PFDialogControl

Simple management of FreeBSD Network Configuration, DHCP/DNS, NAT and Firewall

Release 1.0.3

Eine Vordiplomarbeit von Bogdanovic Theodor für die Höhere Fachschule Uster HFU – Fachrichtung Telekomunikation

5. Mai 2015

Impressum

Danksagungen

Um diese Dokumentation besser Korrekturlesen zu können, wurde sie in einen formellen Teil (Kapitel 1 – 7) sowie in einen technisch orientierten Teil (Kapitel 8 - Ende) aufgeteilt. Daher möchte ich an dieser Stelle meinen beiden Korrekturlesern Hugo Brändle und Reto Grünenfelder recht herzlich für die investierte Zeit und Mühe Danken.

1 Inhaltsverzeichnis

2 Zusammenfassung / Management – Summary

2.1 Management – Summary

PFDialogControl ist ein Steuerskript, welches über mehrere Skripte aufgeteilt ist und ein leicht zu bedienendes Menü bereitstellt, über welches folgende Funktionen bereitgestellt werden sollen:

- Erleichterte Konfiguration der Netzwerkschnittstellen
- Aktivierung und Deaktivierung eines Kombinierten DHCP, -DNS Servers (hier über dnsmasq)
- Aktivierung und Deaktivierung der NAT (Network Address Translation) Funktion. Diese ermöglicht das Umleiten des LAN Verkehrs über eine öffentliche IP – Adresse
- Erstellen von Portweiterleitungen um die Kontaktaufnahme vom WAN (Wide Area Network) ins LAN zu ermöglichen.
- Erstellen von Verbindungssperren um lokalen Computern die Verbindung zu einer bestimmten IP – Adresse oder allgemein die Verbindung nach draussen zu verbieten.

Dies alles baut auf dem Betriebssystem FreeBSD auf und nutzt die per Default mitgelieferte und als Kernel Modul bereitgestellte Firewall PF (Packet Filter). Zusätzlich kommt noch das Softwarepacket dnsmasq dazu, welches DHCP und DNS bereitstellen soll. Mit all diesen Mitteln und einem leicht bedienbaren Menü, soll der bei einer solchen Konfiguration übliche Aufwand um ein erhebliches Reduziert werden und so einen einfachen aber zeittechnisch schnell konfigurierbaren Router ermöglichen.

2.1.1 Projektidee

Diese Projektidee hat ihren Ursprung in einer Notwendigkeit, die dem Entwickler im Laufe der Zeit bei der HFU aufgefallen ist. Denn durch die ständige Bereitstellung von virtuellen Maschinen für seine Mitstudenten bei diversen Gelegenheit und Projekten in gewissen Modulen für die HFU, erkannte ich, dass eine umfangreiche Installation von Webbasierten Firewalls, für die Abschirmung der eigenen privaten Netzwerksegmente auf die Dauer zeitaufreibend und nervend sein kann. So wurde die Idee dieses Projektes geboren.

2.1.2 Schwierigkeiten / unerwartete Probleme

Grössere Schwierigkeiten tauchen aufgrund der Erfahrungen des Entwicklers mit dieser Lösung nicht auf. Bis auf die üblichen kleinen Probleme bei einer Softwareentwicklung, die manchmal beim Entwickler ein aha Erlebnis verursachten. Jedoch gab es einige unerwartete Probleme, die zugegebenermassen doch etwas mehr Zeitaufwand verursachten. Nachfolgend ein kleiner Ausschnitt der gröberen Fälle:

- Dialog, die für den Menüaufbau zuständige Bibliothek, zeigte bei einer mehrstufigen Struktur von Untermenüs das merkwürdige Verhalten, dass erstens einmal im Vorfeld genutzte Variablen wieder auf den Wert „0“ gestellt wurden und somit an bestimmten Stellen aufgrund des leeren Inhalts, logischerweise zu Fehlermeldungen geführt haben. Zweitens führte ein ähnliches Verhalten dazu, dass in Variablen gespeicherte Pfade nicht mehr auslesbar waren.

  Bei der Identifizierung dieses, selbst heute für den Entwickler unerklärlichen Fehlers, wurde als Massnahme entschieden, ab sofort nur direkte (komplett ausgeschriebene) Pfadangaben und Variableninhalte zu nutzen. So konnte das unnatürliche Verhalten behoben werden.

Bereits geschriebene und funktionsfähige Sektionen, welche von diesem Fehler nicht betroffen waren, wurden ignoriert.

- Der zweite nervenaufreibende Fehler entstand durch das Kommandozeilen – Tool „sed“. Diese veraltete Version die in FreeBSD genutzt wird, hat einen kleineren Befehlsumfang als es für dieses Projekt notwendig wäre. Der Entwickler hätte sich gewünscht nur mit den vorinstallierten Standardwerkzeugen von FreeBSD zu arbeiten, jedoch erforderte die Behebung dieses Fehlers das Nachinstallieren von der moderneren Version „gsed“.

2.1.3 Lösungsansatz grob skizziert

Aufgrund der Komplexität des Projekts, folgt nun eine ganz grobe Umschreibung. In erster Linie wurde ein Menü kreiert, welches eine Netzwerkkonfiguration grafisch ermöglichen soll. So sollte ein verwenden der Kommandozeile unnötig werden. In einem zweiten Schritt wurden die beiden Punkte DHCP/DNS – Server und NAT – Funktion so klassifiziert, dass sie mit möglichst wenig Aufwand, schnelle Resultate liefern sollten. Dies führte zur Implementierung von Autoerkennungen, sodass meist nur die Optionen Aktivieren und Deaktivieren blind ausgeführt werden können. Auf Konfigurationsfehler wird stets hingewiesen. Die Kernfunktionen einer Firewall sind zu komplex um an dieser Stelle erklärt zu werden. Aus diesem Grund wird hier auf Punkt 8.3.2.4 „Firewall“ verwiesen.

2.1.4 Erreichen der Zielsetzung

Aufbauend auf der Aufgabenstellung, des Pflichtenhefts sowie des gesetzten Zeitrahmens dieser Vordiplomarbeit, kann seitens des Entwicklers gesagt werden, dass das Ziel erreicht worden ist. PFDialogControl deckt die im Auftrag gestellten, sowie die tatsächlich für den eigenen produktiven Einsatz benötigten Anforderungen vollumfänglich ab. Aber wie so üblich fliessen die Ideen während der Entwicklung wie ein Wasserfall, trotzdem muss man zwangsläufig, terminbedingt zu einem Ende kommen.

2.1.5 Erfahrungen/ Erkenntnisse aus diesem Projekt

Aufgrund der bestehenden Erfahrungen des Entwicklers mit FreeBSD und der sicheren Netzwerkkonfiguration, können als neue Erkenntnisse nur die unter Punkt 2.1.2 „Schwierigkeiten / unerwartete Probleme“ aufgeführten Punkte genannt werden. Definitiv als neue Erfahrung kann aber der Vorgang der Erstellung einer solchen Vordiplomarbeit, in ihrer Länge und im Detailgrad aufgeführt werden. Dies ist sicherlich eine neue Erfahrung für den Entwickler dieses Projektes.

2.1.6 Zeitaufwand (Soll – Ist – Vergleich)

Der Zeitaufwand wurde stets mit genügend Reserven geplant und deckt sich mit dem Inhalt unter Punkt 4 „Projektplanungen“. Eine detaillierte Ansicht folgt unter Punkt 4.

2.1.7 Ausblick

Als kleiner Ausblick auf die Arbeit sei im Vorfeld gesagt, dass sie detailreich aber verständlich gehalten wurde, sowie mit vielen Illustrationen versehen ist um den Leser, erstens zum Lesen zu animieren und zweitens ihn stets mit vielen bunten Bildern bei Laune zu halten, damit er auch wirklich bis zum Ende durchhalten kann.

3 Einleitender Teil

3.1 Motivation zu PFDialogControl

Die Motivation zu einem solchen Projekt reifte wie in dieser Branche üblich, über die Zeit und der langsamen Einsicht über viele „unbrauchbare“ Vorlösungen, die meist nicht dem entsprachen was man wirklich braucht. So kam es, wie eigentlich zu erwarten war, wenn man einen Virtualisierungscluster besitzt und allen davon erzählt und es auch noch über eine Webpräsenz veröffentlicht, dass viele auf einen zukommen und fragen ob sie nicht mal schnell eine virtuelle Instanz zu Testzwecken haben dürfen, am besten natürlich gleich sofort. Als kulanter Hobby – Web Hoster ist es natürlich selbstverständlich, dass man seinen Kommilitone und Freunden diesen Wunsch erfüllen möchte. Doch wie soll man sein privates Netzwerksegment, welches man stets sauber pflegt und wo man auch die Integrität erhalten möchte von so vielen einzelnen, in sich selbst isolierten Instanzen trennen. Die Antwort liegt auf der Hand, man muss sie in einzelne Subnetze oder gar in eigene VLAN’s verschachteln. Hierfür gibt es zwei Lösungsansätze. Erste Möglichkeit; man installiert ein Betriebssystem wie FreeBSD und führt die Regelsätze für Routing und Firewall von Hand in die entsprechenden Konfigurationsdateien. Dies ist jedoch für kurze Laufzeiten eine mühsame Siphos Arbeit und ganz ehrlich, wer hat schon Lust darauf. Die zweite Lösung wäre die Installation einer der vielen verfügbaren Open Source Lösungen wie PFsense, Endian Firewall oder IPcop. Viele dieser Anbieter behaupten selbst mit Web Frontend verbrauche ihre Lösung kaum nennenswerte Systemressourcen. Doch wenn man seine eigenen Virtualisierungsressourcen niedrig halten möchte, hat ein Webserver welcher für das Web Frontend notwendig ist, kaum Platz um bei einer grösseren Anzahl solcher Lösungen Arbeitsspeicher zu verschwenden.

Es musste eine Lösung her die Routingfunktionen sowie Sicherheit durch Firewall bot, vorzugsweise bei minimalem Ressourcenverbrauch. Würde man also auf ein Web Frontend verzichten und ein Betriebssystem nutzen welches von sich behauptet einen niedrigen Ressourcenverbrauch zu besitzen, könnte man ja pro zu isolierende Netzwerkinstanz gerade mal 256 MB RAM verbrauchen. Doch wie soll man sowas bedienen, denn es gibt ja kein Web Frontend? Am besten wäre eine Lösung die wirklich nur die notwendigen Funktionen anbietet und auf einem ressourcenschonenden Menüsystem basiert. Nun war die Idee für PFDialogControl geboren.

3.1.1 Der eigentliche Zweck von PFDialogControl

PfDIalogControl soll so einfach wie möglich installiert werden können, sowie die Konfiguration von Netzwerkdiensten wie DHCP/DNS – Server in wenigen Schritten, am besten in nur einem, ermöglichen. Die Firewall, welche eine Kernimplementation darstellt, soll ebenfalls in nur wenigen Schritten, das Erstellen von Portweiterleitungen ermöglichen, um den eigentlichen Nutzern der Test VM oder des Testnetzwerks den Zugang zu ermöglichen. Ebenfalls soll das Sperren von Verbindungen zu gewissen Ziel – IP‘s möglich sein. So kann beispielsweise die Verbindung auf spezifische Ports, aber auch zu ganzen Verbindungen der VM zu meinem privaten Virtualisierungscluster gesperrt werden. Wenn aber gewünscht, kann eine Verbindung zur OwnCloud erlaubt werden, falls der externe User einen Account hat, welchen er nutzen will. In dieser für die Vordiplomarbeit veröffentlichten Version, sieht die Minimalkonfiguration nur das Definieren von zwei Netzwerkinterfaces vor. Dies erleichtert die Konfiguration und spart Zeit. Dies ist auch das heimliche Ziel dieses Tools, denn der Entwickler und wahrscheinlich häufigster Nutzer ist recht „faul“ was die Art solcher Konfigurationen angeht.

3.1.2 Gesichtspunkte

Der primäre Fokus dieses Tools ist eindeutig die einfache Bedienung in virtuellen Umgebungen. Da der Entwickler auch gleich der häufigste Benutzer sein wird, orientiert sich die Bedienbarkeit nach seinen Wünschen. Da der Aspekt der Einfachheit, resultierend aus der Faulheit des Entwicklers im Vordergrund steht, wurde nach langer, beinahe philosophischer Überlegung auch die sekundäre Betrachtung aus Sicht eines unerfahrenen Benutzers in Betracht gezogen. So wurde das Menü so entwickelt, dass es für einen Anfänger immer genügend hilfreiche Hinweise als Unterstützung gibt. Der erfahrene Anwender wird diese meist einfach ignorieren.

3.1.3 Hinweise zur Struktur

Die Struktur dieser Arbeit entspricht den Regeln zur Erstellung schriftlicher Arbeiten an der HFU. In gewissen Bereichen wie beispielsweise Tests, findet der Leser neben den eigentlichen schriftlichen Abläufen auch noch Illustrationen welche zum besseren Verständnis oder gar zum Nachstellen besagter Tests helfen können.

3.2 Aufgabenstellung – Der Auftrag nach HFU

Der eigentliche Auftrag, wie er dem Entwickler gestellt wurde seitens der HFU ist nachfolgend als Abbildung aufgeführt. Die Darstellung als Abbildung dient der Sicherstellung des Auftrages, um Verfälschungen des Inhaltes auszuschliessen.

Höhere Fachschule Uster

Herr
Theodor Bogdanovic
Aabächliweg 1
8854 Siebnen

Zürich, 5. November 2014

Vordiplomarbeit 2014/15

Sehr geehrter Herr Bogdanovic

Sie erhalten folgende Vordiplomarbeit zugeteilt, die Sie selbständig zu lösen haben.

PFDialogControl

Beschreibung
Für die Firewall-Lösung „PF" (Paket Filter) und die Software „Dnsmasq" (DNS- und DHCP-Server) soll, falls möglich unter Verwendung der Bibliothek „Dialog-Boxes", ein Konsolen-Frontend erstellt werden. Die Lösung soll auf dem Betriebssystem „FreeBSD" lauffähig sein, welches die PF-Firewall standardmässig mitbringt.

Aufgabenstellung
Die Anwendung soll gemäss der im Pflichtenheft definierten Funktionen erstellt werden. Der primäre Fokus ist die Realisierung eines intuitiven Menüs für die Konfiguration von elementaren Funktionalitäten einer Firewall, wie beispielsweise sperren von Ports und Adressen, Weiterreichen von Verbindungen, etc. Weiter soll die Aktivierung und Deaktivierung von NAT sowie eines kombinierten DHCP/DNS Dienstes ermöglicht und entsprechend implementiert werden.

Die Aufgabe umfasst folgende Punkte:
- Erstellung eines Projektplans
- Analyse des technischen Systems und Erarbeitung eine Pflichtenheftes mit Prioritäten
- Entwurf eines Lösungskonzeptes mit Variantenstudien und Entscheidungsfindung
- Zusammenstellung und Aufbau der notwendigen Soft- und Hardware
- Programmierung der notwendigen Funktionen gemäss Pflichtenheft
- Test und gegebenenfalls Optimierung der Applikation
- Erstellung einer Dokumentation der Vordiplomarbeit mit allen Arbeitsschritten

Versand der Aufgabenstellung **25. November 2014**
Abgabe der 2 Dokumentationen **5. Mai 2015**

Freundliche Grüsse

Thomas Gemperli

Höhere Fachschule Uster, Berufsschulstrasse 1, CH-8610 Uster, hfu@bzu.ch, www.hfu.ch
HFU Sekretariat, Haldenweg 10, 8322 Madetswil, Tel. 044 954 11 27

Abbildung 1: Schriftlicher Auftrag vom Dozenten Gemerli Thomas an Bogdanovic Theodor --> Quelle: E-Mail Verkehr

3.3 Einführung in PFDialogControl

PFDialogControl baut auf den folgenden externen Kernkomponenten auf:

- FreeBSD → ein Betriebssystem aus der BSD (Berkeley Software Distribution) Familie
- PF (Packet Filter) Firewall → in FreeBSD als Satz an Betriebssystem – Kern Modulen, welche zur Laufzeit ladbar sind
- Dialog → eine Schnittstellen Bibliothek, welche ressourcenschonend den Aufbau von rudimentären Menüs in einer Shell erlaubt (siehe hierfür diverse Illustrationen innerhalb dieser Arbeit). Anmerkung; in FreeBSD wird eine etwas doch in die Jahre gekommene Version verwendet.
- DNSMASQ → ein Hintergrundservice (daemon), welcher unixartige Betriebssysteme in einen DHCP und DNS Server verwandelt. Dnsmasq gilt als kombinierter und einfach zu bedienender Dienst.
- gsed → in den meisten unixartigen Systemen ist sed der Standard, gsed ist eine aus der Linux – Welt bekannte Erweiterung, welche spezielle Formatierungen neu einführt. Sed oder gsed dienen dem Aufrufen, Abfangen und Modifizieren von Textdateien.

PFDialogControl bietet ein für Shell Verhältnisse leicht bedienbares, menügesteuertes Interface mit deren Hilfe es möglich wird ein FreeBSD Systemmit den oben erwähnten Komponenten in eine leicht zu steuernde Firewall mit Routingfunktionen umzuwandeln. Dabei wurde viel Wert darauf gelegt den Konfigurationsaufwand niedrig zu halten, um in wenigen Schritten an ein nutzbares Ziel zu gelangen. Ebenfalls wurde darauf geachtet, dass stets ein Hilfetext oberhalb eines jeden Abschnittes (Menü/ Untermenü) vorhanden ist, um so ein mühsames Nachschlagen in Manuals überflüssig werden zu lassen. Wurde ein Konfigurationsschritt ausgelassen, weil er vergessen oder aus Unwissenheit nicht gesetzt wurde, so stellt dies kein Problem dar, da eine aktive Fehlererkennung stets die gesamte Konfiguration prüft bevor ein Dienst ausgeführt wird. Hat man sich also „verkonfiguriert“ so ist ein Dienst nicht startbar, sondern ein „Pup – Up “ weist einen auf den Fehler hin und nennt die Lösung gleich mit. Diese aus Sicht des Entwicklers „kinderleichte“ Konfiguration, in Kombination mit der aktiven Fehlererkennung, führte dazu, dass es neben der auf der Kommandozeile ersten Vorerklärung, nur noch einen Quick Guide braucht.

Wie an manchen Stellen innerhalb von PFDialogControl erwähnt, sollte nach dem Top – Down Verfahren gearbeitet werden. Befolgt man dies, so ist eine funktionsfähige Firewall mit Routingfunktionen in nicht einmal 4 Minuten möglich. Nachfolgend eine kleine Einführung:

- Konfiguration der Netzwerk Interfaces
  - Hier gleich mal entscheiden welches das Outside Interface ist, also jenes, welches mit der Aussenwelt kommuniziert.
  - In einem nächsten Schritt das Inside Interface auswählen, dies ist jenes über welches die internen Computer mit PFDialogControl verbunden sind.
  - Über Interface Einstellungen die IP – Adressen der beiden oben genannten Interfaces vergeben. Hier kann zwischen statisch und DHCP gewählt werden, wobei stets Hilfetexte alles ausführlich erklären.
  - DNS und Routing können manuell aber auch dynamisch vergeben werden, je nach Einstellung im vorhergehenden Schritt.
  - Noch die beiden Interfaces Starten und schon ist die für später fehlerfreie Konfiguration erledigt
- Als nächstes muss man sich entscheiden ob man über seine PFDialogControl Instanz selber IP – Adressen vergeben will oder nicht. Dies ist aber sehr empfehlenswert, da so innerhalb des eigenen Netzwerks eine Namensauflösung stattfindet. Es muss lediglich ein Domänenname

angegeben werden. Auch hier gilt; so einfach wie möglich, Fehlkonfigurationen werden sofort mit einem Hinweis gemeldet.

- Hat man ein geroutetes Netzwerk, so kann man den nun folgenden Punkt deaktiviert lassen, muss aber das Routing selbst konfigurieren. Ist man aber wie der Entwickler dieses Tools extrem faul, so aktiviert man diesen Punkt am besten und lehnt sich zurück, denn nun werden die Pakete mittels NAT über eine öffentliche IP geleitet und man muss sich um nichts weiter kümmern. Die Performance – Einbusse über NAT ist beim leistungsstarken Network Stack des FreeBSD kaum der Rede wert.
- Der Punkt Portweiterleitung ermöglicht bei gesetzter Regel, dass über PFDialogControl eine Verbindung von aussen nach innen möglich wird. Hat man NAT deaktiviert, so kann man sich diese Regeln sparen, da PFDialogControl als Host von sich aus nach aussen sichtbar ist. Aufgrund der Komplexität der Konfiguration dieses Punktes wird hier auf den Quick Guide verwiesen.
- Der Punkt Verbindungen sperren, ermöglicht das Sperren von intern laufenden Computern nach aussen. Hier kann spezifisch eine IP – Adresse angegeben werden die auf einen Port zeigt, aber auch mittels der Option „any“ gleich sämtlicher Traffic zu irgendeiner IP – Adresse auf besagten Port unterbunden werden. So kann man beispielsweise sagen, dass ein bestimmter Computer innerhalb des LAN's nirgends (auf der ganzen Welt) sich mit Port 23 (Telnet) verbinden darf.

Dies ist nur ein kleiner „schneller“ Quick Guide, welcher einen groben Umriss der Fähigkeiten, bei einer Standard – Konfiguration wie sie der Entwickler benutzt aufzeigt. Um an dieser Stelle nicht ausschweifend zu werden, da sonst die nächsten 5 – 6 Seiten nur Einführung wären, sei an dieser Stelle ebenfalls für eine ausführlichere Erklärung an den Quick Guide verwiesen.

3.4 Grobbeschrieb des Lösungsansatzes

3.4.1 Menü – Design

Als die Vorabklärungen bezüglich Hardware und des zu verwendeten Hypervisor klar waren, wurden die ersten Gedanken über Menü – Layout und Skriptstruktur vorgenommen. Das Menü – Layout war hierfür ein elementarer Schritt, denn hieraus konnte abgeleitet und auch ergänzt werden, welche Aufgaben noch zusätzlich in PFDialogControl aufgenommen werden mussten. So z.B. wurde bei diesen Überlegungen, der noch nicht ganz geklärte Schritt entschieden ob und wie die Interface – Konfiguration bezüglich Design aussehen sollte. Die nachfolgende Auflistung der Überlegungspunkte soll einen kleinen Einblick in die Denkweise des Entwicklers geben:

- Soll die Interface Konfiguration auch über PFDialogControl laufen, soll sie über ein einfaches Shell – Skript als eine Art Wizard realisiert werden oder gar mittels Anleitung dem Anwender selbst überlassen werden?
  - Da die Konfiguration einfach sein soll, war hier die Entscheidung dies ebenfalls ins Menü aufzunehmen klar. Hierfür waren folgende Überlegungen ausschlaggebend.
    - Da PFDialogControl zwingend zwei Interfaces benötigt und ohne nicht funktioniert, muss an dieser Stelle sicher eine Auswahl des Outside –und Inside Interfaces getroffen werden.
    - Es muss die Möglichkeit da sein, eine statische aber auch eine dynamische IP Konfiguration vorzunehmen.
    - Falls statisch, braucht es die Möglichkeit diverse Punkte wie DNS und Default Route zu setzen.
    - Man muss einen Interfacenamen eingeben können und von diesem Konfigurationsparameter zurückerhalten.
    - Man muss alle Interfaces einzeln ein –und ausschalten können.

- Wieviel Konfigurationsspielraum soll der doch umfangreiche DHCP – Server haben?
  - Die möglichen Konfigurationsparameter sollen eine selbst für Anfänger unproblematische und vor allem schnelle Konfiguration ermöglichen.
    - Hier wäre aus Sicht eines Anfängers und des bereits mehrfach erwähnten konfigurationsfaulen Entwicklers die beste Lösung, wenn nur der notwendige Domänenname als einzige Option verlangt würde.
    - Der IP – Range sollte stets im selben Rahmen vergeben werden und sich automatisch anhand der Inside – Interface – Konfiguration anpassen. Diese Einstellung verlangt kein Vorwissen des Anfängers und entspricht einer Default – Konfiguration wie sie in Testnetzwerken mit einem üppigen Range üblich ist.
    - Ein Ein/ - Ausschaltknopf sollte auch nicht fehlen
- Der NAT Betrieb?
  - Da er sich stets an das Outside Interface richtet und mittels einer abgewandelten Konfiguration in /etc/pf.conf seitens des Entwicklers auch dynamisch an ein per DHCP vergebene IP – Adresse richtet ist hier nichts zu beachten.
    - Ein/ Aus Knopf nicht vergessen.
- Die Firewall muss im Kern zwei Funktionen erfüllen. 1.) Sie muss Verbindungen von aussen nach innen erlauben und 2.) Sie muss Verbindungen spezifische nach Ziel – IP und Ziel – Port sperren, aber wie soll das Menü – Layout aussehen?
  - Hier wurde lange zwischen Wizard und Einzelpunkt Abfrage überlegt. Hieraus folgten folgende Überlegungen:
    - Ein Wizard wäre für Anfänger eine grosse Erleichterung, jedoch könnten Punkte abgefragt werden die ein Anfänger unter Umständen nicht verstehen würde. Das Menü würde auch um ein Vielfaches wachsen, was sich negativ auf die Funktionsweise der veralteten, in FreeBSD verbauten Dialog Version auswirken würde (genaueres zu dieser Aussage siehe Punkt 8 Realisationen).
    - Ein Einzelpunktmenü ist im Grunde die einfachste Lösung. Hier könnte man alles schrittweise bis zum Ende durchgehen. Jedoch ist dies neben dem grösseren Scripting – Aufwand auch nicht wirklich etwas Schönes fürs Auge.
    - Eine Kombination aus beidem wäre die Lösung. Ein kurzes Menü, welches zu einem Formular führt wo die notwendigen Einträge gemacht werden können und dass sich leicht auf Fehler überprüfen lässt. So kam die Dialog Funktion Formbox ins Spiel.

Anhand dieser hier etwas grob dargestellten Überlegungen wurden die Menü – Layout – Fragen beantwortet, welche ausschlaggebend für das Design und somit die eigentliche Implementierung waren.

3.4.2 Dateistruktur – Design

Dieses stand im Grunde schon bei Beginn der Arbeit fest und richtete sich nach dem Standard – Design des Entwicklers. Doch wie so oft fragt man sich; ist dies wirklich das korrekte für eine solch wichtige Arbeit. Nach kurzem Überlegen entschloss sich der Entwickler seinen Idealen treu zu bleiben. Nun wurde PFDialogControl wie folgt strukturiert:

- Das Hauptskript „pfdctl“ enthält nur das Standard – Menü und liegt in /usr/bin
- Die primäre Bibliothek „libpfdctl“ mit sämtlichen relevanten Funktionen ist in /usr/lib/
- Die zusätzlichen Bibliotheken (hier nicht erwähnt) sind ebenfalls in /usr/lib
- Die Konfigurationsdateien sind in /usr/local/etc/pfdcontrol

Diese Splittung hat aus der Sicht des Entwicklers den Vorteil, dass nur das Hauptskript ausführbar sein muss und nur dort entsprechende Rechte vergeben werden müssen. Bei Paranoiden Profis ist somit eine nachträgliche Härtung (genannt Hardening) leichter mittels der Dateirechte durchzuführen. Zusätzlich sind die Bibliotheken vor böswilliger Ausführung geschützt.

4 Projektplan

Auf den nachfolgenden drei Seiten ist der Projektplan ersichtlich. Dieser wurde in drei Teile gegliedert um eine übersichtlichere Darstellung zu ermöglichen. Hierbei wurde folgende Aufteilung gewählt:

1.) Vorstufe zum Einreichen des Antrags: Hier wurden alle Schritte aufgeführt, welche vom stellen des Vordiplomgesuches bis zur Annahme, inklusive erstem Betreuungsgespräch beschreiben.
2.) Beginn mit Arbeit und Dokumentation Teil 1: Zeigt den Verlauf der eigentlichen Realisation der Arbeit, sowie die parallel verlaufende provisorische Dokumentation der einzelnen Schritte.
3.) Beginn mit Arbeit und Dokumentation Teil 2: Legt den Schwerpunkt auf die Testphase, sowie die Fertigstellung der gesamten Arbeit.

Hinweis: In der elektronischen Fassung dieses Dokuments, welche sich auf dem beiliegenden Datenträger befindet, sind auf den nachfolgenden Seiten Links zu finden, welche zur Online – Version des mittels Gantt – Project erstellten Projektplans führen.

Die benötigte Zeit für diese Vordiplomarbeit beträgt **184 Stunden**. Somit mussten **34 Stunden** an Arbeit überzogen werden.

4.1 Vorstufe zum einreichen des Antrags

Folgen Sie dem unten aufgeführten Link zur Online Version des mittels Gantt – Project erstellen Projektplans. Sie können aber auch ein PDF, eine JPEG Datei oder ein html – Dokument nutzen, welches sich auf dem Datenträger befindet. Zusätzlich haben Sie Zugriff auf die Original Gantt – Files auf dem Datenträger. Im Ordner Software auf dem Datenträger finden Sie einen Setup der aktuellen Gantt – Project Version.

http://tb-network.jumpingcrab.com/gantt/gantt_1/Gantt_1.html

4.2 Beginn mit der Arbeit und Dokumentation Teil 1

Folgen Sie dem unten aufgeführten Link zur Online Version des mittels Gantt – Project erstellen Projektplans. Sie können aber auch ein PDF, eine JPEG Datei oder ein html – Dokument nutzen, welches sich auf dem Datenträger befindet. Zusätzlich haben Sie Zugriff auf die Original Gantt – Files auf dem Datenträger. Im Ordner Software auf dem Datenträger finden Sie einen Setup der aktuellen Gantt – Project Version.

http://tb-network.jumpingcrab.com/gantt/gantt_2/Gantt_2.html

4.3 Beginn mit der Arbeit und Dokumentation Teil 2

Folgen Sie dem unten aufgeführten Link zur Online Version des mittels Gantt – Project erstellen Projektplans. Sie können aber auch ein PDF, eine JPEG Datei oder ein html – Dokument nutzen, welches sich auf dem Datenträger befindet. Zusätzlich haben Sie Zugriff auf die Original Gantt – Files auf dem Datenträger. Im Ordner Software auf dem Datenträger finden Sie einen Setup der aktuellen Gantt – Project Version.

http://tb-network.jumpingcrab.com/gantt/gantt_3/Gantt_3.html

5 Pflichtenheft

5.1 Allgemeine Definition

Die allgemein zu erfüllenden Spezifikationen von PFDialogControl sind unter Punkt 3.2 „Aufgabenstellung“ definiert. Die dort grob umschriebenen Funktionen werden nachfolgend noch genauer beschrieben.

5.1.1 Sinn und Zweck

Sinn und Zweck von PFDialogControl ist die Realisierung eines Steuermenüs für die PF Firewall inklusive den daraus resultierenden NAT Funktionalitäten, sowie die Steuerung von DHCP und DNS Funktionalitäten realisiert über dnsmasq. Dies alles basierend auf dem Funktionsumfang, den Spezifikationen, den technischen Möglichkeiten sowie den aktuell geltenden Software – Paketstand von FreeBSD 10 bis aktueller Stand Version 10.1. Die Versionsnummern von PF sowie dnsmasq sind durch das FreeBSD Projekt gebunden, welches stets für eine entsprechende Abwärtskompatibilität sorgt. Die Funktionsweise der aktuellen, für diese Arbeit erstellten Version von PFDialogControl, ist somit an die oben genannten Versionsnummern Von FreeBSD gekoppelt. Funktionstüchtigkeit in neueren Versionen von FreeBSD ist technisch möglich, jedoch wird hier jede Garantie und Haftung abgelehnt.

5.1.2 Geltungsbereich

Dies Arbeit, inklusive der aktuellen Version von PFDialogControl unterstehen dem temporären Copyright des Entwicklers. Nach abschliessender Bewertung der Vordiplomarbeit, aber spätestens am 1. August 2015 geht diese Arbeit inklusive PFDialogControl und aller seiner Komponenten in die Standard BSD Lizenz über, der auch FreeBSD untersteht. Dies geschieht ohne Ankündigung seitens des Entwicklers.

5.2 Zielsetzung

In der Zielsetzung werden die genauen Parameter spezifiziert, welche sich nach Punkt 3.2 „Aufgabenstellung“ orientieren. Hierbei wird in zwei Kategorien unterteilt. 5.2.1 definiert was definitiv implementiert werden muss und was sich als erreichtes oder noch offenes Wunschziel definiert. 5.2.2 ist eine spezielle Auflistung an „zukünftigen Wunschzielen“ welche sich in einer ausserhalb dieser Vordiplomarbeit befindlichen, späteren Version von PFDialogControl implementieren lassen. Die zukünftigen Wunschziele bauen auf sich im Moment befindlichen, aber noch nicht nutzbaren Erweiterungen der aktuellen Version von PFDialogControl auf. Die Implementierung dieser Erweiterungen in PFDialogControl, als mögliche Diplomarbeit, sind noch offen und bleiben dem Entwickler vorbehalten.

5.2.1 Muss- / Wunschziele

Tabelle 1: MUSS- / Wunschziele

Funktion/ Definition	Beschreibung	Muss/ Auftrag
Einfache Menüführung	Die Struktur der Menüführung muss simpel sein um im Profibereich schnelle Konfigurationen zu ermöglichen. Anfänger müssen ohne Schwierigkeiten navigieren können.	**MUSS – Aus Auftrag ersichtlich**
Allgemein Einfache Konfiguration	Erfragt werden nur die nötigsten Optionen. Unklarheiten sollen durch Hilfetexte über dem Menü helfen, zusätzlich noch ein schriftlicher Quick Guide. Autokonfiguration soll so viel wie möglich selbst übernehmen. ACHTUNG gilt nur für NAT und DHCP/DNS – Server!	**MUSS – Aus Auftrag ersichtlich**
NAT Aktivieren- und Deaktivieren	NAT muss sich Aktivieren und Deaktivieren lassen. Eine Warnung weist darauf hin, dass beim Deaktivieren das Routing beachtet werden muss (Konfiguration für Fortgeschrittene).	**MUSS – Aus Auftrag ersichtlich**
DHCP/DNS – Server Aktivieren- und Deaktivieren	DHCP/DNS – Server muss sich auch deaktivieren lassen, wenn ein eigener benutzt wird. Warnung für Anfänger weist auf die Folgen eines deaktivierten Servers hin. Ansonsten Autokonfiguration des IP – Ranges (gute Lösung für schnelles vorgehen für Profi und Anfänger)	**MUSS – Aus Auftrag ersichtlich**
Firewall → Portweiterleitung	Das Menü für die Portweiterleitung muss auf simple Art und Weise erlauben eine Verbindung beliebigen Ursprungs von aussen zu einem Computer im LAN zu leiten. Dabei soll die Zieladresse im LAN angegeben werden können, sowie der Port von assen zur Firewall und der Port zum Computer im LAN. Genauso einfach muss diese Regel auch gelöscht werden können.	**MUSS – Aus Auftrag ersichtlich**
Firewall → Verbindung Sperren	Das Menü für die Sperrung einer Verbindung, muss auf simple Art und Weise erlauben, eine Verbindung von einem Computer innerhalb des LAN’s nach aussen zu Sperren. Dabei soll es	**MUSS – Aus Auftrag ersichtlich**

	möglich sein den zu sperrenden Computer das Ziel mittels IP und Port genau anzugeben. Es soll aber auch möglich sein nur einen Port anzugeben und mittels der Firewall üblichen Direktive „any“ jegliche Zieladresse zu sperren. Genauso einfach muss diese Regel auch gelöscht werden können.	
Konfigurationssicherheit	Für Anfänger sollen Fehlermeldungen „Pup - Ups“ generiert werden, die eine Fehlerbeschreibung und einen Hinweis geben was und vor allem wo noch Konfigurationen fehlen.	**MUSS – Aus Auftrag NICHT ersichtlich**
Bestätigungssicherheit	Wo kritische Entscheidungen (Ein-/ Ausschalten, Löschen, Erstellen) getroffen werden müssen, wird stets eine Bestätigung verlangt.	**MUSS – Aus Auftrag NICHT ersichtlich**
Funktion/ Definition	**Beschreibung**	**Wunschziele**
Status Informationen	Dienste wie NAT und DHCP/DNS – Server sollen ihren Status (Aktiv/ Inaktiv) ersichtlich anzeigen.	**Wunsch**
Auszug aus pftop	Es soll möglich sein einen Auszug aus dem aktuellen Zustand der Firewall Tabelle mit den aktuellen Verbindungen zu erzeugen.	**Wunsch**
Menüpunkt Statusanzeige	Die Firewall soll einen separaten Menüpunkt besitzen aus dem der Status (Running oder Zeilenfehler) ersichtlich ist.	**Wunsch**

5.2.2 Zukünftige Wunschziele

Tabelle 2: Zukünftige Wunschziele

Funktion/ Definition	Beschreibung	Wunschziel
Backup	Die Regelsätze werden in separate Dateien geschrieben. Dies soll später in einer eigenen Version die Möglichkeit bieten, mittels Backup – Button ein Backup des aktuellen Zustandes zu machen und diesen später wieder herzustellen.	Musste aus Zeitgründen aufgegeben werden. PFDialogControl ist aber vom Design her bereits so geschrieben um diese Funktion leicht implementieren zu können.
Multi Homed	Es soll möglich sein mehre Interfaces (mehr als Inside Interface) über das Outside Interface zu leiten.	PFDialogControl ist bereits in diese Richtung geschrieben. Über den Sinn lässt sich streiten, da dies dem Konzept des Betriebes in virtuellen Umgebungen mit gesplittung und isolierten Umgebungen widerspricht.

5.3 Abweichungen und Korrekturen

Von den MUSS Funktionen gibt es keinerlei Abweichungen oder Korrekturen. Diese können nach Auffassung des Entwicklers alle umgesetzt werden.

Bei den Wunschzielen müssen Abstriche gemacht werden. Vorabklärungen haben gezeigt, dass das PF Kernelmodul zwar eine Ausgabe in die Standard Ausgabe schreibt, diese aber zeitgleich mit einer Art implementiertem „exit“ auch abbricht. Es liess sich nicht feststellen ob dies aus Sicherheitsgründen geschieht, oder einfach die Standardvorgehensweise ist. Das Resultat ist auf jeden Fall, dass das in die Standardausgabe geschriebene sich nicht weiterleiten liess. Somit ist ein speichern in eine Variable zur späteren Darstellung nicht möglich. Die Weiterleitung der Ausgabe von pftop zur späteren Darstellung ist ebenfalls nicht möglich, da es sich hierbei um einen konstant lauffähigen Programcode handelt, der sich nicht so ohne weiteres weiterleiten lässt, dies war aber auch so zu erwarten.

6 Vorabklärungen und Analysen

6.1 Allgemein notwendige Vorabklärungen

In diesem Abschnitt sollen alle notwendigen Vorabklärungen getroffen werden, welche notwendig sind um eine solide Grundlage zu haben und Fragen bezüglich aller Schritte zu klären welche für die Arbeit an PFDialogControl notwendig sind. Es sollen aber an dieser Stelle nur Entscheidungen getroffen werden, welche die Peripherie um die eigentliche Arbeit zu PFDialogControl betreffen. Solche Fragen könnten sein:

- Welche Hardware soll in welcher Form eingesetzt werden?
- Welche Entwicklungsumgebung soll verwendet werden?
- Welche Versionen sollen genutzt werden?
- Portabilität der Testumgebung

Nachfolgend sollen diese Punkte mit unterschiedlichen Mitteln so geklärt werden, dass weder Zweifel noch nachträgliche Wünsche nach Änderungen bestehten.

6.1.1 Welche Hardware soll in welcher Form eingesetzt werden?

Dass reale Hardware, aufgesplittet auf einzelne Rechner genutzt wird, kann an dieser Stelle schon einmal ausgeschlossen werden. Denn es müssen neben einem Rechner auf dem PFDialogControl läuft, mindestens noch zwei Rechner vorhanden sein, die sich innerhalb des durch PFDialogControl gemanagten LAN's befinden. Mittels dieser Rechner sollen die Einstellungen getestet und ebenfalls demonstriert werden. Die bei dieser Lösung anfallenden Kosten, wären selbst bei der Verwendung von Occasion – Hardware zu hoch aus Sicht des Entwicklers. Daher war die aus Kostengründen einzig logische Entscheidung auf virtualisierte Hardware zu setzten, da hier bereits eine Infrastruktur existierte, oder weit kostengünstiger realisiert werden konnte. Um nicht die bereits existierende Virtualisierungsinfrastruktur durcheinander zu bringen, wird für diese Arbeit ein Internetschnäppchen nämlich ein IBM Server der e-Serie (Modell 3550) aus einem Occasion Bestand gekauft.

Damit belaufen sich die realen Kosten für die Vordiplomarbeit auf die einmalige Ausgabe von CHF **180.00**.

6.1.2 Welche Entwicklungsumgebung soll verwendet werden?

Da bei dieser Arbeit keinerlei Code anfällt der kompiliert oder sonst in irgend einer Art und Weise in eine Maschinensprache übersetzt werden muss, ist lediglich die Frage zu klären welcher Editor zur Entwicklung genutzt werden soll. Hierbei wird auf eine Entscheidungsfindung mittels Analyse oder Nutzwertanalyse verzichtet. Der Entwickler bevorzugt klar den Editor JEdit, welcher einen Schwerpunkt auf die Hervorhebung von Shellskript Syntax besitzt. Ebenfalls ist JEdit auf vielen unterschiedlichen Plattformen (da er Java nutzt) lauffähig und ermöglicht so die kontinuierliche Arbeit von unterschiedlichen Arbeitsplätzen aus. Der Entwickler nutzte aber gelegentlich auch andere Editoren wenn es die Situation erforderte, da er den Luxus einer Syntaxhervorhebung zwar schätzt, aber ihn nicht als notwendig ansieht. Ausnahmen sind die Editoren vim und emacs. Der Entwickler ist absolut kein Fan solcher antiquierten Lösungen und propagiert für die Kommandozeile klar den

Editor nano, welcher einem durch den Installer von PFDialogControl auch „zwangsweise “ aufgezwungen wird.

6.1.3 Welche Versionen sollen genutzt werden?

Auch in diesem Punkt sind keine speziellen Vorkehrungen zu treffen, da das FreeBSD Projekt selbst für die notwendige Abwärtskompatibilität sorgt. Dies geschieht meist durch Modifikation der einzelnen Tools, oder aber durch die Abstraktionsschicht in den jeweiligen Versionen, die eine Nutzung veralteter Softwareversionen ermöglicht, da FreeBSD den Kernelspace vom Userland trennt. Da aber in vielen Internetforen rund um FreeBSD Diskussionen aufgetaucht sind, die eine Design – Änderung des Initsystems (Startprozedur des Systems) behandeln, wurde entschieden (siehe Pflichtenheft) die für PFDialogControl ausführbaren FreeBSD Versionen auf die zur Erstellung und Entwicklung dieser Arbeit aktuellen Versionen 10 und 10.1 zu beschränken.

6.1.4 Portabilität der Testumgebung

Hier stellt sich die Frage, wie die Test –und Entwicklungsumgebung bezüglich Beweglichkeit aufgebaut werden soll. Sollte der Server auf dem die PFDialogControl – Instanz mit ihren Testsystemen läuft in einem fix installierten Rack verbaut sein, von wo aus mittels Remotezugriff demonstriert werden kann, oder sollte der Server zu Präsentationszwecken mobil sein. Diese Entscheidung sollte zwingend im Vorfeld definiert werden, um bei allfälligen Änderungen den Konfigurationsaufwand zu reduzieren. Diese Entscheidung soll durch nachfolgende Pro und Kontraliste geklärt werden. Wobei die punktuell umfangreichere Liste bzw. Seite gewinnt.

