
all pfsync 192.168.200.40 -> 224.0.0.240       SINGLE:NO_TRAFFIC
all pfsync 224.0.0.240 <- 192.168.200.50       NO_TRAFFIC:SINGLE
all carp 192.168.100.40 -> 224.0.0.18       SINGLE:NO_TRAFFIC
all carp 224.0.0.18 <- 192.168.100.40       NO_TRAFFIC:SINGLE
all tcp 192.168.111.40:22 <- 192.168.111.3:3057       ESTABLISHED:ESTABLISHED
all tcp 192.168.111.50:22 <- 192.168.111.3:3112       ESTABLISHED:ESTABLISHED
all carp 192.168.111.40 -> 224.0.0.18       SINGLE:NO_TRAFFIC
all carp 224.0.0.18 <- 192.168.111.40       NO_TRAFFIC:SINGLE
.
.
.
```

Die Ausgabe vom SLAVE Server:

```
root@slave:~# pfctl -s states
all pfsync 192.168.200.50 -> 224.0.0.240       SINGLE:NO_TRAFFIC
all pfsync 224.0.0.240 <- 192.168.200.40       NO_TRAFFIC:SINGLE
all carp 224.0.0.18 <- 192.168.100.40       NO_TRAFFIC:SINGLE
all carp 192.168.100.40 -> 224.0.0.18       SINGLE:NO_TRAFFIC
all tcp 192.168.111.40:22 <- 192.168.111.3:3057       ESTABLISHED:ESTABLISHED
all tcp 192.168.111.50:22 <- 192.168.111.3:3112       ESTABLISHED:ESTABLISHED
all carp 224.0.0.18 <- 192.168.111.40       NO_TRAFFIC:SINGLE
all carp 192.168.111.40 -> 224.0.0.18       SINGLE:NO_TRAFFIC
.
.
.
```

Man sieht auf beiden ALIX Knoten exakt die selben state-Einträge. Damit *pfsync* arbeitet also fehlerfrei.

Zum Abschluß der *failover Test*. Dazu aus dem externen Netz wieder mit Hilfe von *ping* die externe virtuelle IP anklopfen. Dieses Mal aber mit Zusatzparamter „-t“ für dauerhaftes Anklopfen. Als Ausgabe sollten Antwortpakete erhalten werden wie bereits oben im ersten Test. Jetzt wird ein Ausfall simuliert und der MASTER Rechner vom Strom getrennt. Dies sollte maximal zu einem Paketverlust in der *ping* Ausgabe führen, da der SLAVE die virtuelle IP-Adresse sofort innerhalb von Millisekunden an sich bindet und die Antwortpakete weitersendet. Durch Überprüfung wieder mit *ifconfig* sieht man, dass der SLAVE Rechner auch den Status seiner CARP Schnittstellen auf MASTER gestellt hat. Nach einem Neustart des MASTER Rechners, wird dies wieder in den BACKUP-Status gestellt.

Der Test aus dem internen Netzwerk ist noch interessanter, da man hier durch die eingesetzte *pf* Firewall mit ihrem NAT und Filterregeln deutlicher die Hochverfügbarkeit erkennen kann. Zum Test wird von einem Rechner aus dem internen Netzwerk ein Forum im Internet besucht. Ein Forum deshalb, weil a) durch die http-Verbindung sicherlich *states* generiert wurden und b) weil Forensoftware mit *sessions* arbeiten, die unter an-

derem eine aktive Sitzung anhand der Absender IP-Adresse bestimmen. Auch hier wird während dem Besuch des Forums, der MASTER Rechner heruntergefahren und das Ergebnis ist wie erwartet: Der Anwender im internen Netzwerk bekommt davon nichts mit. Der gegenüberliegende Forumsserver, der die aktive Sitzung aufrecht erhält, hat auch nichts festgestellt und der Anwender bleibt nach wie vor im Forum angemeldet.

Das auf der Begleit-CD beiliegende Video „CARP-failover“ demonstriert in Bildern die oben beschriebenen Tests.

