UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Policy-basierte Internetregulierung durch Router

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Informatik

vorgelegt von

Rainer Weißenfels

Erstgutachter:	Prof. Dr. Rüdiger Grimm Institut für Wirtschafts- und Verwaltungsinformatik
Zweitgutachter:	Dipl.-Inform. Andreas Kasten Institut für Wirtschafts- und Verwaltungsinformatik

Koblenz, im Februar 2014

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....

(Ort, Datum) (Rainer Weißenfels)

Zusammenfassung. Diese Arbeit beschreibt den Entwurf und die Implementierung eines Software-Routers für eine Policy-basierte Internetregulierung. Die Grundlage bildet die von Kasten und Scherp beschriebene Ontologie InFO. InFO dient der System-unabhängigen Beschreibung von Regulierungsmaßnahmen. Zudem ermöglicht InFO eine transparente Regulierung durch Verknüpfung der Regulierungsmaßnahmen mit Hintergrundinformationen. Die InFO-Erweiterung RFCO erweitert die Ontologie um Router-spezifische Entitäten. Es wird ein Software-Router entwickelt, der die RFCO auf IP-Ebene umsetzt. Die Regulierung wird transparent gestaltet, indem betroffene Nutzer vom Router über Regulierungsmaßnahmen informiert werden. Die Router-Implementierung wird exemplarisch in einer virtuellen Netzwerkumgebung getestet.

Abstract. This diploma thesis describes the concept and implementation of a software router for policy-based Internet regulation. It is based on the ontology InFO described by Kasten and Scherp. InFO is destined for a system-independent description of regulation mechanisms. Additionally, InFO enables a transparent regulation by linking background information to the regulation mechanisms. The InFO extension RFCO extends the ontology with router-specific entities. A software router is developed to implement RFCO at the IP level. The regulation is designed to be transparent by letting the router inform affected users about the regulation measures. The router implementation is exemplarily tested in a virtual network environment.

Danksagung
Ich bedanke mich bei Herrn Andreas Kasten für seinen Rat und seine Unterstützung bei der Erstellung dieser Arbeit und seine Geduld.
Meinen Eltern und meiner Familie danke ich für die Unterstützung und den Rückhalt in den ganzen Jahren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

AS	Autonomes System
CIDR	Classless Inter-Domain Routing
DBMS	Datenbankmanagementsystem
DMZ	Demilitarized Zone
DNS	Domain Name System
ENISA	European Network and Information Security Agency
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IHL	IP Header Length
InFO	Information Flow Ontology
IP	Internet Protocol
ISP	Internet Service Provider
NAT	Network Address Translation
JRE	Java Runtime Environment
OSI	Open Systems Interconnection
RFCO	Router-based Flow Control Ontology
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TOS	Type of Service
TTL	Time to live
UDP	User Datagram Protocol
OWL	Web Ontology Language
URL	Uniform Resource Locator
URI	Uniform Resource Identifier

Kapitel 1

Einleitung

Die *OpenNet Initiative*[1] untersucht weltweit Formen der Internetregulierung und dokumentiert diese in ihren Berichten. Darüber hinaus erstellt sie für einzelne Länder oder Regionen Profile, die das Ausmaß der Internetregulierung darstellen. In einer 2006 durchgeführten Studie stellen die Mitarbeiter der Initiative einen Trend für eine weltweite Zunahme der Regulierungsmaßnahmen fest [ZP08]. Auch das *Internet & Jurisdiction*-Projekt[2] dokumentiert in einem 2013 veröffentlichten Bericht [Int13] ähnliche Beobachtungen. Die Regulierungsmaßnahmen äußern sich in unterschiedlichen Formen wie Internetzensur, Zugangsbeschränkung oder Überwachung [DPRZ10]. Die regulierten Inhalte und die Hintergründe für deren Regulierung sind vielfältig und können sich von Staat zu Staat stark unterscheiden [FV08]. So werden neonazistische Inhalte aufgrund der Historie vor allem in Frankreich und Deutschland verfolgt, während diesen in den Vereinigten Staaten aufgrund der weiter ausgelegten Meinungsfreiheit toleranter begegnet wird [Tim03, ZP08].