Pro Punkte:

- Durch die Verwendung eines Externen Routers ist die Testumgebung besser an Bedingungen mit unterschiedlichen Internetanbindungen möglich.
- Die Schulfirewall erlaubt nur eine eingeschränkte Nutzung freier Ports; wirklich einen sehr kleinen Portrange.
- Testsystem kann so konfiguriert werden, dass mitgebrachte Hardware von dritten zu Test- und Demonstrationszwecken angeschlossen werden kann.
- Leitungsunterbrüche seitens des Schulnetzwerkes (Internetanbindung) fällt nicht ins Gewicht.
- Bei Verlust der Internetanbindung, kann eine Testmaschine ausserhalb der Firewall schnell von Hand konfiguriert werden, um dennoch das Durchreichen der Datenpakete zu demonstrieren.
- Im Allgemeinen ist die Fehlerbehebung vor Ort an der defekten Hardware leichter als per Remote Verbindung.
- Zwar liegt der Fokus klar auf PFDialogControl, doch macht es weit mehr Eindruck irgendwo mit einem 2 Höheneinheiten hohen Server aufzutauchen, als nur irgendwohin ein logisches Konstrukt zu umschreiben und auf die Vorstellungskraft der Betrachter zu hoffen.

Kontra Punkte:

- Das Vorhandensein einer stabilen Stromquelle ist ungewiss. Man will ja schliesslich einen Server, den man später noch produktiv nutzen will nicht gefährden.
- Die Netzwerkanbindung ist ebenfalls noch ein wichtiger Punkt; überhaupt vorhanden?

- Einer der wohl „schwerwiegendsten“ Punkte; der Entwickler hat eigentlich nicht wirklich Lust einen 18,6 kg schweren (exklusive 8x 2,5 Zoll Festplatten) Server herumzuschleppen. Dieser Punkt zählt gleich **2-mal.**
- Sicherheit und Integrität; es muss immer in Betracht gezogen werden, dass Teile beim Transport beschädigt oder gar durch kriminelle Handlungen Teile des Systems gestohlen werden können.

Bei der Gegenüberstellung, der Pro –und Kontra – Punkte fällt die Entscheidung klar zu Gunsten der Pro – Seite aus.

6.1.5 Der zu verwendende Hypervisor

Lösungen wie Oracle VirtualBox und diverse VMWare Editionen bieten sich zwar geradezu an für solche Test –und Entwicklungsumgebungen, jedoch sind sie aus Sicht des Entwicklers eine doch etwas „schäbige“ Wahl für eine solche Arbeit. Hinzu kommt noch, dass eine Entwicklung in einer solchen Umgebung mögliche Diskrepanzen aufweisen kann, welche eine Adaption in eine produktive Umgebung schwierig machen könnten. Aus diesem Grund soll von Anfang an eine Lösung gewählt werden, welche mögliche Schwierigkeiten in Produktivumgebungen gleich aufzeigt.

Doch welche Lösung soll hier gewählt werden? Unter den Gesichtspunkten, dass eine saubere Ausführung von FreeBSD möglich sein soll, es aber auch zu keinen mühsamen Nachkonfigurationen bezüglich virtueller Hardwareinkompatibilität geben soll. Sowie eine Beschleunigung der Virtualisierung durch paravirualisierte Hardwaretreiber möglich seien sollte, kommen im Grunde nur folgende Lösungen in Frage:

- KVM (Kernel-based Virtual Machine) auf irgendeiner Linux Distributionsbasis
- KVM auf Solaris Basis (in diesem Fall SmartOS von Joyent)

Die beiden grossen XEN (bspw. XenServer) und VMWare ESXi fallen weg, da XEN nur eine eingeschränkte Unterstützung für FreeBSD bietet (ausser man greift tief in die Konfiguration ein) und VMWare ESXi nicht Open Source ist. An dieser Stelle sei nochmals daran erinnert, dass der Entwickler ein grosser Anhänger der Open Source Bewegung ist und stets versucht nur quelloffene Softwares zu nutzten.

Da die beiden oben erwähnten Lösungen, jeweils entgegengesetzte Vorteile bezüglich der, leichten Bedienbarkeit und effizienten (innovativen) Nutzung der Features bieten, konnte sich der Entwickler nicht entscheiden welche er wählen sollte. Aus diesem Grund soll nachfolgende Nutzwertanalyse eine neutrale und möglichst objektive Entscheidungsfindung ermöglichen.

1) Die notwendigen Auswalkriterien:

*Tabelle 3: **Kriterien** für die Nutzwertanalyse zur Wahl eines geeigneten Hypervisors*

Kriterien	Deren Begründung bzw. Notwendigkeit
Leichte Bedienbarkeit (Kommandozeile)	Im Notfall aber auch sonst, sollte die Bedienbarkeit (erstellen, konfigurieren, ändern) von virtuellen Maschinen leicht von der Hand gehen. → **aus Zeitgründen**
Bedienung durch GUI (grafische Oberfläche)	Ist im Grunde nicht zwingend nötig, aber erleichtert manchmal die Arbeit und gibt bei Präsentationen was her. → **aus Zeitgründen**
Backup Fähigkeiten	Für die Entwicklung durchaus wichtig, wenn man Änderungen an der virtuellen Hardware machen muss und man sich nicht sicher ist wie sich diese Änderungen auf die VM auswirken. Ein Snapshot erleichtert somit die Wiederherstellung des bereits funktionsfähigen Zustandes wieder. → **aus praktischen Gründen**
Erstellungsfähigkeiten	Wenn zu Test- oder Präsentationszwecken eine neue VM her muss, soll dies schnell von der Hand gehen. → **aus praktischen Gründen**
Netzwerkfähigkeiten	Der Hypervisor sollte im Stande sein nicht nur reguläre Netzwerkschnittstellen zu erzeugen, sondern diese auch voneinander zu isolieren (bspw. durch VLAN's) → **technisch notwendig**
Isolationsmethoden → netzwerktechnisch	Gekoppelt an den oberen Punkt wären extra Features wie komplette, eigenständige Netzwerkinfrastrukturen wie bspw. eigenständige Switches von Vorteil. → **wenn möglich**

Die in Tabelle 3 aufgeführten Kriterien erstrecken sich über absolut notwendig, bis hin zu, wäre von Vorteil falls vorhanden. Sie stellen somit das absolute Minimum dar, welches für eine Entscheidungsfindung notwendig ist.

2) Ermittlung des Gewichtungsfaktors für die einzelnen Kriterien (GWF):

*Tabelle 4: **Gewichtungsberechnung** für die Nutzwertanalyse zur Wahl eines geeigneten Hypervisors*

Kriterien / Kriterien	Leichte Bedienbarkei	Bedienung durch GUI	Backup Fähigkeiten	Erstellungsfä higkeiten	Netzwerkfähi gkeiten	Isolationsme thoden → netzwerktec	Summ e der Bew-krit.	Gewich. -faktor
Leichte Bedienbarkeit (Kommandozeile)		50%	70%	50%	35%	35%	240%	22.33%
Bedienung durch GUI (grafische Oberfläche)	25%		25%	35%	15%	10%	110%	10.23%
Backup Fähigkeiten	50%	25%		25%	15%	15%	130%	12.10%
Erstellungsfähigkeiten	50%	15%	35%		50%	50%	200%	18.60%
Netzwerkfähigkeiten	50%	25%	60%	50%		50%	235%	21.86%
Isolationsmethoden → netzwerktechnisch	35%	10%	35%	35%	45%		160%	14.88%
Der sich hierausergebende Umrechnungsfaktor = 10.75 →						**Summe**	1075%	100%

Hinweis zu Tabelle 4: Die Gegenüberstellung folgt folgendem Muster: **Ist Kriterium senkrecht wichtiger als → Kriterium waagrecht, wenn ja wieviel Prozent in Bezug auf 100%.** Im Normalfall sollte bei einer solchen Gegenüberstellung der Kriterien, jeweils der gleiche Punkt in senkrechter wie in waagrechter Position den gleichen prozentualen Wert ergeben. Also senkrecht „Leichte Bedienbarkeit" → waagrecht Bedienung durch GUI, müsste in beiden fällen den gleichen Wert ergeben. Doch der Autor dieser Arbeit benutzt eine eigene Variation, wo in jeder neuen Zeile eine frische Betrachtung der Sichtweise herangezogen wird. Dies erlaubt mehr Dynamik und mehr Spielraum wenn man sich bei einem Punkt nicht ganz sicher ist. Zwar stimmt die Punktesymmetrie nicht ganz, aber man erhält aus Sicht des Autors ein weit ausgewogeneres Resultat als beim klassisch statischen Model.

3) **Bestimmung des Zielerreichungsfaktors, basierend auf teils jahrelanger Erfahrung des Entwicklers mit den beiden Lösungen (ZEF):**

Angewendeter Zielerreichungsfaktor bei nachfolgender Tabelle liegt im Range 1 – 5 (1 = sehr schlecht, 5 = sehr gut)

*Tabelle 5: **Zielerreichungsfaktor** für die Nutzwertanalyse zur Wahl eines geeigneten Hypervisors*

Bewertungskriterien	KVM auf Linux – Basis	KVM auf SmartOS - Basis
Leichte Bedienbarkeit (Kommandozeile)	2	4
Bedienung durch GUI (grafische Oberfläche)	4	3
Backup Fähigkeiten	5	5
Erstellungsfähigkeiten	3	4
Netzwerkfähigkeiten	4	5
Isolationsmethoden → netzwerktechnisch	2	5

4) **Eigentliche Ermittlung des Siegers:**

*Tabelle 6: **Ermittlung eines Siegers** für die Nutzwertanalyse zur Wahl eines geeigneten Hypervisors*

		KVM auf Linux - Basis		KVM auf SmartOS – Basis	
Bewertungskriterien	**GWF**	**ZEF**	**Ermitt. Wert**	**ZEF**	**Ermitt. Wert**
Leichte Bedienbarkeit (Kommandozeile)	22.33%	**2**	**44.66%**	**4**	**89.32%**
Bedienung durch GUI (grafische Oberfläche)	10.23%	**4**	**40.92%**	**3**	**30.69%**
Backup Fähigkeiten	12.10%	**5**	**60.50%**	**5**	**60.50%**
Erstellungsfähigkeiten	18.60%	**3**	**55.80%**	**4**	**74.40%**
Netzwerkfähigkeiten	21.86%	**4**	**87.44%**	**5**	**109.30%**
Isolationsmethoden → netzwerktechnisch	14.88%	**2**	**29.76%**	**5**	**74.40%**
Gesamtnutzwert			**319.08%**		**438.61%**
Rangfolge					

Wie aus Tabelle 6 deutlich zu erkenne ist, haben wir einen Sieger. Das Open Source Projekt SmartOS vom Unternehmen Joyent erfüllt sämtliche verlangten Kriterien. Dieses Ergebnis ist aber auch nicht sonderlich überraschen. Denn betrachtet man den Schwerpunkt der geforderten Kriterien, welcher klar auf leicht zu bedienende Kommandozeilen Tools und gute Netzwerk Fähigkeiten abzielt, stellt SmartOS sicherlich die bessere Wahl darstellt. Somit wäre die Frage nach dem zu verwendenden Unterbau für diese Arbeit geklärt.

6.2 Recherche und Marktanalyse

Vor einreichen des Gesuches um diese Arbeit angehen zu dürfen, wurde im Vorfeld eine Websuche über Google durchgeführt. Hierzu wurden folgende Suchkriterien verwendet, um ein Vorhandensein bereits existierender Lösungen zu untersuchen:

- Kombinationen folgender Suchbegriffe wurden verwendet
  - Pf
  - Commandline tool
  - Dialog
  - Menu
  - Pf shell script

Aus all diesen Kombinationen ergab sich kein Treffer, welcher zu einem gleichen oder annähernd ähnlichem Projekt führte. So sei an dieser Stelle nochmals betont, dass es sich hier um eine Art „Premiere“ in diesem Bereich handelt. Basierend auf der Ansicht des Entwicklers von PFDialogControl, dass eine solche Lösung in der heutigen Zeit der Virtualisierung und der sich hieraus ergebenden Problematik, eine solche Lösung in naher Zukunft immer notwendiger sein wird. Ob dies nun PFDialogControl oder ein ähnliches Produkt sein mag ist im Grunde egal. Der Markt wird nach Einschätzung des Entwicklers in diese Richtung tendieren; Siehe dazu das neue Portfolio von Oracle in Richtung Virtualisierung (Netzwerkvirtualisierung) von Solaris 11.2.

7 Hauptstudie mit Konzeptvarianten

7.1 Allgemeine Definition

In der Hauptstudie werden nur die eigentlichen Aspekte von PFDialogControl selbst behandelt. Zwar sind die Bedingungen durch die Aufgabenstellung, sowie die klare Präzisierung durch das Pflichtenheft klar. Sowie orientiert sich der Entwickler der heute gebräuchlichen Design Vorgabe der üblichen Lösungen. Doch gibt es zwei nachfolgende Aspekte die mittels Entscheidungsfindung definiert werden müssen. Diese beiden Punkte haben in einer Vorüberlegung dem Entwickler einiges an Kopfzerbrechen bereitet. So soll versucht werden, mittels einer Nutzwertanalyse diese beiden Fragen zu klären, um so das bestmögliche Resultat aus Sicht eines erfahrenen, sowie eines unerfahrenen Endanwenders zu generieren.

7.2 Problemstellungen

Nachfolgen die beiden noch zu klärenden Probleme bezüglich Design und Implementierung.

7.2.1 Problem 1: Das Menü Design, Wizard ja oder nein?

In den ersten Gedankenspielen, noch vor Beginn der Arbeit war das grobe Aussehen klar. Dies resultiert aus der Tatsache, dass ich schon vor einreichen des Antrages mit dem Gedanken gespielt habe. Dies aus purer persönlicher Notwendigkeit heraus, so etwas zu schreiben. Gestützt auf das heute übliche Design, dem Endanwender mit wenig Erfahrung mittels Konfigurationsassistenten unter die Arme zu greifen, stellte sich schnell die Frage ob ein Wizard realisiert werden soll. Besonders der Menü Punkt Interface – Konfiguration wäre mit einem Assistenten womöglich einfacher verständlich als eine Standard – Lösung mittels klar strukturierten Menü Punkten. Aus dieser Frage resultierend ergeben sich zwei Varianten:

1) Variante mit Wizard (nachfolgend nur mit **Konzept Wizard** betitelt)

Die Idee wäre mit einem Wizard, oder auf Deutsch mit einem Konfigurationsassistenten zu arbeiten, welcher bei Aufruf aus dem Hauptmenü heraus startet. Dann innerhalb eines eigenen Skriptes heraus die notwendigen Fragen stellt, welche für die Konfiguration dringend notwendig sind. Dies würde jedoch nur bei den Menü Punkten Interface – Konfiguration und Firewall Einstellungen (bzw. Portweiterleitung und Verbindung sperren) Sinn machen.

2) Variante strukturierte Menü Punkte (nachfolgend nur mit **Konzept Struktur** betitelt)

Hier wäre die Idee alles auf einzelne Menü Punkte zu legen, welche klar vom Benutzer ausgewählt und durchlaufen werden müssten. Jedoch besteht die Gefahr, dass der Endanwender einen Punkt vergisst oder sich gar aus Unwissenheit denkt: „das lass ich lieber sein, verstehe ja eh nicht was es bedeutet“. Diese Situation müsste mit vielen Sicherungen und grossem Aufwand so geregelt werden, dass der Endanwender auf seinen Fehler hingewiesen wird und noch einen Wegweiser zu den noch offenen Punkten mit auf den Weg bekommt. Diese Art Sicherung wäre ja aber ohnehin laut Pflichtenheft ein Muss.

Voranmerkung: Konzept Wizard bietet zwar auf den ersten Blick die fürs Auge schönere Lösung, doch sei an diesem Punkt noch erwähnt, dass aus langer Erfahrung seitens des Entwicklers die Implementierung solcher dynamischer Fenster, welche einen „Weiter" –und „Zurück" – Button beinhalten, nur schwer mit einem statischen System wie dialog realisierbar sind. Diese Erkenntnis wird auch in folgende Tabelle mit den Auswahlkriterien fliessen.

Beginnen wir nun mit der Nutzwertanalyse:

1) Die notwendigen Auswalkriterien:

*Tabelle 7: **Auswahlkriterien** für die Nutzwertanalyse zur Wahl eines Konzepts (Wizard Struktur)*

Kriterien	Deren Begründung bzw. Notwendigkeit
Einfachheit	Die Bedienung des Menüs muss einfach und in wenigen Schritten durchlaufbar sein. → **mit Fokus auf Anfänger aber minimale Behinderung für Profi**
Code Einfachheit	Der erzeugte Code soll nicht zu umfangreich sein, da er 1) schwer wartbar wird und 2) bei einem Ressourcenarmen System zu unerwünschten Nebenwirkungen führen kann, wie bspw. sichtbare Verzögerungen in der Ausführung.
Technische Machbarkeit	Hier beziehe ich mich auf die Voranmerkung. Zwar ist eine schrittweise Menüführung möglich, jedoch ist bei einem „Zurück" immer ein „exit" oder" return" im Spiel. Dies führt zur Problematik, dass man nicht ohne weiteres in die gleiche Funktion zurück kann. Es ist erheblicher Aufwand bezüglich Aufsplittung in viele einzelne Skripte notwendig.
Sicherungsaufwand	Sicherungen, welche auf Fehler prüfen sind immer notwendig, jedoch soll deren Aufwand auf das Nötigste reduziert werden, um den Code nicht zu stark aufzublasen.

Die in Tabelle 7 aufgeführten Kriterien umfassen die für die Nutzwertanalyse notwendigen Punkte. Der Punkt „Technische Machbarkeit" wird in der Folge etwas strenger bewertet, da er doch als ein technisches Handicap angesehen wird, welches den Aufwand erheblich erhöhen könnte.

2) Ermittlung des Gewichtungsfaktors für die einzelnen Kriterien (GWF):

*Tabelle 8: **Gewichtungsberechnung** für die Nutzwertanalyse zur Wahl eines Konzepts (Wizard Struktur)*

Kriterien / Kriterien	Einfachheit	Code Einfachheit	Technische Machbarkeit	Sicherungsaufwand	Summe der Bew-krit.	Gewich.-faktor
Einfachheit		50%	15%	50%	115%	19.33%
Code Einfachheit	50%		60%	50%	160%	26.89%
Technische Machbarkeit	70%	70%		50%	190%	31.93%
Sicherungsaufwand	40%	50%	40%		130%	21.85%
Umrechnungsfaktor = 5.95 →				**Summe**	595%	100%

Zur Kalkulation von Tabelle 8 sei auf Definition Seite 19 „Hinweis zu Tabelle 4" verwiesen.

3) Bestimmung des Zielerreichungsfaktors, basierend auf Einschätzung des Entwicklers:

Angewendeter Zielerreichungsfaktor bei nachfolgender Tabelle liegt im Range 1 – 5 (1 = sehr schlecht, 5 = sehr gut)

*Tabelle 9: **Zielerreichungsfaktor** für die Nutzwertanalyse zur Wahl eines Konzepts (Wizard Struktur)*

Bewertungskriterien	Konzept Wizard	Konzept Struktur
Einfachheit	4	3
Code Einfachheit	1	3
Technische Machbarkeit	2	4
Sicherungsaufwand	4	2

4) Eigentliche Ermittlung des Siegers:

Tabelle 10: Ermittlung des Siegers für die Nutzwertanalyse zur Wahl eines Konzepts (Wizard Struktur)

		Konzept Wizard		Konzept Struktur	
Bewertungskriterien	**GWF**	**ZEF**	**Ermitt. Wert**	**ZEF**	**Ermitt. Wert**
Einfachheit	19.33%	**4**	**77.32%**	**3**	**57.99%**
Code Einfachheit	26.89%	**1**	**26.89%**	**3**	**80.67%**
Technische Machbarkeit	31.93%	**2**	**63.86%**	**4**	**127.72%**
Sicherungsaufwand	21.85%	**4**	**87.40%**	**2**	**43.70%**
Gesamtnutzwert			**255.47%**		**310.08%**
Rangfolge					

Analyse des Siegers „Konzept Struktur“ im direkten Vergleich zu „Konzept Wizard“:

Wie aus Tabelle 10 deutlich zu entnehmen ist, hat ein strukturiertes Menü klare Vorteile in Bezug auf technische Machbarkeit sowie Code Einfachheit. Diese Variante hat zwar optisch nicht die besten Karten in Bezug auf die Nutzung seitens eines Anfängers. Sowie ist die Menge an zu verbauenden Sicherungen weit grösser als es bei einem Wizard der Fall wäre, jedoch wird der erzeugte Code weit übersichtlicher. Da die Integration von Sicherungen ein elementarer Punkt im Pflichtenheft ist, sowie die Tatsache, dass ein Wizard vielleicht etwas nachteilig bei Profianwendern angekommen wäre, ist diese Lösung doch die bessere. Dies gilt jedoch nur, wenn eine Sicherung, die bspw. auf eine fehlende IP Adresse hinweist, auch einen sinnvollen Kontext enthält, welcher einen Anfänger erstens nicht abschreckt und zweitens ihm eine klare Ansage mit klarem Verweis erteilt. So kann auch ein strukturiertes Menü mit einzelnen Punkten auch vorteilhaft für einen Anfänger wirken.

Das Konzept Wizard mag zwar optisch vorteilhaft erscheinen, sowie den Sicherungsaufwand zu einem gewissen Teil reduzieren, jedoch überwiegen die Nachteile bezüglich Entwicklungsaufwand, Code – Kompaktheit und Wartbarkeit des Codes klar. Das Konzept Wizard hätte bedeutet, PFDialogControl auf viele kleine Skripte aufzubrechen, um die zurück – Funktion überhaupt realisieren zu können.

7.2.2 Problem 2: Soll der DHCP – Range wählbar sein oder nicht?

Das zweite Problem, welches vor Beginn der Entwicklung einer Entscheidung bedarf, ist die Frage, welchen Spielraum der DHCP – Server bezüglich der Konfiguration des zu vergebenden IP – Bereiches haben soll. Betrachtet man heutige Standard Lösungen, so könnte man sich für ein vom Benutzer konfigurierbares Interface entscheiden. Doch dies widerspricht den beiden, in dieser Arbeit definierten Punkten der Einfachheit für Anfänger, sowie der Tatsache, dass eine möglichst schnelle Konfiguration aus Sicht eines Profis möglich sein soll. Um den bestmöglichen Nutzwert zu generieren, welcher helfen soll eine Entscheidung bezüglich frei wählbarer IP – Ranges, sowie Einfachheit und Schnelligkeit zu treffen, soll nachfolgende Nutzwertanalyse herangezogen werden um eine nüchterne und neutrale Lösung zu finden. Die hierfür in Frage kommenden Varianten wären:

1) Ein frei wählbarer IP –Range (nachfolgend nur als **Konzept Wahl** bezeichnet)

Dies stellt den heutigen Standard in praktisch jeder verfügbaren Lösung dar. Zusätzlich bietet es eine extrem hohe Skalierbarkeit der Konfiguration an, welche in komplexen Netzwerkkonfigurationen auch erforderlich ist.

Die Frage die sich an dieser Stelle aufdrängt; braucht es diesen hohen Grad an Skalierbarkeit? Denn schliesslich soll ein Anfänger ohne grosse Schwierigkeiten bei geringem Aufwand eine Konfiguration zustande bringen. Müsste ein unerfahrener Anwender nun eine Entscheidung bezüglich breite des IP – Bereiches treffen, obwohl er nicht versteht um was es sich dabei handelt. So sind aus dem Erfahrungsbereich seitens des Autors dieser Arbeit, Fehlkonfigurationen vorprogrammiert. Diese müssten mit viel Aufwand, durch Sicherungen die nicht nur auf Korrektheit der Eingabe, sondern auch durch die Logik der vorhergehenden Konfiguration und der Logik der Netzwerktopologie prüfen können, abgefangen werden. Dies würde zeittechnisch einen extrem hohen Aufwand bedeuten, welcher die Rahmenvereinbarungen dieser Arbeit mit grösster Wahrscheinlichkeit überschreiten würde.

2) Eine generische Autokonfiguration (nachfolgend nur als **Konzept Autoconf** bezeichnet)

Bei diesem Verfahren würde eine an die ersten drei Stellen der im Inside Interface eingegebenen IP Adresse, eine statische Autokonfiguration des IP – Bereiches stattfinden. Die ersten drei stellen deswegen, da so eine Autokonfiguration unabhängig vom gewählten Subnetzbereich möglich ist. Zwar tendiert die Skalierbarkeit dieser Lösung gegen Null, jedoch erlaubt sie eine schnelle und durch Reduzierung des Konfigurationsaufwandes auf Aktivieren und Deaktivieren des Dienstes, eine anfängerfreundlichen Konfiguration. Dies würde auch dem Grundprinzip von PFDialogControl und der Philosophie des Entwicklers, KISS (keep it small and simple) entsprechen. Der für den Benutzer bleibende Aufwand reduziert sich bei dieser Lösung auf zwei Punkte „wähle einen Domänennamen und entscheide dich ob Aktivieren oder eben nicht“. Bei dieser Lösung wäre der Aufwand an zu implementierenden Sicherungen minimal. Jedoch ergeben sich bei einer generischen Konfiguration immer Definitionsprobleme, da einem Benutzer, in diesem Fall einem Profi, Konfigurationen aufgezwungen werden, welche ihm vielleicht nicht passen. Daher ist ein gewisser Bewegungsspielraum notwendig um einen Profi nicht zu behindern oder gar zu verärgern. Mögliche einzuhaltende Punkte wären bspw.:

- Breite des IP –Adressbereichs: Diese sollte gross genug sein um viele Clients unterzubringen, jedoch auch einen freien Raum besitzen für statische IP’s. Möglich wäre bspw. xxx.xxx.xxx.20 – xxx.xxx.xxx.250, genug Spielraum für DHCP. Der Bereich .2 - .19 sowie .251 – 254 sollten einem Profi genügen. Grössere Ranges währen in einer virtualisierten Umgebung, sowie im realen (Hardware) gebrauch bspw. zuhause übertrieben.

- Die Lease Time der DHCP Clients: Da ein Anfänger kaum etwas mit einem solchen Wert anzufangen weiss (vor allem wenn der Wert in Sekunden anzugeben wäre), ist auch hier ein generischer Wert empfehlenswert. Dieser Zeitrahmen sollte so gewählt werden, dass er bei schwankenden Laufzeiten der Computer dennoch sicherstellt, dass die IP der Clients stets konstant bleiben, um so eine Art künstliche statische Konfiguration zu ermöglichen (geeignet für Anfänger, da so die Firewall Konfiguration auf IP Basis einfacher wird). Mögliche Lease Times könnten sein 2 – 4 Wochen.
- Einschränkung des Inside Default Gateways: Ein Anfänger könnte bei der Konfiguration der IP – Adresse des Inside Interfaces aus Unwissenheit eine IP – Adresse aus dem DHCP – Pool vergeben. Dies würde die Firewall zwar nicht grossartig beeinträchtigen, jedoch zu unschönen Fehlermeldungen bei den Clients führen, wenn sich die IP – Adressen überschneiden. Hier könnten Sicherungsmassnahmen helfen, welche die Auswahlmöglichkeiten der Inside IP – Adresse einschränken. Jedoch könnte dies Profis missfallen, welche sich so womöglich bevormundet fühlen könnten. Daher macht es mehr Sinn den etwas Unerfahrenen Anwender mittels einer Warnung, oder besser vielleicht mit einer Empfehlung auf seine Auswahlmöglichkeiten hinzuweisen. Dies würde dem Anfänger ein sichereres Gefühl geben und den Profi in seinem Stolz nicht verletzten.

Der grösste Nachteil dieser Lösung ist nicht im technischen, aber dafür im ästhetischen Bereich zu finden. Denn selbst ein unerfahrener Anwender wird heute mal einen Router eines Providers konfiguriert haben. So hat sich in der heutigen Zeit die freie Auswahl des IP –Ranges selbst bei ihm eingebürgert.

Nachfolgend eine nüchterne und neutrale Lösungsfindung mittels Nutzwertanalyse, um ein Frage zu klären die dem Entwickler dieser Arbeit einiges an Kopfzerbrechen verursacht hat:

1) **Die notwendigen Auswalkriterien:**

*Tabelle 11: **Auswahlkriterien** für die Nutzwertanalyse zur Wahl eines Konzepts (Wahl, Autoconf)*

Kriterien	Deren Begründung bzw. Notwendigkeit
Einfachheit	Die Bedienung des Menüs muss einfach und in wenigen Schritten durchlaufbar sein. → **mit Fokus auf Anfänger aber minimale Behinderung für Profi**
Code Einfachheit	Der erzeugte Code soll nicht zu umfangreich sein, da er 1) schwer wartbar wird und 2) bei einem Ressourcenarmen System zu unerwünschten Nebenwirkungen führen kann, wie bspw. sichtbare Verzögerungen in der Ausführung.
Sicherungsaufwand	Sicherungen, welche auf Fehler prüfen, sind immer notwendig, jedoch soll deren Aufwand auf das nötigste reduziert werden, um den Code nicht zu stark aufzublasen.
Eingebürgerte Sichtweise **Zählt durchgängig durch die komplette Nutzwertanalyse 2x**	Da das Konzept Autoconf eine Art Bevormundung darstellt, sowie von gängigen Standards abweicht, soll dieser Punkt in Bezug auf die heute übliche Methode nach Konzept Wahl, ein Gegengewicht darstellen. Denn im Grunde liefern beide Methoden das gleiche Ergebnis, womit der einzige Unterschied die optische Erscheinung des Interfaces ist. Da dies schwer zu erfassen ist, soll dieses Kriterium mittels doppelter Gewichtung eine Faire Betrachtung der beiden Ansätze ermöglichen → **Schwerpunkt auf Konzept Wahl**

Die in Tabelle 11 aufgeführten Kriterien stellen die für die Nutzwertanalyse notwendigen Auswahlkriterien dar. Dabei wurde dem Konzept von PFDialogControl gefolgt, welches eine leicht zu bedienende Software mit gut wartbarem und kompaktem Code zum Ziel hat. Speziell an dieser Stelle, ist das Kriterium „Eingebürgerte Sichtweise“ welcher doppelt zählt. Mit ihm soll eine fairere Beurteilung der optischen bzw. ästhetischen Gesichtspunkte der beiden Konzepte möglich werden, da diese Frage sonst schwer zu beantworten wäre und in reine „Geschmackssache“ münden würde.

2) Ermittlung des Gewichtungsfaktors für die einzelnen Kriterien (GWF):

*Tabelle 12: **Gewichtungsberechnung** für die Nutzwertanalyse zur Wahl eines Konzepts (Wahl, Autoconf)*

Kriterien / Kriterien	Einfachheit	Code Einfachheit	Sicherungsaufwand	Eingebürgerte Sichtweise	Summe der Bew-krit.	Gewich.-faktor
Einfachheit		50%	50%	70%	170%	32.69%
Code Einfachheit	50%		60%	70%	180%	34.61%
Sicherungsaufwand	40%	30%		20%	90%	17.31%
Eingebürgerte Sichtweise	40%	30%	10%		80%	15.38%
Umrechnungsfaktor = 5.2 →				**Summe**	520%	100%

Zur Kalkulation von Tabelle 12 sei auf Definition Seite 19 „Hinweis zu Tabelle 4“ verwiesen.

3) Bestimmung des Zielerreichungsfaktors, basierend auf Einschätzung seitens des Entwicklers:

Angewendeter Zielerreichungsfaktor bei nachfolgender Tabelle liegt im Range 1 – 5 (1 = sehr schlecht, 5 = sehr gut)

*Tabelle 13: **Zielerreichungsfaktor** für die Nutzwertanalyse zur Wahl eines Konzepts (Wahl, Autoconf)*

Bewertungskriterien	Konzept Wahl	Konzept Autoconf
Einfachheit	2	5
Code Einfachheit	2	4
Sicherungsaufwand	1	4
Eingebürgerte Sichtweise 2x	8 (2x 4 → max. Wert)	1

Hinweis zu Tabelle 13: Punkt „Eingebürgerte Sichtweisen“ erreichte den Wert 4. Durch Multiplikationsfaktor mal 2 wurde der Range überschreitende Wert 8 erzielt.

4) Eigentliche Ermittlung des Siegers:

*Tabelle 14: **Ermittlung des Siegers** für die Nutzwertanalyse zur Wahl eines Konzepts (Wahl, Autoconfig)*

		Konzept Wahl		Konzept Autoconf	
Bewertungskriterien	**GWF**	**ZEF**	**Ermitt. Wert**	**ZEF**	**Ermitt. Wert**
Einfachheit	32.69%	**2**	**65.38%**	**5**	**163.45%**
Code Einfachheit	34.61%	**2**	**69.22%**	**4**	**138.44%**
Sicherungsaufwand	17.31%	**1**	**17.31%**	**4**	**69.24%**
Eingebürgerte Sichtweise	15.38%	**8**	**123.04%**	**1**	**15.38%**
Gesamtnutzwert			**274.95%**		**386.51%**
Rangfolge					

Analyse des Siegers „Konzept Autoconfig“ im direkten Vergleich zu „Konzept Wahl“:

Wie im Ergebnis deutlich zu erkennen ist, herrscht ein deutlicher prozentualer Abstand zwischen den Kandidaten, trotz der grosszügigen Unterstützung seitens des Entwicklers mit Extrapunkten. Dieses Ergebnis resultiert aus der Tatsache, dass Autoconf implementierungstechnisch die einfachere, aber prinzipiell auch schnellere Lösung ist, welche sich stark in Richtung Anfänger und somit in Richtung simplere Lösung bewegt. Dies wiederspiegelt im Allgemeinen das gesamte Konzept von PFDialogControl. Hinzu kommt die Tatsache, dass durch die Extraktion der ersten drei Stellen der IP – Adresse des Inside Interfaces, die Korrektheit des Subnetzes immer gewährleistet sein wird, auch wenn ein Anfänger fälschlicherweise ein etwas zu grosses Subnetz eingibt. Dies setzt aber voraus, dass das durch den DHCP – Server verteilte Subnetz immer auf einem Default – Wert steht, z.B. 24 Bit. Diese „Sicherung“ war nie so beabsichtigt und fiel dem Entwickler erst in diesem Schritt auf. Was sicherlich noch stark zu Gunsten von Autoconf ausfiel, ist die extrem komplexe Implementierung des Konzepts „Wahl“. Da bei der Eingabe von Benutzerseite aus viele tiefgreifende Sicherungen nötig geworden wären, welche auch den logischen Aspekt der Netzwerkkonfiguration hätten berücksichtigen müssen. Wie etwa die Frage, befindet sich der eingegebene Bereich innerhalb des zulässigen Bereiches oder ist der Bereich breiter als es die Subnetzmaske des Inside Interface überhaupt zulässt. Solche Sicherungen sind mit Skript, beziehungsweise Programmiersprachen wie Perl oder Python ohne weiteres realisierbar. Doch bei Shell – Skripten ist hierfür ein extrem hoher und codetechnisch aufgeblasener Einsatz der beiden Kommandozeilen Tools awk und sed erforderlich. Darauf wurde aus Zeitgründen seitens der gesetzten Zeitfristen dieser Vordiplomarbeit und aus dem Wunsch nach Codeeinfachheit verzichtet.

Ein klarer Nachteil dieser Lösung wird sich mit grösster Wahrscheinlichkeit in negativem Feedback seitens professioneller Anwender, in möglichen produktiven Szenarien, nach Veröffentlichung dieser Arbeit unter der BSD – Lizenz zeigen. Da vermutet wird, dass eine Autokonfiguration als Bevormundung oder gar Behinderung aufgefasst werden wird. Doch aus Gründen der Schnelligkeit und Einfachheit, hofft der Entwickler dieser Arbeit, dass sein Produkt mit der Zeit ankommen wird.

7.3 Abschliessende Analyse der Hauptstudie

Die Hautstudie mag mit zwei Problemstellungen und je zwei Varianten auf den ersten Blick etwas mickrig erscheinen. Doch beinhalten diese beiden mittels Nutzwertanalyse beantworteten Fragen einen elementaren Wert. Denn durch die klar formulierte Aufgabenstellung, welche kaum Spielraum für Alternativmittel lässt, welche aber auch nicht nötig sind, bleiben eben nur Design Fragen bezüglich Aussehen (7.2.1) und ähnliche Fragen wie unter Punkt 7.2.2 übrig. Doch diese Fragen sind von elementarer Bedeutung, denn die Entscheidungen welche im Hintergrund getroffen werden sind meist nicht ersichtlich, sondern eben diese die das Aussehen betreffen. Da genau hier durch ein klar strukturiertes, schnell bedienbares und simples Menü dem Endanwender am meisten geholfen werden kann. Das Aussehen ist meist ein ausschlaggebender Punkt, welcher unter Anfängern die Wahl beeinflusst und unter Profianwender den Anreiz gibt eine Community um das Projekt zu bilden, oder was dem Entwickler dieser Lösung noch besser gefallen würde, die Bildung ähnlicher Projekte basierend auf der Inspiration durch PFDialogControl.

8 Realisation / Strukturplan vom Projekt PFDialogControl

8.1 Einleitung zum Aufbau der Realisation

In den nachfolgenden Abschnitten sollen die eingesetzte Hardware, mit einem groben Beschrieb der Konfiguration, sowie ein der Menüstruktur von PFDialogControl folgender, dokumentarischer Ablauf beschrieben werden. Hierbei wird versucht die Struktur sowie den funktionellen Aufbau, von PFDialogControl mittels Illustrationen vertieft darzustellen.

8.1.1 Information zur strukturellen Darstellung des Abschnitts betreffend PFDialogControl

Mit Schwerpunkt auf die Realisierung von PFDialogControl, soll versucht werden möglichst viele relevante und auch wirklich nötige Daten schriftlich zu erfassen. Daher folgt der softwareseitige Aufbau einer klaren Struktur, wo jeder Menü Punkt, der in sich selbst auch eine Funktion darstellt, nach folgendem Muster aufgebaut ist (sofern strukturell möglich):

- Kurze Umschreibung des Primär-, Sekundär-,…. Menüs
- Kurze Umschreibung des Menü Punktes und dessen Funktion
- Meist eine Illustration
- Ein Flowchart
- Eine Beschreibung des Ablaufs einer Funktion (Menü Punkt) mit Erklärungen zu den Fragen: wie, warum, was geschieht und falls erforderlich Verweise auf Code Passagen oder deren Kommentare
- Falls nötig, ein paar Worte zu aufgetauchten Schwierigkeiten

In jeder Erklärung sind die Suchstrings enthalten. Hierbei handelt es sich um Tags, welche helfen sollen eine „function“ im Code wiederzufinden. Die Tags folgen dabei folgender Syntax:

#---<*Name des Menü Punktes*>--- bspw. **#---NAT Aktivieren---**

So kann die zu einem Menü Punkt gehörende Funktion mittels Suchfunktion des Editors schnell ermittelt werden.

Abweichungen zu oben genannten Muster in grösserem Umfang sollten nicht vorkommen, falls doch, so werden diese in der einleitenden Kurzbeschreibung des Menü Punktes speziell erwähnt. Kleinere Abweichungen die aus Gründen des Umfangs vom Menü Punkt oder des Seiten Layouts resultieren werden vernachlässigt und nicht weiter beschrieben.

8.2 Aufbau und Struktur der Sekundärelemente

Nachfolgend wird grob der Aufbau der eingesetzten Hardware und Software umschrieben, welcher als Unterbau für die Entwicklung und das Testing von PFDialogControl verwendet wird.

8.2.1 Die eingesetzte Hardware

Für die Realisierung und mögliche Präsentierung der Arbeit wurde ein Server des Typs IBM 3650 M2 aus einem Versandhaus für Occasion – Hardware erworben.

Abbildung 2: Eingesetzte Hardware (Server). Typ IBM x3650 M2 -- Quelle: https://www.techdata.com/reseller/secure/product_info/ibm/x3650%20M2.aspx

Dieser Server soll die nötige Mobilität bei möglichen Präsentationen liefern, ebenfalls erfüllt er die nötigen Voraussetzungen bezüglich hardwareseitiger Virtualisierung. Die Wahl fiel auf ihn, da er EPT (auch bekannt als Nested Paging) unterstützt, welches seitens SmartOS für den Betrieb erforderlich ist. Die Kosten belaufen sich auf **CHF 180.00** für den Server und ca. **CHF 800.00** für 8x Toshiba 2.5 Zoll Desktop Festplatten, welche für einen sicheren (redundanten) Betrieb sorgen sollen. Da sein technischer und optischer Zustand überraschend gut ist wird dieser Server nach der Vordiplomarbeit als Produktivsystem in den bestehenden Virtualisierungscluster integriert. An dieser Stelle noch eine kleine Schleichwerbung. Der Server wurde bei http://benno-shop.ch gekauft, wo ab und zu ein günstiges Occasion Schnäppchen vorhanden ist.

8.2.2 Der softwareseitige Aufbau (Hypervisor)

Für den Aufbau der Entwicklungs- und Testumgebungen wurde mit SmartOS virtualisiert. Um einen möglichst authentischen Aufbau zu gewährleisten wurde ein virtueller Switch (vswitch0) kreiert, welcher sich nur im Kernelspace befindet und somit keinen Link zu einem realen Netzwerkanschluss besitzt. An vswitch0 können somit alle virtuellen Maschinen angeschlossen werden, welche zu Testzwecken (Packet Übermittlung) nötig sind. Hierzu wurden meist fertige Abbilder der Linux Distribution Debain ab Version 7 verwendet. Ebenfalls an vswitch0 angeschlossen ist eine FreeBSD 10 oder 10.1 Instanz, welche in direkter Kommunikation mit den Testmaschinen steht. Die FreeBSD Instanz besitzt aber ein zweites Interface, welches einen direkten Link zu einem realen Netzwerk Controller besitzt, um das Routing ins Internet zu ermöglichen. Eine Zone (Solaris Ressourcencontainer) befindet sich parallel zur FreeBSD Instanz, um die Firewall Funktion „Portweiterleitung“ zu testen. Siehe nachfolgendes Schema:

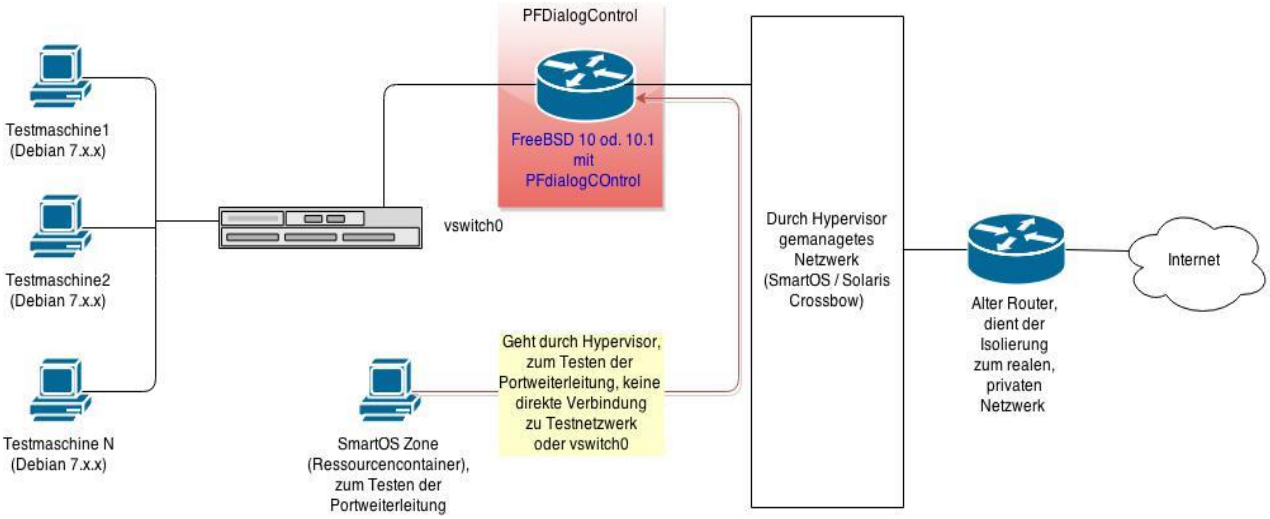

Abbildung 3: Netzwerkschema der Testumgebung auf SmartOS

8.2.3 Sonstige Peripherie

Neben den Netzwerkkabeln die für die Verkabelung notwendig sind, welche auf Ethernet 1000BASE-T basiert, wurde der Isolation gegenüber dem realen und produktiven Netzwerk ein alter WLAN Router dazu geschaltet. Besagter Router des Typs D – Link dir-615, welcher nach langem Suchen nur mit Mühe und Not gefunden wurde, dient neben der Netzwerkisolation, auch der Portabilität um die gesamte Entwicklungs- und Testumgebung zu Präsentationszwecken mobil zu halten.