8. Fazit

Diese Diplomarbeit hat deutlich gemacht, dass OpenBSD im Bereich sicherer und funktionaler Firewallsysteme bedenkenlos eingesetzt werden kann. Man kann sehr gut erkennen, dass die Entwickler des Projekts, ihre Entwicklungen für den praktischen Einsatz programmiert haben und sich in der Materie auskennen. Viele der gebotenen Funktionen sind natürlich auch bei den kommerziellen Herstellern von Firewallsystemen zu finden, jedoch sicherlich nicht kostenfrei und so flexibel in der Anwendung wie bei OpenBSD. Ergänzt wird das ganze noch durch die vollständige Verfügbarkeit der Quelltexte. So lassen sich bei Bedarf, und vor allem durch die freie BSD-Lizenz, eigene Anpassungen vornehmen oder Sicherheitslöcher stopfen.

Im zweiten Teil der Arbeit ging es um Hochverfügbarkeitssysteme. Diesem Aspekt wird leider noch nicht genug Achtung in der Praxis geschenkt. Oft kommt es vor, dass an Redundanz erst gedacht wird, wenn bereits ein Ausfall stattgefunden hat. Und sei es nur durch eine nicht vorhandene Datensicherung. Dabei beweist gerade diese Diplomarbeit, dass Sicherheit und Hochverfügbarkeit nicht unbedingt eine reine Frage der Kosten darstellt.

Sehr gut in dieser Arbeit hat sich auch die Verwendung der ALIX-Plattform als Firewallsystem erwiesen. Äußerst stabiler und stromsparender Betrieb stehen neben einer durchaus beachtlichen Leistung. So konnte mit dem 2-Knoten System und dem Netzwerk-Benchmark Werkzeug „iperf“ ein Durchsatz von beachtlichen 85MBit/s gemessen werden. Und das, bei aktiviertem NAT und mehreren Filterregeln. Dies stellt ein Durchsatz dar, der sicherlich im produktiven Einsatz nicht erreicht wird, dennoch aber für die meisten kleinen bis mittelständischen Unternehmen ausreichen würde, da

diese Zielgruppe meistens keine schnellere Außenanbindung besitzt. Im VPN-Einsatz ist dank der in den AMD Geode Prozessoren integrierten Crypto-Beschleuniger ebenfalls eine gute Leistung zu erwarten. Und falls das nicht ausreichen sollte, lassen sich dank der Mini-PCI Schnittstelle noch Erweiterungskarten einbauen.

Zum Abschluss möchte ich nochmals betonen, dass es keine 100-prozentige Sicherheit gibt. Technische, aber vor allem auch menschliche Faktoren sind natur bedingt, und führen zu Fehlern. OpenBSD und freie Software können allgemein die Sicherheit fördern, aber die wachsenden Gefahren nicht immer verhindern. Außerdem ist ein gut eingerichtetes System nur halb so gut, wenn die Wartung nicht im regelmäßigen Abständen durchgeführt wird.

A. Abkürzungsverzeichnis

AAA	Authentication, Authorization and Accounting
BSD	Berkeley Software Distribution, ein Unix-Derivat der Berkeley Universität Kalifornien/USA
BSI	Bundesamt für Sicherheit in der Informationstechnik
CARP	Common Redundancy Resolution Protocol
DHCP	Dynamic Host Configuration Protocol, Netzwerkprotokoll zur automatischen IP-Adressenvergabe
DMZ	Demilitarized Zone, eine eigenständiges Nettzwerk in einer LAN-Architektur
ftp	file transfer protocol, unsicheres Dateiübertragungsprotokoll
HA	High Availability, Hochverfügbarkeit
ICMP	Internet Control Message Protocol
IDS-System	Intrusion Detection System, eine Erweiterung für höhere Firewallsysteme, die eine Einbruchserkennung ermöglicht und auch Maßnahmen zur Abwehr ergreifen kann
IPSec	Internet Protocol secure, ein sicheres Netzwerkprotokoll, dass vorrangig für VPN genutzt wird