Auch die eingesetzten technischen Mittel zur Regulierung unterscheiden sich. Sie reichen von Zugangssperren für bestimmte IP-Adressen und DNS-Manipulation über den Einsatz von Proxy-Servern und Filterung von Suchmaschinenergebnissen bis hin zur Sperrung von Internetseiten oder Abschaltung von Servern, über die die betroffenen Inhalte verfügbar gemacht werden [MA08]. Je nachdem, wie die Regulierungsmaßnahme technisch durchgeführt wird, ist sie für den einzelnen betroffenen Nutzer oft nicht als solche erkennbar. Geschieht die

[1] https://opennet.net/ - Abruf: 24.02.2014
[2] http://www.internetjurisdiction.net/ - Abruf: 24.02.2014

Regulierung auf Anwendungsebene, kann der betroffene Nutzer leicht darüber unterrichtet werden: Bei der Filterung von Suchmaschinenergebnissen kann der Betreiber auf die vorgenommene Regulierung hinweisen. Bei der Sperrung von Internetseiten kann durch den Internet Service Provider oder die regulierende Behörde ein Hinweis durch Umleitung auf eine andere Internetseite erfolgen. Auch wenn hier in beiden Fällen eine Nennung der genauen Gründe oft entfällt.

Im Gegensatz dazu ist die Regulierung auf der IP-Ebene durch Router oder DNS-Server unter Umständen nicht als solche erkennbar: So lässt sich eine Zugangssperre nicht direkt von Störungen oder temporären Überlastungen im Netzwerk unterscheiden. Zudem können bei einer derartigen Regulierung ungewollt Seiteneffekte produziert werden. So führte im Februar 2008 der Versuch der pakistanischen Telekom, den Zugriff auf das Videoportal *YouTube*[3] für die pakistanische Bevölkerung zu blockieren, durch eine fehlerhafte Router-Einstellung dazu, dass weltweit der Zugriff auf YouTube für mehrere Stunden nicht möglich war [RIP08].

Der geringen Transparenz und Nachvollziehbarkeit von Maßnahmen zur Internet-Regulierung begegnet die von Kasten und Scherp beschriebene Information Flow Ontology (InFO) [KS13]. Sie liefert ein einheitliches Sprachmodell (*Ontologie*) für unterschiedliche technische Systeme zur Internetregulierung. Die Ontologie ermöglicht es, Regulierungsmaßnahmen unabhängig von ihrer technischen Umsetzung in Policys zu beschreiben. Zusätzlich bietet InFO die Möglichkeit, die Regulierung transparent zu gestalten. Es wird eine Verknüpfung zwischen der technischen Regulierungsmaßnahme und den Motiven der Regulierung hergestellt. So können von der Regulierung betroffene Internet-Nutzer beispielsweise auf konkrete Gesetzesparagraphen verwiesen werden. Die Abbildung auf die unterschiedlichen technischen Systeme zur Durchsetzung der Regulierung wird mit Domänen-spezifischen Erweiterungen der InFO vorgenommen. Die InFO-Erweiterung für Router (RFCO) [Kas12b] wird für diese Arbeit als Grundlage genommen. Basierend auf der RFCO wird ein softwaretechnisches System entwickelt, dass die Policy-basierte Internetregulierung auf Routern ermöglicht. Das zu entwickelnde System übersetzt die in Policys verfassten Regulierungsmaßnahmen zur Anwendung auf einem Software-Router mit Firewall. Weiter erfolgt durch das System eine Benachrichtigung betroffener Nutzer, in der auf Art und Hinter-

[3] http://www.youtube.com/ - Abruf: 24.02.2014

grund der Regulierung verwiesen wird. Das System wird zudem exemplarisch in einer virtuellen Netzwerkumgebung getestet.

Der weitere Aufbau dieser Arbeit stellt sich wie folgt dar: Zunächst werden in Kapitel 2 die Grundlagen zu Routern und Firewalls und der verwendeten Ontologie InFO erläutert. In Kapitel 3 werden die Anforderungen an das zu entwickelnde System zur Internetregulierung mit Routern beschrieben. Daran anschließend wird das Konzept und die Architektur der entwickelten Lösung vorgestellt. In Kapitel 4 wird eine mögliche Implementierung mit einer Firewall beschrieben. Zudem wird die Funktion der Implementierung in einer Netzwerksimulation exemplarisch nachvollzogen. Die Arbeit schließt in Kapitel 5 mit einer Zusammenfassung und Bewertung der Arbeit und einem Ausblick auf mögliche Weiterentwicklungen.