8.3 Realisierung von PFDialogControl Version 1.0.3

Nachfolgend die strukturierte Dokumentation der Funktionen und derer Entwicklungsschritte. Der Aufbau der Dokumentation richtet sich nach der Menü Struktur von PFDialogControl und dem Top – Down Verfahren.

8.3.1 Das Hauptmenü

Filename → pfdctl
Position in Filesystem → /usr/bin
Function Name → Keiner, Direktausführung
Rechte → Default: Besitzer root, Gruppe wheel, Lesend – Schreibend – Ausführend (rwx - rwx - ---)

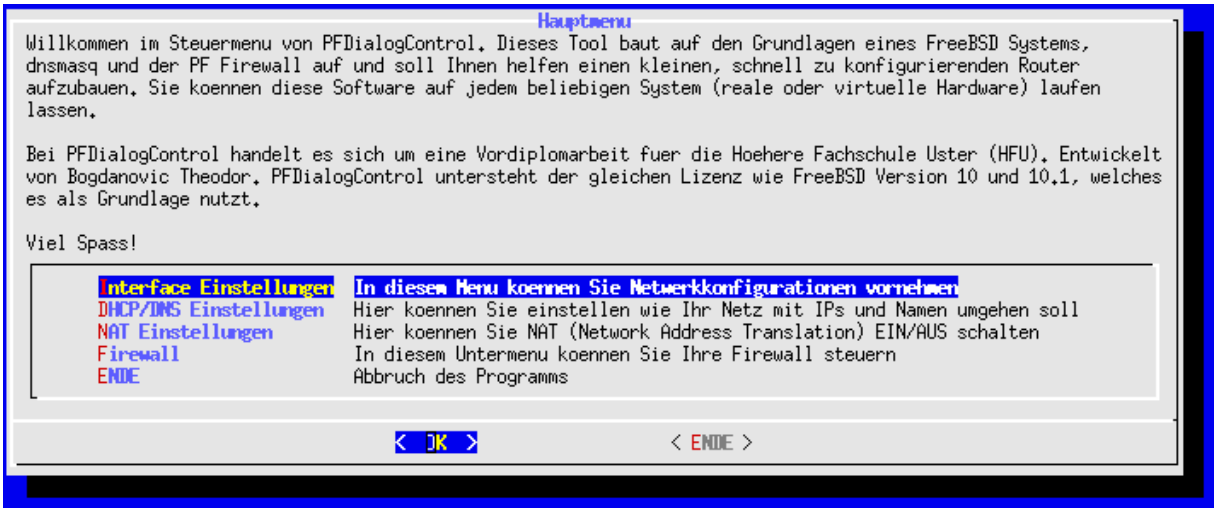

Abbildung 4: Hauptmenü von PFDialogControl

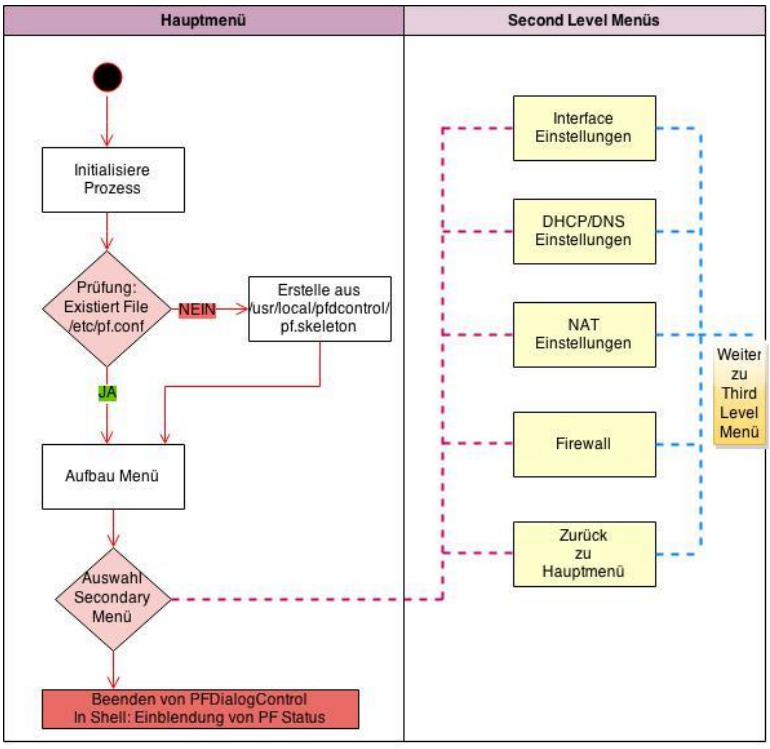

Abbildung 5: Flowchart, PFDialog Hauptmenü

Der sequenzielle Aufbau von pfdctl ist simpel, er umfasst wenige Zeilen die hauptsächlich nur das Menü aufbauen. Dies hat mehrere Vorteile, doch der wichtigste ist die Isolierung der Functions aus Sicherheitsgründen. Denn wie später ersichtlich wird, sind in den restlichen Files nur reine Funktionen gekapselt, was eine Ausführung direkt aus den Files verhindert. Somit ist pfdctl die einzige Möglichkeit den Funktionsumfang von PFDialogControl zu nutzen. An den Rechten von pfdctl ist nichts verändert worden, sie sind so belassen worden wie sie FreeBSD per Default vorsieht. Dies ist aus Sicht des Entwicklers absolut akzeptabel bezüglich Sicherheit

und soll aus Kompatibilitätsgründen auch so bleiben. Zusätzlich schafft es einen kleinen Anreiz für Profianwender, dies als zu unsicher einzustufen und ein „paranoides“ Hardening durchzuführen, wo bspw. die Rechte weiter auf Root eingeschränkt werden. Dies soll helfen eine mögliche Community schneller aufzubauen. Bis auf die Prüfung auf das Vorhandensein der PF Firewall Konfigurationsdatei pf.conf, gibt es in diesem Skript nichts spezielles zu dokumentieren, es dient einzig dem Zweck PFDialogControl zu starten und die Ausführung der gekapselten Functions zu ermöglichen.

8.3.2 Die Secondary Menüs

Die Secondary Menüs werden aus diversen Main Functions aus libpfdctl gestartet und umfassen meist eine strukturierte Menüführung welche neben diversen Konfigurationsmöglichkeiten meist die Punkte zum Aktivieren und Deaktivieren umfasst. Diese Punkte sollen nachfolgen, dem Menüdesign von PFDialogControl aufgeführt werden.

8.3.2.1 Interface Einstellungen

Filename → libpfdctl
Position in Filesystem → /usr/lib
Function Namen → menu_network
Rechte → Default: Besitzer root, Gruppe wheel, Lesend – Schreibend (rw- – rw- - ---)

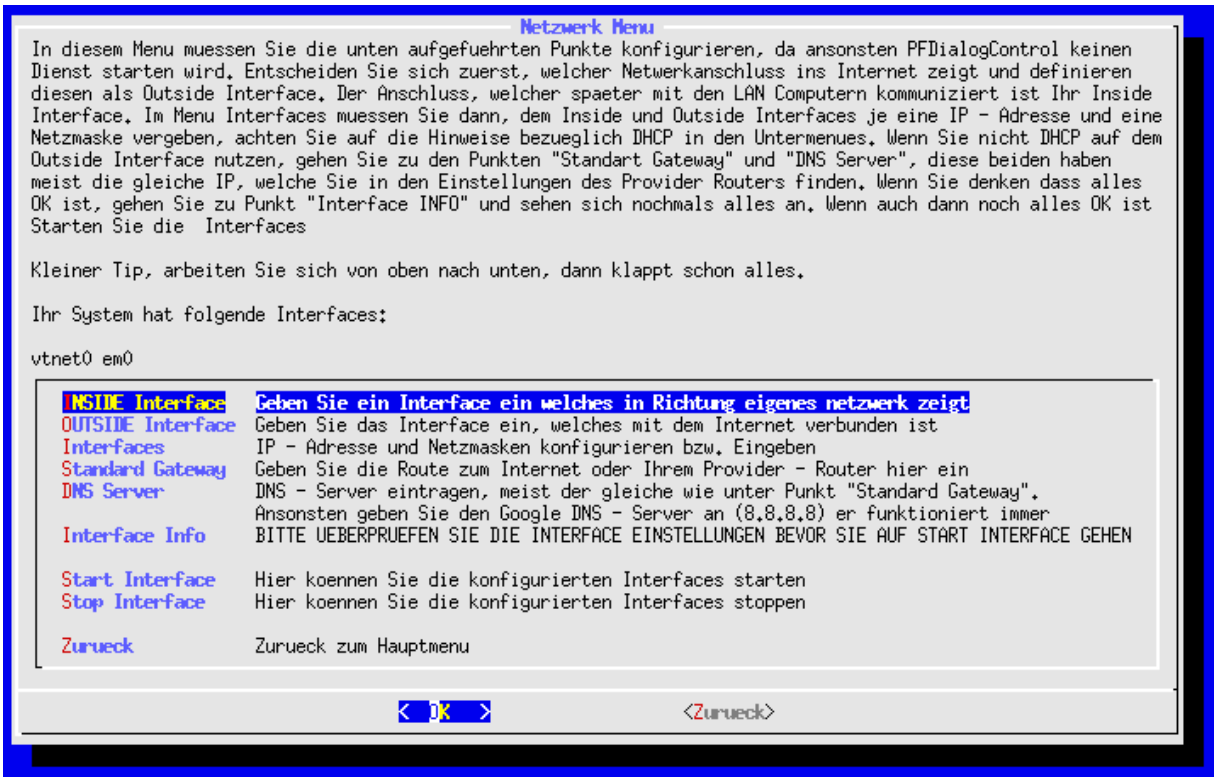

Abbildung 6: Secondary Menü: Interface Einstellungen

Dieses Secondary Menü umfasst sämtliche relevanten Einstellungen bezüglich der Netzwerkkonfiguration des FreeBSD Systems und soll so ein Eingreifen des Users in die Config Files wie bspw. /etc/rc.conf verhindern. An mehreren Stellen des Menüs, sowie im Quick Guide wird darauf hingewiesen an dieser Stelle mit der Konfiguration im Top – Down Verfahren zu beginnen. Denn nur wenn hier sämtliche Einstellungen vorhanden und korrekt sind, lässt sich ein Dienst

überhaupt starten. Der Ansatz des Top – Down Verfahrens zur Konfiguration ist im Grunde nicht notwendig, wird aber so forciert um Anfängern ein sichereres Gefühl zu vermitteln. Beginnen wir nun mit den einzelnen Menü Punkten:

Die Menüstruktur hat folgende Verknüpfungen zu den Skript Functions in libpfdctl:

- **INSIDE Interface** → run_inside_netconfig
- **OUTSIDE Interface** → run_outside_netconfig
- **Interfaces** → warning_1, input_netconfig_name, input_netconfig_ip, input_netconfig_netmask
- **Standard Gateway** → input_netconfig_gway
- **DNS Server** → input_netconfig_dns
- **Interface INFO** → info_netconfig
- **Start Interface** → run_netconfig
- **Stop Interface** → run_netconfig_stop
- **Zurück** → Keine Funktion, ist direct im Dialog Menü integriert. Zusätzlich existiert eine if – Anweisung, welche den Button die gleichen „Zurück" Fähigkeiten verleiht.

→ INSIDE Interface:

Beim Aufruf dieses Menü Punktes erfolgt die Aufforderung eine Netzwerkschnittstelle einzugeben, die in das File inside_iface unter /usr/local/etc/pfdcontrol geschrieben wird.

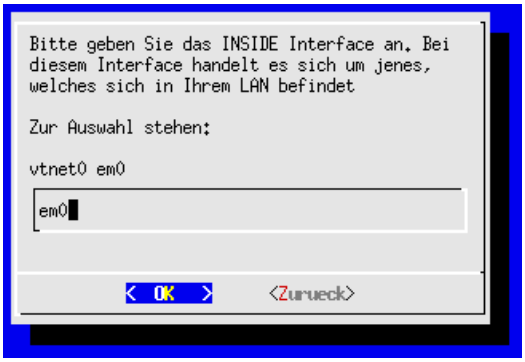

Abbildung 7: Secondary Menü: Inside Interface Eingabebox

Mittels der Function chk_iface_network, welche bei jedem Aufruf auf neue Schnittstellen prüft, wird eine Liste mit Möglichkeiten zur Auswahl, gleich oberhalb des Eingabefeldes, gestellt. Wählt man ein Interface aus, so prüft eine Routine in run_inside_netconfig ob der eingegebene Interfacename schon für das Outside Interface genutzt wird. Ist alles in Ordnung, erfolgt eine Bestätigung mittels Dialog MessageBox welche mit OK bestätigt werden muss.

Gleich hier zu Beginn der Realisation soll noch auf eine Implementierung hingewiesen werden. Bei Eingabeboxen wie in Abbildung 7, wird zu Beginn der Function der Inhalt einer der Konfigurationsdateien in /usr/local/etc/pfdcontrol in eine Variable geschrieben und dort eingefangen. Die Variable wird in das Eingabefeld geplottet, um erstens dem User mittzuteilen was in der aktuellen Konfiguration gerade gilt, aber um zweitens auch gleich eine Sicherung zu implementieren damit bei Benutzung der Zurück- oder Abbruchtaste der Inhalt in der Konfigurationsdatei erhalten bleibt. Dies führt auf die Default Implementierung von Dialog zurück, welches bereits beim Öffnen der Eingabebox den Inhalt der Datei überschreibt. Ohne diese Sicherung, würde es zum Löschen des Inhaltes im Config File kommen. Diese Sicherung wird überall dort verwendet, wo eine Eingabebox erscheint und wird im weiteren Verlauf der Realisation nicht mehr spezifisch erwähnt.

Das hier eingegebene Interface wird im Config File gespeichert, hat aber noch keinen Einfluss auf die Funktion von PFDialogControl. Es dient nur der Definition, welches das Inside Interface ist. Die Übernahme der Funktionalität wird erst mit „Start Interface" initialisiert.

➔ OUTSIDE Interface:

Der Menü Punkt OUTSIDE Interface ist identisch mit dem Punkt INSIDE Interface. Lediglich aus optischen Gründen, wurde hier entschieden die Eingabeboxen zu splitten. Der einzige Unterschied besteht im Ziel File der zu speichernden Eingabe, diese wird im File outide_iface in /usr/local/etc/pfdcontrol hinterlegt.

An dieser Stelle noch die Definition der Speicherinhalte der bereits erwähnten und noch folgenden Konfigurationsdateien. In den Files befinden sich nur Klartextinformationen der Eingabe der jeweiligen Eingabe- bzw. Formularboxen, ohne irgendwelche Formatierungstags. Dies wurde aus praktischen Gründen gewählt, da so die Prüfung auf Vorhandensein oder enthaltener Inhalt einfacher wird.

➔ Interfaces:

Dieser Menü Punkt erlaubt das Setzen der IP – Adresse und der Subnetzmaske für ein Interface. Bei Auswahl dieses Punktes erscheint immer ein Warnfenster (Function warning_1), welches den User daran erinnert beim Wechseln von Konfigurationseinstellungen wie Interface, IP oder Netzmaske, die Dienste DHCP/DNS und NAT neu zu starten um die Einstellungen zu übernehmen. Zugleich mahnt es den Anfänger hier wahllos Änderungen vorzunehmen, indem es ihn darauf hinweist, dass die Firewall Einstellungen statisch sind und somit angepasst werden müssen.

Die Eingabebox listet alle verfügbaren Interfaces auf und differenziert welche als Inside und Outside vorgewählt wurden. Durch Eingabe eines Interfaces wird in einem ersten Schritt, mittels des externen Skripts libchkin12 welches sich in /usr/lib befindet, geprüft ob der Interfacename korrekt eingegeben wurde. Hierbei wird der Eingabename in eine Variable gespeichert und an libchkin12 übergeben, welches eine statische Prüfung des Namens vornimmt. Libchkin12 ist wegen seiner statischen Form auf die Prüfung von 12 Interface – Endungen (bei Unix Systemen jeweils eine Zahl von 0 - N) beschränkt. Diese Zahl sah der Entwickler als eine logische Grenze des Anwendungszweckes von PFDialogControl an.

Abbildung 8: Secondary Menü: Interfaces Eingabebox

Hat man sich für ein Interface entschieden, so entstehen je nach Auswahl unterschiedliche Situationen. Erstens, wenn man das Inside- oder Outside Interface gewählt hat, erfolgen folgende beiden Hinweismöglichkeiten:

Abbildung 9: Secondary Menü: Interfaces, Hinweis letzte Ziffer

Dieser Hinweis ist für Anfänger wie Profis und soll sie darauf hinweisen, dass der DHCP Range per Autokonfiguration gemanagt wird und somit nur ein eingeschränkter Bereich für die Inside Konfiguration zur Verfügung steht.

Dieser Hinweis soll zeigen, dass auch eine DHCP Konfiguration des Outside Interfaces möglich ist. Dieser Hinweis erscheint nur beim Outside Interface. Da bei aktivem DHCP Server nur eine statische Konfiguration verwendet werden kann, soll dieser Dialog erst gar nicht beim Inside Interface auftauchen, da er einen Anfänger in höchsten Masse verwirren könnte.

Abbildung 10: Secondary Menü: Interfaces, Hinweis DHCP

Eine if – Anweisung innerhalb der zu Menü „Interfaces“ gehörenden Function input_netconfig_name, führt diese Prüfungen in Bezug auf den Inhalt der beiden Files inside_iface und outside_iface zu der Eingabe durch. Hieraus resultiert der zweite Fall, denn wenn keine der genannten Sicherungen gegriffen hat, wird auch kein Hinweis erzeugt. Somit erlaubt PFDialogControl die Konfiguration weiterer Interfaces, welche vielleicht mit den Grundfunktionen von PFDialogControl nichts zu tun haben.

Sind nun alle Eingaben gemacht und alle gewünschten Konfigurationen getätigt, so wird die Konfiguration in folgende Files, mit spezifischer Syntax geschrieben:

ip.<Interfacename> → bspw. ip.em0 und

netmask.<Interfacename> → bspw. netmask.em0

Diese Files sind bei der Installation von PFDialogControl nicht erzeugt worden. Sie werden erst mit ausführen des Menü Punktes Interfaces generiert und gespeichert. Auch hier greifen viele mittels if – Anweisung implementierte Sicherungen, welche auf korrekte Eingaben prüfen. Der Abschluss wird mit einem kleinen Infofenster, wo IP und Netzmaske nochmals aufgelistet sind bestätigt. Aufgrund des Fehlverhaltens eines hier nicht weiter namentlich genannten Beat – Testers, findet an dieser Stelle mittels if – Anweisung eine Prüfung auf Eingabe von IP – Adressen im gleichen Subnetz, in Bezug auf Inside und Outside Interface statt. Diese Aktion ist nicht zulässig. Nebenstehendes Flowchart soll helfen den Vorgang besser nachzuvollziehen. Anbei eine kleine Anmerkung: In jedem Schritt lässt sich mittels „Zurück“ der gesamte Prozess stoppen, ohne irgendwelche Änderungen zu generieren. Das Zurück führt einen aber immer zum Secondary Menü, da Dialog kein geeignetes Werkzeug für Vor- und Zurückaktionen ist, wie man es vielleicht bei GUI’s kennt.

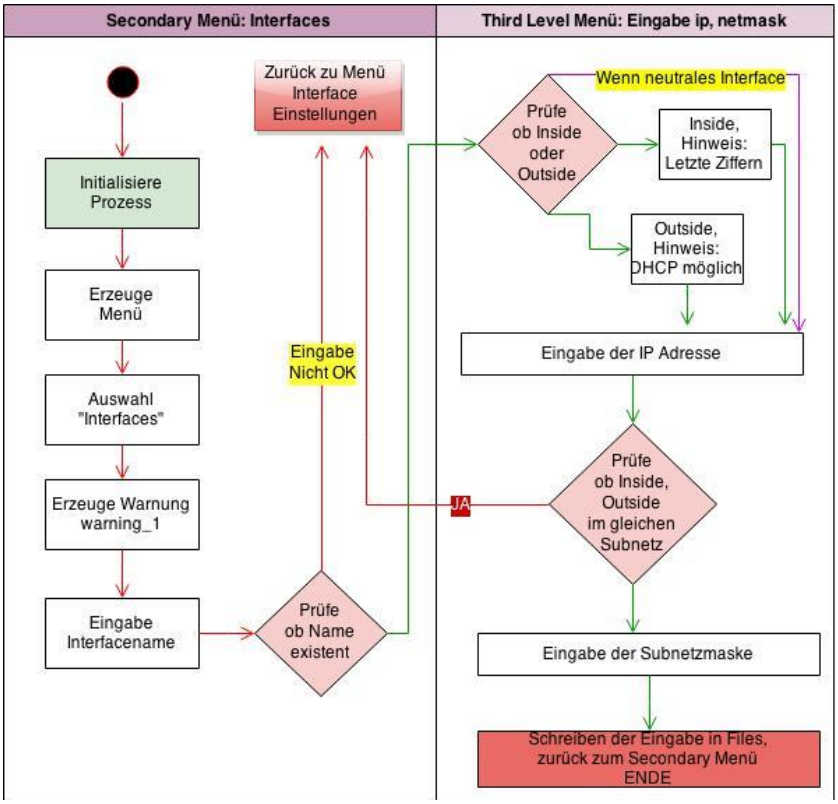

Abbildung 11:Flowchart: Secondary Menü, Interfaces

➔ Standard Gateway:

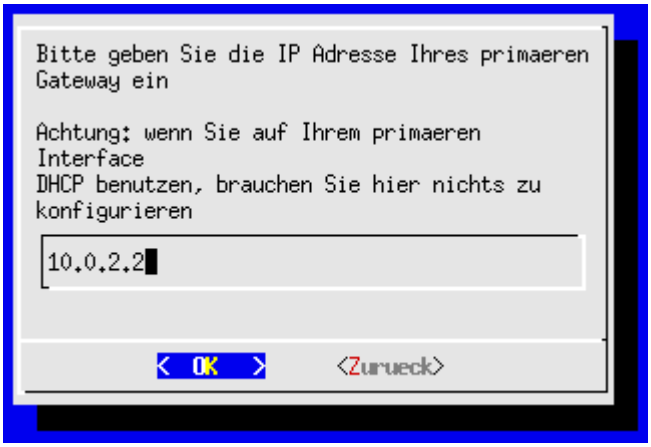

Abbildung 13: Secondary Menü: Standard Gateway Eingabe

Auch der Punkt „Standard Gateway“ besitzt die gleichen Sicherungen bezüglich Prüfung auf leere Felder wie die restlichen Eingabeboxen. Ebenfalls wird hier der aus dem Konfigurationsfile gespeicherte Wert in eine Variable gespeichert, um im Falle eines „Zurück“ nicht die Config Datei mit einem Leer – Wert zu überschreiben. Die Funktion dieser Function ist simpel, man gibt die Adresse ein und diese wird im File gateway.addr unter /usr/local/etc/pfdcontrol gespeichert. Jedoch greift an dieser Stelle noch eine if – Anweisung welche prüft ob das Outside Interface, welches in /usr/local/etc/pfdcontrol/ip.<Outside Interface> gespeichert ist, den Wert „DHCP“ oder „dhcp“ enthält. Dazu wird ein Suchpfad generiert, welcher nach der Datei ip.<In Variable Gespeicherter Name> genutzt wird. Diese Technik wird allgemein für die Ermittlung von Interfaceabhängigen Aktionen genutzt, was auch der ausschlaggebende Punkt für die Verwendung von einzelnen Dateien zur Speicherung der Konfigurationsdateien war.

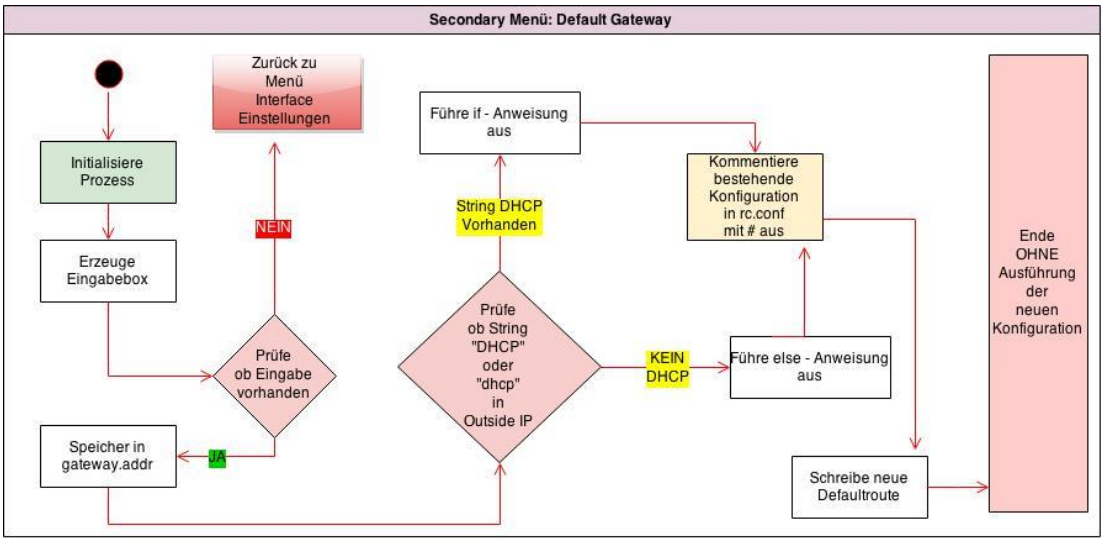

Abbildung 12: Flowchart: Secondary Menü, Gateway eingeben

Wurde nun ein String „DHCP“ oder „dhcp“ ermittelt, so gilt folgende Aktionen im if – Codeblock:

- Mittels des Befehls sed wird im Hauptkonfigurationsfile von FreeBSD (/etc/rc.conf) nach dem String „defaultrouter“ gesucht. Wurde er gefunden, so wird er mittels „#“ auskommentiert und mit dem PFDialogControl Deafault – Tag „--DECONFI--durch--PFDiaolControl“ ergänzt. Da bei DHCP hier keine Einträge für Defaultgateway nötig sind.

Wurde jedoch festgestellt, dass kein DHCP verwendet wir, so gelten folgende Schritte aus dem „else“ Codeblock der if – Anweisung.

- Der oben beschriebene Teil mit suchen und auskommentieren bleibt sich gleich, jedoch wird zusätzlich mittels des Befehls echo ein neuer Konfigurationsstring erstellt mit dem aktuellen Defaultgateway. Dieser sieht wie folgt aus:

  → defaultrouter=“<Variable die gateway.addr ausliest> “

  Dies wird in rc.conf geschrieben, hat aber noch keine Auswirkungen auf das Netzwerkverhalten von FreeBSD.

Die Tatsache, dass an dieser Stelle Konfigurationen geschrieben, aber nicht ausgeführt werden, liegt an einer Schutzüberlegung. Anfänger könnten hier aus Versehen eine Änderung vornehmen welche nicht funktionsfähig ist, sie es aber zu spät merken, was zur Folge hätte, dass ein Kommunikationsunterbruch stattfindet. Es wird im Allgemeinen davon ausgegangen, dass die bestehende Gateway Konfiguration korrekt ist, weshalb Änderungen eher seltener Natur sind und so vom Anfänger allfällige Fehlkonfigurationen abgefangen werden können. Diese Änderungen werden nochmals unter Menü Punkt „Start Interface“ überschrieben und ausgeführt. Nun könnte man meinen, dass die Schreibaktion in rc.conf sinnlos war, doch im Falle eines „Anfänger Neustarts“, also Strom weg, kann FreeBSD ohne Eingriffe korrekt booten obwohl die Konfiguration nicht übernommen wurde.

Wie meistens üblich bei PFDialogControl erfolgt auch hier am Schluss eine Informierung des Users mittels Infofenster.

→ DNS Server:

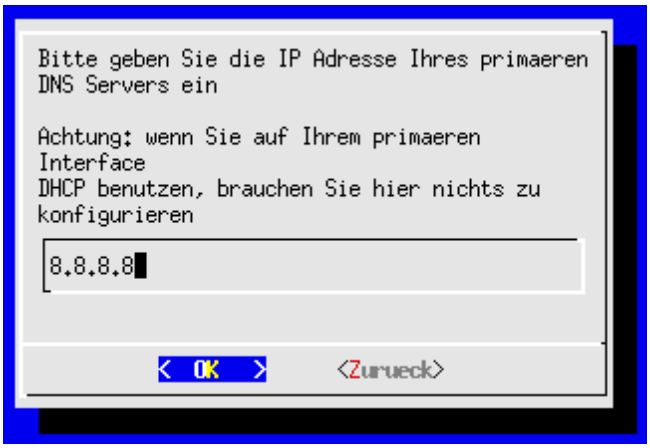

Abbildung 14: Secondary Menü: DNS Server, Eingabebox

Dieser Menü Punkt entspricht designtechnisch komplett dem Punkt „Gateway“. Da er dasselbe Arbeitsprinzip aufweist und danach entwickelt wurde. So wird an dieser Stelle auf eine genauere Erklärung und Illustration mittels Flowchart verzichtet und an den vorhergehenden Punkt „Standard Gateway“ verwiesen.

Was genauer betrachtet wird sind die Diskrepanzen in Bezug auf Punkt „Standard Gateway“. So generiert diese Funktion einen Konfigurationsstring welcher folgender Notation folgt:

→ nameserver=“<Variable die dns.addr ausliest> “

Zusätzlich wird hier der besseren Lesbarkeit, kein Deafault – Tag „—DECONFI--durch--PFDiaolControl“ erzeugt, sondern gleich der ganze zu ändernde Eintrag entfernt. Das Änderungs – Ziel ist in diesem Fall die Datei resolver.conf welche unter /etc gespeichert ist. Hier wird die Änderung aber auf Anhieb übernommen, da dies die übliche Implementierung bei Unix Systemen ist. Sie wird aber dennoch der Sicherheit halber unter Punkt „Start Interface“ wiederholt um 100% Konfigurationssicherheit zu gewährleisten.

➔ Interface INFO:

Wie zu erkennen ist, besteht die Interface INFO lediglich aus einer Eingabebox und der daraus resultierenden Ausgabe. Auch hier wird die Eingabe auf Leer – Eingaben und Existenz des Interfaces geprüft. Ist das eingegebene Interface als korrekt identifiziert worden, so wird es in einer Variable gespeichert. Dieser Variableninhalt wird später dazu verwendet um die Infofelder IP – Adresse und Netzmaske auszulesen und ebenfalls in eine neue Variable zu speichern. Hier folgt die Suche nach dem Muster ip.<Interfacename> bzw. netmask.<Interfacename>, welche aus der Eingabe herausgelesen werden kann. Auch hier sein nochmals auf die Einfachheit und Effektivität der Einzelabspeicherung in einzelne Dateien erwähnt. Die restlichen Variablen sind statisch und können auf simple Art und Weise mittels des Befehls „cat“ aus den Config Files herausgelesen werden. Falls nötig, kann noch eine einfache Ausgabeformatierung mittels des Befehls „cut“ generiert werden.

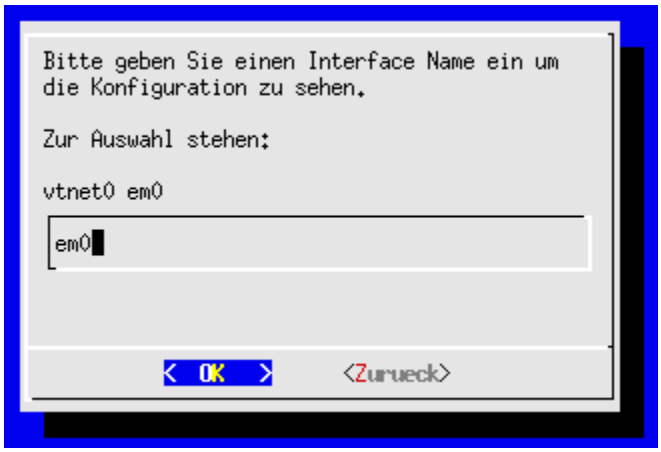

Abbildung 16: Secondary Menü: Interface INFO, Eingabebox

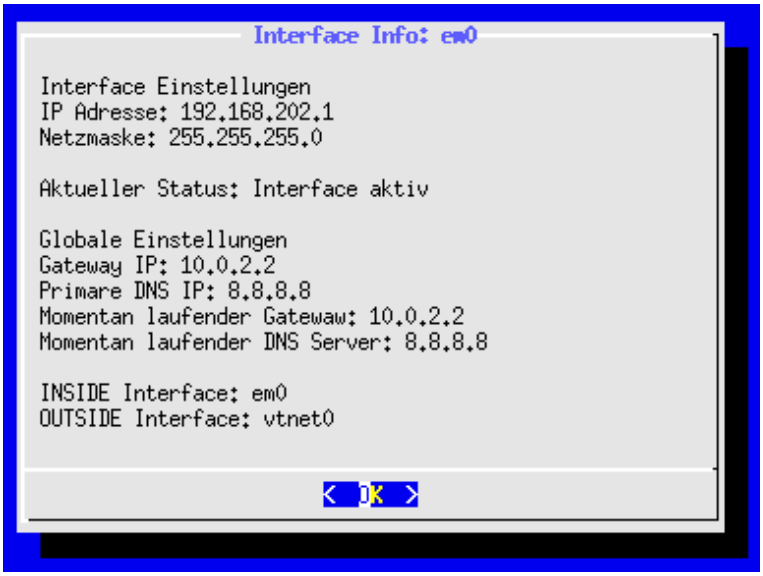

Abbildung 15: Secondary Menü: Interface INFO, Output Infos

Die beiden Punkte „Momentan laufender Gateway bzw. NDS Server“ sind hier etwas spezieller. Der DNS Server wird auf simple Art und Weise aus /etc/resolv.conf herausgelesen. Gateway hingegen wird aus einer Kombination der Befehle netstat, grep und cut (genauer netstat –rn | grep default | cut –d „ <1 Leerschlag>“ -f13-20) herausgelesen. So ist es möglich die beiden Konfigurationen unabhängig von den in Konfigurationsfiles gespeicherten Werten zu ermitteln und gegenüberzustellen. Dies hat reine Debugging Überlegungen und soll Fehler während möglichen Wechseln zwischen statischen und dynamischen (DHCP) Konfigurationen sichtbarmachen, wo ein Reboot helfen könnte. In einigen Vor – Tests während der Entwicklung dieser Funktion wurde festgestellt, dass vor allem ein Reboot die DHCP Einstellungen sauber übernimmt, obwohl eine Funktion in run_netconfig eigentlich mittels dhclient <Interfacename> dies erledigen sollte. Manchmal funktionierte der dhclient Befehl manchmal aber auch nicht. Dieses Verhalten wurde auf eine, zu Testzwecken schnell eingerichtetem zweiten Hypervisor verifiziert, was die Vermutung zulässt, dass dhclient unter FreeBSD nur ungerne und mit langen Ausführzeiten einen Wechsel von statisch zu dynamisch unter Last zulässt.

Interface Info ist ein nützliches Debugging Tool in PFDialogControl und liefert immer Aufschluss über mögliche Fehlkonfigurationen. Dies ist der Grund, weswegen es auch speziell hervorgehoben und isoliert zu den anderen Punkten im Menü ist.

→ Start Interface:

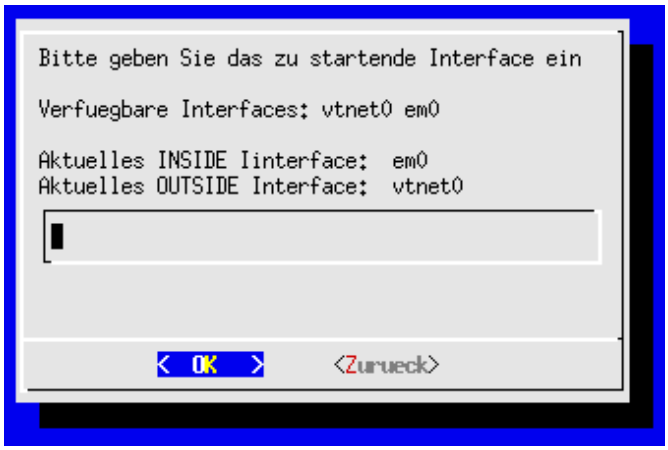

Abbildung 17: Secondary Menü: Start Interfae, Eingabebox

Neben den üblichen Prüfungen auf Leer – Eingaben und Vorhandensein des eingegebenen Interfaces, wird an dieser Stelle auch noch eine Konfigurationsprüfung durchgeführt. Diese Konfigurationsprüfung nutzt hauptsächlich die Befehle „cat“ und „cut“ um diverse Informationen aus den Konfigurationsdateien in /usr/local/etc/pfdcontrol zu verifizieren. Hierbei wird das eingegebene Interface wieder in eine Variable gespeichert, um als Suffix für die Identifizierung von ip.<Interfacename> und netmask.<Interfacename> zu dienen. Dazu wird in einem Umfangreichen if – elif – Block auf folgende Punkte geprüft:

Abbildung 18: Secondary Menü: Start Interface Info Output

- Ist Inside Interface (inside_iface) vorhanden und hat es einen String Wert höher als „0“
- Ist Outside Interface (outside_iface) vorhanden und hat es einen String Wert höher als „0“
- Ist die IP –Adresse des Inside Interface (ip.<cat von inside_iface>) gesetzt, also einen String Wert höher „0“
- Ist die IP –Adresse des Outside Interface (ip.<cat von outside_iface>) gesetzt, also einen String Wert höher „0“
- Ist die Netmaske des Inside Interface (netmask.<cat von inside_iface>) gesetzt, also einen String Wert höher „0“
- Ist die Netmaske des Outside Interface (netmask.<cat von outside_iface>) gesetzt, also einen String Wert höher „0“

Erst wenn diese Fragen durch diverse if – Anweisungen (Negativwert Bestimmung mit „oder“ Verknüpfungen) positiv durchgelaufen sind, kann im restlichen Skripttext fortgefahren werden. Ansonsten erfolgt eine Fehlermeldung mittels Dialog Massagebox, welche auf den Fehler hinweist und den „Weg“ zum Menü Punkt nennt wo noch Konfigurationen nötig sind. Ein Beispiel für einen solchen „Wegbeschrieb“ könnte sein „Ihr Internes Interface ist nicht gesetzt. Sie können dies unter → Interface Einstellungen → INSIDE Interface eingeben.“ Das Ergebnis eines korrekten Starts ist in Abbildung 17 ersichtlich.

Die oben beschriebene Prüfung ist ein erster Schritt, welcher aber bei nichtvorhandensein eines Prüfpunktes zum Abbruch von „Start Interface“ führt. Sollte aber alles in Ordnung sein, so wird mit den nötigen Instruktionen innerhalb der Function fortgefahren, welche die Konfiguration durchführt.

Hierbei kommt ein grosser Satz an anzuwendenden Befehlen ins Spiel, weswegen nachfolgend eine nummerierte Auflistung der Schritte genutzt wird, um eine bessere Übersicht zu ermöglichen:

1. Die System Anweisung „ifconfig <*Variable mit Interfacename*> netmask <*Subnetzmaske des Interface*> up“ wird abgeschickt um das Interface online zu bringen. Variablen welche DHCP enthalten, werden auch mitgeschickt. Dies führt zu einer Fehlermeldung, doch diese ist nicht ersichtlich und damit auch nicht relevant.
2. Die Standard Route wird gesetzt mit „route add default <*cat aus Config File gateway.addr*>“. Auch dies verursacht bei DHCP eine Fehlermeldung welche aber nicht ersichtlich ist. Diese beiden DHCP spezifischen Fehlkonfigurationen verursachen keinen Fehler, daher können sie so angewendet werden. Statische Konfigurationen werden aber korrekt und sofort übernommen.
3. Basieren auf der Eingabe als Suffix Name, wird jede Zeile in der Basiskonfigurationsdatei in /etc/rc.conf welche auf das Interface zutrifft mittels „#“ auskommentiert. Da es alsalte und somit stillgelegte Konfiguration gilt.
4. Hier erfolgt die dynamische Konfiguration. Ist DHCP in ip.<Eingegebener Name in Eingabebox> als String vorhanden, so wird der If – Codeblock aktiv. Dieser schreibt die aktuelle Konfiguration
   „ ifconfig_<*Eingegebener Name in Eingabebox*>=“inet ip. <*Ermittelt aus Name in Eingabebox → in Konfigurationsdatei*>**=“DHCP“ netmask.** <*Ermittelt aus Name in Eingabebox → in Konfigurationsdatei*>=**“DHCP“** “ in die Basiskonfiguration von FreeBSD /etc/rc.conf geschrieben.
5. Hier wird auch versucht mittels des Befehls dhclient, die DHCP Konfiguration anzustossen. Doch wie bereits erwähnt, funktioniert dies nur ab und an. Einige Anstrengung wurde in die Lösung dieses Problems gesteckt, doch konnte keine Lösung gefunden werden. Der Fehler scheint sich nicht im Aufbau des Skriptes oder des Designs zu befinden, sondern FreeBSD spezifisch zu sein. Als Lösung wird ein Warnhinweis mittels Dialog Massagebox generiert, welcher einen Neustart des Systems empfiehlt. Dies klappt immer.
6. Gekapselt in obere If – Anweisung wird hier eine zweite ausgeführt, welche spezifisch überprüft ob die IP – Adresse des Outside Interface den String Wert DHCP oder dhcp enthält. Trifft dies zu, wird eine mögliche Defaultrouter Zeile in /rct/rc.conf gesucht und falls vorhanden gelöscht. Diese Braucht es bei DHCP nicht.
7. Ist die Konfiguration aber statischer Natur, so werden bis auf die ersten beiden Kommandos „ifconfig“ und „route add“ alle oben erwähnten if – Anweisungen ignoriert, lediglich die else – Anweisung des ersten „if“ kommt hier zum Tragen.
   An dieser Stelle werden die Konfigurationswerte von IP –Adresse, Netzmaske, Standart Gateway und Nameserver aus den entsprechenden Konfigurationsdateien ermittelt und in Variablen gespeichert. Dann werden mittels des Befehls „sed“ der Standard Gateway in /etc/rc.conf und der DNS – Server Eintrag in /etc/resolv.conf ermittelt und im ersten Fall auskommentiert und im zweiten Fall gelöscht. Die Alten IP Konfigurationen des eingegebenen Interfaces werden ebenfalls auskommentiert. Nun können mittels des Befehls „echo“ die IP Konfigurationen (z.B ifconfig_em0=“inet 10.200.1.1 netmask 255.255.255.0“) und die Defaultroute in /etc/rc.conf niedergeschrieben werden. Der DNS Server wird mittels gleichem Befehl in /etc/resolv.conf eingetragen.
   Innerhalb dieser else – Anweisung wird die Funktion calc_netmask, welche sich im ausgelagerten Skript libnmcalc16 unter /usr/lib befindet ausgeführt. Die Funktion dieser Function ist das Berechnen der Subnetzmaske in Bit für das jeweilige netmask.<*Interfacename*>. Gespeichert wird stets in suffix.<Interfacename>. Diese Finktion wird nur bei statischen Konfigurationen herangezogen, da sie primär nur für das Inside

Interface relevant ist. Zwar wäre es möglich das Inside Interface auch per DHCP zu konfigurieren, jedoch widerspricht dies der Tatsache, dass DHCP fähige Geräte immer eine statische IP benötigen. Auf Sicherungen diesbezüglich wurde verzichtet, da die Infofenster sowie der Quick Guide klar eine statische Konfiguration suggerieren. Profi Anwender muss das nicht gesagt werden, den für sie ist das selbstverständlich.