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
MSS	maximum segment size
NDA	Non Disclosure Agreement, eine Verschwiegenheitsvereinbarung
nfs	network file system
OpenSSH	Open SecureShell, Standardsoftware zur sicheren Fernsteuerung von Betriebssystemen auf Kommandozeilenebene
policy	eine Regel im Kontext von Sicherheitssystemen
RFC	Request for Comments

TCP	Transmission Control Protocol, ein zustandsbehaftetes Netzwerkprotokoll mit Fehlerkorrektur
TTL	Time to live
UDP	User Datagram Protocol, ein zustandsloses Netzwerkprotokoll
VPN	Virtual Private Network, Vernetzung von Netzwerken über gesicherte Protokolle
VRRP	Virtual Router Protocol
WLAN	Wireless LAN, ein drahtloses Funknetzwerk

B. Abbildungsverzeichnis

2.1. OpenBSD Logo [Proc] . . . 3
2.2. Software quality measurement [Hol] . . . 9
2.3. OpenBSD Benutzeranteil [Gro] . . . 10

3.1. einfache Firewall Topologie . . . 14
3.2. Firewall mit DMZ . . . 15
3.3. Mehrstufiger Aufbau, bestehend aus einer Hintereinanderschaltung von Paketfilter, Application-Level-Gateway und einem weiteren Paketfilter (DMZ) Abb.2 [BSI] . . . 19
3.4. source-NAT [Wika] . . . 21
3.5. destination-NAT [Wika] . . . 21

4.1. Packetfluss von pf
Eigene Darstellung in Anlehnung an [Tab] . . . 25
4.2. einfache CBQ-Hierarchie . . . 28
4.3. erweiterte CBQ-Hierarchie . . . 29
4.4. PRIQ-Hierarchie . . . 30

5.1. Netzwerk mit WLAN Erweiterung . . . 57
5.2. Netzwerk mit load balancer . . . 63

7.1. ALIX 2D13 Innenansicht . . . 72
7.2. zwei ALIX 2D13 in ihren Gehäusen . . . 73
7.3. ALIX Firewallsystem mit CARP und pfsync . . . 76

C. Literaturverzeichnis

[Art03] Artymiak, Jacek: *Building Firewalls with OpenBSD and PF*. devguide.net Jacek Artymiak, 2. Auflage, 2003.

[Bec] Beck, Bob: *pf Presentation foils*. http://www.ualberta.ca/~beck/nycbug06/pf/.

[BSI] BSI: *Sicherheits-Gateway (Firewalls)*. BSI https://www.bsi.bund.de/cln_164/ContentBSI/Themen/sinet/Sicherheitskomponenten/Sicherheitsgateway/konzsichgateway.html besucht am 02.09.2009.

[Dix] Dixon, Jason: *pf haiku*. pf - Mailingliste http://marc.info/?l=openbsd-pf&m=108507584013046&w=2 besucht am 11.08.2009.

[Ell94] Ellermann, Uwe: *Firewalls: Isolations- und Audittechniken zum Schutz von lokalen Computernetzen*. DFN-Bericht Nr.76, 1994.

[Floa] Floyd, Sally: *CBQ (Class-Based Queuing)*. http://www.icir.org/floyd/cbq.html besucht am 22.08.2009.

[Flob] Floyd, Sally: *RED (Random Early Detection) Queue Management*. http://www.icir.org/floyd/red.html.

[GeN] GeNUA: *GeNUGate Firewall*. http://www.genua.de/produkte/firewall/genugate/index.html besucht am 02.09.2009.

[Gro] Group, BSD Certification: *BSD Usage Survey*. BSD Usage Survey http://www.bsdcertification.org/downloads/pr_20051031_usage_survey_en_en.pdf.

[Han08] Hansteen, Peter N.M.: *The Book of PF*. No Starch Press, 2008.

[Hol] Holwerda, Thom: *The only valid measurement of software quality*. comic: http://www.osnews.com/story/19266/WTFs_m besucht am 15:08.2009.

[HPK] Handley, Mark, Vern Paxson und Christian Kreibich: *Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics*. http://www.icir.org/vern/papers/norm-usenix-sec-01-html/index.html besucht am 28.08.2009.

[IET81] IETF: *Internet Protocol*. RFC 791 http://www.ietf.org/rfc/rfc791.txt besucht am 02.09.2009, September 1981.