Kapitel 2

Grundlagen

In diesem Kapitel werden die inhaltlichen und technischen Grundlagen der Arbeit beschrieben. Das Verständnis dieser Grundlagen wird im weiteren Verlauf vorausgesetzt. Zunächst werden Funktion und technische Grundlagen eines Routers behandelt (Abschnitt 2.1). Darauf folgt die Darstellung des Aufbaus und der Funktionalität von Firewalls (Abschnitt 2.2). Beide Abschnitte dienen im Wesentlichen einer Bestimmung der Begriffe „Router“ und „Firewall“ und ihrer Verwendung in dieser Arbeit. Die technischen Aspekte werden hier nur verkürzt dargestellt. Das Kapitel schließt mit einer Einführung in die InFO und deren Routerspezifische Erweiterung RFCO (Abschnitt 2.3).

2.1 Grundlagen zu Routern

Die Grundlagen zu Routern werden in diesem Abschnitt behandelt. Zunächst wird in Abschnitt 2.1.1 zur Einführung ein Überblick über die Schichten des OSI-Referenzmodells und des TCP/IP-Modells gegeben. Abschnitt 2.1.2 behandelt IPv4 als das in dieser Arbeit behandelte Protokoll der Internetschicht. Aufbau und Funktion eines Routers werden in Abschnitt 2.1.3 erklärt.

2.1.1 OSI- und TCP/IP-Referenzmodell

Zur Beschreibung der Kommunikation im Internet haben sich zwei Modelle etabliert. Das formale OSI-Referenzmodell [Int94] und das aus der Praxis entwi-

ckelte TCP/IP-Referenzmodell [Bra89]. Beide Modelle sollen in diesem Abschnitt kurz vorgestellt werden.

Das OSI-Modell ist ein Referenzmodell zur Verbindung von Rechnern [PD11]. Es unterteilt eine Verbindung zwischen Rechnern in sieben Schichten (siehe linke Spalte der Tabelle 2.1) [Int94]. Die unterschiedlichen Schichten bauen aufeinander auf. Für jede Schicht gibt es ein oder mehrere Protokolle, die die Funktionen dieser Schicht bereitstellen. Die benötigten Funktionen einer Schicht werden von der darunter liegenden Schicht bereitgestellt. Ebenso stellt eine Schicht Funktionen für die darüber liegende Schicht bereit.

OSI-Schicht		**TCP/IP-Schicht**	
7	Anwendungsschicht (*application layer*)	**4**	Anwendungsschicht (*application layer*)
6	Darstellungsschicht (*presentation layer*)		
5	Kommunikationssteuerungsschicht (*session layer*)		
4	Transportschicht (*transport layer*)	**3**	Transportschicht (*transport layer*)
3	Vermittlungsschicht (*network layer*)	**2**	Internetschicht (*internet layer*)
2	Sicherungsschicht (*data link layer*)	**1**	Netzzugangsschicht (*link layer*)
1	Bitübertragungsschicht (*physical layer*)		

Tabelle 2.1 Schichten des OSI-Referenzmodells (links) und des TCP/IP-Referenzmodells (rechts) im Vergleich [Hun02, TW10].

Die *Bitübertragungsschicht* kümmert sich um die Übertragung von Bits über ein Medium. Aufgabe der *Sicherungsschicht* ist es, diese Bits in Frames zu unterteilen und fehlerfrei zu übertragen. Die *Vermittlungsschicht* ist dafür verantwortlich, dass Pakete von ihrer Quelle zum Ziel über mehrere Netzwerkknoten (*Hops*) geleitet werden. Die *Transportschicht* stellt eine Ende-zu-Ende-Verbindung zwischen Quelle und Ziel her. Sie unterteilt die eingehenden Daten der höheren Schichten in kleinere Einheiten, damit sie von den unteren Schichten übertragen werden können. Am Ziel stellt die Transportschicht sicher, dass die empfangenen Daten

wieder in der richtigen Reihenfolge an die höheren Schichten geleitet werden. Die *Kommunikationssteuerungsschicht* sorgt für den Aufbau und Erhalt von Sitzungen (*Sessions*) zwischen entfernten Prozessen. Sie bündelt verschiedene Datenströme der Transportschicht [PD11]. Aufgabe der *Darstellungsschicht* ist die korrekte Darstellung der ausgetauschten Informationen in Abhängigkeit vom darunter liegenden System. Die *Anwendungsschicht* enthält Protokolle wie HTTP, die vom Nutzer benötigt werden und stellt die Verbindung zu den darunter liegenden Schichten her.