8. An dieser Stelle wird noch in einer separaten if – Anweisung überprüft, ob es sich um eine DHCP Konfiguration handelt. Sollte dies zutreffen, so muss der Nameserver Eintrag aus /etc/resolv.conf noch entfernt werden. Dieser wird bei DHCP nicht benötigt.
9. Da im Falle einer nachträglichen Konfigurationsänderung die aktuell eingetragenen Makros (Variablen in pf.conf) nicht mehr korrekt sind, muss folgender Schritt noch abgearbeitet werden. Mittels des Befehls „grap“ wird in pf.conf nach einem definierten Such Tag gesucht, welcher #---ACT--- heisst. Dieser Tag, falls so vorhanden, besagt dass die darunter folgenden vier Makros aktiv und in Gebrauch sind. Eine if – Anweisung ändert nun den bereits genannten Such Tag in #---NOTACT--- um und löscht die darunter liegenden Makros, mithilfe von angefügten Kommentaren wie #var1, #var2 usw.
   Nun kommt eine globale Function namens chk_iface_status_on, welche sich zuoberst in libpfdctl befindet zum Zuge. Sie macht das genaue Gegenteil vom oben beschriebenen, denn sie sucht nach dem Tag #---NOTACT--- und ändert ihn in #---ACT--- zurück. Gleichzeitig liest sie aus den Konfigurationsdateien in /usr/local/etc/pfdcontrol die aktuellen Inside/Outside Werte aus. Damit das Gleiche immer wieder funktioniert, wird jedem Makro wieder der Such – Tag #var1-4 als Kommentar hinzugefügt. Die Function warning_1 hat bereits darauf hingewiesen, dass bei nachträglichen Änderungen ein Neustart der Dienste erforderlich ist, was auch getan werden muss, um die neuen Interfaces überall anzumelden. Die Firewall hingegen wird mit dieser Funktion automatisch initialisiert, da sie ein fixer Bestandteil von FreeBSD und PFDialolControl ist. Eine kleine Anmerkung noch zu „gsed“, welches hier verwendet wurde. Es ist eine modernere, aus der Linux Welt bekannte Alternative zu „sed“. Sie wurde an dieser Stelle eingeführt, da das klassische „sed“ nicht mit Zeilenumbrüchen umgehen kann, jedoch war dies an dieser Stelle notwendig.

Viele Sicherungen haben bis hier hin ihre Funktion unter Beweis gestellt, um diese Function in ihrer sauberen Ausführung zu unterstützen. Wurde Start Interface ohne Fehlermeldung ausgeführt, so sollten alle relevanten Interfacekonfigurationen präsent sein um die nachfolgenden Dienste korrekt auszuführen.

„Start Interface“ ist in seiner Funktion sehr umfangreich, weswegen sich dieser Abschnitt nur auf eine Grobbeschreibung reduziert. Für eine genauere Analyse sei an dieser Stelle auf den Code verwiesen. Doch mit diesem Beschrieb und dem nachfolgenden Flowchart, welches als Reflexion mit besserer Übersicht dient, sollte ein klares Verständnis bezüglich der Funktion dieser Function möglich sein.

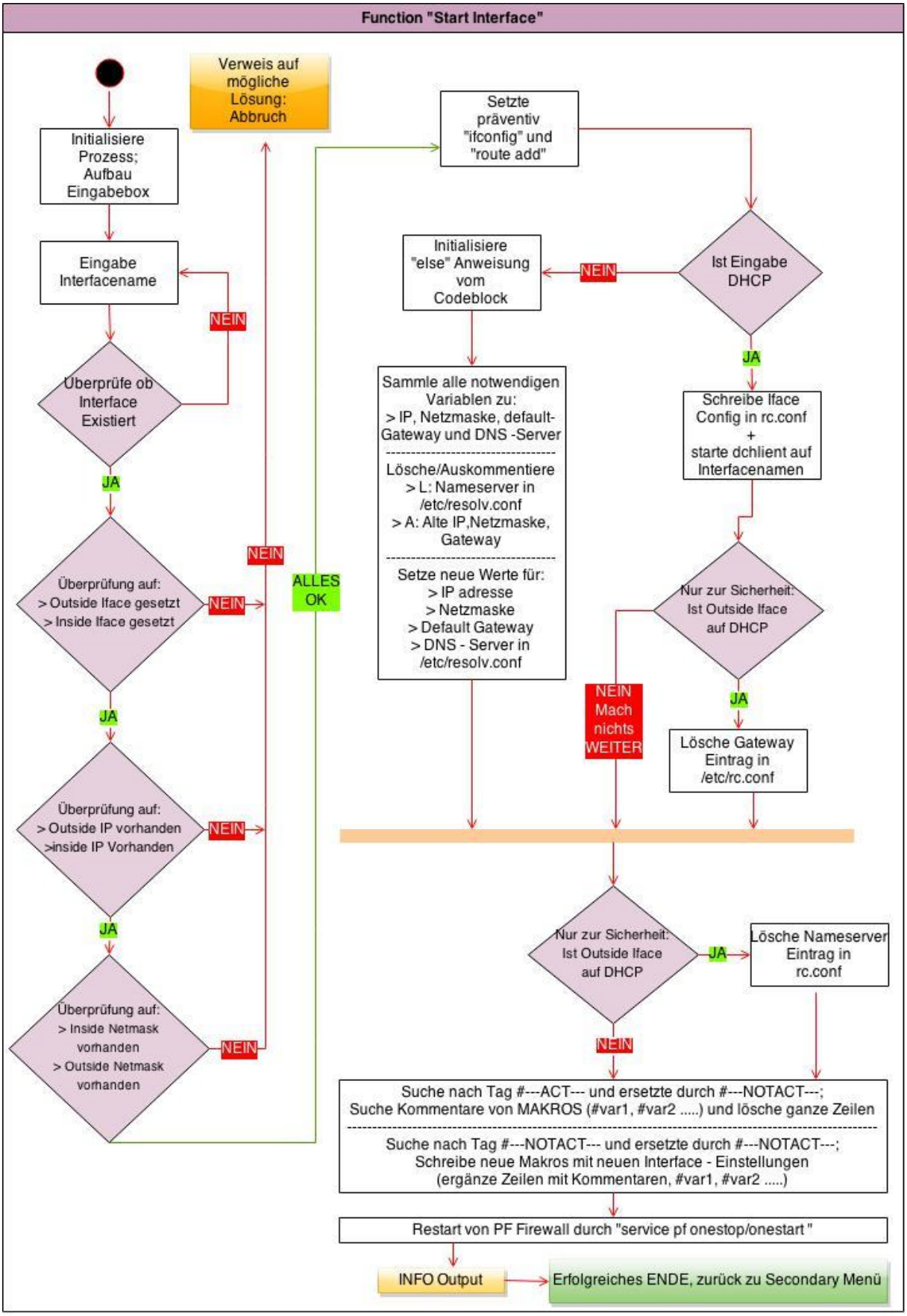

Abbildung 19: Flowchart: Secondary Menü, Start Interface

➔ Stopp Interface:

Die Stopp Funktion ist in der Function run_netconf_stop definiert und äusserst bescheiden gehalten. Die meisten der Zeilen befassen sich mit dem Prüfen auf Leer Eingaben und die Existenz der Interfacenamen. Sicherungen die einen daran hindern eines der relevanten Interfaces zu deaktivieren sind nicht implementiert worden, da wie in Abbildung 20 ersichtlich ist, die relevanten Interfaces aufgeführt werden. Wurde das Interface korrekt erkannt, wird lediglich aus der Input Variable folgender Befehl generiert und ausgeführt:

Abbildung 20: Secondary Menü: Stop Interface, Eingabebox

ifconfig <*Input aus Eingabebox*> **down delete**

Dieser Befehl deaktiviert das angesprochene Interface. In einem zweiten Schritt wird mithilfe von „sed“ die Zeile für das besagte Interface ermittelt und auskommentiert. Somit ist das Interface gestoppt und wird auch nicht mehr gestartet. Ein Beispiel für eine solche auskommentierte Zeile in /etc/rc.conf könnte sein:

ifconfig_<Interfacename>=“**inet** <IP – Adresse> **netmask** <Netzmaske>“ geändert in →

#<Interfacename>--DECON—duc--PFDialogControl=“<IP -Adresse“ #Diese Regel wurde durch…..

Die Schreibweise oben entspricht dem Standard Vorgehen bei diesem Projekt. Wie ersichtlich bleibt ein Teil der Informationen (IP) erhalten. Dies ist für Profis, um mögliches Debugging ihrer Konfiguration zu erleichtern.

Die Deaktivierung und Dekonfigurierung des Interfaces ist alles was hier abgearbeitet wird. Ein Refresh der Einstellungen mittels der Function chk_iface_status_on wird ausdrücklich abgelehnt. Denn erstens macht dies keinen Sinn, da die Konfiguration der nötigen Parameter nicht bekannt ist. Zum Beispiel ist Outside noch immer gültig wenn der User „nein“ sagt. Zweitens will man ja bewusst ein Interface ausserbetrieb nehmen. Warum also soll die Konfiguration der Makros aufgefrischt werden, wenn sie ja noch immer gleich ist. Auch im Notfall, wie bspw. Trennung zur Aussenwelt aufgrund dringenden Tatverdachts auf Schadsoftwarebefall, ist es von Vorteil wenn man mit wenigen Aktionen eine Verbindung roh stoppen kann, um sie vielleicht später wieder mit gleichen Parametern in Betrieb zu nehmen. So kann mit dieser simplen Function die Verbindung einfach gestoppt werden, sowie mit einem einfachen „Start Interface“ wieder mit den gleichen Parametern online geschaltet werden. Da die Function reell betrachtet nur aus zwei Kernfunktionen besteht, wird an dieser Stelle auf die detailliertere Betrachtung mittels Flowchart verzichtet.

➔ Kurzer Exkurs zur „Zurück Funktion“:

Diese Funktion wird per Default von diaolg zur Verfügung gestellt, jedoch hat sie in ihrer rohen Funktion eine simple Abbruchfunktion. Möchte man sie etwas umfangreicher nutzen, so muss man ihr Instruktionen mitgeben. Die Auswahl eines OK/JA Buttons löst einen booleschen Rückgabewert von „0“ aus. Demzufolge muss für NEIN/ABBRECHEN/ZURÜCK der Wert „1“ generiert werden. Dieser Wert wird mittels einer standardisierten if – Anweisung aufgefangen und realisiert, welche aus Lesbarkeitsgründen in einen Einzeiler verpackt wurde. Eine solche Anweisung findet sich, falls notwendig, immer unterhalb der dialog – Anweisung zu der sie gehört. Dabei folgt sie meist folgendem Muster:

If [$opt != 0]; **then** <*Mögliche Aktionen bspw. exit, return….*>

Wird also eine Aktion mittels Abbruch- oder Zurück – Button gewählt, so kommt immer eine solche Funktion zum Tragen. In unterschiedlichen Umgebungen kann es je nach Lokation der Sprache im System auch vorkommen, dass anstelle eines OK ein JA definiert wird. Dies lässt sich nicht ändern, da die in FreeBSD verwendete Dialog Version dies nicht zulässt. Dies gilt aber meist nur für Linux Systeme, FreeBSD folgt hier einer anderen Philosophie wo immer OK verwendet wird, aber dafür mittels der dialog Option --cancel-lable „Zurueck“ das englische Cancel durch ein eigenes Label ersetzt werden kann.

Um die vor- und zurück Funktionen zu realisieren, wird meist eine Anweisung wie bspw. menu_network, gefolgt vom Shell typischen Operationsverbinder „&&“ in Verbindung mit einem exit genutzt. Dies alles wird in den Codeblock der Zurück – if – Anweisung geschrieben. So kann man der Shell klar sagen: Geh gleich zur besagter Funktion und führe hier wo du gerade bist ein exit aus, um diesen Prozess hier zu beenden. Das „exit“ ist notwendig, da der Prozess ansonsten weiterlaufen würde und ein erneuter Aufruf der Function verunmöglicht wird, was zur Folge hätte, dass die Shell diesen Prozess einfach überspringt und zum nächsten übergeht.

Mehr gibt es zu dieser Funktion nicht zu sagen. Sie taucht praktisch in jeder Box bzw. jedem Fenster auf und verweist meistens einen oder mehrere Schritte zurück. Lediglich die „Ende“ Funktion im Hauptmenü hat einen etwas umfangreicheren Code, da sie sich um das Löschen der Temporär Dateien kümmert, welche beim Speichern diverser Eingaben entstehen.

Da die Zurück – Funktion Global ist und an dieser Stelle umschrieben wurde, wird eine erneute Definition in den nachfolgenden Abschnitten nicht mehr erfolgen.

8.3.2.2 DHCP/DNS Einstellungen

Filename → libpfdctl
Position in Filesystem → /usr/lib
Function Namen → menu_dnsmasq
Rechte → Default: Besitzer root, Gruppe wheel, Lesend – Schreibend (rw- – rw- - ---)

Abbildung 21: Secondary Menü: DHCP/DNS Einstellungen

Dieses Secondary Menü DHCP/DNS Einstellungen bietet die Möglichkeit PFDialogControl um die Funktionalität eines DHCP und DNS Servers zu erweitern. Zu diesem Zweck wurde dnsmasq installiert, der beide Funktionen in einem Simplen Konfigurationsfile vereint. Diese Wahl erleichtert einerseits das Updaten des FreeBSD Unterbaus, da nur ein Tool aktuell gehalten werden muss, sowie reduziert sich der mir DNS verbundene Konfigurationsaufwand um ein vielfaches, da dnsmasq per Default alles im Hintergrund managet. So ist für den Betrieb dieser Funktion in PFDialogControl lediglich die Eingabe eines Domainnamens erforderlich. Die Aktivierung- bzw. Deaktivierung des DHCP/DNS Services ist neben der Tatsache, dass es so in der Aufgabenstellung steht auch aus praktischer sich nützlich. Denn es bietet die Möglichkeit einen zweiten DHCP/DNS Service, an einem anderen Punkt zu betreiben. PFDialogControl kann aber auch nur mit dieser Funktion aktiv, als simpler DHCP/DNS Server im Netzwerk oder in Testumgebungen wie es schon öfters in gewissen Modulen an der HFU vorkam, genutzt werden.

Hier kommt auch das in der Hauptstudie (Punkt 7.2.2, Seite 24) definierte Konzept der Autogeneration des IP Ranges zum Zuge. So kann der Anwender lediglich sagen ob er den Dienst nutzten will oder nicht. Alle weiteren Schritte bezüglich Konfiguration und automatischer Start beim Reboot übernimmt PFDialogControl. Bei aktivem Dienst, steht dem Anwender ein DHCP Server zur Verfügung, welcher dnsmasq typisch IP Adressen nach dem Zufallsprinzip vergibt und die Namen der Hosts in einer eigenen DNS Datenbank mit Reverse Funktion führt.

Nachfolgen ein kleines Schema, welches die Funktion des Dienstes verdeutlicht.

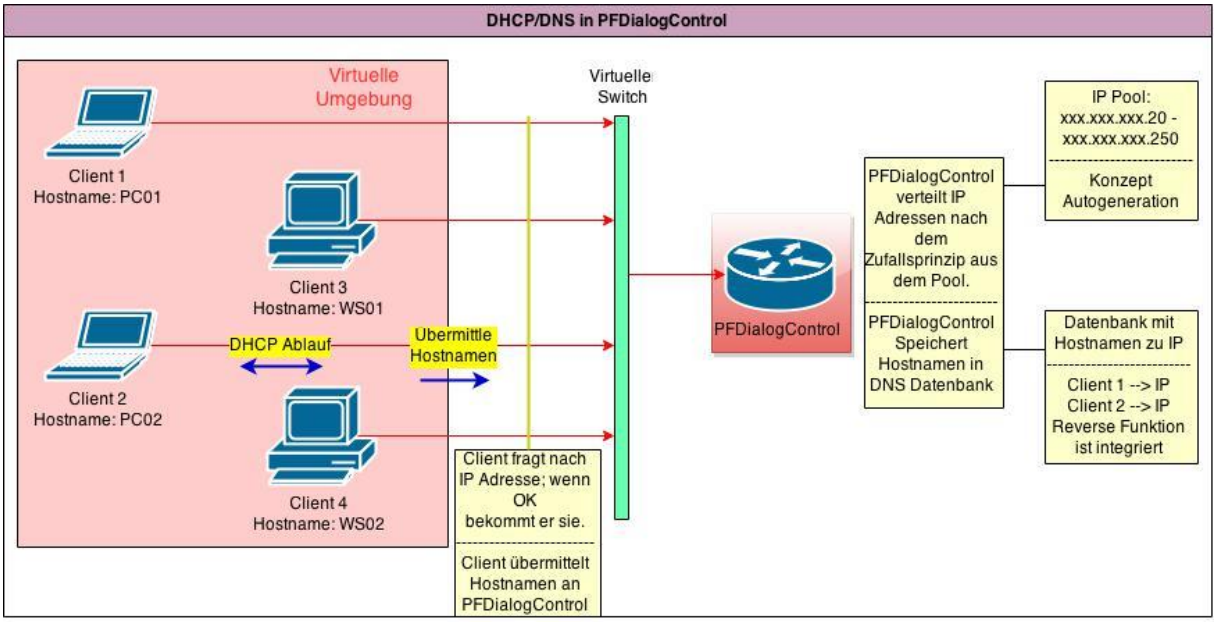

Abbildung 22: Schematische Darstellung des Funktionsprinzips von DHCP/DNS in PFDialogControl

Die Menüstruktur hat folgende Verknüpfungen zu den Skript Functions in libpfdctl:

- **Domainname** → input_dnsmasq_domainname
- **DHCP/DNS INFO** → info_dnsmasq
- **Start DHCP/DNS** → run_dnsmasq
- **Stop DHCP/DNS** → stop_dnsmasq
- **Zurück** → Keine Funktion, ist direct im Dialog Menü integriert. Zusätzlich existiert ein if – Anweisung, welche den Button die gleichen „Zurück" Fähigkeiten verleit.

Die erste Prüfung die beim Start getätigt wird ist eine Statusprüfung, welche mit zwei ineinander verschachtelten if – Anweisungen realisiert wird. Hierbei wird überprüft ob der Status von dnsmasq auf gestoppt zeigt. Sollte dies zutreffen, so wird gleich ein „DHCP/DNS ist DEAKTIVIERT" in eine Variable geschrieben, welche im Infotext des Secondary Menü erscheint. Sollte dnsmasq hingegen gestartet sein, so wird im zweiten if – Block geprüft, ob das Inside Interface über den Befehl „ifconfig" ansprechbar ist, da nur nach positiver Bestätigung dieser beiden Punkte ein lauffähiger DHCP/DNS Service gewährleistet ist.

```
if [ $(ps -ax | grep dnsmasq | wc -l) = 1 ]; then
  varRUNNING="DNS/DHCP ist DEAKTIVIERT"
else
  ifconfig $(cat /usr/local/etc/pfdcontrol/inside_iface) | grep inet
  if [ "$?" != "0" ]; then
      varRUNNING="DNS/DHCP ist DEAKTIVIERT"
  else
      varRUNNING="DNS/DHCP ist AKTIVIERT"
  fi
fi
```

Abbildung 23: Code Ausschnitt zur Statusanzeige des DHCP/DNS Services

→ Domainname:

Die Function input_dnsmasq_domainname ist simpel realisiert und umfasst nur wenige Zeilen Code. Ihre Hauptaufgabe ist das erfragen des gewünschten Domainnamen über eine Dialog Inputbox. Dabei wird der String unter /usr/local/etc/pfdcontrol/ in dem File domainname gespeichert.

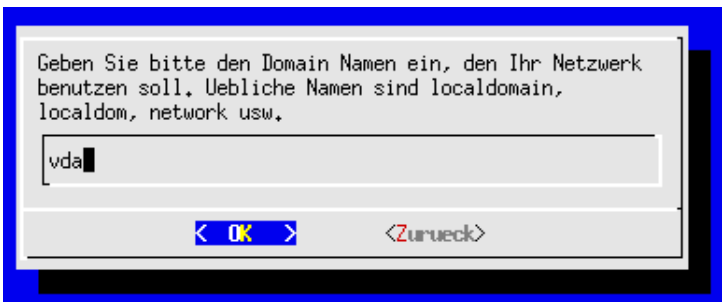

Abbildung 24; Secondary Menü: DHCP/DNS, Inputbox Domainname

Hier gibt es keinerlei Einschränkungen durch implementierte Filterfunktionen, es kann jeder beliebige Name verwendet werden der einem passt. Es wird lediglich mittels if – Anweisung geprüft, ob die bei der Installation erzeugte Datei „domainname“ einen String Wert höher als null besitzt. Auf diese Weise werden Leereingaben abgefangen. Zusätzlich wird an dieser Stelle wieder der Inhalt des Config Files präventiv in eine Variable gespeichert, um den Inhalt in der Inputbox anzuzeigen, sowie ihn im Falle eines „Zurück“ in das Config File zurückzuschreiben.

Die Bestätigung erfolgt mittels einer Dialog Infobox mit dem Text „Ihr neuer Domain Name lautet → <Eingegebener String>“

→ DHCP/DNS INFO:

Die Funktion DHCP/DNS INFO ermöglicht einem die Einsicht in die mögliche Konfiguration, im Falle eines Dienststartes. Dazu werden die benötigten Werte in folgende Variablen gespeichert:

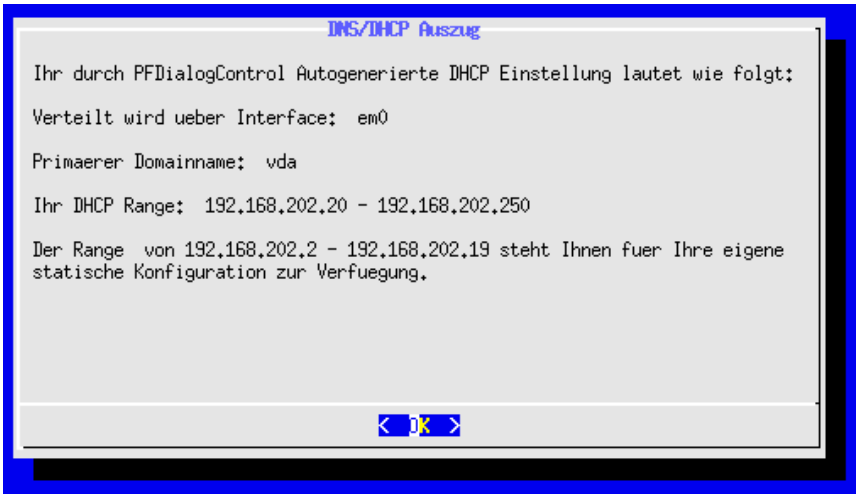

Abbildung 25: Secondary Menü: DHCP/DNS, DHCP/DNS INFO, Dialog Infobox

- inside= → holt sich mittels des Befehls „cat“ den Namen des Inside Interfaces aus dem Config File inside_iface
- range= → holt sich die IP Adresse des Inside Interfaces mittels „cat“ und schneidet ihn sich mittels „cut –d „.“ –f1-3“ so zurecht, dass nur die ersten drei Ziffernblöcke der IP Adresse übrig bleiben, wie bspw. 192.168.202
- dom= → ermittelt aus /usr/local/etc/pfdcontrol/domainname den Domainnamen und speichert ihn.

Diese drei Schritte werden mit in den Variablen gespeicherten Befehlsaufrufen durchgeführt, wobei meist der Befehl „cat“ genutzt wird. Lediglich der Range wird nochmals nachformatiert, indem er zum Befehl „cut“ gepipet wird. Diese Funktion wird nachfolgend im Abschnitt „Start DHCP/DNS“ benötigt, um den autogenerierten IP Range zu erzeugen.

Das Ergebnis ist rein informativ und zeigt die mögliche Konfiguration, bzw. es zeigt gleich an dieser Stelle wo noch offene Konfigurationspunkte sind. Diese werden nicht im Output Fenster dargestellt. Abbildung 24 zeigt einen Auszug aus einer korrekt konfigurierten und in Betrieb stehenden Konfiguration.

Die Function info_dnsmasq obliegt keinerlei Prüfrutinen und kann in konfiguriertem, sowie in noch rohem Zustand des DHCP/DNS Services ausgeführt werden. Sie ist als minimales Debugging Tool mit rein informativer Funktion zu sehen.

➔ Start DHCP/DNS:

Die Konfiguration dieses Services spielt sich im dnsmasq eigenen Konfigurationsfile ab, welches FreeBSD spezifisch als Userland Programm in /usr/local/etc/dnsmasq.conf zu finden ist. Hierbei wird ein Musterfile unter /usr/local/etc/pfdcontrol/skel/ mit dem aussagekräftigen Namen dnsmasq.skeleton verwendet. Dieses File hat eine minimale Grundkonfiguration, welche nach Ausführen dieser Function um weitere Parameter ergänzt wird. Nachfolgendes Flowchart soll dies verdeutlichen.

Abbildung 26: Secondary Menü; DHCP/DNS Start DHCP/DNS, Infobox

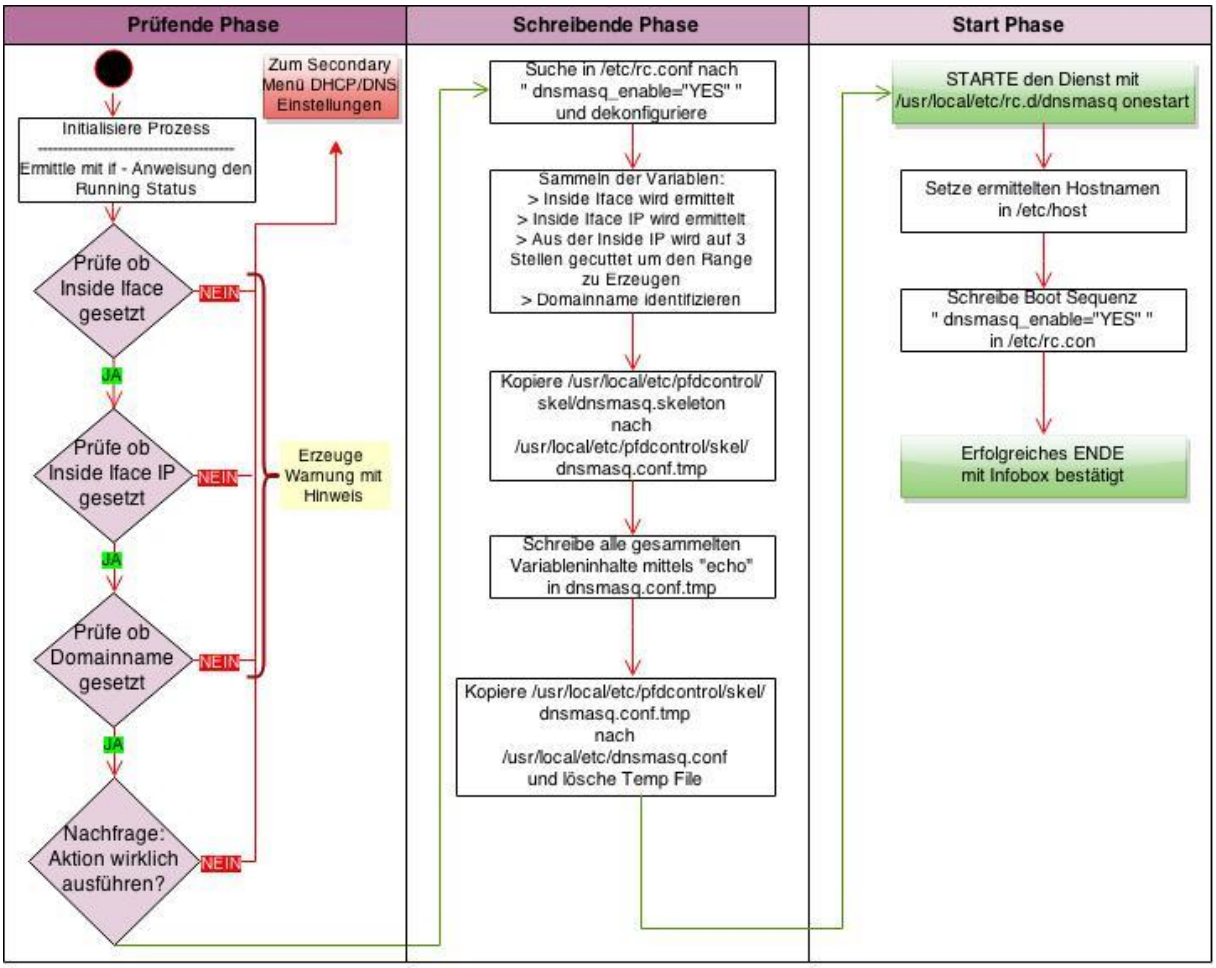

Abbildung 27: Secondary Menü: DHCP/DNS, Start DHCP/DNS, Phasen Schema

An dieser Stelle wird mittels einer kombinierten und einer separaten if – Anweisung auf das Vorhandensein von Strings in den benötigten Config Files geprüft. So prüft die kombinierte if - Anweisung auf vorhandene Einträge in inside_iface sowie in ip.<Inside Interface> , wo ermittelt wird, ob das Inside Interface überhaupt gesetzt und konfiguriert ist. Auf die Prüfung der Outside Interface Einstellungen kann an dieser Stelle verzichtet werden, da sie für den DHCP/DNS Service irrelevant sind. Der separate if – Block ist für die Prüfung auf vorhandenen Domainname zuständig in /usr/local/etc/pfdcontrol. Fehlende Einstellungen würden bereits an dieser Stelle abgefangen und mittels Warnfenster dem User mitgeteilt. Ebenfalls kann auf das Prüfen der Netzmaske verzichtet werden, da das Interface unter „Interface Einstellungen → Start Interface“ gar nicht gestartet worden wäre und somit die Statusprüfung (siehe Abbildung 27) dies sofort melden würde.

```
### Skeleton "dnsmasq" modifiziert durch PFDialogControl ###

domain-needed
bogus-priv
expand-hosts

interface=em0
dhcp-option=option:router,192.168.202.1
dhcp-range=192.168.202.20,192.168.202.250,336h
local=/vda/
domain=vda
```

Abbildung 28: Code Ausschnitt aus dnsmasq.conf, in fertigem Zustand

Zusätzlich wird mittels des Aufrufs **sed “s/dnsmasq_enable/#dnsmasq_enable—Deconfig—duch PFDialogControl/g** in rc.conf nach dem vorhanden Startparameter **dnsmasq_enable=“YES“** gesucht und dekonfiguriert.

Sind sämtliche Prüfungen positiv ausgefallen, wird die Arbeitsbestätigung nochmals mittels dialog – yesno – Box abgefragt. Bestätigt der User so werden folgende Werte mittels der beiden Befehle „cat und cut„ ermittelt und in Variablen gespeichert:

- inside= → Name des Inside Interface
- range= → der autogenerierte IP Range wird mittels der Befehlskette „cat <Inside IP> | cut –d „.“ –f1-3“ auf drei Blöcke der IP – Adresse zugeschnitten. Die Ziffern 20 und 250 werden statisch vom Skript übernommen. So entsteht immer die gleiche Poolbreite in Relation zur eingegebenen Inside Interface IP.
- dom= → Der eigegebene Domainname
- router_ip= → Die IP –Adresse des Inside Interface
- w_hostname= → der Hostname des Systems wird mittels des Befehls „hostname“ ermittelt und gespeichert
- ip= → IP – Adresse des Inside Interfaces

Nun wird aus dem Skeleton Ordner unter /usr/local/etc/pfdcontrol/skel das vorkonfigurierte Config File für dnsmasq in dnsmasq.conf.tmp kopiert, wo sämtliche oben genannten Werte aus den Variablen mittels des Befehls echo kopiert werden. Danach wird das Temp – File an seinen Bestimmungsort unter /usr/local/etc/dnsmasq.conf kopiert. Das Design des dnsmasq Config Files hat dabei folgendes Layout (Abbildung 28):

- Fix im Skeleton File vordefiniert
  - domain-needed; falls nur ein leerer Rechnername ohne Punkt angegeben wird, so soll der DNS Aufruf nicht an einen externen DNS weitergeleitet werden. Auch nicht wenn er in /etc/hosts gefunden wurde.
  - bogus-priv; gibt bei nichtauflösbaren Reverse DNS Aufrufen im privaten IP Range eine Fehlermeldung zurück
  - expand-host; fordert dnsmasq dazu auf, den Domainname in DNS zu ergänzen falls es der User bei einer Eingabe vergessen hat.

Die drei Konfigurationen sind bereits im Skeleton integriert, da der Entwickler dieser Arbeit dies jeweils in diesem Stil praktiziert.

- Die nachfolgenden Punkte werden bei jedem Aufruf von „Start DHCP/DNS“ neu generiert.
  - Interface; wurde aus einer Variablen geschrieben und entspricht dem Inside Interface. Dies ist notwendig um dnsmasq anzuweisen, lediglich das Inside Interface zu nutzen. Ansonsten würde auch über das Outside Interface IP – Adressen verteilt.
  - dhcp-option=option:router; teilt per DHCP den Client mit wer ihr defaultrouter ist.
  - dhcp-range; entspricht dem autogenerierten Default Range von PFDialogControl.
  - local=/<Domainname>/; schränkt den Suchradius von dnsmasq bezüglich des Domainnamens auf /etc/hosts und die interne DNS Datenbank ein um zu verhindern, dass lokale nichtauflösbare Aufrufe an einen externen DNS Server gelangen.
  - domain; definiert die Suchdomäne. Diese wird den Clients mitgeteilt, damit sie überhaupt wissen in welcher Domäne sie sich befinden.

Nun befindet sich dnsmasq.conf an seinem Bestimmungsort und der Daemon kann gestartet werden. Dies geschieht mit dem Aufruf von /usr/local/etc/rc.d/dnsmasq onestart. Die Tatsache, dass der Start bereits hier erfolgt, obwohl noch zwei Schreiboperationen notwendig wären, hat rein sicherheitstechnische Gründe. Denn sollte etwas schieflaufen beim Start, so verhindert dies das Schreiben der Start – Up Konfiguration in rc.conf. So wird verhindert, dass ein fehlerhafter Dienst per Default beim Systemstart versucht zu starten und so den Bootprozess von FreeBSD kompromittiert.

War der Start erfolgreich, so wird mittels der Befehlskette **sed “/$w_hostname/d“** versucht den aktuellen Hostnamen zu finden und zu löschen. Sowie mittels „echo“ ihn gleich wieder neu zu setzen.

Im letzten Schritt wird in /etc/rc.conf die Zeile dnsmasq_enable=“YES“ gesetzt, um den DHCP/DNS Dienst per Default als Start – Up Daemon zu definieren.

➔ Stop DHCP/DNS:

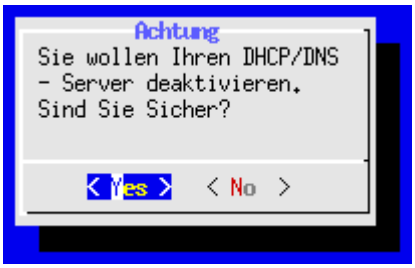

Abbildung 30: Secondary Menü: DHCP/DNS: Stop DHCP/DNS, Warnfenster

Die Funktion der Function stop_dnsmasq ist simpel, sie soll den DHCP/DNS Service stoppen und deaktivieren. Genau so simpel ist die Realisierung dieser Funktion, da sie lediglich aus einem Dialog –yesno Aufruf besteht. Beantwortet der User diesen mit yes/ja, so versucht der Befehl „sed“ den String dnsmasq_enable= in /etc/rc.conf zu identifizieren und mit dem PFDialogControl typischen Dekonfigurationstext auszukommentieren. Im nachfolgenden Schritt wird **mittels /usr/local/etc/rc.d/dnsmasq stop** dnsmasq gestoppt.

Das Konfigurationsfile unter /usr/local/etc/dnsmasq.conf wird nicht gelöscht, da es keine Funktion erfüllt, sowie ohnehin beim nächsten „Start DHCP/DNS“ überschrieben wird.

Eine Abschaltbestätigung wird mittels dialog –infobox generiert. Dies beinhaltet auch den Hinweis, dass kein DHCP-Relay-Agent zur Verfügung steht und somit dringend darauf zu achten ist, dass eine Ausweichlösung DHCP – technisch nötig wird.

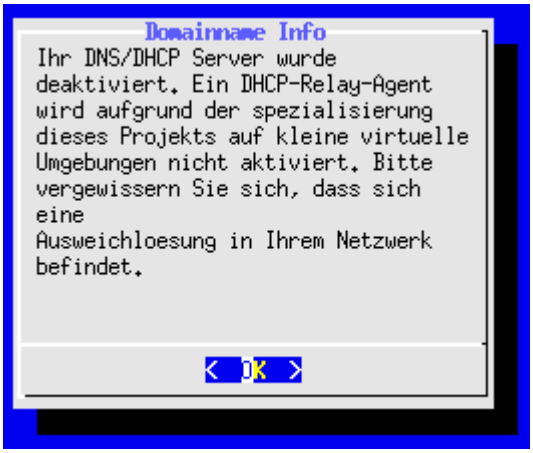

Abbildung 29: Secondary Menü: DHCP/DNS: Stop DHCP/DNS, Infobox

8.3.2.3 NAT Einstellungen

Filename → libpfdctl
Position in Filesystem → /usr/lib
Function Namen → menu_nat
Rechte → Default: Besitzer root, Gruppe wheel, Lesend – Schreibend (rw- – rw- - ---)

Abbildung 31: Secondary Menü: NAT Einstellungen

Die Funktionsweise diese Menü Punktes ist schnell erklärt. Man kann die NAT Funktion aktivieren bzw. deaktivieren, dies alles erfordert an dieser Stelle keinerlei Konfigurationen. Der User muss sich lediglich entscheiden ob er NAT nutzen will oder eben nicht. Viele moderne Router bieten diese Funktion zwar an, jedoch lässt sie sich in den meisten Ländern nicht nutzen, da man vom Provider nur eine öffentliche IP bekommt und keine Konfigurationsmöglichkeit seitens statischer Routingtabellen hat. Im eigenen Netzwerk sieht dies aber ganz anders aus, denn hier möchte man vielleicht aus Performance Gründen auf NAT verzichten. So war es ein grosses Anliegen des Entwicklers diese Funktion zu implementieren.

Jedoch benutzt PFDialogControl ein spezielles Design, wo Routing -und NAT Funktionen zeitgleich existieren können. Im Normalfall muss man sich entscheiden ob die Firewall das Management zwischen den Interfaces steuert, oder eine Routingtabelle den Takt vorgibt. PFDialogControl hat per Definition den Syscontrol Forwarding Eintrag aktiv und funktioniert ab Installation als dynamischer Router zwischen den Interfaces. Durch Verzicht auf komplexe NAT Optionen innerhalb von pf.conf, ist es möglich mit nur einer klaren NAT Regel, welche sich im File zuoberst befindet, die Routingtabelle von FreeBSD ausser Kraft zu setzen. Ist also NAT aktiv, kann man eine Route zu einem Host über die direkte Eingabe der IP – Adresse des Zielrechners nicht erreichen, da die NAT Regel dies verbietet. Es steht nur die Möglichkeit zur Verfügung, PFDialogControl über seine Outside IP anzusprechen und eine Portweiterleitung zu definieren. Deaktiviert man aber NAT, kann man den Zielrechner über seine eigene IP erreichen indem PFDialogControl als simpler Gateway fungiert. Zeitgleich ist es aber auch weiterhin möglich PFDialogControl über die Portweiterleitungsregel zu nutzen. Im Falle eines Wechsels von NAT zu konventionellem Routing, wäre die statische Routingtabelle auf jedem Client im LAN anzupassen. Mit diesem speziellen Design von PFDialogControl, ist lediglich die Routingtabelle im WAN zu modifizieren, da es den Clients im LAN egal ist wie PFDialogControl den Verkehr managet. Diese Technik wird vom Entwickler seit langem mit positiver Erfahrung eingesetzt und musste daher in PFDialogControl einfliessen. Nachfolgendes Flowchart soll dies noch etwas besser verdeutlichen.

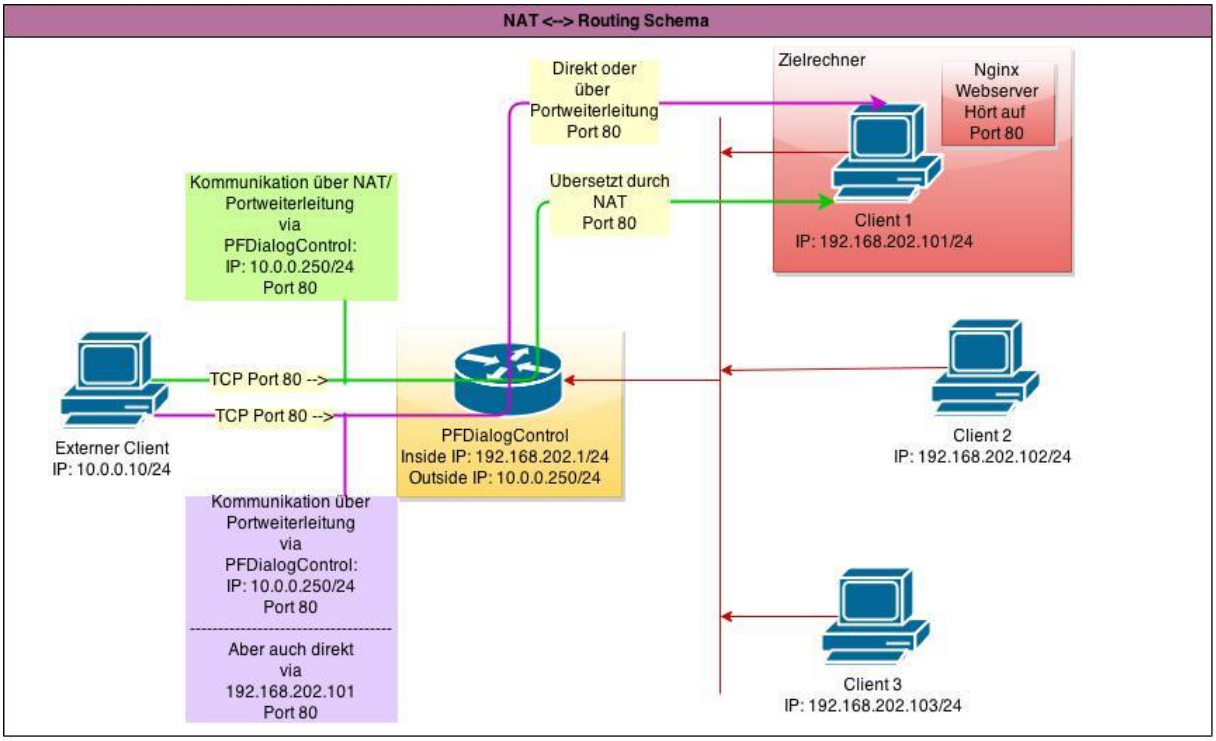

Abbildung 32: NAT <--> Routing Schema

Auch an dieser Stelle wird als erstes mittels einer ineinander verschachtelten if – Anweisung geprüft ob das Outside Interface eine gültige IP – Adresse besitzt. Sowie ob ein boolescher Wert „0“ erhalten wird, wenn mittels „grep“ nach dem Such String „nat on“ in /etc/pf.conf gesucht wird. Sind beide Bedingungen erfüllt, kann mit klarer Sicherheit ein „NAT ist Aktiviert“ in das Secondary Menü NAT Einstellungen gesetzt werden. Das Design dieser Funktion entspricht dem in „DHCP/DNS Einstellungen“ womit an dieser Stelle auf Seite 47, Abbildung 27 verwiesen sei. Abbildung 33 zeigt den verwendeten Code in Relation zu Abbildung 27.

```
cat /etc/pf.conf | grep "nat on"

if [ "$?" != "0" ]; then
    varRUNNING="NAT ist DEAKTIVIERT"
else
    ifconfig $outside | grep inet
    if [ "$?" != "0" ]; then
      varRUNNING="NAT ist DEAKTIVIERT"
    else
      varRUNNING="NAT ist AKTIVIERT"
    fi
fi
```

Abbildung 33: Code Ausschnitt zur Statusanzeige des NAT Aktivieren

➔ NAT Aktivieren:

Beim Aktivieren der Function run_nat wird wie in PFDialogControl üblich, eine Prüfung der Interfaces Inside und Outside auf Vorhandensein eines String Wertes grösser als „0“ geprüft. Dasselbe gilt auch für die entsprechenden IP – Adresse der beiden Interfaces. Sollte hier ein Fehler vorliegen, so wird wieder mittels dialog Massagebox auf den Fehler hingewiesen. Sollte alles in Ordnung sein, wird vor Beginn der eigentlichen Konfigurationsarbeit nochmals nachgefragt ob diese Aktion wirklich durgeführt werden soll. Bestätigt der User dies, wird mittels des Befehls „sed“ folgende statische Operation gestartet:

```
sed 's/#---nat---/nat on $out_iface from $in_iface_ip to any -> $out_iface/g'
```

Abbildung 34: Code Ausschnitt: NAT Aktivieren, setzten des NAT Parameters in pf.conf

Diese Zeile sucht nach dem String „#---nat---„ und ersetzt ihn durch „nat on $out_iface ……“ im PF Konfigurationsfile pf.conf. Die Werte $out_iface, in_iface sind in den Makros als Variablen definiert

und wurden bei „Interface Einstellungen → Interfaces“ gesetzt. Um auf Nummer sicher zu gehen, kommt an dieser Stelle die Globale Funktion chk_iface_status_on zum Zuge, welche auf Seite 42 Punkt 9 beschrieben wurde. Sie sorgt dafür, dass stets das aktuelle Outside und Inside Interface sich in pf.conf befinden. Der Such Flag „#---nat---„ befindet sich immer im oberen Abschnitt des bei der Installation, aus pf.skeleton erzeugten pf.conf Datei. Die Raute vor dem Such Flag deklassiert ihn als simplen Kommentar. Durch das Ersetzten mittels der sed Operation, wird aber eine vollwertige PF Anweisung erzeugt, welche besagt, dass alles NAT anwenden soll was aus dem Inside Interface ($in_iface) kommt und in Richtung Outside Interface ($out_iface) geht. Mehr Kontext übermittelt diese Anweisung nicht an die PF Firewall.