[IET94] IETF: *The IP Network Address Translator (NAT)*. RFC 1631 http://www.ietf.org/rfc/rfc1631.txt besucht am 02.09.2009, Mai 1994.

[IET96] IETF: *Address Allocation for Private Internets*. RFC 1918 http://www.ietf.org/rfc/rfc1918.txt besucht am 02.09.2009, Februar 1996.

[IET98] IETF: *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460 http://www.ietf.org/rfc/rfc2460.txt besucht am 02.09.2009, Dezember 1998.

[IET01a] IETF: *The Addition of Explicit Congestion Notification to IP*. RFC 3168 http://www.ietf.org/rfc/rfc3168.txt, September 2001.

[IET01b] IETF: *Protocol Complications with the IP Network Add Translator*. RFC 3027 http://tools.ietf.org/html/rfc3027, Januar 2001.

[KHP05] Korff, Yanek, Paco Hope und Bruce Potter: *Mastering FreeBSD and OpenBSD Security*. O´Reilly Media, 2005.

[McB] McBride, Ryan: *Interview: Ryan McBride*. CARP functionality: http://kerneltrap.org/node/2873.

[Ogl00] Ogletree, Terry William: *Practical Firewalls*. QUE Corporation, 2000.

[Opea] OpenBSD, Team: *Cryptography in OpenBSD*. OpenBSD Website: http://www.openbsd.org/crypto.html besucht am 25.08.2009.

[Opeb] OpenBSD, Team: *OpenBSD Security*. OpenBSD Website: http://www.openbsd.org/security.html besucht am 27.08.2009.

[Opp97] Oppliger, Rolf: *Internet and Intranet Security*. Artech House, 1997.

[PN04] Palmer, B. und J. Nazario: *Secure Architectures with OpenBSD*. Addison Wesley Pearson Education, 2004.

[Proa] Project, OpenBSD: *Install Guide*. http://www.openbsd.org/faq/faq4.html besucht am 11.09.2009.

[Prob] Project, OpenBSD: *OpenBSD FAQ*. http://www.openbsd.org/faq.

[Proc] Project, OpenBSD: *Reference Artwork*. http://www.openbsd.org/art4.html.

[sec] securityfocus.com: *BugTraq Mailing Liste*. BugTraq: http://www.securityfocus.com/archive/1.

[Sol02] Soltau, Michael: *Unix/Linux Hochverfügbarkeit*. mitp, 2002.

[Ste00] Stevens, W. Richard: *TCP/IP Ilustrated Volume 1- The Protocols*. Addison Wesley, 18. Auflage, 2000.

[Tab] Tabot, Trevor: *pf packet flow*. http://homepage.mac.com/quension/pf/flow.png.

[Tza] Tzanidakis, Manolis: *Using OpenBSD on the desktop*. http://www.linux.com/archive/feature/52930 besucht am 17.08.2009.

[UK] UK, SC Magazine: *WiFi is no longer a viable secure connection*. http://www.scmagazineuk.com/WiFi-is-no-longer-a-viable-secure-connection/article/119294/ besucht am 6.9.2009.

[Way00] Wayner, Peter: *Free For All: How Linux and the Free Software Movement Undercut the High Tech Titans*. HarperCollins, 2000.

[Wei] Weiss, Toodd R.: *OpenBSD drops firewall program in licensing dispute*. Computerworld Development: http://www.computerworld.com/s/article/61038/OpenBSD_drops_firewall_program_in_licensing_dispute?taxonomyId=063 besucht am 10.08.2009.

[Wika] Wikipedia: *Network Address Translation*. Funktionsweise NAT: http://de.wikipedia.org/wiki/Network_Address_Translation besucht am 29.08.2009.

[Wikb] Wikipedia: *Software Quality*. Source code quality http://en.wikipedia.org/wiki/Software_quality besucht am 01.09.2009.

D. BSD-Lizenz

[1] Berkeley hob den 3.Punkt am 22.Juli 1999 auf, so dass OpenBSD ohne diesen Punkt in der Lizenz ausgeliefert wird

OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

E. Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.
Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Gersthofen, 30. September 2009 _____

Unterschrift