Eine vereinfachte Sicht auf die Verbindung zweier Rechner bietet das TCP/IP-Referenzmodell (siehe rechte Spalte der Tabelle 2.1). Obwohl das OSI-Modell eine detailliertere Darstellung bietet, wird zur Veranschaulichung der Internet-Kommunikation zumeist das TCP/IP-Modell verwendet [PD11]. Das TCP/IP-Modell unterteilt sich in vier statt sieben Schichten [Bra89]. Die drei oberen Schichten des OSI-Modells werden zur *Anwendungsschicht* zusammengefasst. Darstellung und Präsentation werden hier nicht mit gesonderten Schichten unterschieden [TW10]. Die Bitübertragungsschicht und die Sicherungsschicht werden im TCP/IP-Modell zu einer *Netzzugangsschicht* zusammengefasst. Die Vermittlungsschicht des OSI-Modells entspricht in etwa der *Internetschicht* des TCP/IP-Modells. Die *Transportschicht* nimmt im TCP/IP-Modell die gleichen Aufgaben wahr wie die gleichnamige Schicht des OSI-Modells. Für den weiteren Verlauf soll das TCP/IP-Modell als Referenz für die unterschiedlichen Kommunikationsschichten benutzt werden.

Den einzelnen Schichten des TCP/IP-Modells (wie auch des OSI-Modells) können bestimmte Protokolle zugeordnet werden. Verbreitete Protokolle sind *IP* [Pos81b] für die Internetschicht und *TCP* [Pos81c] und *UDP* [Pos80] für die Transportschicht [Hun02]. Im nächsten Abschnitt wird das für Router essentielle Internet Protocol (IP) in der Version 4 vorgestellt.

2.1.2 IPv4

Das Internet Protocol (IP) ist ein verbindungsloses Protokoll der Internetschicht (siehe Tabelle 2.1). Es überträgt die Daten der Transportschicht von einem Quell-Host zu einem Ziel-Host ohne zuvor eine Verbindung aufzubauen. Die zu übertragenden Daten werden dazu nach Bedarf fragmentiert und in *Paketen* gekapselt.

Die Pakete werden über verschiedene Netzwerkknoten[1] bis zum Ziel weitergereicht. Ein IP-Paket besteht aus einem Kopf (*Header*) mit Übertragungsinformationen und den Nutzdaten (*Payload*) der Transportschicht. Zu Beginn des Jahres 2014 ist Version 4 des Internet Protocol (IPv4) noch immer die verbreitetste Version. Die Ablösung durch Version 6 geschieht nur sehr langsam[2] [TW10]. Daher beschränkt sich die Betrachtung dieser Arbeit auf IPv4.

Der Header eines IP-Pakets ist in Abbildung 2.1 schematisch dargestellt. Der Header ist mindestens 20 Bytes groß. Zusätzliche Optionen können den Header auf insgesamt 60 Bytes vergrößern. Die ersten vier Bits legen die IP-*Version* fest. Der Wert ist in IPv4 stets 4. Die *IP Header Length (IHL)* gibt die Länge des Headers an. Das Feld *Type of Service (TOS)* wurde ursprünglich zur Priorisierung von Paketen eingeführt. Inzwischen dient es zur Flusskontrolle [RFB01]. Die *Gesamtlänge* bestimmt die gesamte Länge des Pakets. Die Felder *Identifikation*, *Flags* und *Fragment Offset* dienen der Fragmentierung und späteren Assemblierung von Paketen. *Time to live (TTL)* enthält die verbleibende Anzahl an Hops bevor das Paket verworfen wird. *Protokoll* gibt den Transportprotokolltyp der Payload an. Die *Header-Prüfsumme* erlaubt eine Integritätsprüfung der Header-Daten zum Ausschluss von Übertragungsfehlern. Die *Quell-IP-Adresse* und die *Ziel-IP-Adresse* sind jeweils 32 Bit breit. Den Schluss des Headers bilden optionale *Optionen*.