Nach setzen der unter Abbildung 34 definierten Regel, muss die PF Firewall noch neu geladen werden, um die dynamische Regel Tabelle welchem im RAM geführt wird zu aktualisieren. Dies geschieht mit den Anweisungen „service pf onestop“ und „service pf onestart“. Eine Besonderheit bei der PF Version unter FreeBSD ist, dass PF die Regeländerung erst annimmt, wenn tatsächlich eine Änderung im pf.conf stattgefunden hat. Aus diesem Grund wird mittels der Befehlskette „**echo ““ >> /etc/pf.conf“** eine Leerzeile in die Datei geschrieben und anschliessend mit „pfctl –f /etc/pf.conf“ eine Regelüberprüfung erzwungen. Diese beiden Schritte sind notwendig, da PF für FreeBSD nicht wie unter OpenBSD für die dynamische Regeländerung konzipiert wurde. Ein „sysctl net.inet.ip.forward=1“ wird ebenfalls präventiv mitgeschickt, da FreeBSD in diversen Vor – Tests die Neigung zeigte diesen Parameter zu deaktivieren. Die genaue Ursache dieses Verhaltens konnte nicht ermittelt werden, wird aber mit dieser Massnahme behoben. Abschliessend wird mittels dialog Massagebox noch die korrekte Ausführung bestätigt und mit einem Auszug

```
sysctl net.inet.ip.forwarding=1
service pf onestop
service pf onestart
echo "" >> /etc/pf.conf
pfctl -f /etc/pf.conf
```

Abbildung 35: Startsequenz für einen PF Neustart. Diese Regel wird immer verwendet, wenn PF modifiziert wurde.

Abbildung 36: Third Level Menü zu NAT Aktivieren, Infobox mit Infos zu Outside Interface

versehen, welcher das aktuelle Outside Interface und dessen IP – Adresse anzeigt.

Auf der nachfolgenden Seite ein kleines Flowchart zur besseren Verdeutlichung:

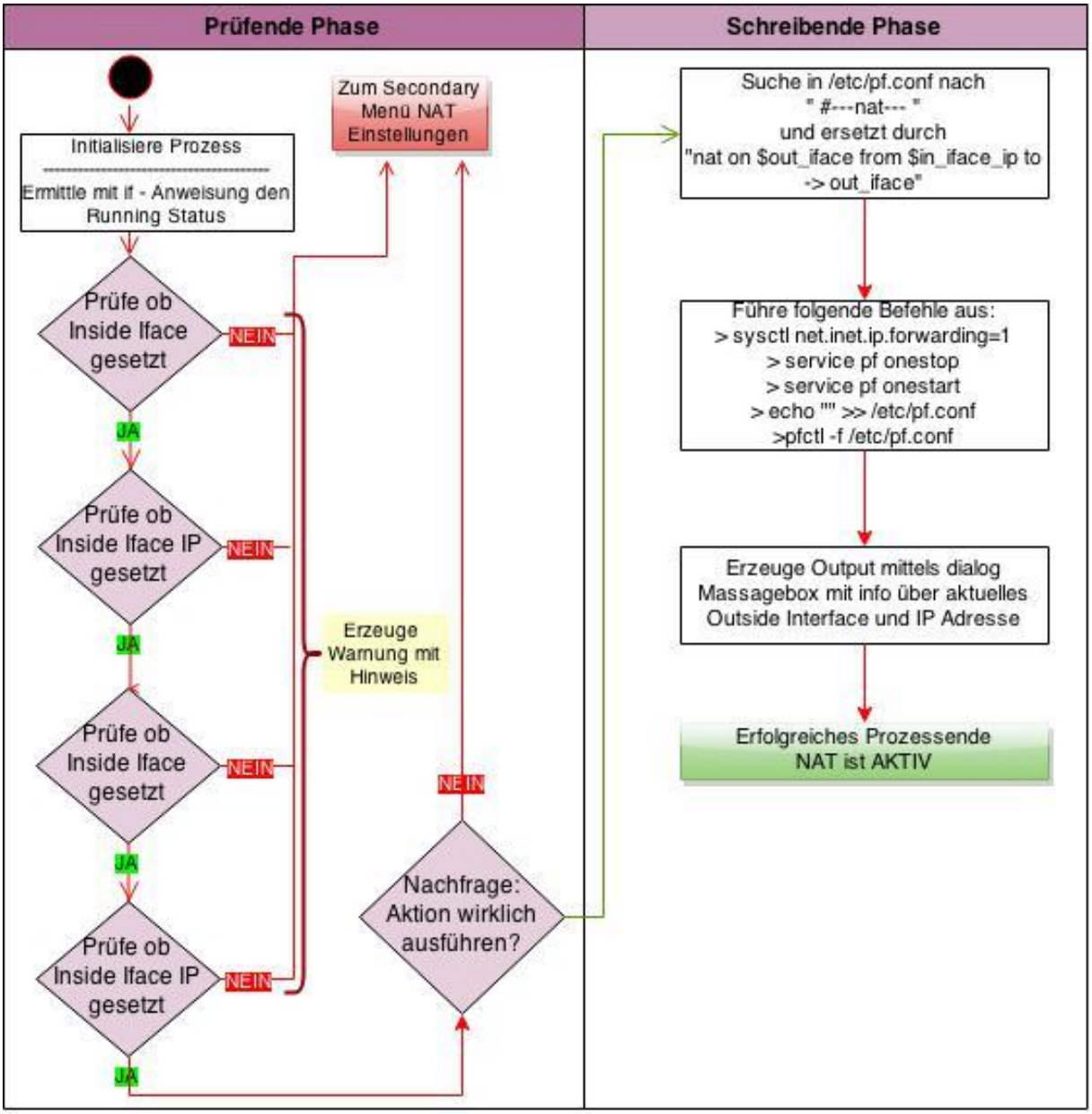

Abbildung 37: Secondary Menü: NAT Aktivieren, Flowchart zur Funktion run_nat

➔ NAT Deaktivieren:

„NAT Deaktivieren“ entspricht dem Design von „NAT Aktivieren“, mit dem einzigen Unterschied, dass die sed Zeile (Abbildung 34, Seite 54) in umgekehrter Reihenfolge ausgeführt wird. So kann der Such Tag „#---nat---“ wiederhergestellt werden, um im Falle eines „NAT Aktivieren“ diesen zu identifizieren und wieder entsprechend zu modifizieren. Auf die if – Prüfungen bezüglich Inside/Outside Interface/IP kann an dieser Stelle verzichtet werden. Durch das Entfernen der NAT Regel in pf.conf wird auch die NAT Funktionalität in PF deaktiviert.

```
sed 's/nat on $out_iface from $in_iface_ip to any -> $out_iface/#---nat---/g'
```

Abbildung 38: Code Ausschnitt, NAT Deaktivieren. Der einzige Unterschied zur Funktion NAT Aktivieren

8.3.2.4 Firewall

Filename → libpfdctl
Position in Filesystem → /usr/lib
Function Namen → menu_fw
Rechte → Default: Besitzer root, Gruppe wheel, Lesend – Schreibend (rw- – rw- - ---)

Abbildung 39: Secondary Menü: Firewall

Die in diesem Abschnitt enthaltenen Funktions sind:

- **Firewall Status →** status_fw
- **Portweiterleitung EIN →** list_pw und input_pw
- **Portweiterleitung AUS →** list_pw_stop und delete_pw
- **Verbindung Sperren →** list_block und input_block
- **Verbindung Lösen →** list_block_stop und delete_block

Das Sicherheitsdesigne der Firewall folgt dem Prinzip, dass bei aktivem NAT keine Verbindung zu einem Rechner im LAN möglich ist, da eine Network Address Translation per Definition dies nicht zulässt. Dies geschieht nicht durch klares Sperren von Verbindungen, sondern durch einfache Logik, da NAT eine frische Verbindung von aussen nicht logisch zu einem Client im LAN zuweisen kann. Aus diesem Grund wurde darauf verzichtet ein Menü zu kreieren, welches Sperrregeln von aussen nach innen definiert. Was aber sicherlich vorhanden sein muss, ist die Möglichkeit Verbindungen von aussen, über einen spezifischen Port, zu einem Client ins Innere zu ermöglichen. Dies wurde über die beiden Menü Punkte Portweiterleitung EIN und Portweiterleitung AUS implementiert. Eine elementare Funktion, welche bei keiner Firewall fehlen darf, ist die Möglichkeit einem Client aus dem LAN die Verbindung zu einer Zieladresse zu sperren. PFDialogControl bietet über die Menü Punkte Verbindung sperren und Verbindung lösen die Möglichkeit, eine spezifische IP – Adresse in Verbindung mit einem Port anzugeben, welche dann nicht mehr zum Outside Interface geleitet werden können. So ist der betreffende Client nicht mehr in der Lage sein Ziel zu kontaktieren. Die Regelsätze wurden aus Performancegründen und der besseren Übersicht halber auf ein Minimum reduziert. So wird mit den beiden PF Parametern „rdr on….“ und „block in quick….“ gearbeitet, obwohl die PF Firewall hier noch viele weitere Kombinationsmöglichkeiten bieten würde. Jedoch sind diese aus Sicht des Entwicklers nicht notwendig bei einer First – Match Firewall. Nachfolgendes Schema soll dies noch verdeutlichen:

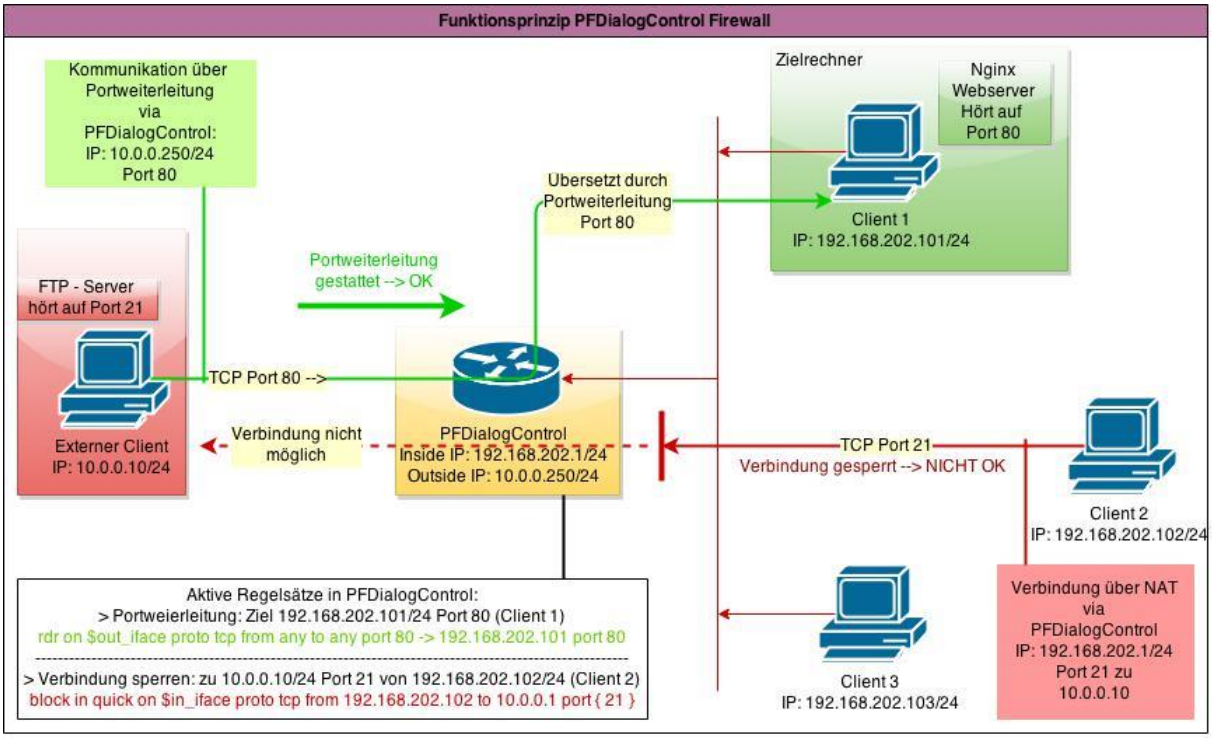

Abbildung 40: Funktionsprinzip der in PFDialogControl implementierten Firewall

➔ Firewall Status:

Die Function status_fw ist als Debugging Funktion implementiert und stellt zwei Statuszustände bereit. Der erste wird mittels dialog Infobox realisiert, indem der zu erzeugende Text direkt in eine Variable als Kommandoaufruf gespeichert wird. Hierbei wird die Befehlskette „**service pf onestatus**“ in der Variable ausgeführt, welche sich als Textoutput im dialog Befehl befindet. Das Ergebnis ist in Abbildung 41 zu sehen, wo diverse Optionen bezüglich zutreffende Regeln, Suchoperationen und die dafür benötigten Zeiten zu sehen sind. Der reale Informationsgehalt dieser Ausgabe ist minimal und selbst für Profis meist nicht von Bedeutung. Die Funktion dient im Grossen und Ganzen rein dekorativen Zwecken. Lediglich bei absoluter Maximalbelastung der Firewall, könnten hier relevante Informationen bezogen werden. Doch aufgrund der hohen Performance von PF ist dies ein eher seltener Anwendungszweck. Jedoch soll diese Informationsquelle Profianwendern nicht vorenthalten werden, da sie in einigen Fällen doch helfen kann einen Engpass in PF zu identifizieren. Für Anfänger ist dies doch etwas schwer bis gar nicht verständlich, weswegen im Info Text dieser Funktion darauf hingewiesen wird, dass dies nur für Profis gedacht ist. Derselbe Auszug erscheint

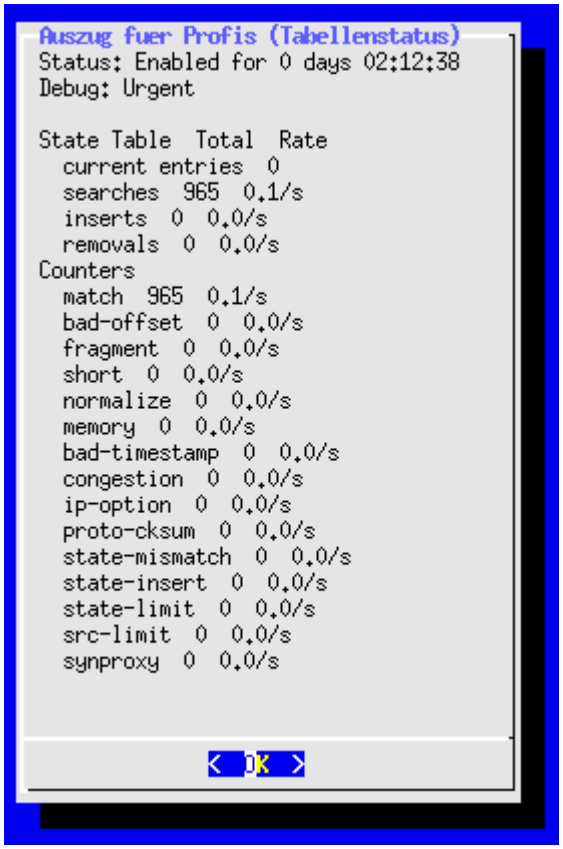

Abbildung 41:Third Level Menü: Firewall Status, Stufe 1

übrigens auch beim Beenden von PFDialogControl, um auf diese Weise die aktuelle Laufzeit der Firewall auszulesen.

Hingegen ist für alle Anwender die folgende Infobox relevant. Sie wird mittels des in Abbildung 42 definierten Info Textes eingeleitet und informiert den User, dass er fünf Sekunden Zeit hat um einen nachfolgenden Auszug aus der Regelsatzprüfung von PF im linken unteren Bildschirmrand zu betrachten. Diese doch etwas unschöne Lösung musste so implementiert werden, da das PF Kernelmodul nach Ausgabe von Laufzeitinformation sich weigert diese in irgendeiner Art und Weise zu manipulieren. So kann die hier gewünschte Information nicht weitergeleitet werden um sie als simplen Text zu plotten. Da die hieraus gewonnenen Informationen aber äusserst wichtig sind für mögliche Regel Eingabefehler, wurde entschieden diese Ausgabevariante so hinzunehmen.

Abbildung 42: Third Level Menü: Firewall Status, Stufe 2

Der Text in Abbildung 42 unterstützt Anfänger, indem er ihnen mitteilt, dass sie sich nicht gross Sorgen über die Bedeutung möglicher Zeilenfehler (Syntax errors) machen müssen, sondern ihre eigegebenen Regeln unter „Portweiterleitung EIN“ und „Verbindung sperren“ nochmals näher betrachten sollen. Profis werden an dieser Stelle schnell ihren Lieblings Editor aufrufen, um in /etc/pf.conf die betreffende Zeile von Hand zu untersuchen.

Der eigentliche Code in Stufe zwei besteht lediglich aus der Befehlskette „**pfctl –f /etc/pf.conf**“, welche die Standard Konfigurationsdatei von PF auf Logik –und Syntaxfehler hin überprüft. Dabei werden beide Arten von Fehlern klar deklariert und mit „Syntax error (mit Zeilennummer)“ oder falsche Reihenfolge ausgeschrieben. Ein nachfolgendes „**sleep 5**“ sorgt dafür, dass der User bequeme fünf Sekunden Zeit hat die Ausgabe zu analysieren.

Ein korrekt konfigurierter Regelsatz wird folgende Ausgabe erzeugen.

```
No ALTQ support in kernel
ALTQ related functions disabled
```

Abbildung 43: Third Level Menü: Firewall Status, korrekter Regelsatz Auszug

Eingabefehler währen an dieser Stelle mit Syntax error deklariert. „Falsche Reihenfolge“ ist bei Verwendung von PFDialogControl nicht möglich. Die beiden Zeilen bezüglich ALTQ betreffen eine PF Funktion welche per Default nicht aktiv ist und spezielle Hardware voraussetzt, welche in virtuellen Maschinen ohnehin nicht zur Verfügung stehen. Deshalb wurde auf die Implementierung dieser Funktion verzichtet.

➔ Portweiterleitung EIN:

Die Startsequenz dieses Menü Punktes ist list_pw, jedoch wurde der besseren Übersicht halber entschieden, gleich mit der Primärfunktion input_pw zu beginnen. Nachfolgendes Flowchart soll einen ersten Überblick geben.

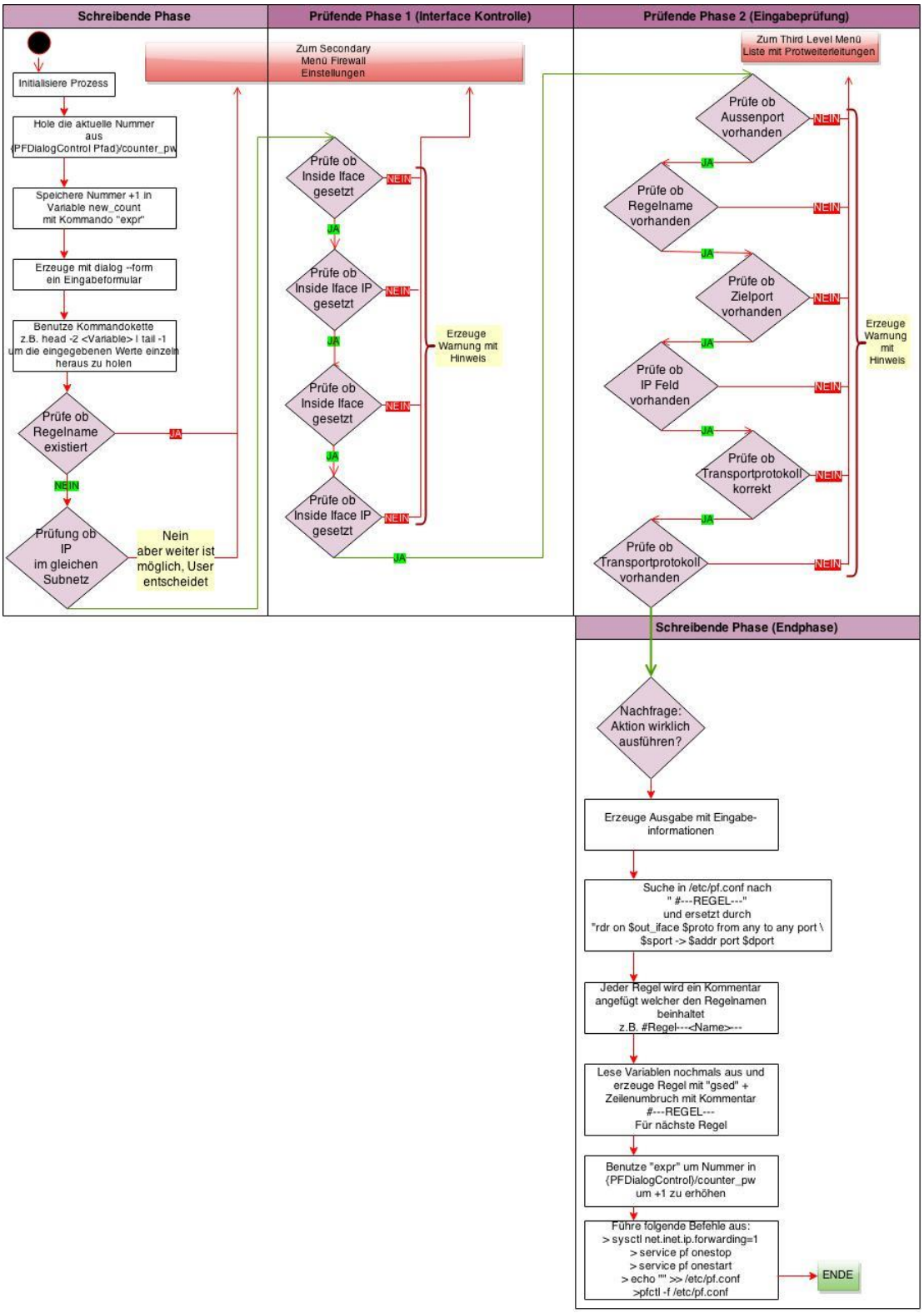

Abbildung 44: Flowchart zu Firewall --> Portweiterleitung Input. Dient als Basis für "Verbindung sperren"

Der besseren Darstellung halber wird nachfolgend Schritt für Schritt im Top – Down Verfahren eine Auflistung der einzelnen Funktionsschritte durchgeführt.

1. Beim Start der Function input_pw aus list_pw kommt man zu einem mit dialog –form (Formular) generierten Menü. Hier müssen die notwendigen Werte eingegeben werden, welche für eine Weiterleitung von aussen nach innen notwendig sind. In diesem Fall; Regelname, Aussenport, Zieladresse, Zielport und Transportprotokoll. Dialog –form speichert diese gleich bei der Eingabe in das neue File. Der Filename ist Regel_<Nummer> und wird unter /usr/local/etc/pfdcontrol/ portfor gespeichert. Die Nummer wird in der Function aus dem File /usr/local/etc/pfdcontrol/counter_pw gelesen, welche mit dem String „1“ bei der Installation erzeugt wurde. Sie wird jedes Mal herausgelesen und in eine Variable gespeichert, die sie automatisch um eins erhöht.

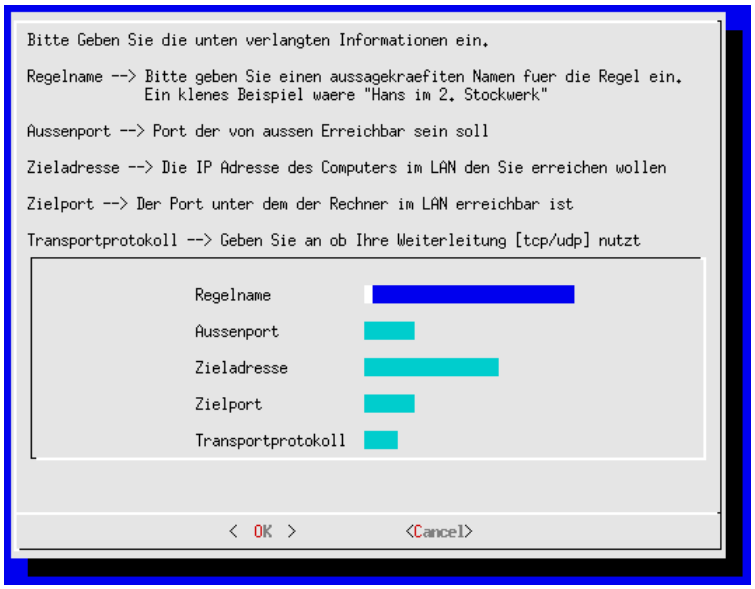

Abbildung 45: Secondary Menü; Firewall, Portweiterleitung Eingabe von Primärwerten

2. Da das neue Regel – File nun schon existiert, kann mittels der Kombination der Befehle „head“ und „tail“ jeweils eine Zeile herausgelesen werden. Dies geschieht am Beispiel „head -2 | tail -1“, indem aus dem File die ersten beiden Zeilen extrahiert werden und mit tail -1 immer nur die letzte Zeile erfasst wird. So kann man sich durch das File Zeile für Zeile arbeiten. Dies funktioniert nur deshalb, da dialog—form per Default zeilenweise speichert. Die herausgelesenen Werte werden nach dem Muster in Abbildung 46 in Variablen gespeichert, welche später in input_pw und list_pw benötigt werden.

```
fname=`head -1 $tmp_pw_input`
sport=`head -2 $tmp_pw_input | tail -1`
addr=`head -3 $tmp_pw_input | tail -1`
dport=`head -4 $tmp_pw_input | tail -1`
proto=`head -5 $tmp_pw_input | tail -1`
```

Abbildung 46: Code Ausschnitt zur Auslesung von dialog --form Input

3. Nun wird mittels zweier verschachtelter if – Anweisungen und der Variable $fname überprüft ob der Regelname bereits existiert. Dazu wird in jedem File in /usr/local/etc/pfdcontrol/portfor mittels der in Abbildung 46 aufgeführten head tail Kombination zur Variable $fname überprüft, ob sich ein String mit dem Input String überschneidet. Falls ja, wird eine Fehlermeldung mittels dialog generiert und das angelegte Regelfile wieder gelöst.
4. Als kleines Gadget wurde an dieser Stelle noch eine Subnetzprüfung implementiert. Diese benutzt die „cat“ und „cut“ Befehlskombination, um aus der Konfigurationsdatei ip.<Inside Interface> die IP zu extrahieren und auf drei Blöcke von links nach rechts zuzuschneiden. Dieser neue Wert wird mit der Eingabe ($addr) verglichen und bei Abweichungen eine entsprechende Warnung mittels dialog –yesno Box generiert. Man kann an dieser Stelle abbrechen oder mittels yes weitermachen, da ja die Möglichkeit besteht, dass das Subnetz grösser ist und somit die IP entsprechend zulässig ist.

5. War alles bis hier hin i.O. startet die in PFDialogControl obligatorische und mehrfach umschriebene Prüfung auf Vorhandensein der Konfigurationen bezüglich Inside/Outside Interface und deren IP –Adressen. Dieser Schritt wäre im Grunde nicht notwendig an dieser Stelle, da er bereits in list_pw ausgeführt wird. Doch die allgemeine Devise des Entwicklers ist stets „sicher ist sicher“, weshalb eine zweite Prüfung nie schadet, da sie kaum Performance kostet.
6. Diese Prüfungen welche in einem umfangreichen if – elif – Block ausgeführt werden sind jedoch von grösster Wichtigkeit. Den hier wird auf folgende Notwendigkeiten geprüft:
    a. Ist ein Regelname eingegeben worden. Nachträgliche Prüfung zu identischem Regelnamen.
    b. Ist ein Aussenport angegeben worden.
    c. Ist der eingegebene Port zulässig, also nicht grösser als 65535.
    d. Ist eine Ziel – IP – Adresse angegeben worden.
    e. Ist ein Zielport definiert.
    f. Ist ein Transportprotokoll eingegeben worden und ist dieses entweder TCP oder UDP.

    Erst wenn diese Bedingungen erfüllt sind, kann mit dem restlichen Code weitergefahren werden. Ansonsten wird zu jedem Punkt eine dialog Warnung generiert, welche den User auf seinen Fehler hinweist.

7. Ist bis hier alles in Ordnung, so kann davon ausgegangen werden, dass die Eingabe korrekt war. Nun wird mittels der unter Abbildung 46 ersichtlichen Variablen eine dialog –yesno Anweisung generiert, welche vom User eine Ausführgenehmigung einfordert. Diese kann wie folgt aussehen:

Abbildung 47: Third Level Menü; Firewall --> Portweiterleitung, Regelbestätigung

8. Der nächste Schritt nutzt wieder die in Abbildung 46 definierten Variablen, um mittels „gsed“ den Such Tag „#---REGEL---„ in /etc/pf.conf zu identifizieren und diesen mit den definierten Variablen zu überschreiben. Zeitgleich wird nach der neuen Regel ein Zeilenumbruch generiert und der Such Tag wieder hergestellt. So kann der Schreibprozess bei einer neuen Regel seine korrekte Position in pf.conf wieder lokalisieren.

```
gsed "s/#---REGEL---/rdr on \$out_iface proto\ $proto from any to any port $sport -> $addr port $dport  # Regel--$fname--\n#---REGEL---/"
```

Abbildung 48: Code Ausschnitt einer möglichen Konstruktionszeile zum Absetzten einer Regel in pf.conf

9. Wie in Abbildung 48 am Ende ersichtlich ist, wird jeder Regel das Such Flag **„# Regel---<Regelname>---„** beigefügt. Dabei dient „Regel--- ---“ als Delimiter um es sed und gsed später zu erleichtern, den darin enthaltenen Regelnamen besser zu identifizieren. Doch dazu später mehr im Abschnitt delete_pw.

10. Wie in Abbildung 35, Seite 55 definiert, kommt an dieser Stelle noch die Reload Befehlskette zum Zug, welche die neue Regel in die dynamische Datenbank von PF aktualisiert.
11. Abschliessen wird mittels des Mathematik Befehls „expr“ die Nummer in counter_pw um eins erhöht, womit sie der aktuellen Regelnummer gleicht. Diese wird später beim Erstellen einer neuen Regel wieder im obersten Abschnitt des Codes nochmals um eins erhöht.

➔ Listing Funktionen (list_pw, list_pw_stop, list_block und list_block_stop):

Wie bereits zu Beginn von input_pw erwähnt, ist diese Function die erste welche ausgeführt wird. Ihr Zweck ist die Auflistung der vorhandenen Regelsätze in „Portweiterleitung“ bzw. „Verbindung Sperren“. Dabei sind die Functions list_pw und list_pw_stop identisch, wobei das gleiche für list_block und list_block_stop gilt. Im Grunde sind alle vier vom funktionellen Design her betrachtet absolut gleich. Jedoch ist bei letzteren beiden der Output ein anderer, da sie Informationen über gesperrte Verbindungen beinhalten. Die Isolierung von identischen Funktionen in eigene Functions wurde erforderlich, da bei der vierten Ebene innerhalb einer while – Schlaufe ein nicht nachvollziehbarer Fehler auftrat. Dieser Fehler verursachte, dass das Working Directory innerhalb des Skripts stets zum Root Accout sprang und somit die benötigten Files nicht mehr auffindbar waren. Durch Kapselung der Funktions und des statischen Setzens des Working Directorys (/usr/local/etc/pfdcontrol/ portfor bzw. block) konnte dieser Fehler behoben werden und zeitgleich ein stabilerer Bildaufbau entstehen.

Da wie bereits erwähnt sich die Functions im Design ähneln, soll an dieser Stelle mithilfe von list_pw die Funktionsweise allgemein erklärt werden. Die spezifischen Unterschiede zu list_block werden im späteren Abschnitt kurz angeschnitten.

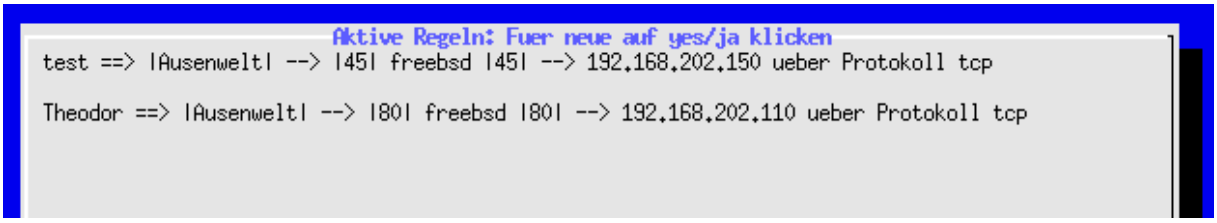

Abbildung 49: Mögliches Listing einer list_<> Function

Der Start einer solchen Function führt immer zur Prüfung auf Vorhandensein der Werte in Inside/Outside Interface bzw. deren IP – Adresse. So ist ein Setzen von Regeln im allgemeinen nicht möglich, da der bereits mehrfach erwähnte if – Block stets zu einer Fehlermeldung führt, welche einen Abbruch mit Hinweis generiert. Ist diese Prüfung erfolgreich, so wird im nächsten Schritt das Verzeichnis wo sich die Regeln befinden (portfor → Portweiterleitung, block → Verbindung sperren) mittels des Befehls „ls“ ausgelesen und die Dateinamen in eine Variable als Liste gespeichert. Diese Liste wird nun in einer for – Schleife abgearbeitet, indem die Dateiinhalte mittels einer ähnlichen head und tail Kombination wie in Abbildung 46, Seite 61 einzeln ausgelesen werden. Dabei wird wieder jede Zeile einzeln in eine Variable abgepackt, welche ihren exakten Platz im Output bekommt. Diese Outputzeile wird mittels „echo“ in ein Tempfile geschrieben. Dies geschieht mit dem Inhalt jeder Datei, welche in den Ordnern portfor und block identifiziert wurde. Sind alle Outputzeilen in das Tempfile geschrieben worden, kann eine dialog –yesno Box generiert werden, welche als Text eine Variable beinhaltet die mittels „cat“ den Inhalt des Tempfiles ausgibt. So wird Zeile für Zeile in das dialog Menü geschrieben. Das Ergebnis ist in Abbildung 49 zu sehen. Bestätigt man nun mit „yes“ um eine neu Regel zu generieren, so wird das Tempfile gelöscht und man gelangt zur bereits beschreibenen Function input_pw. Trotz der Tatsache, dass dieser Prozess sehr schreibintensiv ist,

konnten auch bei einer grossen Anzahl an Regeln keine Geschwindigkeits –oder Performanceeinbussen festgestellt werden. Nachfolgend ist ein kleiner Codeausschnitt zu sehen, welcher die gesamte Aufbau Funktionalität aller Listing Funktionen im Allgemeinen, am Beispiel von list_pw zeigt.

```
while true; do
cd /usr/local/etc/pfdcontrol/portfor/

list=$(ls /usr/local/etc/pfdcontrol/portfor/)

for i in $list
  do
    fname=`head -1 $i`
    sport=`head -2 $i | tail -1`
    addr=`head -3 $i | tail -1`
    dport=`head -4 $i | tail -1`
    proto=`head -5 $i | tail -1`

    echo "$fname ==> |Ausenwelt| --> |$sport| $(hostname) |$dport| --> $addr ueber Protokoll $proto">> /usr/local/etc/pfdcontrol/tmp_pw_list
    echo "" >> /usr/local/etc/pfdcontrol/tmp_pw_list
  done


  dialog --backtitle "PFDialogControl"\
     --cancel-label "Zurueck" \
     --title "Aktive Regeln: Fuer neue auf yes/ja klicken" --yesno "$(cat /usr/local/etc/pfdcontrol/tmp_pw_list)" 30 100

  case $? in
    0) rm /usr/local/etc/pfdcontrol/tmp_pw_list; input_pw;
    ;;
    1) rm /usr/local/etc/pfdcontrol/tmp_pw_list; menu_fw;
    ;;
  esac
done
cd /usr/local/etc/pfdcontrol/portfor/
```

Abbildung 50: Allgemeines Design Beispiel für die vier Listing Funktionen

➔ Portweiterleitung AUS (Verbindung lösen):

Die Functions delete_pw und delete_block sind vom Design her identisch, da beide in pf.conf schreibend eingreifen. Was sich unterscheidet, ist der Such Flag welcher bei delete_pw „Regel--<Regelname>--„ und bei delete_block „Filter--<Regelname>--„ ist. Ansonsten ist das funktionelle Design der Functions dasselbe. Aus diesem Grund findet die Dokumentation beider Functions (delete_pw und delete_block) parallel an dieser Stelle statt.

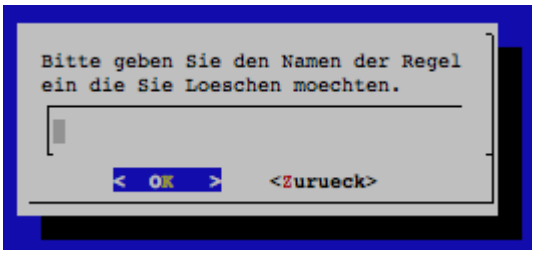

Abbildung 51: Third Level Menü: Firewall --> Portweiterleitung AUS, Untermenü gestartet aus list_pw_stop

Startet man Portweiterleitung AUS (bzw. Verbindung lösen) aus einem der Listing Menüs mittels „yes“, so wird als erstes eine dialog –yesno Box generiert mit der Aufforderung einen Regelnamen anzugeben. Ist die yesno – Box ohne Eingabe betätigt worden, so prüft eine if –Anweisung ob der Inhalt in der temporären Eingabevariable grösser als null ist (Prüfoption „-z“). Sollte dies zutreffen, kann eine Leereingabe abgefangen und mittels dialog –infobox darauf hingewiesen werden.

Ist eine Eingabe vorhanden, also Variabel Inhalt grösser als null, muss dieser noch auf Vorhandensein eines Regelnamens überprüft werden. Hierzu kommt eine for – Schleife zum Zuge, ähnlich der in Abbildung 50, Seite 64. Es wird der Inhalt des betreffenden Verzeichnisses (…/portfor bzw. …/block) mittels „ls“ in eine Variable gespeichert, welche als Input für die for – Schleife dient. Innerhalb dieser for – Schleife wird eine if – Anweisung geprüft, welche in jedem gefunden File in /portfor bzw. /block mittels der Suchkombination „head -1“ den Regelnamen extrahiert und mit dem Eingabewert der yesno – Box vergleicht. Endet die for – Schleife ohne Treffer, so wird in die Variable „chk_input_name_(pw bzw. block)“ nichts gespeichert und der Default Wert bleibt auf „10“. So kann die nachfolgende if – Anweisung eine Prüfung auf den besagten Variablenwert vollziehen und bei Übereinstimmung (Wert = 10) die Fehlermeldung in Abbildung 52, mittels dialog –infobox generieren.

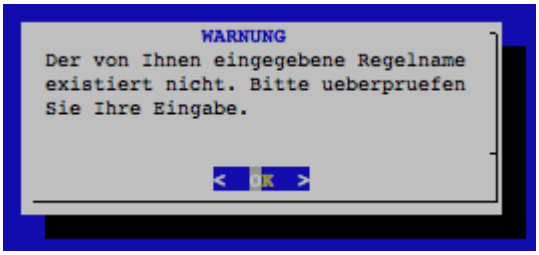

Abbildung 52: Third Level Menü: Portweiterleitung AUS bzw. Verbindung lösen, Regelname existiert nicht!

Sollte die Eingabe aber als existierender Regelname identifiziert worden sein, wird der Variabel Wert von chk_input_name_(pw bzw. block) auf 5 gesetzt, was dazu führt das die oben genannte if – Anweisung nicht erfüllt wird und somit weiter im Code gefahren werden kann. Nun kann der eigentliche Löschvorgang eingeleitet werden, welcher aus zwei Stufen besteht. In Stufe eins wird mittels „sed“ eine Zeile generiert, welche den Such Flag beinhaltet. Dieser ist bei Portweiterleitung AUS ➔ Regel--<Regelname>-- und bei Verbindung lösen ➔ Filter--<Regelname>--. Diese Tags werden als Kommentar, jeder mittels „Portweiterleitung EIN“ (input_pw) und „Verbindung sperren“ (input_block) erzeugten Regel am Zeilenende mitgegeben. Filter--…-- und Regel--…-- dienen als Delimiter welche dafür sorgen, dass ein Regelname sich nicht mit einem Funktionsparameter in pf.conf überschneidet oder Regelnamen mit teilweise gleichem Inhalt gleichzeitig mitgelöscht werden. Kann der so erzeugte Such Flag in pf.conf identifiziert werden, wird die gesamte Zeile mittels der in Abbildung 53 generierten sed – Anweisung gelöscht.

```
sed "/Regel--$(cat $del_input)--/d"
```

Abbildung 53: Code Ausschnitt: sed Anweisung zum Zeile in pf.conf zu löschen.

Somit ist die Regel in einem ersten Schritt aus pf.conf gelöscht. Nun muss noch das Regelfile in …/portfor bzw. …/block entfernt werden, welches die gespeicherten Werte der Regel beinhaltet. Dies geschieht mit einer for – Schleife welche dem Arbeitsmuster der oben beschriebenen Namensprüfung entspricht. Es wird wieder mit „head -1“ der Regelname extrahiert und mit der Eingabe verglichen. Entspricht eine Datei dem Eingabewert, wird diese mittels „rm $i“, wobei $i dem aktuell gefundenen Dateinamen entspricht, gelöscht.

Nun muss nur noch PF neu geladen werden um die Regelsätze zu aktualisieren und die Firewall kann mit einem Unterbruch im Millisekunden – Bereich weiterarbeiten.

```
del_pw=`ls /usr/local/etc/pfdcontrol/portfor/`
for i in $del_pw
  do
    str_search_pw=`head -1 $i`
    if [ "$str_search_pw" = "$(cat $del_input)" ]; then
            rm $i
    fi
done
```

Abbildung 54: Codeausschnitt Portweiterleitung, Löschvorgang Stufe 2

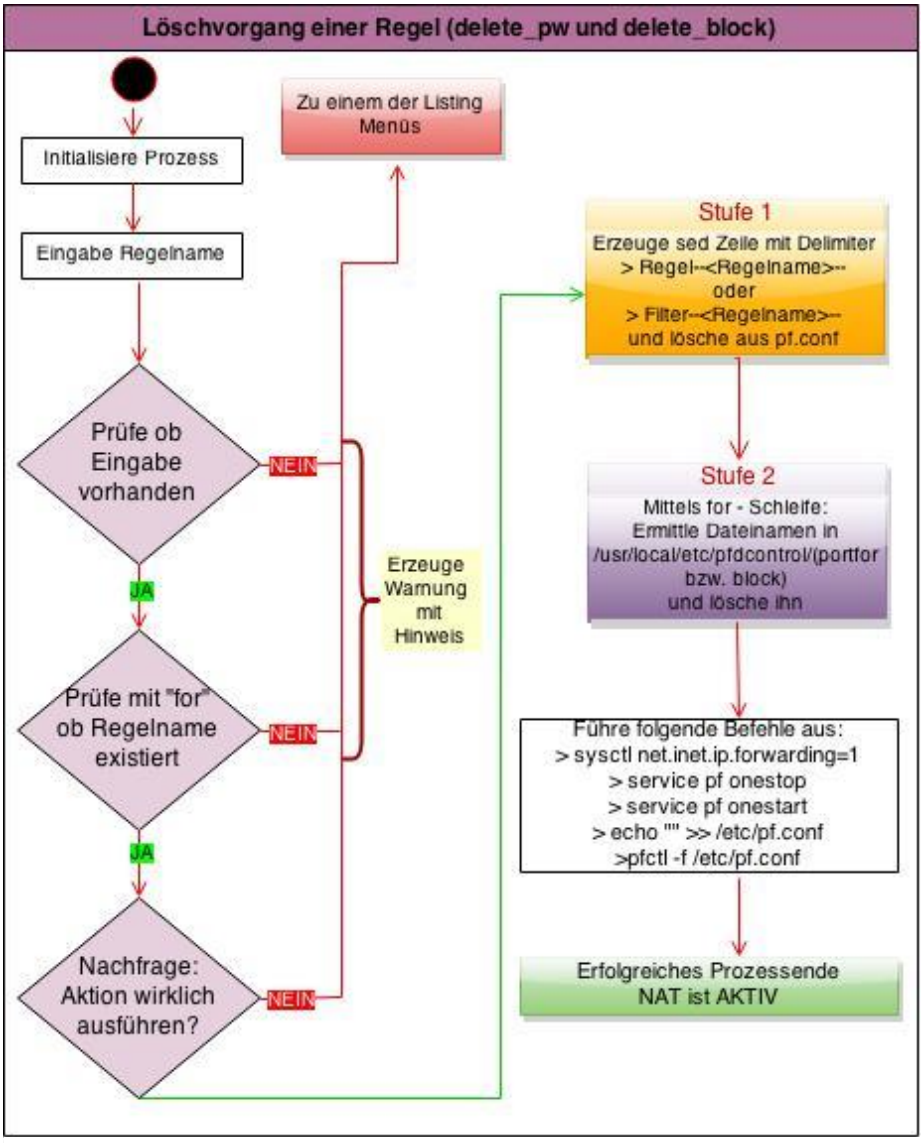

Abbildung 55: Flowchart zu Regel löschen. Gilt allgemein für "Portweiterleitung AUS" und "Verbindung lösen"

➔ Verbindung sperren (Designvergleich zu Portweiterleitung EIN):

Da die meisten Functions im Arbeitsdesign gleich bis ähnlich sind, wurde versucht die Dokumentation etwas zusammenzufassen. So konnten die designtechnischen Arbeitsweisen übersichtlich und parallel erklärt werden. Dasselbe gilt auch für die Funktion input_block, welche dem Arbeitsmuster von input_pw entspricht. Da die eigentliche Funktion aber eine andere ist, sollen an dieser Stelle die relativen Unterschiede zu „Portweiterleitung EIN“ erläutert werden. Hierbei gilt das Arbeitsdesign von Abbildung 44, Seite 60 (Flowchart Portweiterleitung) sowie die Auflistung der Arbeitsschritte weiterhin. Es bestehen aber folgende Unterschiede:

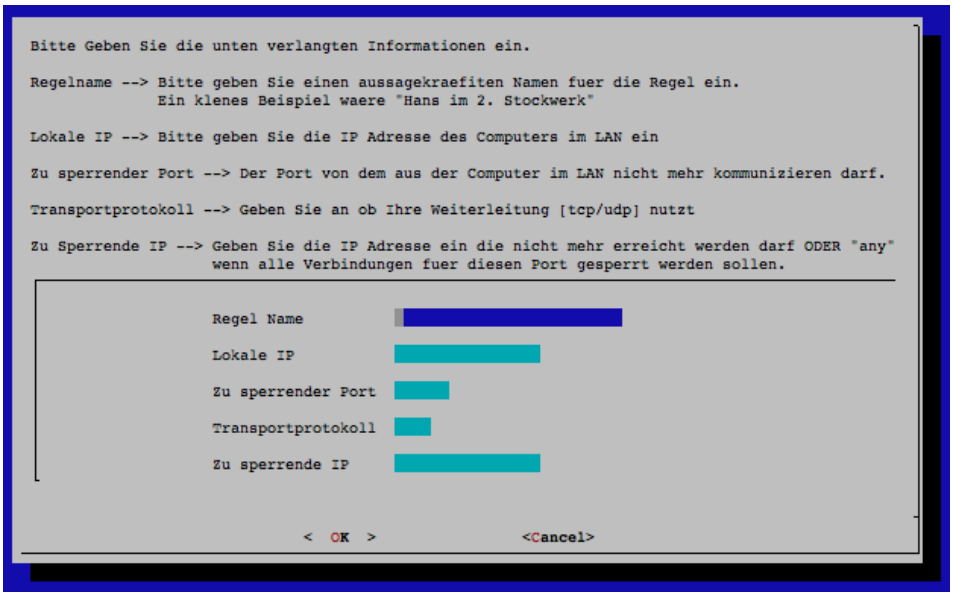

Abbildung 56: Secondary Menü; Firewall, Verbindung sperren Eingabe von Primärwerten

- Da hier eine Verbindung gesperrt werden soll, müssen folgende Parameter gesetzt werden:
  - Regelname, wie üblich
  - Lokale IP – Adresse, ist an dieser Stelle ähnlich implementiert wie in „Portweiterleitung EIN“
  - Zu sperrender Port, ebenfalls ähnlich implementiert
  - Transportprotokoll, entspricht der gleichen Implementierung bezüglich Sicherungen wie in „Portweiterleitung EIN“
  - Zu sperrende IP, ist ein neuer Punkt an dieser Stelle
- Das Speichern erfolgt auf die gleiche Art und Weise wie allgemein üblich bei der Verwendung von dialog —form Boxen. Jedoch kommt hier das Default Working Directory /usr/local/etc/pfdcontrol/block, anstelle von …/portfor wie in „Portweiterleitung EIN“ zum Zuge.
- Die head tail – Kombination ist funktionstechnisch die gleiche wie in Portweiterleitung. Da die Anzahl an Parametern dieselbe ist, kommt auch die gleiche Nummerierung (z.B. head -2 | tail -1) zum Einsatz. Der Unterschied liegt lediglich in einem Teil der Variablennamen die wie in Abbildung 56 gesetzt sind.
- Was im Vergleich zu in Abbildung 58 (oben) ebenfalls ändert, ist die erzeugte „gsed“ Zeile welche aufgrund der unterschiedlichen Variablennamen und des neuen PF Sperrbefehls anders lautet. Die neu erzeugte Zeile trägt nun den Anfang „block in quick“ und erzwingt im FreeBSD Network Stack eine Sperrung

```
bname=`head -1 $tmp_block_input`
sip=`head -2 $tmp_block_input | tail -1`
sport=`head -3 $tmp_block_input | tail -1`
proto=`head -4 $tmp_block_input | tail -1`
tip=`head -5 $tmp_block_input | tail -1`
```

Abbildung 57: Code Ausschnitt zur Auslesung von dialog --form Input. Im Vergleich zu Portweiterleitung EIN unten.

```
fname=`head -1 $tmp_pw_input`
sport=`head -2 $tmp_pw_input | tail -1`
addr=`head -3 $tmp_pw_input | tail -1`
dport=`head -4 $tmp_pw_input | tail -1`
proto=`head -5 $tmp_pw_input | tail -1`
```

des Paketflusses, im Falle einer Übereinstimmung. Nachfolgend ein kleiner Vergleich der gsed – Zeile in „Verbindung sperren“ zu „Portweiterleitun EIN“:

```
gsed "s/#---FILTER---/block in quick on \$in_iface\ proto $proto from $sip to $tip port { $sport }  # Filter--$bname--\n#---FILTER---/"
```

Abbildung 58: Code Ausschnitt einer möglichen Konstruktionszeile zum Absetzten einer Regel in pf.conf, im Vergleich zu "Portweiterleitung EIN" unten

```
gsed "s/#---REGEL---/rdr on \$out_iface proto\ $proto from any to any port $sport -> $addr port $dport  # Regel--$fname--\n#---REGEL---/"
```

- Auch hier wird ein Delimiter als Kommentar hinzugefügt um die Regel später identifizieren und löschen zu können. Jedoch heisst dieser „Filter--<Regelname>--“ und wird in pf.conf im dritten Abschnitt (am Schluss) gespeichert.
- Der Zähler für die Regelfiles ist unter /usr/local/etc/pfdcontrol/counter_block erreichbar.

Das war es schon mit den Unterschieden. Alle Prüfrutinen bezüglich existierender Regelnamen, Schreibbestätigung, Erhöhung des Counters und natürlich die Tatsache, dass auch diese Function aus einer Listing Funktion (list_block) heraus gestartet wird, sind funktionstechnisch identisch.

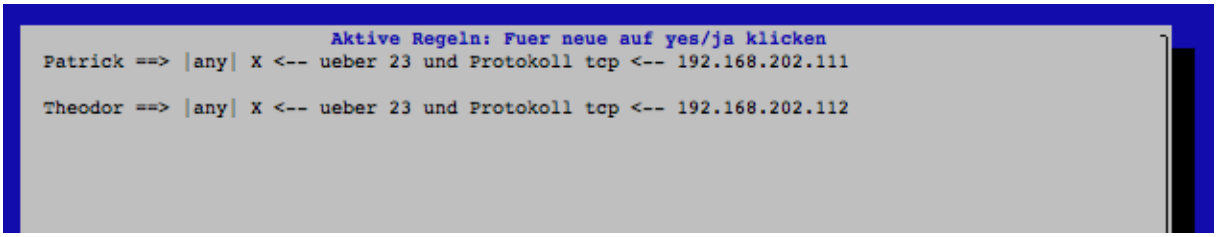

Abbildung 59: Darstellung einer Verbindungssperrung in Listing "Verbindung sperren" (Function: list_block)

➔ Allgemeines zu den Statusleisten (dialog --gauge):

Die Statusleisten werden beim Aufruf der Menüs Interface Einstellungen, DHCP/DNS Einstellungen und Firewall Einstellungen als eigene Functions gestartet. Ihre Namen sind chk_iface_(network, dnsmasq oder fw) und sie befinden sich zu oberst in jedem Menüblock.

Abbildung 61: Statusanzeige Interface Einstellungen

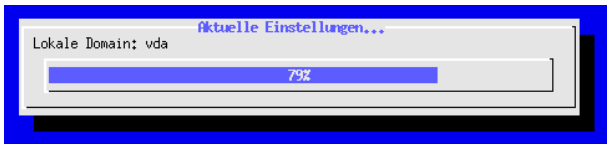

Abbildung 60: Statusanzeige DHCP/DNS Einstellungen

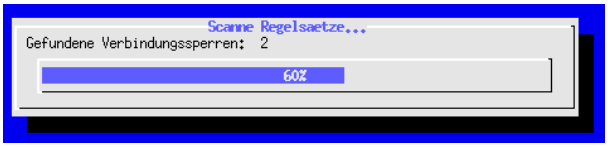

Abbildung 62: Statusanzeige Firewall Einstellungen

Ihre Implementierung ist rein statischer Natur wie am Bespiel von chk_iface_fw in Abbildung 63 zu sehen ist. Dies bedeutet, dass sämtliche Zeitwerte welche die dialog –gauge Funktion bezüglich Fortschritt anzeigt, in den Zeilen wie echo 1; sleep 0.16 Sekunden definiert sind. Man führt also eine echo – Anweisung mit einem Zahlenwert, welcher dem prozentualen Anzeigewert in der Statusleiste entspricht aus und pipet diesen zur dialog –gauge Funktion. Der sleep Befehl sorgt dafür, dass die –gauge Anzeige für einen Moment verharrt. Meist folgt nach der Steuerzeile eine Befehlszeile, welche einen Text plottet, wie bspw. „Gefundene Verbindungssperren: 2“. Um einen solchen Text in die –gauge Funktion zu bekommen, muss eine echo – Zeile mit dem String „XXX“ gesendet werden sowie auch entsprechend mit dem gleichen Trenner wieder beendet werden. Dazwischen befindet sich der Text, welcher ebenfalls als echo – String mitgegeben werden muss. Meist wird dem Text noch eine mittels der bewährten cat/cut Kombination, formatierter Suchstring mitgegeben. Ein gutes Beispiel eines solchen Suchstrings ist die erste Zeile des Codeblocks in Abbildung 63. Dabei wird innerhalb einer Variablen mittels „cat“ ein Auszug generiert, welcher dann mit „grep“ auf einen Suchbegriff eingeschränkt werden kann, der wiederum mittels „cut“ auf die gewünschte Formatierung zugeschnitten wird. Nach dem Prinzip der Stapelverarbeitung wird dann Text um Text in die –gauge geschrieben. Das Ergebnis ist meist ein Auszug aus den verschiedenen Config Files in /usr/local/etc/pfdcontrol.

```
chk_iface_fw() {
    {
      echo "XXX"; echo "Aktive NAT Regeln: $(cat /etc/pf.conf | grep "nat on" | wc -l | cut -d " " -f7-9)"; echo "XXX"
      echo 1; sleep 0.18; echo 5; sleep 0.18; echo 13; sleep 0.18; echo 12 ; sleep 0.18
      echo 13; sleep 0.18; echo 16; sleep 0.18; echo 20; sleep 0.18; echo 26; sleep 0.18
      echo "XXX"; echo "Gefundene Portweiterleitungen: $(cat /etc/pf.conf | grep "rdr on" | wc -l | cut -d " " -f7-9)"; echo "XXX"
      echo 30; sleep 0.18; echo 36; sleep 0.18; echo 40; sleep 0.18; echo 45; sleep 0.18
      echo 47; sleep 0.18; echo 49; sleep 0.18; echo 50; sleep 0.18; echo 55; sleep 0.18
      echo "XXX"; echo "Gefundene Verbindungssperren: $(cat /etc/pf.conf | grep "block in quick" | wc -l | cut -d " " -f7-9)"; echo "XXX"
      echo 57; sleep 0.18; echo 60; sleep 0.18; echo 63; sleep 0.18; echo 67; sleep 0.18
      echo 79; sleep 0.18; echo 82; sleep 0.18; echo 86; sleep 0.18; echo 94; sleep 0.18
      echo "XXX"; echo Komplett; echo "XXX"
      echo 100; sleep 1
      } | \
      dialog --backtitle "PFDialogControl" --title "Scanne Regelsaetze..." --gauge ""  6 75 0
}
```

Abbildung 63: Code Ausschnitt der Stautusleiste von chk_iface_fw (Firewll)

Die Leisten erfüllen keinen relevanten Anteil an der Funktionsweise der betreffenden Menüs. Ihre Funktion ist rein informativer Natur und soll dem etwas unerfahreneren User eine gewisse Sicherheit geben. Da er die von ihm eingegebenen Konfigurationswerte innerhalb der Statusleiste sieht, soll er davon ausgehen, dass sie korrekt sind und PFDialogControl ordnungsgemäss Funktioniert.

Es wurde zu Beginn versucht die dialog –gauge Funktion real in Functions zu implementieren, um so mit dem Ausführen eines Prozesses auch den aktuellen Status zu erhalten. Doch gab es nur eine Funktion (Interface Einstellungen → Scannen der Verfügbaren Interfaces) wo dies Sinn gemacht hätte. Diese eine Funktion musste aber dennoch Zeitangaben mittels „sleep“ aufweisen, was sie erstens unberechenbar machte bei einer grossen Anzahl an auszuführenden Prozessen, da so die statische Zeit nicht exakt bestimmt werden konnte und je nach Anzahl der Vorgänge zu lange dauerte oder bei einer kleinen Prozesszahl keine Zeit blieb den Text zu erkennen. Zweitens wurde der Prozess der Function stark instabile und führte zu unerwarteten Fehlermeldungen.

8.3.2.5 Ausgelagerte Functions

Die nachfolgende Functions wurden der besseren Übersicht halber in eigene Files ausgelagert, welche meist von libpfdctl bei Gebrauch eingebunden werden können. Ihr Default Working Directory ist /usr/lib, da sie aus Sicht des Entwicklers als Librarys zu sehen sind.

➔ Libchkin12:

Die primäre Funktion der Function chk_iface_name ist das Überprüfen von eingegebenen Interfacenamen auf ihre Existenz. Dabei ist die Funktionsweise statisch und simpel wie im kleinen Beispielausschnitt in Abbildung 64 zu sehen ist. Der eingegebene Interfacename wird in eine Variable gespeichert und dann in einem 12 – Zeiligen if – elif – Anweisungsblock jeweils auf Übereinstimmung geprüft. Hierbei wird das Konfigurationsfile „identified_ifaces“, welches alle durch PFDialogControl erkannten Interfaces beinhaltet ausgelesen und als Liste in eine Variable gespeichert. Nun kann mittels der cat/cut – Befehlskombination, Interface um Interface aus der Variable geholt und verglichen werden. Passt eines davon wird eine dialog Massagebox erzeugt, welche den Namen und eine Interface Verifizierung plottet. Zeitgleich wird in das Tempfile /usr/local/etc/pfdcontrol/tmp_iface_chk der String „OK“ geschrieben. Dieser ist für eine unabhängige if – Anweisung am Ende des Skripts wichtig, welche prüft ob das Tempfile existiert. Sollte dies nicht auffindbar sein, so muss der if – Block davon ausgehen, dass der eingegebene Interfacename von dem oberen if – elif – Block nicht gefunden wurde und somit auch nicht existiert. Die erzeugte dialog Massagebox Ausgabe beinhaltet den eingegeben Namen und einen Hinweis auf Nichtexistenz des Interfaces (siehe Code Ausschnitt Abbildung 65).

Die statische Prüfung ist auf zwölf Interfaces beschränkt, da der Entwickler der Meinung ist, dass dies für das Konzept von PFDialogControl ausreichend ist.

```
nics=`cat /usr/local/etc/pfdcontrol/identified_ifaces`
interface_name=`cat $iface_name`

[ $interface_name = $(echo $nics | cut -d " " -f1) ]
      if [ $? = 0 ]; then
       dialog --backtitle "PFDialogControl" \
                --title "Betstaetigt" --msgbox "Der von Ihnen eingegebene Interface Name --> $interface_name kann als existent verifiziert werden." 10 35
            echo "OK" > /usr/local/etc/pfdcontrol/tmp_iface_chk
      fi
```

Abbildung 64: Code Ausschnitt 1 aus libchkin12

```
iface_OK_file=/usr/local/etc/pfdcontrol/tmp_iface_chk
if [ ! -e $iface_OK_file ]; then
   dialog --backtitle "PFDialogControl"\
      --title "WARNUNG" --msgbox "Der von Ihnen eingegebene Interface Name --> $interface_name kann NICHT als existent verifiziert werden.
   rm /usr/local/etc/pfdcontrol/tmp_iface_chk 2> /dev/null
   menu_network && exit
fi
   rm /usr/local/etc/pfdcontrol/tmp_iface_chk 2> /dev/null
```

Abbildung 65: Code Ausschnitt 2 aus libchkin12

Anmerkung: Diese Funktion wurde durch den Entwickler implementiert, da bei der Entwicklung durch Falscheingabe des Interfacenamens zwei Mal je 20 Minuten verloren gingen. Deshalb wurde beschlossen, jede Interfaceeingabe zu verifizieren bzw. mit einer Hinweismeldung abzulehnen, damit mögliche Fehleingaben schneller erkannt werden.

➔ Libnmcalc16:

Die Funktion von libnmcalc16 ist die Übersetzung der konventionellen blockbasierten Schreibweise der Subnetzmaske, in die Bit basierte Notation. Beispielsweise entspricht die blockbasierte Netzmaske 255.255.0.0, der Bit basierten Schreibweise /16 bit.

Hierbei wird wieder auf eine statische if – elif – Anweisung zurückgegriffen, welche in Abbildung 66 ersichtlich ist. Diese Function ist in input_netconfig_netmask integriert und wird am Ende der Funktion ausgeführt. Dabei liest sie die eingegebene blockbasierte Netzmaske aus der Variable aus und versucht eine passende Stelle im Anweisungsblock zu finden. Wurde eine Stelle gefunden, wird der Bit Wert mittels „echo“ in ein File namens suffix.<*eingegebener Interfacename*> geschrieben. Kam es zu keiner Identifikation, so kommt der Fallback Modus zum Zuge, welcher als else – Abschnitt des Anweisungsblockes implementiert ist. Dieser setzt stets den Bit Wert „24“ in das besagte Suffix File. So können Falscheingaben seitens eines Anfängers immer abgefangen werden und der Betrieb ist mit der heute üblichen Subnetzbreite von 24 Bit (254 Clients) gewährleistet.

Der Suffix Wert wird primär für den Betrieb von NAT und der Firewall benötigt, da das Inside Interface, welches als Makro in pf.conf gespeichert ist, diese Notation benötigt.

Die maximal erkennbare Suffix – Breite umfasst 16 – 30 Bit (255.255.0.0 – 255.255.255.252), was aus Sicht des Entwicklers für PFDialogControl ausreicht. Der Startpunkt von 16 Bit an aufwärts ist der ausschlaggebende Punkt für das 16 im Namen der Library.

```
calc_netmask() {
if [ $netmask = "255.255.0.0" ]; then
    echo "16" > /usr/local/etc/pfdcontrol/suffix.$run
elif [ $netmask = "255.255.128.0" ]; then
    echo "17" > /usr/local/etc/pfdcontrol/suffix.$run
elif [ $netmask = "255.255.192.0" ]; then
    echo "18" > /usr/local/etc/pfdcontrol/suffix.$run
elif [ $netmask = "255.255.224.0" ]; then
    echo "19" > /usr/local/etc/pfdcontrol/suffix.$run
elif [ $netmask = "255.255.240.0" ]; then
    echo "20" > /usr/local/etc/pfdcontrol/suffix.$run
elif [ $netmask = "255.255.248.0" ]; then
    echo "21" > /usr/local/etc/pfdcontrol/suffix.$run
elif [ $netmask = "255.255.252.0" ]; then
    echo "22" > /usr/local/etc/pfdcontrol/suffix.$run
elif [ $netmask = "255.255.254.0" ]; then
    echo "23" > /usr/local/etc/pfdcontrol/suffix.$run
elif [ $netmask = "255.255.255.0" ]; then
    echo "24" > /usr/local/etc/pfdcontrol/suffix.$run
elif [ $netmask = "255.255.255.128" ]; then
    echo "25" > /usr/local/etc/pfdcontrol/suffix.$run
elif [ $netmask = "255.255.255.192" ]; then
    echo "26" > /usr/local/etc/pfdcontrol/suffix.$run
elif [ $netmask = "255.255.255.224" ]; then
    echo "27" > /usr/local/etc/pfdcontrol/suffix.$run
elif [ $netmask = "255.255.255.240" ]; then
    echo "28" > /usr/local/etc/pfdcontrol/suffix.$run
elif [ $netmask = "255.255.255.248" ]; then
    echo "29" > /usr/local/etc/pfdcontrol/suffix.$run
elif [ $netmask = "255.255.255.252" ]; then
    echo "30" > /usr/local/etc/pfdcontrol/suffix.$run
else
    echo "24" > /usr/local/etc/pfdcontrol/suffix.$run
fi
}
```

Abbildung 66: Kompletter Code Ausschnitt der Function libnmcalc16

8.3.3 Nachträgliche Änderungen

Wie meistens üblich in der Entwicklung erkennt man manche Zusammenhänge erst später bei der Dokumentation eines nachfolgenden Schrittes. So sollen minimale Änderungen, welche keine schwerwiegenden Abweichungen bezüglich der Funktionsweise einer Funktion haben, aber dennoch von der Dokumentation abweichen, nachfolgend dokumentiert werden.

8.3.3.1 Interface Statusinformation in Menü Firewall Einstellungen

Nach der Dokumentation der Menüs NAT und DHCP/DNS fiel dem Entwickler auf, dass eine Statusmeldung auch bei „Firewall Einstellungen“ nett aussehen würde. Dies ist aus technischer Sicht nicht notwendig, da die Prüfungen wie bereits dokumentiert innerhalb der Listing Funktionen durchgeführt werden. Der aus dieser Ergänzung gewonnene Informationsgehalt ist zwar nur bescheiden, aber es ist ein nettes kleines optisches Extrafeature, welches das Konzept von Staus Informationen innerhalb der Menüs abrundet.

Die Implementierung ähnelt der in „NAT Einstellungen“, wo ein „ifconfig“ Aufruf innerhalb einer if – Anweisung genutzt wird, um mittels „grep inet“ zu prüfen ob das Interface online ist. Diese Prüfung wird auf das Outside sowie auf das Inside Interface angewendet. Abbildung 68 zeigt den ergänzten Code, inklusive der Textpassage im dialog Textoutput (Abbildung 67).

```
ifconfig $(cat /usr/local/etc/pfdcontrol/outside_iface) | grep inet
if [ "$?" != "0" ]; then
  varRUNNING_OUT="Interface INAKTIV!"
else
  varRUNNING_OUT="Interface aktiv"
fi

ifconfig $(cat /usr/local/etc/pfdcontrol/inside_iface_iface) | grep inet
if [ "$?" != "0" ]; then
  varRUNNING_IN="Interface INAKTIV"
else
  varRUNNING_IN="Interface aktiv"
fi
```

Abbildung 67: Ergänzung in menu_fw um Status zu generieren

```
Status Outside Interface : $varRUNNING_OUT\n \
Status Inside Interface: $varRUNNING_IN" 21 130 21\
```

Abbildung 68: Ergänzender Text, durch zusätzliche Variablen im Menü Output

Abbildung 69: Neues Design von menu_fw. Zu erkennen sind die Zeilen "Status Outside/Inside Interface: Interface INAKTIV/aktiv

9 Funktionale Testphase

9.1 Testumgebung/ Testaufbau

Für die Durchführung aller relevanten Tests werden neben PFDialogControl auch die nachfolgenden Komponenten benötigt:

- IBM 3650 M2 Server: Hier wird mittels eines SmartOS Live Systems eine virtuelle Umgebung erzeugt, welche die Inside Clients und PFDialogControl betreibt.
- D – Link WLAN Router des Typs DIR – 615: Dieser dient der Managementverbindung zu SmartOS, sowie der Mobilität der gesamten Test/ -Betriebsumgebung.
- Neutrales Gerät: In diesem Fall das MacBook Pro 9.2 des Entwicklers, mit dessen Hilfe die Kontrollverbindung und einige Tests durchgeführt werden.
- Softwareseitige Komponenten:
    - PFDialogControl in der endgültigen Version 1.0.3
    - Ein KVM Abbild eines Debian Linux Version 7.x.x innerhalb der Smartos Virtualisierung. (Eine zweite Debian Instanz wird als Notfall Backup vorhanden sein, um schneller auf Fehlkonfigurationen reagieren zu können.)
    - Ein SmartOS Container des Typs base64 (entspricht Solaris Zones), welcher ausserhalb des virtuellen Switches sein wird um „Portweiterleitung EIN/AUS“ zu testen.
    - Eine OpenSUSE 13.2 Installation: Diese ist auf dem Heimrechner des Entwicklers installiert. Hier können Verbindungssperren getestet werden.

Mit den oben genannten Hard –bzw. Softwarekomponenten können sämtliche Tests nach Pflichtenheft (5.2.1 Muss- / Wunschziele, Seite 12), aber auch allgemein betrachtet Funktionstests durgeführt werden.

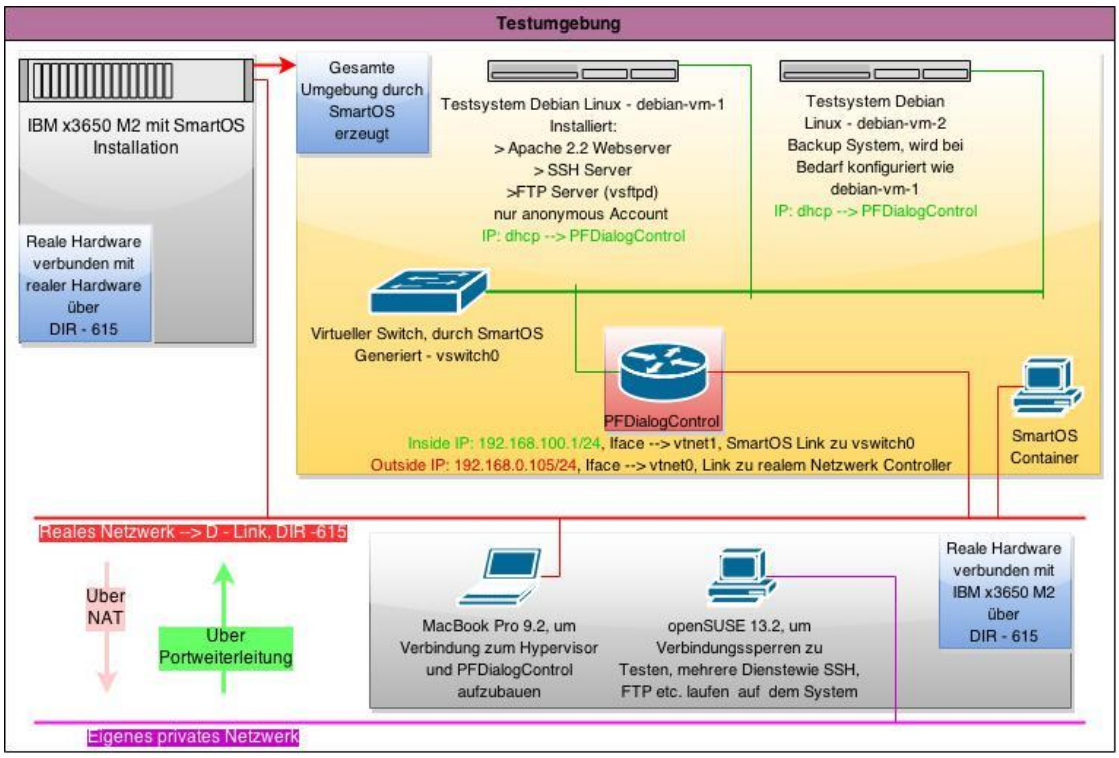

Abbildung 70: Aufbau der Testumgebung zu 9.1

9.2 Testkonfiguration

Die Testkonfiguration ist in drei Teile gegliedert und umfasst primär die Kategorien Netzwerkkonfiguration sowie Dienstkonfiguration. Nachfolgend eine Auflistung der Konfigurationen:

- Netzwerkseitige Konfiguration:
  - SmartOS: Der Hypervisor erzeugt drei virtuelle Switches, wobei für die Testkonfiguration nur vswitch0 benötigt wird. Dieser ist an keinen realen Link gebunden und existiert nur in der Netzwerk – Virtualisierungsschicht (Crossbow) von SmartOS. Ein Netzwerkanschluss ist so konfiguriert, dass SmartOS unter einer IP – Adresse kontaktierbar ist, sowie ein Link zu den virtuellen Maschinen möglich wird, welchen diese nutzen können um eine Verbindung zu DIR – 615 herzustellen (z.B. PFDialogControl).
  - Reale Hardware: Diese nutzt die „öffentliche“ Verbindung über DIR – 615 mittels Portweiterleitung um in das Testnetzwerk zu kommen. Das MacBook Pro 9.2 kann aber auch direkt ins Testnetzwerk über die WLAN Funktion von DIR – 615 gelangen.
  - Test Container: Hierbei handelt es sich um einen base64 SmartOS Container, welcher sich innerhalb des Testnetzwerks in DIR – 615, aber ausserhalb des virtuellen Switches befindet.
- Dienstkonfiguration: Um diverse Test bezüglich Portweiterleitung und dem Sperren von Verbindungen zu ermöglichen wurden die drei Default Dienste Webserver, FTP – Server und SSH – Server gewählt. Zusätzliche Dienste können aber bei Bedarf hinzugezogen werden. Nachfolgend die Konfigurationen:
  - Testsystem debian-vm-1: Auf dieser Maschine sind die Dienste Webserver → Apache 2.2, FTP – Server → vsftpd, nur anonymous Login und SSH – Server → Default Version 2 im Einsatz.
  - Testsystem debian-vm-2: Ist eine Rohinstallation und kann bei Bedarf entsprechen debian-vm-1 konfiguriert werden.
  - openSUSE 13.2: Der eigene Heimrechner ist nicht mit dem Testnetzwerk verbunden, kann aber über die NAT Funktion von DIR – 615 damit kommunizieren. Auf ihm laufen ein FTP – Server (vsftpd) und ein SSH – Server.
- PFDialogControl: Die PFDialogControl Instanz befindet sich innerhalb des Hypervisors und hat einen Link zu vswitch0 und einen Link ins Testnetzwerk zu DIR – 615 über den konfigurierten Netzwerkanschluss von SmartOS. Somit verbindet PFDialogControl wie in seinem Grundkonzept vorgesehen zwei unabhängige Netzwerksegmente.
  - Der Link zu vswitch0: Dieser innerhalb von PFDialogControl als Inside Interface bezeichnete Anschluss nutzt das paravirtualisierte Netzwerkinterface vtnet1 und hört auf die IP –Adresse 192.168.100.1/24.
  - Der Link zum Testnetzwerk: Dieser heisst PFDialogControl typisch Outside Interface und läuft ebenfalls über das paravirtualisierte Netzwerkinterface vtnet0. Seine Konfiguration ist IP – Adresse 192.168.0.105/24.

9.3 Definition der Testszenarien

Die Tests werden in zwei Kategorien unterteilt. Kategorie 1 richtet ihren Schwerpunkt auf die eingebauten Sicherungen in PFDialogControl, welche fehlende Eingaben oder Falschkonfigurationen abfangen und mit einem Hinweis versehen werden sollen. Kategorie 2 richtet den Fokus auf die eigentliche Ausführung der möglichen Dienste bzw. Einstellungen. Nachfolgend eine detailliertere Auflistung:

9.3.1 Kategorie 1

In Bezug auf die Sicherungen sollen die durch dialog generierten Reaktionen getestet werden. Solche Tests könnten sein:

- Prüfung auf Leereingaben.
- Prüfung auf nicht erlaubte Eingaben wie bspw. Inside/Outside IP – Adresse im gleichen Subnetz.
- Prüfen der Wiederherstellungsfunktion der bestehenden Eingabe bei Betätigung des Zurück – Buttons.
- Prüfung auf Vorhandensein von notwendigen Vorkonfigurationen, wie etwa das vorgängige setzten des Inside/Outside Interfaces bzw. deren IP – Adressen.
- Prüfung auf das Abfangen von gleichen Konfigurationsparametern, wie etwa das Setzen des gleichen Interface für Inside/outside, oder die gleiche IP – Adresse.
- Prüfung auf Hinweis bezüglich möglicher Subnetzverletzungen, wie bspw. befindet sich die Eingabe ausserhalb des zurzeit genutzten Subnetzes.
- Allgemeine Abhängigkeitsprüfungen welche das Vorhandensein gewisser Vorkonfigurationen überprüfen bevor ein Dienst gestartet werden kann.

9.3.2 Kategorie 2

Tests dieser Kategorie sollen die eigentliche Funktionsweise der einzelnen Dienste bzw. Funktionen verifizieren. Solche Tests könnten sein:

- Wurden die Netzwerkeinstellungen als Parameter im Hintergrund übernommen.
- Ist der DHCP – Server lauffähig und verteilt IP – Adressen aus dem definierten Pool.
- Wird eine DNS Tabelle von dnsmasq geführt und funktioniert dementsprechend die IP ←→ Rechnername Auflösung.
- Funktioniert NAT nach Aktivierung.
- Stoppen die Dienste bei Betätigung der jeweiligen AUS Funktionen.
- Funktioniert eine erstellte Portweiterleitung.
- Ist eine Portweiterleitung auch wirklich deaktiviert wenn man sie löscht.
- Funktioniert die Sperrung von Verbindungen.
- Lässt sich eine Verbindung auch wieder aufnehmen, wenn sie aus dem Regelsatz gelöscht wurde.
- Der duale Betrieb von Routingtabelle zu NAT soll sich entsprechenden dem Zustand von NAT EIN/AUS verhalten.
- Die allgemein bekannte Problematik zwischen FTP Protokoll und NAT Prinzip soll getestet werden. Wie verhält sich FTP bei aktivem NAT aus Sicht eines Clients und eines Servers.

9.4 Testausführung

Nachfolgend sind sämtliche Tests nach Testaufbau (9.1 und 9.2) und Testdefinition (9.3) ersichtlich. Die Testresultate sind innerhalb der Testsequenzen selbst ersichtlich.

9.4.1 Tests nach Kategorie 1

Die hier durchgeführten Tests sollen die Funktionstüchtigkeit der verbauten Sicherungen innerhalb von PFDialogControl basierend auf den dialog Outputs verifizieren.

9.4.1.1 Tests auf Leereingaben allgemein

Testkategorie/ Name	Beschreibung	Zu erwartendes Resultat
Prüfung auf Leereingaben	Die Leereingaben werden quer durch PFDialogControl durchgeführt. Dabei wird ein Eingabefeld leer gelassen und die Reaktion ermittelt.	Sämtliche Leereingaben sollen stets mit einer Dialog Warnung/Hinweis versehen sein. Der ursprüngliche Wert soll bei einem zweiten Aufruf erhalten bleiben (falls vorhanden).

Test 9.4.1.1 – 1

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Leereingabe in „Interface Einstellungen“ → „Inside Interface“ → Eingabe ohne Wert	Fehler wurde erkannt und abgefangen. Bei erneutem Aufrufen erscheint letzter Eintrag wieder.	Alles i.O. Keine Korrekturen notwendig

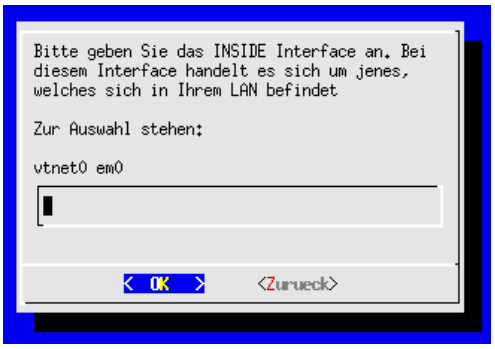

Abbildung 72: Test 9.4.1.1 - 1, Leereingabe Inside Interface

Abbildung 71: Test 9.4.1.1 - 1 Warnung, Test OK

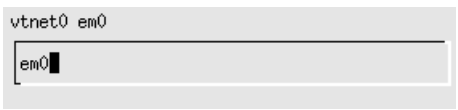

Abbildung 73: Test 9.4.1.1 -1 Letzter Eintrag erscheint wieder

Test 9.4.1.1 – 2

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Leereingabe in „Firewall“ → „Porteiterleitung EIN“ → Eingabe Regelname	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

Abbildung 75: Test 9.4.1.1- 2, Regelname vergessen

Abbildung 74: Test 9.4.1.1 - 2 Warnung OK

Test 9.4.1.1 – 3

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Leereingabe in „Firewall“ → „Verbindung Sperren“ → Eingabe ohne Regelname	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

Abbildung 77: Test 9.4.1.1 -3, Regelname vergessen

Abbildung 76: Test 9.4.1.1 3, Warnung OK

Test 9.4.1.1 – 4

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Leereingabe in „Interface Einstellungen“ → „Interfaces“ → Eingabe Subnetzmaske. Spezialtest, ohne DHCP Eingabe sollte hier eine Fehlermeldung kommen	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

Abbildung 79: Test 9.4.1.1 - 4, Keine Netzmaske, ohne DHCP

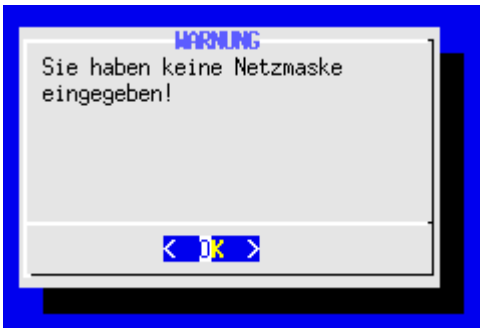

Abbildung 78: Test 9.4.1.1 -4, Warnung OK

Test 9.4.1.1 – 5

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Alternative Eingabe in „Interface Einstellungen“ → „Interfaces“ → Eingabe Subnetzmaske. Spezialtest, mit DHCP Eingabe, sollte keine Fehlermeldung erzeugen.	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

Abbildung 81: Test 9.4.1.1 -5 Interface konfiguriert mit DHCP

Abbildung 80: Test 9.4.1.1 -5, Netzmaske nicht eingegeben

Abbildung 82: Bestätigung erfolgreich mit DHCP Sicherung

9.4.1.2 Tests auf nicht korrekte/ nicht mögliche Eingaben

Testkategorie/ Name	Beschreibung	Zu erwartendes Resultat
Prüfung auf Falscheingaben oder sich überschneidende Eingaben.	In dieser Testserie sollen mögliche Fehleingaben geprüft werden. Wie bspw. sich überschneidende Eingaben, oder nicht mögliche Eingaben.	Diese sollen stets eine Fehlermeldung erzeugen, sowie einen Korrekturhinweis falls möglich.