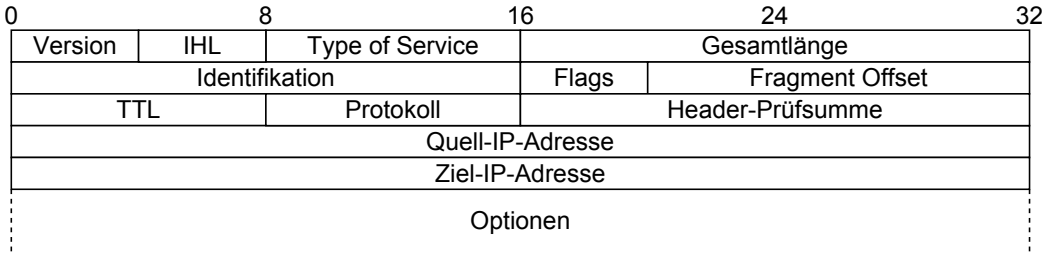


Abbildung 2.1 Header eines IPv4Pakets [Pos81b].

Anhand der IP-Adressen ist eine eindeutige Adressierung von Hosts möglich. Das 32 Bit breite Adressfeld erlaubt jedoch theoretisch nur 2^{32} Adressen. Um u.a. der eng begrenzten Menge an möglichen IP-Adressen zu begegnen, wurde Network Address Translation (NAT) eingeführt [SH99]. NAT ermöglicht es, die

[1] Sofern sich Quell- und Ziel-Host nicht im selben Netzwerk befinden, müssen die Pakete über einen oder mehrere Router transportiert werden.

[2] http://www.google.com/intl/en/ipv6/statistics.html - Abruf: 21.02.2014. Zu Beginn des Jahres 2014 griffen weniger als 3% der Nutzer von Google über IPv6 auf den Dienst zu.

IP-Adressen eines Netzwerks nach außen hin zu maskieren. So ist es möglich, ganze Adressbereiche nach außen hinter einer einzigen IP-Adresse zu versammeln [Per99]. NAT übersetzt dazu bei ausgehenden Paketen die *lokale* IP-Adresse und Portnummer in eine *globale* IP-Adresse kombiniert mit einer eindeutigen, temporären Portnummer. Bei eingehenden Paketen von außen wird diese globale IP-Adress-Portnummer-Kombination dann in die lokalen Adresse übersetzt. NAT muss dazu eine Tabelle führen, die eine Zuordnung zwischen lokalen und globalen Adressen erlaubt. Durch Übersetzung der Adressen manipuliert NAT Informationen der Internet- und Transportschicht und verhindert eine Ende-zu-Ende-Konnektivität [PD11]. Die Hosts eines NAT-Netzwerks sind von außen nicht direkt adressierbar. Die Architektur des Netzwerks bleibt somit verborgen.

NAT kann entweder durch dedizierte Netzwerkkomponenten anwendet werden oder als Funktion in einen Router (siehe folgenden Abschnitt 2.1.3) integriert werden.

2.1.3 Aufbau und Funktion von Routern

Ein Router ist eine Netzwerkkomponente, die mehrere Netzwerke miteinander verbindet und dazu maßgeblich das Internet Protocol nutzt [PD11, TW10]. Dazu besitzt ein Router mehrere Netzwerkschnittstellen. Eingehende Pakete werden durch den Router weitergeleitet, indem die Ziel-IP-Adresse ausgewertet wird. Basierend auf dieser IP-Adresse wird die Netzwerkschnittstelle ausgewählt, über die das Paket weitergeleitet wird. Der Router führt dazu eine *Forwarding-Tabelle*, die Netzwerkadressen bzw. -adressbereiche einer Netzwerkschnittstelle zuordnet. Die direkt angeschlossenen Netzwerke sind dem Router nach seiner Initialisierung bekannt. Weiter entfernte Netzwerke erfährt der Router über regelmäßige Benachrichtigungen durch andere Router. Diese Benachrichtigungen werden auf dem Router von einem *Routing-Algorithmus* verarbeitet. Der Routing-Algorithmus erzeugt aus diesen Benachrichtigungen eine *Routing-Tabelle* und aktualisiert sie fortwährend. Basierend auf dieser Routing-Tabelle aktualisiert der Routing-Algorithmus dann die Forwarding-Tabelle, indem er für alle unterscheidbaren Ziele den optimalen Pfad und damit die entsprechende ausgehende Netzwerkschnittstelle auswählt. Der optimale Pfad kann durch verschiedene Parameter

bestimmt sein. Für eine genaue Darstellung der Funktionsweise von Routern und Routing-Algorithmen sei auf [PD11] und [TW10] verwiesen.