Test 9.4.1.2 – 1

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Falscheingabe in „Interface Einstellungen“ → „Outside Interface“ → Das Inside Iface ist bereits em0, jetzt wird versucht das Outside Iface auf ebenfalls auf em0 zu setzten. Test ist äquivalent zu Inside Interface. Dies sollte nicht möglich sein.	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

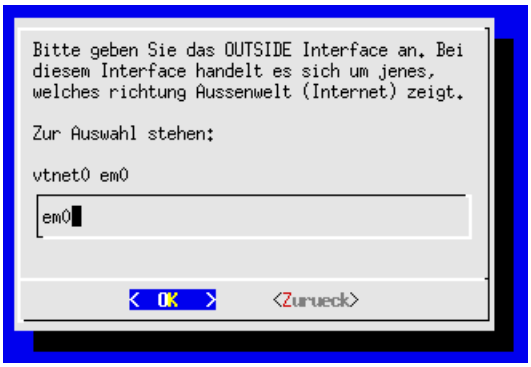

Abbildung 83: Test 9.4.1.2 -1 Gleiches Inside/Outside Iface

Abbildung 84: Test 9.4.1.2 - 1 Feheler abgefangen, WEITER nicht möglcih

Test 9.4.1.2 – 2

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Falscheingabe in „Interface Einstellungen“ → „Interfaces“ → Das Inside Iface hat die IP 192.168.202.1, Outside soll eine IP im gleichen Subnetz erhalten. Dies sollte eine Fehlermeldung erzeugen.	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

Abbildung 85: Test 9.4.1.2 -2, Gleich IP von Inside/Outside Interface

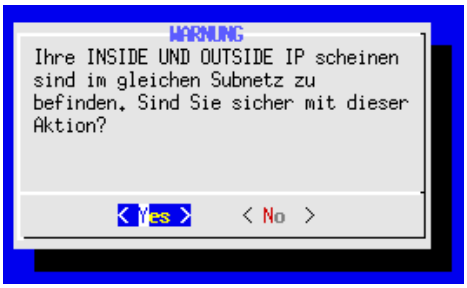

Abbildung 86: Test 9.4.1.2 -2, Gleiche IP wurde erkannt. Vorgehen bleibt dem User offen

Test 9.4.1.2 – 3

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Gleiche Eingabe in „Firewall“ → „Verbindung sperren“ → Dieser Test gilt auch für „Portweiterleitung EIN“. Es wird versucht eine Regel mit dem bereits existierenden Namen Theodor zu erzeugen. Dies sollte abgefangen werden mit einem Hinweis + Abbruch	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

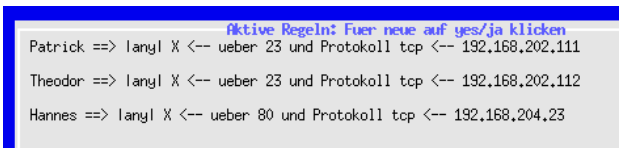

Abbildung 87: Test 9.4.1.2 -3, Erkennung des gleichen Regelnamens

Abbildung 88: Test 9.4.1.2 -3 Eingabe bestehender Regelname

Abbildung 89: Test 9.4.1.2 -3 Gleicher Regelname wurde abgefangen

Test 9.4.1.2 – 4

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Mögliche Falscheingabe in „Firewall“ → „Verbindung sperren“ → Dieser Test gilt auch für „Portweiterleitung EIN“. Die eingegebene IP befindet sich nicht im lokalen Subnetz (192.168.202.XXX). Eine Warnung sollte erscheinen. Weiteres Vorgehen bleibt dem User überlassen mittels yes/no	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

Abbildung 91: Test 9.4.1.2 - 4: IP ausserhalb des lokalen Subnetzes

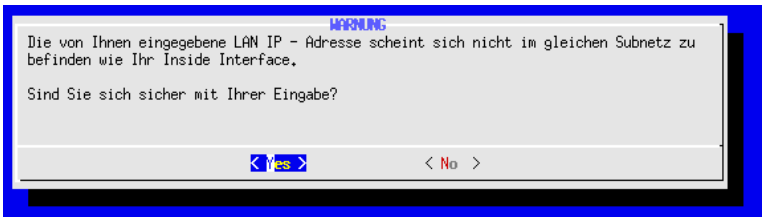

Abbildung 90: Test 9.4.1.2 - 4, Ungleiches Subnetz wurde erkannt. Weiteres Vorgehen bleibt beim User

Test 9.4.1.2 – 5

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Falscheingabe in „Firewall“ → „Verbindung sperren“ → Dieser Test gilt auch für „Portweiterleitung EIN“. Dieser Test gilt allgemein für jeden einzugebenden Port, da der Code immer derselbe (oder ähnlich) ist. Es wird versucht einen viel zu hohen Port als IPv4 erlaubt, einzugehen (in diesem Fall +1)	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

Abbildung 93: Test 9.4.1.2 - 5, Zu hohe Portnummer

Abbildung 92: Test 9.4.1.2 -5, Zu hoher Port abgefangen

Test 9.4.1.2 – 6

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Falscheingabe in „Firewall“ → „Verbindung sperren“ → Dieser Test gilt auch für „Portweiterleitung EIN“. Dieser Test gilt allgemein für jeden einzugebenden Port, da der Code immer derselbe (oder ähnlich) ist. Es wird versucht TCP versehentlich falsch einzugeben. (hier tsp)	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

Abbildung 95: Test 9.4.1.2 -5, Falscheingabe von TCP

Abbildung 94: Test 9.4.1.2 - 5, Zu hoher Port wurde abgefangen

---

Test 9.4.1.2 – 6

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Leereingabe in „Firewall“ → „Verbindung sperren“ → Dieser Test gilt auch für „Portweiterleitung EIN“. Da die Leereingaben stets auf demselben Code basieren (-z <Variable>), soll dieses Beispiel für sämtliche Eingaben in „Portweiterleitung“ und „Verbindung sperren“ gelten.	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

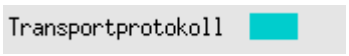

Abbildung 96: Test 9.4.1.2 -6, Allgemeine Leereingabe in Firewall Sektion

Abbildung 97: Test 9.4.1.2 -6, Allgemein ausgelöste Fehlermeldung

9.4.1.3 Abhängigkeitsprüfung von notwendigen Konfigurationsparametern

Testkategorie/ Name	Beschreibung	Zu erwartendes Resultat
Prüfung auf Konfigurationsparameter welche notwendig sind um Dienste/ Funktionen zu betreiben.	Es soll geprüft werden ob Sicherungen greifen, welche die überprüfen ob Konfigurationen vorhanden sind um einen Dienst zu starten.	Diese sollen stets eine Fehlermeldung erzeugen, sowie einen Korrekturhinweis falls möglich.

Test 9.4.1.3 – 1

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Für diese Testreihe wurde die Outside Interface IP manuell gelöscht. So konnte es von PFDialogControl nicht erfasst werden. Dies sollte stets zu Fehlermeldungen mit Hinweis führen. Hier der Versuch das Outside Interface in „Interface Einstellungen“ → „Start Interface“ zu starten.	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

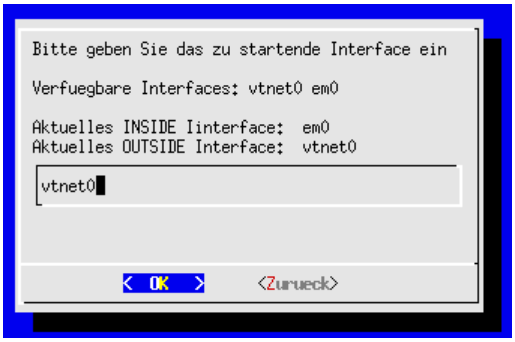

Abbildung 98: Test 9.4.1.3 -1, Start Outside Iface ohne IP

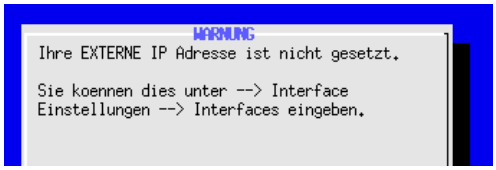

Abbildung 99: Test 9.4.1.3 - 1, Warnung mit Hinweis

Test 9.4.1.3 – 2

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Für diese Testreihe wurde die Outside Interface IP manuell gelöscht. So konnte es von PFDialogControl nicht erfasst werden. Dies sollte stets zu Fehlermeldungen mit Hinweis führen. Hier der Versuch die Liste mit den aktuellen Regeln Portweiterleitung/Verbindung sperren zu öffnen.	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

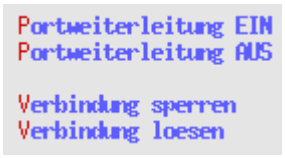

Abbildung 100: Test 9.4.1.3 - 2, Es wurden alle vier Menüs aufgerufen

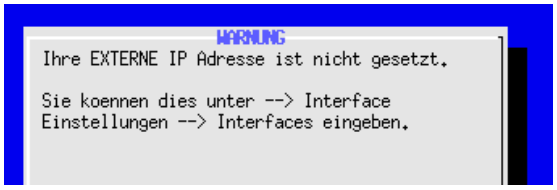

Abbildung 101: Test 9.4.1.3 - 2, Fehlende IP wurde immer erkannt

Test 9.4.1.3 – 3

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Test 9.4.1.3 – 1 und 2 wurden wiederholt mit manuell gelöschtem Outside Interface, anstelle von Outside IP. Das Ergebnis ist dasselbe wie in den vorhergehenden Tests.	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

Abbildung 102: Test 9.4.1.3 - 3, Führt zu gleichem Ergebnis mit dieser Fehlermeldung

Test 9.4.1.3 – 4

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Basierend auf den Test aus Serie 9.4.1.3 wurde der Test für das Inside Interface/IP durchgeführt aber aus Platzgründen nicht dokumentiert. Die Ergebnisse waren dieselben, da der Code technisch der Gleiche ist.	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

Test 9.4.1.3 – 5

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Basierend auf den Test aus Serie 9.4.1.3 wurde ein Test im Bereich DHCP/DNS (Start des Dienstes ohne Inside Interface/IP+ Statusanzeige) durchgeführt, um zu sehen ob dieser Abschnitt den Fehler ebenfalls abfängt. Dem war so, jedoch wurde auf Screenshots verzichtet da die Verbindung bei diesen Tests zur Remote Maschine unterbrochen wurde.	Fehler wurde erkannt und abgefangen	Alles i.O. Keine Korrekturen notwendig

9.4.2 Auswertung der Tests nach Kategorie 1

Ein Vergleich zu den Muss Zielen:

Tabelle 15: Verifizierung der Muss Ziele mit den durchgeführten Tests nach Kategorie 1

Funktion/ Definition	Beschreibung	Muss/ Auftrag	Bestätigt durch Testen
Einfache Menüführung	Die Struktur der Menüführung muss simpel sein um im Profibereich schnelle Konfigurationen zu ermöglichen. Anfänger müssen ohne Schwierigkeiten navigieren können.	**MUSS – Aus Auftrag ersichtlich**	**Bestätigt**
Allgemein Einfache Konfiguration	Erfragt werden nur die nötigsten Optionen. Unklarheiten sollen durch Hilfetexte über dem Menü helfen, zusätzlich noch ein schriftlicher Quick Guide. Autokonfiguration soll so viel wie möglich selbst übernehmen. ACHTUNG gilt nur für NAT und DHCP/DNS – Server!	**MUSS – Aus Auftrag ersichtlich**	**Bestätigt**
Konfigurationssicherheit	Für Anfänger sollen Fehlermeldungen „Pup - Ups“ generiert werden, die eine Fehlerbeschreibung und einen Hinweis geben was und vor allem wo noch Konfigurationen fehlen.	**MUSS – Aus Auftrag NICHT ersichtlich**	**Bestätigt**
Bestätigungssicherheit	Wo kritische Entscheidungen (Ein-/ Ausschalten, Löschen, Erstellen) getroffen werden müssen, wird stets eine Bestätigung verlangt.	**MUSS – Aus Auftrag NICHT ersichtlich**	**Bestätigt**

Die ersten beiden Punkte von Tabelle 15 können seitens des Entwicklers als bestätigt angesehen werden, da die Konfiguration simpel ist und Falscheingaben abgefangen werden und stets mit einem Hinweis versehen sind. Auch ist es kaum möglich Fehler zu produzieren, da diese von PFDialogControl stets abgefangen werden. So mussten für diese Tests manuelle Eingriffe in den Konfigurationsdateien vorgenommen werden, welche von einem Anfänger so kaum nachvollziehbar oder gar nachstellbar wären.

Zu den durchgeführten Tests muss noch gesagt werden, dass sie stark zusammengefasst worden sind, um den Seitenzahlrahmen nicht unnötig aufzublasen. Sie wurden aber so gewählt, dass sie aussagekräftige Resultate liefern, welche den Grossteil der nach Punkt 9.3 „Definition der Testszenarien“ verlangten Tests abdecken. So wurde nach Testserie 1 aus Kategorie 1 ermittelt, dass sämtliche verbauten Sicherungen bezüglich Leereingaben und Falscheingaben korrekt funktionieren. Dieses Ergebnis wurde aber auch so erwartet, da die meisten Tests während der Entwicklung der einzelnen Funktionen bereits in einer ähnlichen Form durchgeführt wurden.

9.4.3 Tests nach Kategorie 2

Die hier durchgeführten Tests beziehen sich auf die eigentliche Funktionstüchtigkeit der Dienste DHCP/DNS und NAT sowie auf die Funktionsweise von Portweiterleitung und Verbindung sperren. Innerhalb dieser Testserie wird auf die korrekte Ausführung der Eingaben unter Menü Punkt „Interface Einstellungen“ verzichtet. Da deren korrekte Einstellungen in direkter Abhängigkeit zur Funktionsfähigkeit der oben genannten Dienste/ Funktionen stehen.

9.4.3.1 Verifizierung der Funktion von DHCP/DNS Einstellungen

Testkategorie/ Name	Beschreibung	Zu erwartendes Resultat
Prüfung auf korrekten Start des Dienstes sowie Verifizierung der Funktionalität	Es soll geprüft werden, ob der DHCP Server startet und entsprechend IP Adressen an seine Clients im Inside Interface vergibt. Es soll auch geprüft werden, ob die Namen sich mittels nslookup in IP Adressen auflösen lassen. Eingabesicherungen werden vernachlässigt, da sie in Testserie 1 durgeführt worden sind.	PFDialogControl kann nach autogenerierten IP Pool, Adressen an seine Clients vergeben und die Namen auf IP Adressen auflösen. Bei Deaktivierung des Dienstes soll das oben genannte nicht mehr funktionieren.

Test 9.4.3.1 – 1

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Starte über Menü Punkt „DHCP/DNS Einstellungen → Start DHCP/DNS“ den Dienst. Zum Testen wird ein Auszug aus der Prozessliste eingeholt. Kommando: **ps –ax | grep dnsmasq**	Der dnsmasq wird in der Prozessliste als aktiver Hintergrunddienst erkannt. Statusanzeige ist vorhanden „AKTIVIERT“.	Alles i.O. Keine Korrekturen notwendig

Abbildung 103: Test 9.4.3.1 - 1, Starten des DHCP/DNS Dienstes aus Menü

```
root@freebsd:/usr/home/test # ps -ax | grep dnsmasq
  521  -  I      0:00.00 /usr/local/sbin/dnsmasq -x /var/run/dnsmasq.pid -C /usr/local/etc/dnsmasq.conf
```

Abbildung 104: Test 9.4.3.1 - 1, Bestätigung des Starts von dnsmasq. Hier über die PID 521 verifiziert

Test 9.4.3.1 – 2

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Überprüfen ob IP's an Clients vergeben werden. Dies wird auf den virtuellen Maschinen debian-vm-1 und debian-vm-2 durchgeführt. Kommando: **ifconfig,** hier sollte allgemein eine IP stehen. Vorzugsweise eine aus dem konfiguriertem Subnetz 192.168.100.(20 – 250)	Beide Systeme haben eine IP aus dem Pool erhalten. Für genaue IP's siehe Abbildungstext. Die IP werden bei dnsmasq nach dem Zufallsprinzip aus dem Pool bereitgestellt.	Alles i.O. Keine Korrekturen notwendig

```
root@debian-vm-1:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 52:b8:18:91:a1:8c
          inet addr:192.168.100.242  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::50b8:18ff:fe91:a18c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3213 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1966 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4087348 (3.8 MiB)  TX bytes:147323 (143.8 KiB)
```

Abbildung 106: Test 9.4.3.1 - 2, ifconfig von debian-vm-1

```
root@debian-vm-2:~# ifconfig
eth0      Link encap:Ethernet  HWaddr c2:db:a2:05:d9:4b
          inet addr:192.168.100.224  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::c0db:a2ff:fe05:d94b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2758 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1524 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4043358 (3.8 MiB)  TX bytes:109413 (106.8 KiB)
```

Abbildung 105: Test 9.4.3.1 - 2, ifconfig von debian-vm-2

Test 9.4.3.1 – 3

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
DNS soll getestet werden. Dazu wird auf jeder VM versucht die IP von der jeweils anderen Maschine zu ermitteln. + Es wird ein Reverse Lookup versucht, um den Namen von der PFDialogControl Instanz zu ermitteln.	Die Auflösung des Namens zur IP funktioniert unter den virtuellen Maschinen. Die Reverse Auflösung von der IP zum Namen in Richtung PFDialogControl funktioniert ebenfalls. Nslookup google.ch funktioniert auch.	Alles i.O. Keine Korrekturen notwendig

```
root@debian-vm-1:~# nslookup debian-vm-2
Server:         192.168.100.1
Address:        192.168.100.1#53

Name:   debian-vm-2.testdom
Address: 192.168.100.224

root@debian-vm-1:~# nslookup 192.168.100.1
Server:         192.168.100.1
Address:        192.168.100.1#53

1.100.168.192.in-addr.arpa      name = th-firewall.testdom.
```

Abbildung 107: Test 9.4.3.1 - 3, erfolgreicher nslookup Test zu debian-vm-2

```
root@debian-vm-2:~# nslookup debian-vm-1
Server:         192.168.100.1
Address:        192.168.100.1#53

Name:   debian-vm-1.testdom
Address: 192.168.100.242

root@debian-vm-2:~# nslookup 192.168.100.1
Server:         192.168.100.1
Address:        192.168.100.1#53

1.100.168.192.in-addr.arpa      name = th-firewall.testdom.
```

Abbildung 108: Test 9.4.3.1 - 3, erfolgreicher nslookup Test zu debian-vm-1

```
root@debian-vm-1:~# nslookup google.ch
Server:         192.168.100.1
Address:        192.168.100.1#53

Non-authoritative answer:
Name:   google.ch
Address: 173.194.116.95
Name:   google.ch
Address: 173.194.116.79
Name:   google.ch
Address: 173.194.116.87
Name:   google.ch
Address: 173.194.116.88
```

Abbildung 109: Test 9.4.3.1 - 3, erfolgreicher nslookup Test zu google.ch

Test 9.4.3.1 – 3

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Deaktiviere DHCP/DNS Dienst und prüfe ob er wirklich down ist. Kommando**: ps –ax | grep dnsmasq**; und auf debian-vm-2 wird **dhclient eth0** ausgeführt	Dnsmasq ist ausgeschaltet. Statusanzeige bestätigt dies. Ein dhclient zum Auffrischen der IP auf debain-vm-2 lief in einen Timeout ohne Ergebnis. Ein Auszug aus ps –ax | grep dnsmasq zeigt nur den „grep“ Auftrag, dnsmasq läuft nicht mehr als Daemon im Hintergrung.	Alles i.O. Keine Korrekturen notwendig

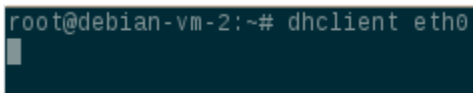

Abbildung 111: Test 9.4.3.1 - 3, dhclient auf debain-vm-2 läuft in Timeount ohne eine IP zu beziehen

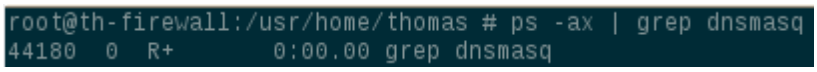

Abbildung 110: Test 9.4.3.1 - 3, Eine Prozesssuche nach dnsmasq in PFDialogControl blieb ohne Ergebnis, keine PID

9.4.3.2 Verifizierung der Funktion von NAT Service

Testkategorie/ Name	Beschreibung	Zu erwartendes Resultat
Prüfung auf korrekten Start des Dienstes, sowie Verifizierung der Funktionalität	Es soll geprüft werden ob der NAT Service wenn er aktiviert ist auch NAT anwendet. An dieser Stelle soll auch der duale Betrieb von Routingtabelle zu NAT bei deaktiviertem NAT getestet werden. Dazu wird das MacBook Pro einmal in Richtung debain-vm-2 und umgekehrt angepingt.	Die beiden VM's sollen über die „öffentliche" IP 192.168.0.105 mittels NAT geleitet werden können. Bei Deaktivierung soll die Verbindung nicht möglich sein. Es sollte möglich sein bei deaktiviertem NAT über die Routingtabelle von PFDialogControl in beide Richtungen zu pingen. Dazu wird sich das MacBook im Testnetzwerk befinden und einen statischen Routingeintrag (192.168.100.0/24 over 192.168.0.105) besitzen.

Test 9.4.3.2 – 1

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
NAT ist aktiv und es wird versucht ein **apt-get update** (debian Paketquellen aktualisieren) von debain-vm-2 durchzuführen. Zusätzlich wird ein **ping google.ch** ausgeführt	NAT war vorgängig schon aktiv, die Statusanzeige bestätigte dies. **apt-get update** war erfolgreich, siehe teilweisen Auszug unten. Ping in Richtung google.ch funktionierte ebenfalls.	Alles i.O. Keine Korrekturen notwendig

```
root@debian-vm-2:~# apt-get update
Hit http://security.debian.org wheezy/updates Release.gpg
Hit http://security.debian.org wheezy/updates Release
Hit http://security.debian.org wheezy/updates/main Sources
Hit http://security.debian.org wheezy/updates/main amd64 Packages
Hit http://security.debian.org wheezy/updates/main Translation-en
Get:1 http://http.us.debian.org wheezy Release.gpg [2,390 B]
```

Abbildung 113: Test 9.4.3.2 - 1, teilweiser Auszug, erfolgreicher durchlauf von apt-get update

```
root@debian-vm-2:~# ping google.ch
PING google.ch (173.194.116.88) 56(84) bytes of data.
64 bytes from zrh04s08-in-f24.1e100.net (173.194.116.88): icmp_req=1 ttl=56 time=13.8 ms
64 bytes from zrh04s08-in-f24.1e100.net (173.194.116.88): icmp_req=2 ttl=56 time=12.9 ms
^C
--- google.ch ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 12.944/13.420/13.897/0.490 ms
```

Abbildung 112: Test 9.4.3.2 - 1, erfolgreicher ping in Richtung google.ch

Test 9.4.3.2 – 2

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
NAT wurde über Menü deaktiviert.	Die unter Test 9.4.3.2 – 1 getätigten Aktionen funktionieren nicht mehr.	Alles i.O. Keine Korrekturen notwendig

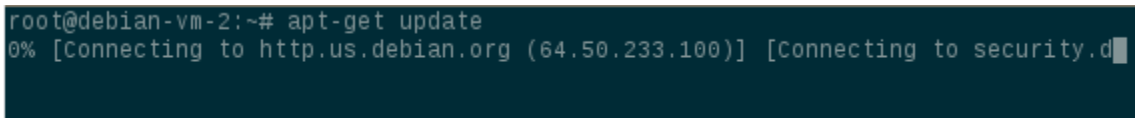

Abbildung 115: Test 9.4.3.2 - 2, apt-get update läuft in einen Timeout, KEIN Erfolg

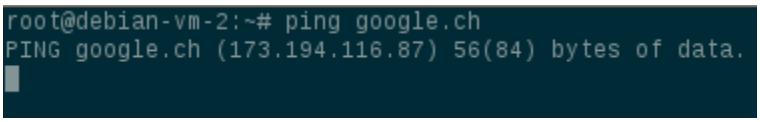

Abbildung 114: Test 9.4.3.2 - 2, Ping kann nicht beantwortet werden, daher wartet das Tool Ping unendlich auf eine Antwort

---

Test 9.4.3.2 – 3

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Spezialtest: NAT ist deaktiviert. Es wird einmal vom MacBook in Richtung debain-vm-1 gepingt und umgekehrt. Dabei wurde ein statischer Routingeintrag im MacBook erzeugt, welcher ausserhalb von PFDialgoControl nötig ist. PFDialogControl selbst ist nach Deaktivierung von NAT automatisch im Routingmodus.	Ping funktioniert in beide Richtungen, dank dem Default Routingeintrag in PFDialogControl und dem statischen Eintrag auf dem MacBook.	Alles i.O. Keine Korrekturen notwendig

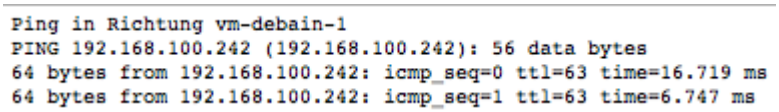

Abbildung 117: Test 9.4.3.2 - 3, Ping von MacBook (IP=192.168.0.101) in Richtung debian-vm-1 (IP=192.168.100.242)

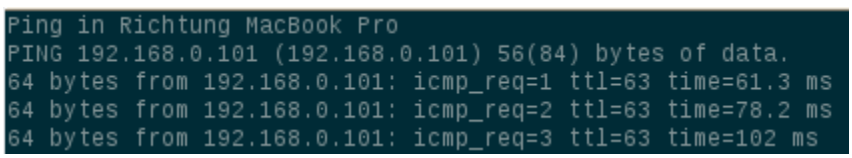

Abbildung 116: Test 9.4.3.2 - 3, Ping von debian-vm-1 (IP=192.168.100.242) in Richtung MacBook (IP=192.168.0.101)

9.4.3.3 Verifizierung der Portweiterleitung Funktion

Testkategorie/ Name	Beschreibung	Zu erwartendes Resultat
Prüfung auf korrekten Start der Funktion, sowie Verifizierung der Funktionalität	Es soll geprüft werden, ob eine erstellte Portweiterleitung tatsächlich wirksam ist. Zu diesem Zweck werden auf debain-vm-1 ein Webserver und ein FTP Server betrieben. Die Ziel IP des Systems im Testnetzwerk ist 192.168.100.242. Die Tests werden zweifach durchgeführt, einmal mit NAT und einmal im Routingmodus, um den Dualbetrieb zu demonstrieren.	Die Portweiterleitung zum Webserver sollte mit NAT einfach und schnell, über die öffentliche IP (192.168.0.105) von PFDialogControl möglich sein. Im Routingmodus sollte sowohl die öffentliche als auch die LAN IP des Zielsystems ansprechbar sein. FTP stellt einen Spezialfall dar, da FTP hinter NAT immer spezielle Bedingungen voraussetzt, siehe dazu mehr in den Tests.

Test 9.4.3.3 – 1

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Eine Portweiterleitung wurde unter PFDialogControl eingerichtet. Diese erlaubt das Weiterleiten von Port 80 nach Port 80 zu 192.168.100.242. Diese Tests wurden einmal mit und einmal ohne NAT durgeführt. Genutztes externes System ist das MacBook innerhalb des Testnetzwerks.	Im NAT Modus ist der Webserver nur unter der öffentlichen IP erreichbar, da die PF Firewall die Routingtabelle unterdrückt. Im Routingmodus hingegen kann sowohl über die öffentliche, wie auch über die Private IP Adresse angesprochen werden. Dies stellt den Dualbetrieb dar.	Alles i.O. Keine Korrekturen notwendig

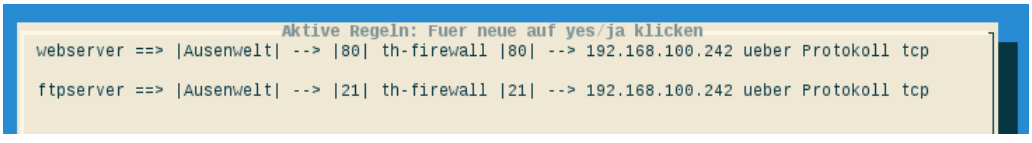

Abbildung 118: Test 9.4.3.3 - 1, Aktive Portweiterleitungen

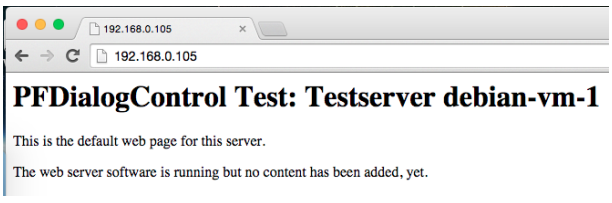

Abbildung 119: Test 9.4.3.3 - 1, Website aufgerufen über öffentliche IP von PFDialogControl im NAT Modus und im

Abbildung 120: Test 9.4.3.3 - 1, Website aufgerufen über private IP im Routingmodus. Statischer Routingeintrag auf MacBook

Test 9.4.3.3 – 2

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Eine Portweiterleitung wurde unter PFDialogControl eingerichtet. Diese erlaubt das Weiterleiten von Port 21 nach Port 21 zu 192.168.100.242. Diese Tests wurden einmal mit und einmal ohne NAT durgeführt. Genutztes externes System ist das MacBook innerhalb des Testnetzwerks.	Im NAT Modus ist der FTP Server nur unter der öffentlichen IP erreichbar, da die PF Firewall die Routingtabelle unterdrückt. Im Routingmodus hingegen kann sowohl über die öffentliche, wie auch über die Private IP Adresse angesprochen werden. Dies stellt den Dualbetrieb dar. Wie zu erwarten war, ist der FTP Server im NAT Modus nur im aktiven Modus (Portmode) ansprechbar. Dies resultiert daraus, dass NAT beim passiven Modus vom Client zuerst angesprochen werden muss. Dies ist aber nicht möglich, da NAT nicht weiss wohin mit der Verbindung. Der Aktivmodus gilt zwar als veraltet, aber er ist nutzbar. Im Allgemeinen wird seitens des Entwicklers stets sftp empfohlen. Dies funktioniert so simpel wie FTP, vereint aber die Sicherheit von FTPS.	Alles i.O. Keine Korrekturen notwendig

```
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
229 Entering Extended Passive Mode (|||5334|).
ftp: Can't connect to `192.168.0.105': Connection refused
```

Abbildung 121: Test 9.4.3.3 - 2, Im Passivmodus kann man sich über NAT Verbinden, aber sonst funktioniert nichts.

```
macservtb:~ thebog$ ftp -A 192.168.0.105
Connected to 192.168.0.105.
220 (vsFTPd 2.3.5)
Name (192.168.0.105:thebog): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 EPRT command successful. Consider using EPSV.
150 Here comes the directory listing.
drwxr-xr-x    2 0        0            4096 Apr 27 18:29 PFDialogControl
226 Directory send OK.
ftp>
```

Abbildung 122: Test 9.4.3.3 -2, Im aktiven Modus funktioniert die Verbindung im NAT, wie im Routing Modus

```
macservtb:~ thebog$ ftp 192.168.100.242
Connected to 192.168.100.242.
220 (vsFTPd 2.3.5)
Name (192.168.100.242:thebog): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
229 Entering Extended Passive Mode (|||34493|).
150 Here comes the directory listing.
drwxr-xr-x    2 0        0            4096 Apr 27 18:29 PFDialogControl
226 Directory send OK.
ftp>
```

Abbildung 123: Test 9.4.3.3 - 2, Im Routingmodus funktioniert allgemein alles, da NAT nicht eingreift. Hier ein Verbindungsaufbau zur privaten IP

9.4.3.4 Verifizierung der Portweiterleitung Funktion

Testkategorie/ Name	Beschreibung	Zu erwartendes Resultat
Prüfung auf korrekten Start der Funktion, sowie Verifizierung der Funktionalität	Es soll geprüft werden, ob eine erstellte Verbindungssperre tatsächlich wirksam ist. Zu diesem Zweck werden zwei Webserver genutzt. Ein sich im Testnetzwerk befindlicher SmartOS Container (192.168.0.100), sowie der Bewerbungs – Webserver des Entwicklers (10.0.0.178, bewerbung.localdom), welcher sich im privaten Netzwerksegment des Entwicklers befindet. Hier wird ebenfalls einmal mit und einmal ohne NAT getestet, um auch an dieser Stelle den Dualbetrieb zu demonstrieren.	Die beiden gesperrten Webserver können weder im NAT Betrieb, noch im Routingbetrieb angewählt werden. Wobei für den Routingtest nur der SmartOS Container genutzt wird. Da sonst komplexe Eingriffe in private Netzwerk erforderlich werden.

Test 9.4.3.4 – 1

Getätigte Aktion	Erzieltes Resultat	Interpretation/ Fehler
Zwei Verbindungssperren wurden eingerichtet. Einmal für Webserver SmartOS (192.168.0.100, Testnetzwerk) und eine zweite für bewerbung.localdom (10.0.0.178, privates Netzwerk). Aus Platzgründen werden alle Tests an dieser Stelle definiert. Siehe dazu Screenshots unten. Getestet wurde immer von debain-vm-1 (192.168.100.242).	Durch das Sperren der beiden Webserver, konnte keine Verbindung zu ihnen aufgebaut werden. Dies gilt für den NAT Betrieb, wie auch für den Routingmodus. Wobei der Routingmodus nur auf 192.168.0.100 getestet wurde. Im Allgemeinen gilt, eine gesperrte Verbindung ist **immer** gesperrt. Da PF lediglich auf die Interface Transfers achtet. Ob nun die NAT Regel gesetzt ist oder nicht, spielt hierbei keine Rolle.	Alles i.O. Keine Korrekturen notwendig

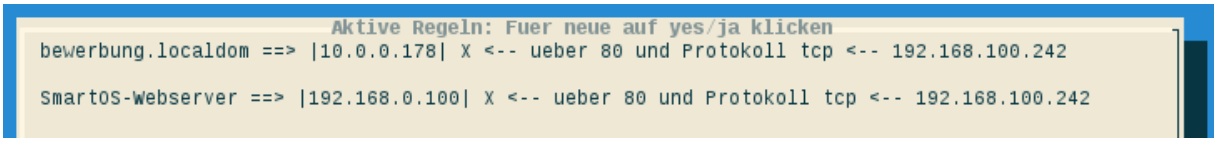

Abbildung 125: Test 9.4.3.4 - 1, Die beiden für debain-vm-1 gesperrten Webserver

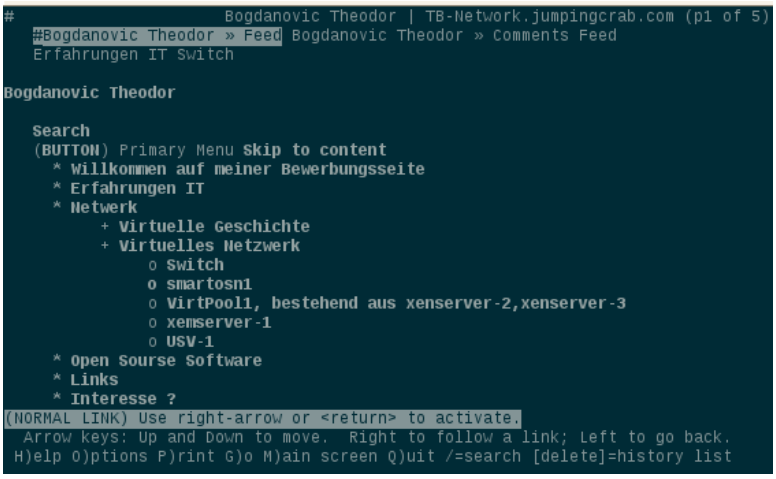

Abbildung 124: Test 9.4.3.4 - 1, Verbindungstest ohne Sperre über NAT zu bewerbung.localdom (von debian-vm-1)

Abbildung 126: Test 9.4.3.4 - 1,Verbindungstest ohne Sperre über Routingmodus zu SmartOS Webserver (von debian-vm-1)

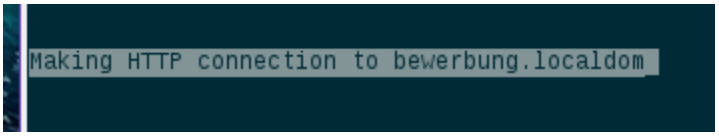

Abbildung 127: Test 9.4.3.4 - 1, Nach der Sperrung (über NAT) läuft der Verbindungsversuch zu bewerbung.localdom in einen Timeout

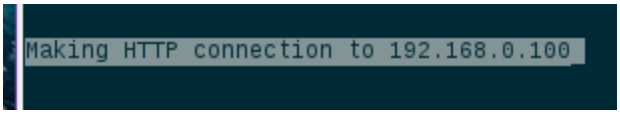

Abbildung 128: Test 9.4.3.4 - 1, Nach der Sperrung (Routingmodus) läuft der Verbindungsversuch zu SmartOS Webserver in einen Timeout

9.4.4 Auswertung der Tests nach Kategorie 2

Ein Vergleich zu den Muss Zielen:

Tabelle 16: Verifizierung der Muss Ziele mit den durchgeführten Tests nach Kategorie 2

Funktion/ Definition	Beschreibung	Muss/ Auftrag	Bestätigt durch Testen
NAT Aktivieren- und Deaktivieren	NAT muss sich Aktivieren und Deaktivieren lassen. Eine Warnung weist darauf hin, dass beim Deaktivieren das Routing beachtet werden muss (Konfiguration für Fortgeschrittene).	**MUSS – Aus Auftrag ersichtlich**	**Bestätigt**
DHCP/DNS – Server Aktivieren- und Deaktivieren	DHCP/DNS – Server muss sich auch deaktivieren lassen, wenn ein eigener benutzt wird. Warnung für Anfänger weist auf die Folgen eines deaktivierten Servers hin. Ansonsten Autokonfiguration des IP – Ranges (gute Lösung für schnelles vorgehen für Profi und Anfänger)	**MUSS – Aus Auftrag ersichtlich**	**Bestätigt**
Firewall → Portweiterleitung	Das Menü für die Portweiterleitung muss auf simple Art und Weise erlauben eine Verbindung beliebigen Ursprungs von aussen zu einem Computer im LAN zu leiten. Dabei soll die Zieladresse im LAN angegeben werden können, sowie der Port von assen zur Firewall und der Port zum Computer im LAN .Genauso einfach muss diese Regel auch gelöscht werden können.	**MUSS – Aus Auftrag ersichtlich**	**Bestätigt**
Firewall → Verbindung Sperren	Das Menü für die Sperrung einer Verbindung, muss auf simple Art und Weise erlauben, eine Verbindung von einem Computer innerhalb des LAN's	**MUSS – Aus Auftrag ersichtlich**	**Bestätigt**

	nach aussen zu Sperren. Dabei soll es möglich sein den zu sperrenden Computer das Ziel mittels IP und Port genau anzugeben. Es soll aber auch möglich sein nur einen Port anzugeben und mittels der Firewall üblichen Direktive „any“ jegliche Zieladresse zu sperren. Genauso einfach muss diese Regel auch gelöscht werden können.		

Da auch hier während der Entwicklung in ähnlicher Form getestet wurde, war eine vollständige Bestätigung der MUSS Ziele zu erwarten. Vor allem der Dualbetrieb erwies sich als perfekt Funktionierend und ermöglicht so den permanenten Wechsel von NAT zu Routingbetrieb. Zwar wäre der Einsatz im Dualbetrieb ohne weiteres garantierbar, doch erfordert dies gute Kenntnisse im Bereich statischer Routen bzw. Routing Information Protocol (RIP v1 – 2). Da dies einem Anfänger nicht zuzumuten wäre, wird an manchen Stellen in PFDialogControl darauf hingewiesen, dass ein Wechsel von NAT zu Routing die Regelsätze zum Teil ausser Kraft setzten könnte. Dies wurde so implementiert um einen Anfänger daran zu hindern in den Routingmodus zu wechseln und so PFDialogControl lahmzulegen.

9.5 Beurteilung und Fazit

Die Tests verliefen alle wie gewünscht und bestätigten das Vorhandensein sämtlicher MUSS Ziele. Alle Eingabesicherungen haben wie gewünscht funktioniert, sowie wo erforderlich auch Hinweise zur Problemlösung geliefert. Der Dualbetrieb, welcher nicht direkt aus dem Pflichtenheft ersichtlich ist, da er die Grundfunktionsweise von PFDialogControl nicht beeinträchtigt, sondern diese ohne grossen Konfigurationsaufwand ergänzt, funktionierte ebenfalls wie gewünscht. Daher kann mit Gewissheit gesagt werden, dass PFDialogControl so funktioniert wie nach Pflichtenheft definiert, aber auch so wie es sich der Entwickler für seinen privaten Einsatz auch gewünscht hat. Denn bereits die auf dem Testsystem durchgeführten Tests zeigten ein schnell –und leicht konfigurierbares Menü, mit dessen Hilfe die diversen Portweiterleitungen, sowie Verbindungssperren erstellt werden konnten.

Seitens des Entwicklers kann PFDialogControl nach dieser Testserie als funktionierend und bereit für den Produktiveinsatz eingestuft werden. Daher wird an dieser Stelle das Release 1.0.3 von PFDialogControl freigegeben.

10 Dokumentation

10.1 Installationsanleitung

Zum Installer gibt es nicht viel zu sagen, da er so simpel wie möglich gehalten wurde und sämtliche Instruktionen über dialog mittgeteilt werden. Dazu muss der gezippte Tarball (pfdialogcontrol-1.0.3.tar.gzip) lediglich entpackt und mittels des Aufrufs **./install** gestartet werden. Es kann, sollte aber nicht vorkommen, dass das install – Skript nicht ausführbar ist nach dem entpacken. In diesem Fall reicht die Befehlskette **chmod 700 install** innerhalb des Verzeichnisses wo sich der Installer befindet, um ihn ausführbar zu machen. Der Installer besteht lediglich aus 2 – 3 Dialogboxen, welche die Installationsbestätigung einholen und den Abschluss mit einer Reboot Meldung bestätigen. Sollte ein System nicht mindestens 2 Interfaces zur Verfügung stellen, so wird gleich nach der Installationsbestätigung darauf hingewiesen, dass es eine Minimumzahl gibt. Der User kann aber dennoch mit „OK“ den Installationsprozess durchführen. Dieser Schritt ist für Profi Anwender, welche wissen wie sie dieses Problem mittels virtuellem Interface lösen können. Nach Abschluss des Installationsprozesses wird ein Reboot durgeführt und es kann mit der Konfiguration von PFDialogControl begonnen werden.

10.2 Bedienungsanleitung (Quick Guide)

Dier Quick Guide soll anhand eines heute typischen Netzwerkszenarios eine Anleitung zur Konfiguration von PFDialogControl liefern. Dazu werden folgende Komponenten benötigt:

- Ein realer Rechner oder eine virtuelle Maschine mit mindestens zwei Interfaces.
- Ein realer Switch oder ein virtueller Switch

Diese Anleitung nutzt eine neutrale Beschreibung, welche für reale wie auch für virtuelle Hardware genutzt werden kann. Um die Funktionalitäten der Portweiterleitung zu erklären, wird im Anleitungsszenario ein Webserver vorhanden sein.

Das Anleitungsszenario basiert auf folgendem Schema:

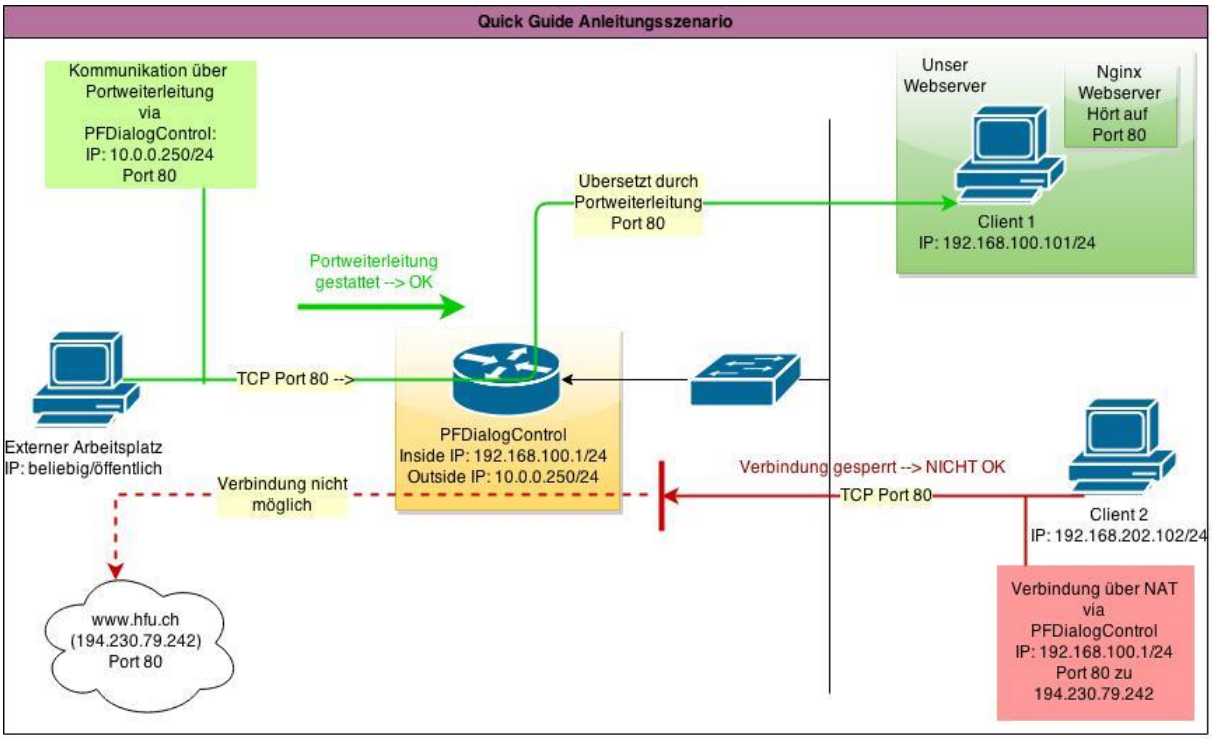

Abbildung 129: Quick Guide Schema

Für dieses Szenario gelten folgende Parameter:

- PFDialogControl wird wie folgt eingerichtet:
  - Inside Interface = vtnet1, IP = 192.168.100.1/24
  - Outside Interface = vtnet0, IP = 10.0.0.250/24
- Wir haben zwei Rechner in unserem durch PFDialogControl gemanagten Netzwerk:
  - Client 1, mit der IP = 192.168.100.101/24. Dies ist unser Webserver auf den wir mittels Portweiterleitung zugreifen wollen.
  - Client 2, mit der IP = 192.168.100.102/24. Diesem Rechner soll die Verbindung zu www.hfu.ch gesperrt werden.
- Unser Externer Arbeitsplatz kann sich irgendwo befinden und eine beliebige IP besitzen.

Die Verkabelung über den Switch, sowie die Anbindung an den Provider Router sollten aus dem Quick Guide Anleitungsszenario oben gut ersichtlich sein.

Haben Sie eine frische FreeBSD Installation (siehe Empfehlung für FreeBSD Installation in Anhang A), auf welcher Sie PFDialogControl installiert haben, so können wir mit der Konfiguration beginnen.

➔ Schritt 1

Loggen Sie sich als Superuser „root“ ein und starten Sie PFDialogControl mittels **pfdctl**. Sie sehen nun das Startmenü.

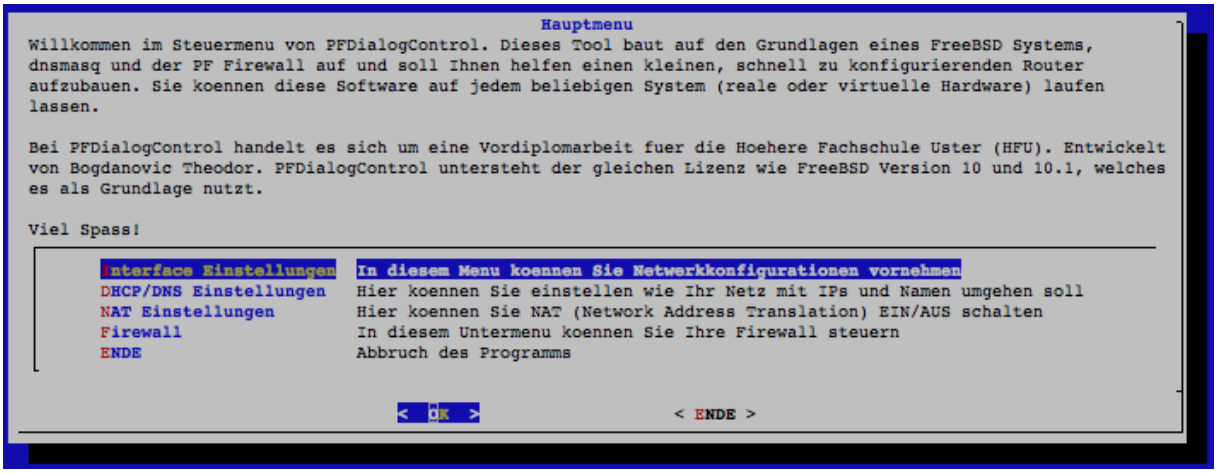

Abbildung 130: Quick Guide, Start Menü

➔ Schritt 2

Beginnen Sie stets mit der Netzwerkkonfiguration, da ansonsten die Sicherungen von PFDialogControl Ihnen weitere Konfigurationsschritte verbieten.

Gehen Sie auf Menüpunkt **Interface Einstellungen**. Dann erscheint folgendes Menü.

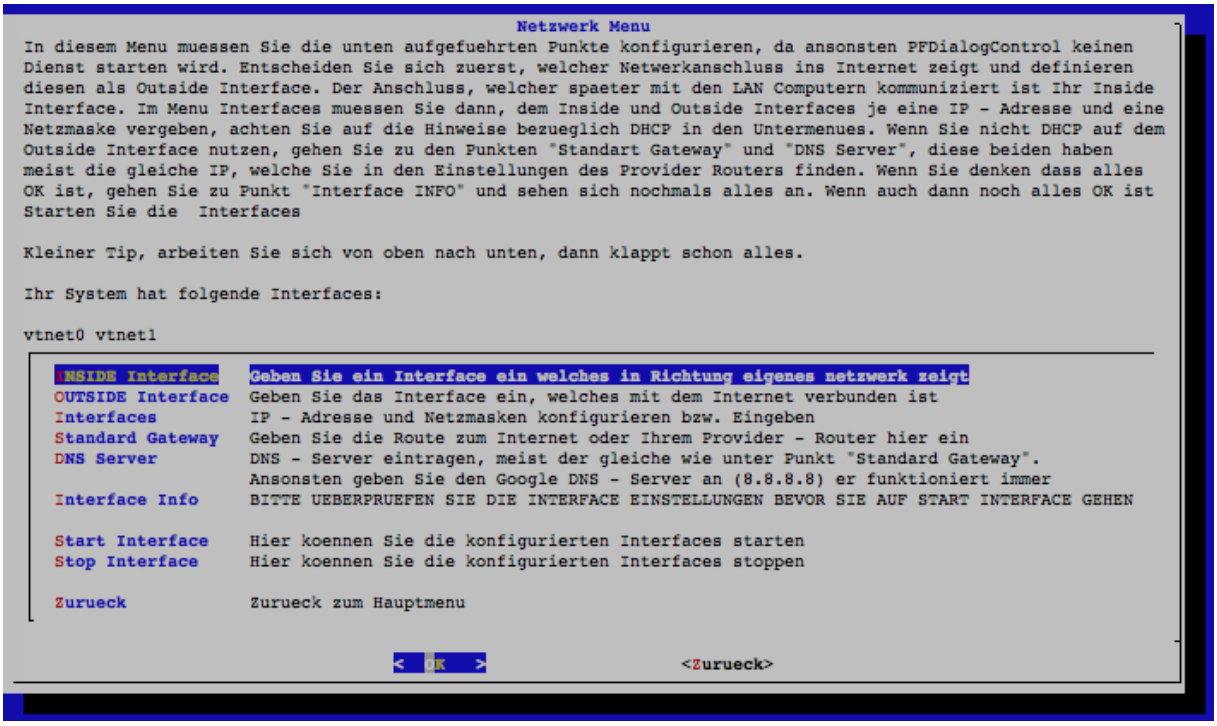

Abbildung 131: Quick Guide, Interface Einstellungen

An dieser Stelle müssen Sie sich entscheiden, welches Interface Ihr Inside Interface ist. Eine Liste mit verfügbaren Interfaces sehen Sie gleich zuoberst bei der Menüauswahl. Gehen Sie nun auf **INSIDE Interface** und wählen Ihr Interface.

➔ Schritt 3

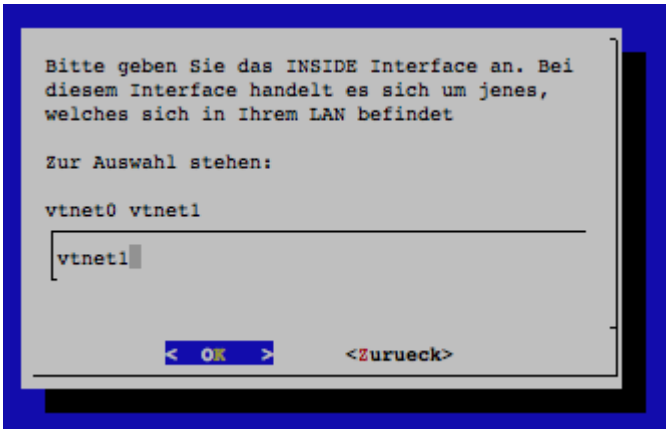

Abbildung 132: Quick Guide, Inside Interface wählen

Wir werden an dieser Stelle das Interface „vtnet1“ als unser INSIDE Interface wählen. Bestätigen Sie dies mit OK. Eine nachfolgende informative Zusammenstellung können Sie ebenfalls mit OK bestätigen.

➔ Schritt 4

Nun sind wir wieder automatisch im Menü „Interface Einstellungen“. Wir wählen nun den Punkt **OUTSIDE Interface** und wiederholen die Prozedur von Schritt 3. Hier wählen wir das Interface „vtnet0“ als jenes, welches mit dem Internet verbunden ist.

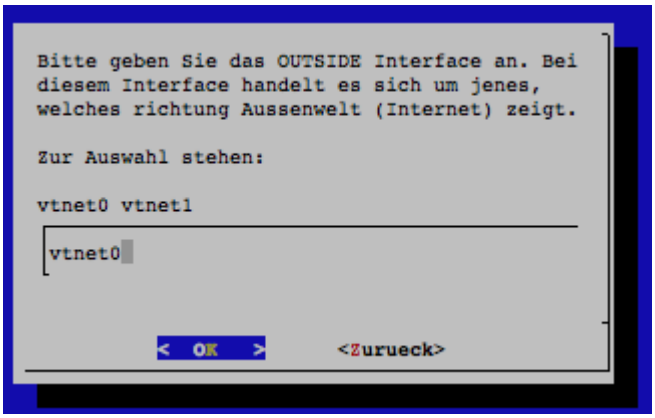

Abbildung 133: Quick Guide, Outside Interface wählen

➔ Schritt 5

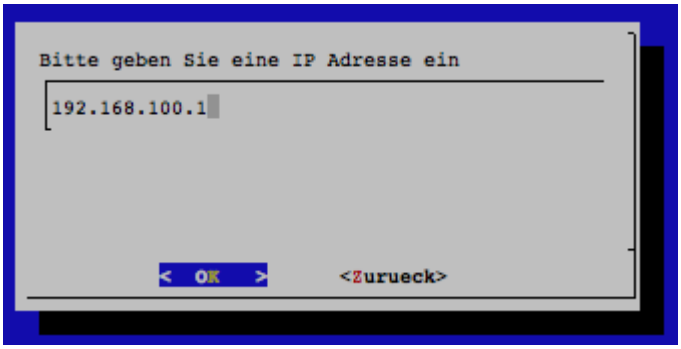

Abbildung 134: Quick Guide, Interface vtnet1 konfigurieren

Nun wählen wir im Menü „Interface Einstellungen“ den Punkt **Interfaces** und geben den Interface Namen „vtnet1“ ein. Danach gelangen wir zum nebenstehenden Fenster. Hier geben wir die IP – Adresse des Inside Interface ein. Ein Warnhinweis wird Sie vorgängig darauf aufmerksam machen, dass Sie eine spezifische Endnummer der IP nutzten müssen. Hier wird Ihnen empfohlen für das Inside Interface stets die Endziffer „1“ zu nutzen. Dies ist so üblich.

Nach Bestätigung der IP müssen Sie noch die Netzmaske eingeben. Der übliche Standard ist meist 255.255.255.0. Geben Sie diese so wie in folgender Abbildung ein.

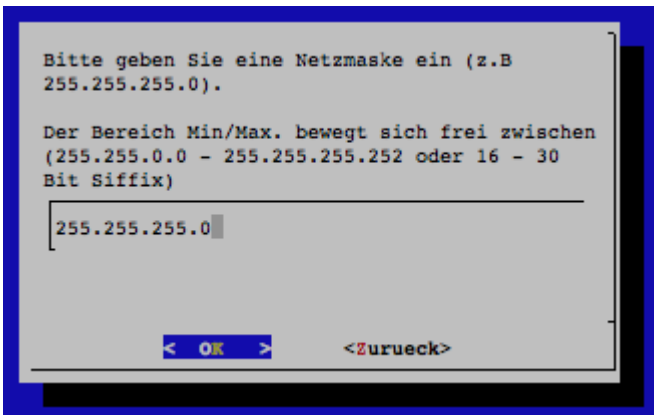

Abbildung 135: Quick Guide, Inside Interface Eingabe Netzmaske

➔ Schritt 6

Nun sind wir wieder im Menü „Interface Einstellungen“ angelangt. Wiederholen Sie nun diesen Schritt für das **OUTSIDE Interface**.

Hier besteht auch die Möglichkeit „dhcp oder DHCP“ einzugeben und FreeBSD sich selbst bei der Interfacekonfiguration zu überlassen. Wenn Sie diesen Punkt wählen, so müssen Sie die Punkte „Standard Gateway“ und „DNS Server“ nicht mehr konfigurieren. Jedoch wird an dieser Stelle davon abgeraten. Eine statische Konfiguration ist meistens die bessere Wahl. Falls Sie sich für DHCP entscheiden, so vergessen Sie nicht FreeBSD an dieser Stelle neu zu Booten.

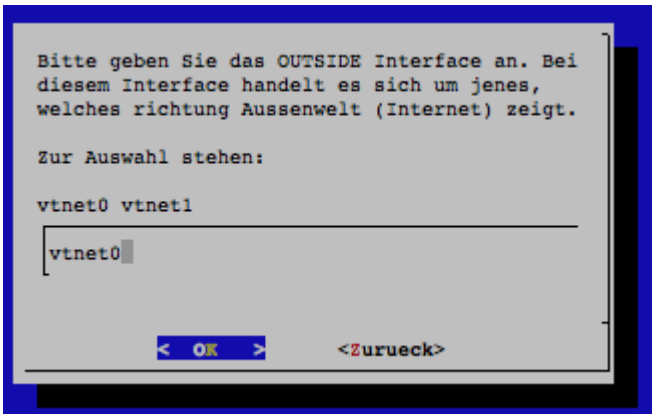

Abbildung 136: Quick Guide, Interface vtnet0 konfigurieren

➔ Schritt 7

Wenn Sie sich im Outside Interface für DHCP entschieden haben, so können Sie diesen Schritt überspringen. Andernfalls gehen wir jetzt zu Menü Punkt **Standard Gateway** und geben unseren Gateway an. Dieser wird meist der Provider Router sein. In unserem Beispiel geben wird die IP – Adresse 10.0.0.1 ein.

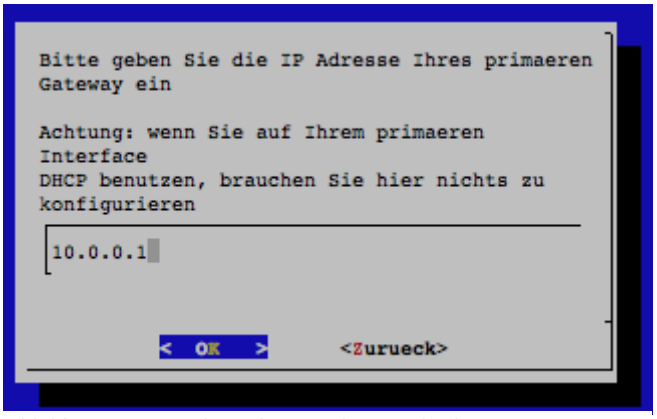

Abbildung 137: Quick Guide, Eingabe Standard Gateway

➔ Schritt 8

Nun gehen wir zu Menü Punkt **DNS Server**. Hier gilt das gleiche wie in Schritt 7. Wir geben an dieser Stelle wieder die IP – Adresse 10.0.0.1 ein und bestätigen die Eingabe.

➔ Schritt 8

Wenn Sie möchten, können Sie an dieser Stelle noch mittels des Menü Punktes **Interface Info** die konfigurierten Interfaces ansehen. Ansonsten gehen wir nun zum Punkt Start Interface und geben den Namen des zu startenden Interfaces ein. Sollte alles in Ordnung sein, wird das Interface gestartet und die Konfiguration permanent lauffähig gespeichert. Diesen Schritt führen Sie für beide konfigurierten Interfaces aus.

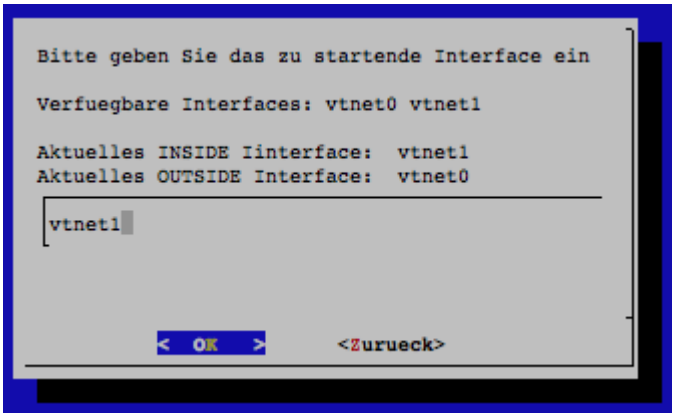

Abbildung 138: Quick Guide, Starten der Interfaces

Starten des DHCP/DNS Dienstes

➔ Schritt 9

Wenn Sie sich entschliessen den PFDialogControl eigenen DHCP/DNS Dienst zu nutzten, was empfehlenswert ist, gehen Sie nun mittels „Zurück“ ins Hauptmenü. Dort wählen Sie nun den Menü Punkt **DHCP/DNS Einstellungen** aus, mit dessen Hilfe Sie ins DHCP/DNS Menü gelangen.

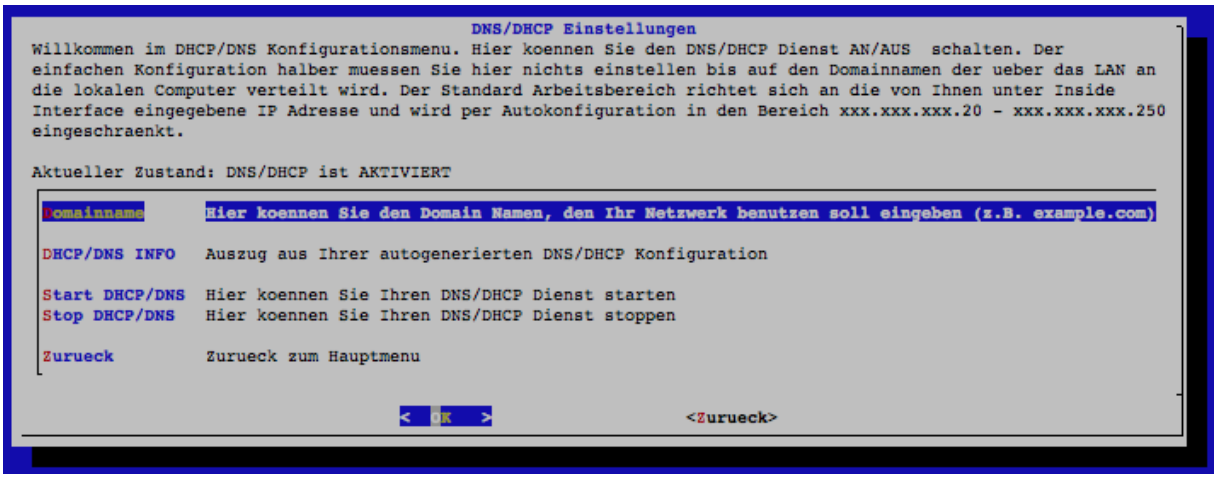

Abbildung 139: Quick Guide, DHCP/DNS Hauptmenü

➔ Schritt 10

Hier gehen wir als erstes zum Menü Punkt **Domainname** und geben einen Domänennamen für unser internes Netzwerk an. Dieser wird automatisch allen unseren Rechnern im Inside Network am Ende angeheftet. In unserem Beispiel vergeben wir den Namen „vda“. Bestätigen Sie nun noch die Eingabe.

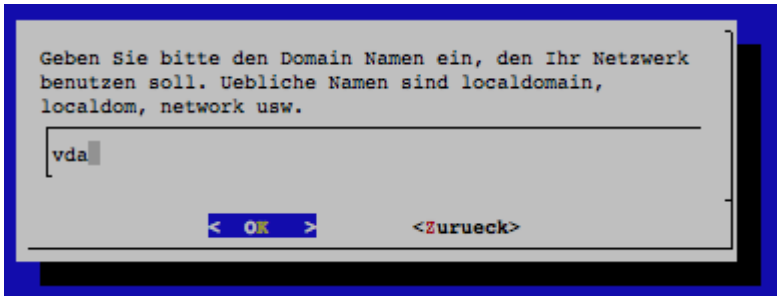

Abbildung 140: Quick Guide, Eingabe Domänennamen

➔ Schritt 11

Mittels **DHCP/DNS Info** können Sie sich die Primäreinstellungen ansehen. Dort sehen Sie, dass der Dienst per Default einen automatisch generierten IP Pool, von in unserem Beispiel 192.168.100.20 – 192.168.100.250 vergibt. Sie müssen sich also nicht um den IP Range kümmern, dieser wurde bereits erstellt.

➔ Schritt 12

Gehen wir nun zum Menü Punkt **Start DHCP/DNS** und aktivieren den Dienst. Wenn alles OK war, wird eine Einschaltbestätigung erscheinen und Sie sehen in der Statusanzeige den aktuellen Zustand des Dienstes.

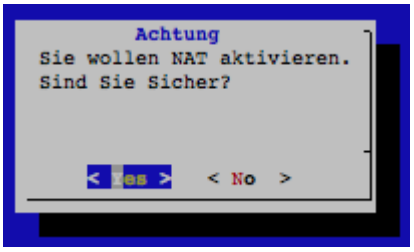

Abbildung 141: Quick Guide, Einschaltbestätigung DHCP/DNS Dienst

Das war’s schon, nun läuft der DHCP/DNS Dienst bereits im Hintergrund und wir können mit der restlichen Konfiguration fortfahren.

Starten des NAT Modus

PFDialogControl läuft bereits jetzt im Routing Modus, wenn sie diesen Modus wüschen, müssen Sie hier nichts tun. Sagen Ihnen aber Begriffe wie Routingtabelle oder statische Route – Konfiguration nichts, dann sollten wir an dieser Stelle besser NAT aktivieren. Zu diesem Zweck gehen wir zurück ins Hauptmenü und wählen den Punkt **NAT Einstellungen** aus. So gelangen wir in das NAT Menü, wo wir NAT aktivieren können.

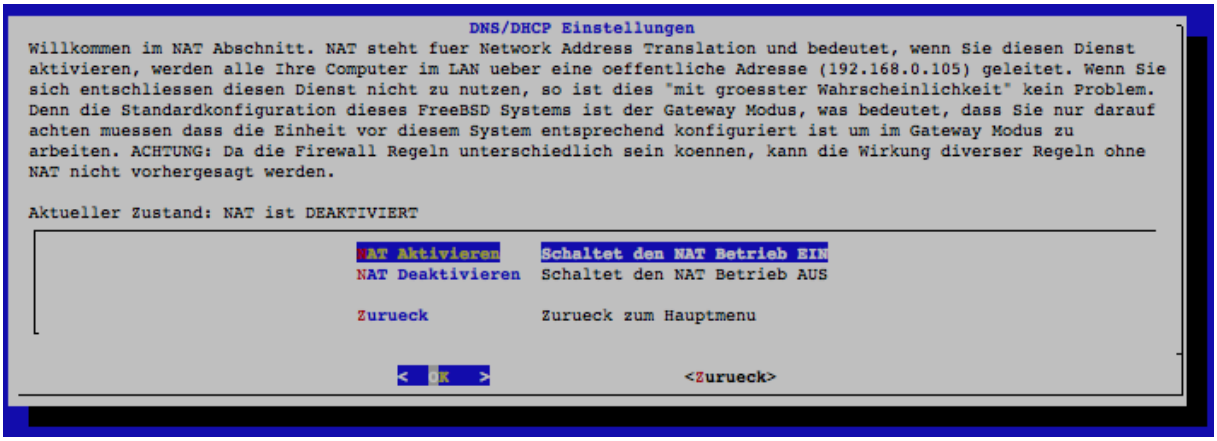

Abbildung 142: Quick Guide, NAT Hauptmenü

➔ Schritt 13

Hier wählen wir den Punkt **NAT Aktivieren** aus und das war's schon. Sollten Sie in „Interface Einstellungen" etwas vergessen haben zu konfigurieren, so wird dieser Fehler hier abgefangen und mittels Infofenster darauf hingewiesen.

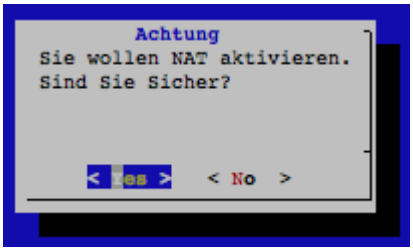

Abbildung 143: Quick Guide, NAT aktivieren

Die Portweiterleitung

In unserem Anleitungsszenario haben wir einen Webserver namens Client 1 (IP = 192.168.100.101/24). Diesen möchten wir von aussen, durch PFDialogControl erreichbar machen. Hierzu verwenden wir die Portweiterleitung unter Menü Punkt Firewall. Wenn wir nun vom Hauptmenü in Menü Firewall gehen, dann sehen wir folgendes Bild:

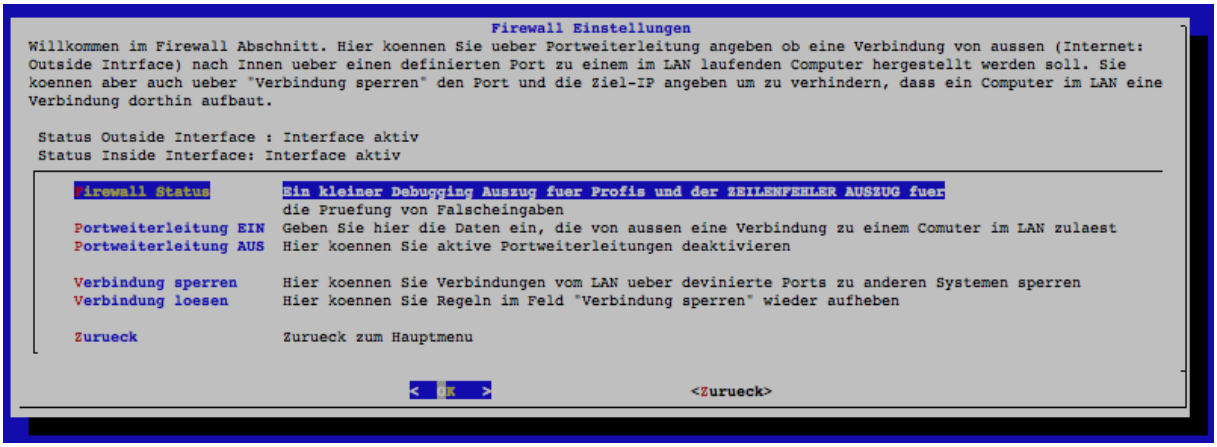

Abbildung 144: Quick Guide, Hauptmenü Firewall

➔ Schritt 14

In diesem Menü wählen **Portweiterleitung EIN** und gelangen zum Listing Menü, wo wir alle unser bereits erstellten Portweiterleitungen sehen. In unserem Fall ist die Liste noch leer. Nun wählen wir **yes** um ins Erstellungsmenü zu gelangen.

Wenn alles geklappt hat, sehen wir nebenstehendes Menü. Hier müssen folgende Punkte eingetragen werden.

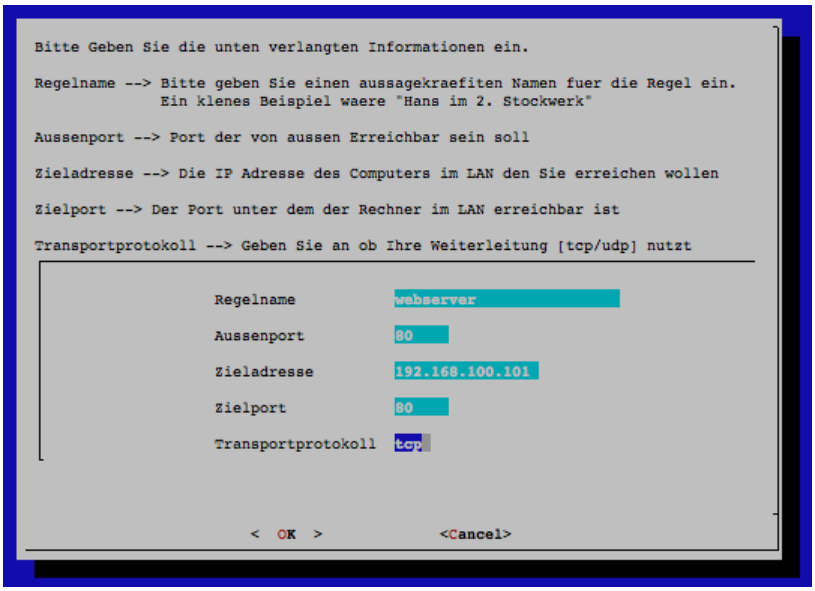

Abbildung 145: Quick Guide, neue Portweiterleitung erstellen

- Regelname: Geben Sie hier einen aussagekräftigen Namen ein. In unserem Beispiel **webserver**.
- Aussenport: Hier geben Sie den Port, welcher von aussen angesprochen werden muss, an. In unserem Fall **Port 80**
- Zieladresse: Das ist die IP des Rechners innerhalb des Inside Networks. In unserem Fall **192.168.100.101**
- Zielport: Ist der Port, unter welchem Client 1 den Webserver bereitstellt, also hier ebenfalls **Port 80**.
- Transportprotokoll: Unterstützt werden TCP und UDP. In unserem Beispiel geben Sie **tcp** ein.

Nun nur noch mit OK bestätigen.

Sie erhalten eine Zusammenstellung mit der gerade eingegebenen Portweiterleitung, welche Sie Prüfen und bestätigen müssen.

Abbildung 146: Quick Guide, Regelbestätigung

➔ Schritt 15

Nach Bestätigung der Korrektheit gelangen Sie wieder ins Listing Menü, wo die neue Portweiterleitung in der Liste ersichtlich sein sollte.

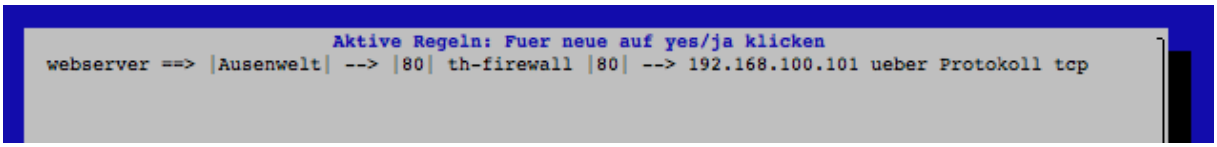

Abbildung 147: Quick Guide, neu erstellte Portweiterleitung

Das war's schon, die Portweiterleitung ist ab jetzt aktiv und kann genutzt werden.

Wenn Sie eine Portweiterleitung wieder entfernen möchten, so wählen Sie den Menü Punkt **Portweiterleitung AUS**. Sie werden wieder in ein Listing Menü gelangen, wo Sie mittels **yes** in ein Eingabemenü weitergeleitet werden. Dort geben Sie den Namen der Portweiterleitung ein und bestätigen die Löschung. Schon ist die Regel aus dem Regelwerk verschwunden.

Eine Verbindung sperren

Nun solange wir im Menü Firewall sind, können wir gleich Client 2 (IP = 192.168.100.102), wegen exzessiver Nutzung der HFU Website, die Verbindung dorthin verbieten. Hierzu gehen wir in Menü **Verbindung sperren**, wo wir wieder vor einem noch leeren Listing Menü stehen.

➔ Schritt 16

Hier wählen wir wieder **yes** aus, um ins Eingabemenü zu gelangen. Hier sind folgende Parameter anzugeben:

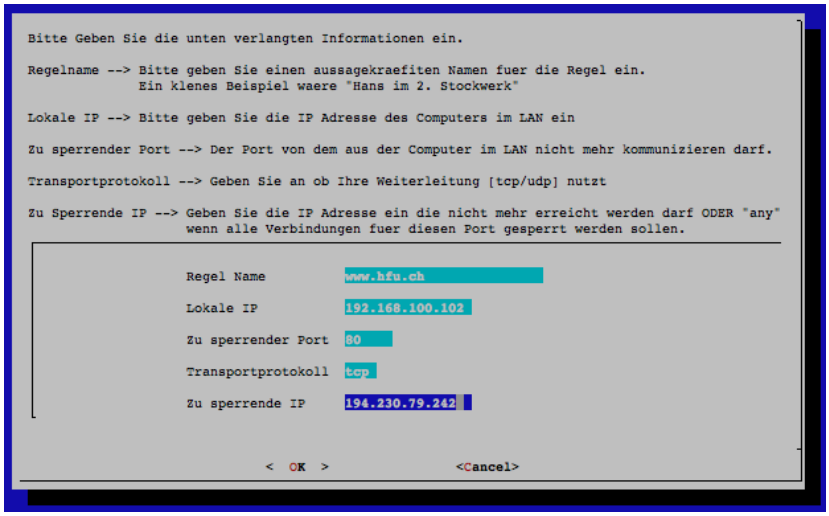

Abbildung 148: Quick Guide, Eingabe einer neuen Verbindungssperre

- Regelname: Auch hier geben wir einen aussagekräftigen Namen ein. Hier **www.hfu.ch**.
- Lokale IP: Ist die IP des Rechners im Inside Network, für welchen die Regel gelten soll. In unserem Fall Client 2 (192.168.100.102)
- Zu sperrender Port: Ist der spezifische Port, welchen Client 2 nicht mehr ansprechen darf. Hier **Port 80** (Webserver).
- Transportprotokoll: Hier geben wir http typisch **tcp** ein.
- Zu sperrende IP: Ist die IP – Adresse des fremden Zielsystems, welche nicht mehr Angesprochen werden darf In unserem Beispiel ist das die IP von www.hfu.ch (194.230.79.242). Sie können aber auch **any** benutzen, wenn der zu sperrende Port für jede fremde IP gelten soll.

Nun noch mit OK bestätigen und schon ist die Verbindung gesperrt.

Auch hier können Sie die Verbindungssperre wieder aufheben, indem Sie zu Menü Punkt Verbindung lösen gehen und den zu löschenden Regelnamen eingeben.

Mithilfe des auf dem Anleitungsszenarios basierenden Quick Guide und etwas Übung durch probieren, sollte die Bedienung von PFDialogControl schnell erlernbar sein. Das intuitive Menü sollte im Allgemeinen die Bedienung erleichtern. Es gilt das gleiche wie für jede neu installierte Software, je mehr Sie sie benutzten, desto besser können Sie damit umgehen.

11 Persönliches Fazit über die Vordiplomarbeit

Die Arbeit in einem solchen Umfang, stelle sich als grosse Herausforderung dar. Weniger im technischen, als im dokumentarischen Sinne, da mir eine solch umfangreiche Dokumentation meiner Arbeit eher neu war. Bisher war ich es mir nur gewohnt Arbeiten in ähnlicher Form wie PFDialogControl für mich selbst zu erstellen, wobei die Dokumentation lediglich in Notizform existierte, oder gar nicht vorhanden war. Daher war die detaillierte Planung absolutes Neuland, wo ich mich bei gewissen Punkten stets fragen musste; reicht dies wirklich oder entspricht diese überhaupt der geforderten Aufgabe. Doch nach einer gewissen Zeit ging auch diese Aufgabe in eine Routine über, wo doch langsam ein Gefühl für die strukturierte Dokumentation entstand. Im Grossen und Ganzen kann ich sagen, dass dies doch eine eher positiv prägende Erfahrung war, welche mir zeigte wie viel dokumentarisches hinter einem kleinen Tool wie PFDialogControl stecken kann. Was mir aber am meisten gefallen hat, ist das eigentliche Ergebnis welches ich mit PFDialogControl erzielt habe. Denn die Idee zu diesem Werkzeug entsprang der Notwendigkeit innerhalb der von mir betreuten Netzwerkinfrastruktur, wo schon seit langem ein Bedürfnis nach einer solchen Lösung da war. Bereits während der Entwicklung, aber spätestens bei den Tests erkannte ich, dass PFDialogControl genauso geworden ist wie ich es wollte, aber auch brachte. Es wird in naher Zukunft bereits seinen produktiven Platz in meiner Netzwerkinfrastruktur finden und hoffentlich die ersehnte Entlastung im Bereich Konfigurationsfreundlichkeit liefern, welche ich mir erhofft habe.

12 Verzeichnisse und Quellen

12.1 Glossar

Dialog	Ein Programm welches innerhalb der Shell ausführbar ist und Menüstrukturen mit Buttons bilden kann.
Dnsmasq	Ist ein in der Unix Welt bekanntes Tool, welches einen DHCP und DNS vereint. So ist eine simple Konfiguration möglich, ohne sich gross um den DNS Dienst kümmern zu müssen.
EPT – Nested Paging	Eine Hardware basierte Erweiterung die virtuellen Maschinen beschleunigt, indem der Prozessor die virtuell geführten Adressräume in seine realen umwandelt.
FreeBSD	Ein Betriebssystem (Derivat) der BSD (Berkeley Software Distribution) Familie, welches für seine hohe Netzwerkperformance berühmt ist.
GUI	Grafischer Benutzer Interface, meist durch Fenstersteuerung realisiert. Im Fall von PFDialogControl ist dies dialog.
Hypervisor	Eine softwarebasierte Abstraktionsschicht, welche das Erstellen von virtueller Hardware ermöglicht, auf der Betriebssysteme installiert werden können
Jedit	Ein Editor auf Java Basis, welcher die Shell Syntax hervorheben kann.
Kernel	Der Kern eines Betriebssystems. In dieser Arbeit ist mit Kernel stets der FreeBSD Systemkern gemeint.
Lease Time	Die Zeitdauer, welche eine durch DHCP vergebene IP – Adresse gültig ist, bis sie erneuert werden muss.
NAT	Ein Verfahren, meist auf Routern implementiert, welches das Übersetzten vieler lokaler IP – Adresse in eine öffentliche ermöglicht.
Sed / gsed	Stream Editor; Ein Tool welches eine Textdatei auslesen kann und den Inhalt nach Pattern absucht bzw. modifiziert.
Shell	Die Kommandozeilen Schnittstelle zwischen Betriebssystemkern und dem User.
SmartOS	Ein auf Solaris basierendes Betriebssystem, mit dessen Hilfe Solaris Container und echte virtuelle Maschinen virtualisiert werden können.
VLAN	Ein virtuelles LAN, das meist in Layer 2 Switches konfiguriert werden kann, um innerhalb dem physikalischen LAN weitere logische Trennungen zu ermöglichen.
Wizard	Meist ein strukturiertes Menü, welches gestartet wird und den User durch einen Prozess bis zum Ende führt.
Zone (Solaris)	Ist ein Ressourcen Container, welcher nach aussen als echtes Betriebssystem auftritt, aber tatsächlich einen Master Kernel des Hypervisors nutzt.

12.2 Abbildungsverzeichnis

12.3 Tabellenverzeichnis

12.4 Quellenverzeichnis

12.4.1 Bücherverzeichnis

- **Shell – Skript Programmierung:** ISBN 978-3-8266-9134-8. Ist ein Standardwerk zur Shell – Skript Programmierung, welches alle Shells und die meisten Unix/ -aähnlichen Systeme umfasst.
- **Shell – Programmierung:** ISBN 3-89842-683-1. Ist ebenfalls ein Shell Standardwerk, welches als Online – Buch unter http://openbook.rheinwerk-verlag.de/shell_programmierung/ Open Source zur Verfügung steht.

12.4.2 Internetadressen

- **FreeBSD Handbuch:** http://www.freebsd.org/doc/de_DE.ISO8859-1/books/handbook/index.html
- **SmartOS:** Zum Herunterladen der Software für das Testsystem (Hypervisor). https://smartos.org/
- **Datasets:** Hier wurden die Abbilder für SmartOS heruntergeladen. http://datasets.at/ui/#!/home

13 Beilagen

13.1 Der Dokumentation beiliegend

Beilage	Titel	Bemerkungen
1	Aufgabenstellung	Original, siehe Seite 5
2	Datenträger (CD – Rom)	
2.1	PFDialogControl in einem gezippten Tar – Archiv	
2.2	PFDialogControl im entpackten Zustand	
2.3	Ein elektronische Form dieser Arbeit im docx und pdf Format	
2.4	Projektpläne in den Formen PDF, JPEG, html und original gan – Format	
2.5	Externe Software auf dem Datenträger für Windows, MacOS, div. Linux Distributionen	
2.5.1	Jedit 5.2.0	
2.5.1	Gantt – Project 2.7 –r 1891	
3	Formelles	
3.1	Original Aufgabenstellung	
3.2	Original VDA Antrag	
3.3	2x Betreuungssitzung	

14 Anhang

14.1 Anhang A (Empfehlung für FreeBSD Installation)

Für den Betrieb von PFDialogControl benötigen Sie ein System mit FreeBSD Installation. Aktuelle ISO Abbilder können Sie unter **http://www.freebsd.org/de/where.html** herunterladen und auf eine CD brennen bzw. in Ihrer Virtualisierungsumgebung ausführen. Der FreeBSD Installer ist simpel im Aufbau und erlaubt Ihnen eine schnelle Installation. Wenn Sie Schwierigkeiten bei der Installation haben sollten, finden Sie Hilfe unter **http://www.freebsd.org/doc/de_DE.ISO8859-1/books/handbook/install-start.html**. Der Installer ist aber in seiner Struktur so einfach gestaltet, dass Sie lediglich immer auf „OK“ klicken müssen. So werden Sie zwar ein FreeBSD mit englischem Tastaturlayout erhalten, aber es wird funktionsfähig sein. Für den korrekten Betrieb von PFDialogControl müssen Sie aber bei der Installation auf folgende Punkte achten:

- Beim Punkt „Einen User Erstellen“, **müssen** Sie einen User anlegen. Da ein SSH Login unter FreeBSD mit dem root Account aus Sicherheitsgründen nicht möglich ist.
- Wenn Sie sich als Default User unter FreeBSD per SSH anmelden, müssen Sie auch in den Superuser Modus wechseln können. **Hierfür ist es von grösster Dringlichkeit, dass Sie während der Erstellung des neuen Users, diesen auch bei der Frage nach zusätzlichen Gruppenmitgliedschaften in die Gruppe wheel eintragen.**

Diese speziellen Richtlinien bezüglich des Default Users müssen beflogt werden, um eine SSH – Verbindung mit FreeBSD herstellen zu können, welche die Bedienung von PFDialogControl erst ermöglicht.

Dies ist alles was Sie beachten müssen. Die restlichen Abhängigkeiten erledigt der Installer von PFDialogControl, welcher Sie sicher durch den Installationsprozess führt.

14.2 Anhang B (Versionsverwaltung)

PFDialogControl erscheint in der Version 1.0.3 Release. Hierbei wird ein System mit drei Ziffern verwendet, welches Aussagen über die darin vorgenommenen Änderungen vornimmt. Die erste Ziffer kann nur geändert werden, wenn elementare Änderungen in der Grundfunktionsweise von PFDialogControl vorgenommen wurden. Ziffer zwei zeigt Änderungen in den einzelnen Diensten bzw. in deren Funktionsprinzip. Die dritte Ziffer gibt Auskunft über kleinere Änderungen in den einzelnen Diensten bzw. Funktionen, welche keinen direkten Einfluss auf das Grundfunktionsprinzip haben. Solche Änderungen betreffen meist die Integration von zusätzlichen Sicherungen, oder bspw. Wechsel im Darstellungsverfahren, wie etwa die Verwendung von dialog Infoboxen anstelle von Massageboxen.