Ursprünglich war die Herstellung von Konnektivität zwischen Netzwerken die einzige Aufgabe von Routern. Heutige Router integrieren aber zusätzliche Komponenten wie NAT (siehe Abschnitt 2.1.2) oder eine Firewall (siehe folgenden Abschnitt 2.2). Statt auf dedizierten Netzwerkkomponenten kann die Software eines Routers auch auf einem Rechner mit mehreren Netzwerkschnittstelle betrieben. Derartige *Software-Router* können mit geringen finanziellen Ressourcen aufgebaut werden [BFG+05].

2.2 Grundlagen zu Firewalls

In diesem Abschnitt werden die Grundlagen zu Firewalls erläutert. Es wird im Verlauf dieses Abschnitts eine Definition für *Firewall* erarbeitet, die für diese Arbeit verwendet wird. In Abschnitt 2.2.1 werden zunächst verschiedene Definitionen des Begriffs „Firewall“ dargelegt. Im darauf folgenden Abschnitt 2.2.2 werden die Bestandteile einer Firewall dargelegt. Im letzten Abschnitt 2.2.3 werden verschiedene Einsatzszenarien für Firewalls vorgestellt.

2.2.1 Definition

Für den Begriff *Firewall* gibt es in der Literatur sehr unterschiedliche Definitionen, die sich teilweise überschneiden.[3] Hier soll ein Überblick gegeben werden über verschiedene Definitionen. Die Definitionen sind nach ihrem Abstraktionsgrad geordnet. Begonnen wird mit der abstrakten Auffassung einer Firewall als Sicherheitskonzept. Die Liste endet mit Darstellung der Firewall als konkrete Definition Am Ende soll eine Definition bestimmt werden, die für diese Arbeit verwendet wird.

Sicherheitskonzept Abstrakt kann eine Firewall als ein *Sicherheitskonzept* verstanden werden [Bar01]. Ziel dieses Sicherheitskonzeptes ist es, den Zugriff auf schützenswerte Güter innerhalb eines Netzwerks einzuschränken. Insbesondere sollen Angriffe von außen abgewehrt werden. Zur Erreichung dieses Ziels werden u.a. Zugriffe von außen, aber auch Zugriff nach außen

[3] Vergleiche beispielsweise [Hun02] mit [Bar01].

eingeschränkt. Die Umsetzung des Sicherheitskonzepts ist nicht singulär sondern kann verschiedene Maßnahmen (z.B. Benutzerechteverwaltung) und technische Komponenten miteinbeziehen.

Netzwerkkomponente Als dedizierte *Netzwerkkomponente* kapselt eine Firewall den Zugriff auf ein Netzwerk [Hun02]. Eine solche *Hardware-Firewall* wird an den Übergangen des zu schützenden Netzwerks zu den angeschlossenen Netzwerken platziert. Sie fungiert dabei als Gateway zwischen dem zu schützenden Netzwerk und der Außenwelt.

Software Eine Firewall kann in einer engeren Definition auch als die *Software* bezeichnet werden, die auf einer Netzwerkkomponente läuft und den Zugriff zwischen Netzwerken beschränkt [ZCC00]. Eine solche *Software-Firewall* kann aber auch auf einem einzelnen Rechner betrieben werden und den Zugriff auf diesen einschränken. Letztere wird als *Personal Firewall* bezeichnet.

Funktion Zuletzt kann mit dem Begriff Firewall auch eine spezifische *Funktion* einer Software bezeichnet werden. Beispielhaft ist hier vor allem die Paketfilterung (siehe folgenden Abschnitt 2.2.2).

Für diese Arbeit wird unter einer Firewall eine Software verstanden, die auf einer Netzwerkkomponente installiert ist. Diese Netzwerkkomponente verbindet mehrere Netze miteinander. Die Software reguliert die Kommunikation zwischen den Netzen, d.h. sie schränkt den Zugriff zwischen den Netzen ein.

2.2.2 Aufbau und Funktion von Firewalls

Eine Firewall besteht grundlegend aus einer Kombination von *Paketfilter* und/ oder *Proxys* [Fre00]. Beide Begriffe sollen hier erläutert werden. Dabei wird auf den Paketfilter ausführlicher eingegangen als auf den Proxy, da letzterer im weiteren Verlauf der Arbeit nicht weiter behandelt wird.

Ein *Paketfilter* besteht aus einem Satz aus Regeln, die auf empfangene Pakete der Firewall angewendet werden. Die Regeln sind so verfasst, dass sie ein Paket zulassen (*allow*), es abweisen (*deny*) oder es anderweitig behandeln (z.B. umleiten) [Fre00]. Wird ein Paket zugelassen, passiert es die Firewall unverändert. Im Fall der Abweisung wird das Paket verworfen. Zur Umleitung des Pakets werden Änderungen am Paket-Header vorgenommen. Trifft keine der Regeln auf ein

eintreffendes Paket zu, wird eine Standardregel (*Default Policy*) angewendet, die gesondert definiert wird. Zur Definition der Regeln werden Informationen der Verbindungs- und Transportschicht genutzt, wie etwa Ziel- und Absenderadressen oder Portnummern (siehe Abschnitt 2.1.1) [Fuh00]. In ihrer Grundform erfolgt die Paketfilterung zustandsunabhängig. Jedes Paket wird unabhängig von den bereits vorher zwischen Hosts ausgetauschten Paketen betrachtet. Dynamische Paketfilter dagegen merken sich anhand ausgetauschter Pakete die logische Verbindung zwischen zwei Hosts. Damit ermöglichen sie es, dass beispielsweise Antwortpakete von außerhalb die Firewall passieren können. Zustandsabhängige Paketfilter nutzen zur Verbindungserkennung zusätzlich Informationen der Schichten oberhalb der Transportschicht (siehe Abschnitt 2.1.1) [Fuh00].

Ein *Proxy* wertet zur Regulierung die Informationen der Anwendungsschicht (siehe Abschnitt 2.1.1) aus. Der Proxy fungiert als „Stellvertreter“ für einen Host des zu schützenden Netzes. Verbindungsanfragen zu einem entfernten Rechner führt der Proxy für den Host aus. Auch die empfangenen Antworten nimmt der Proxy entgegen, bevor er sie dem Host weiterleitet. Dadurch hat der Proxy die Möglichkeit, Inhalte der Anwendungsebene zu regulieren. Ein Proxy kann auf ein Anwendungsprotokoll angepasst sein (*Application-Level-Proxy*) oder generisch (*Circuit-Level-Proxys*) sein [Bar01].

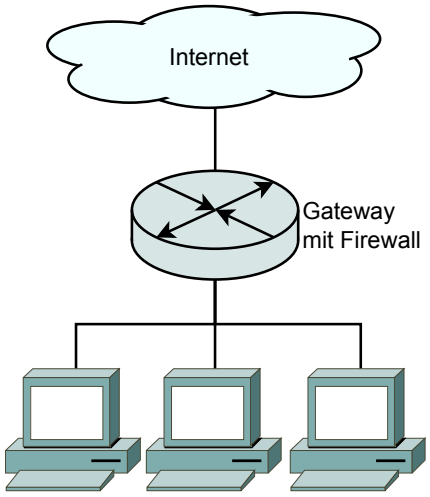
Die in Abschnitt 2.2.1 vorgenomme Definition einer Firewall soll hier erweitert werden: Zur Funktionalität der Firewall gehört die zustandsunabhängige Paketfilterung. Für die Paketfilterung können Regeln verfasst werden, die Kommunikation erlauben (*allow*) oder verbieten (*deny*). Es wird für die Paketfilterung eine *Default Policy* definiert, falls keine der Regeln auf ein Paket zutrifft.

2.2.3 Einsatz von Firewalls

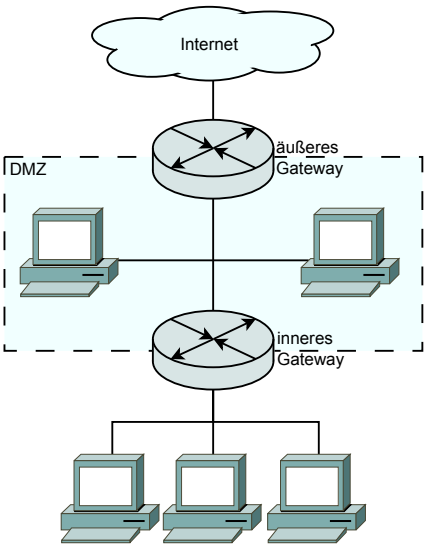
Für den Einsatz von Firewalls gibt es verschiedene Möglichkeiten. Als *Personal Firewall* ist sie direkt auf dem zu schützenden Rechner installiert. Sie reguliert dann direkt die Zugriffe zwischen dem Rechner und den verbundenen Netzen. Soll die Firewall ein Netzwerk nach außen Zugriffen vor außen schützen, wird sie auf dem *Gateway*[4] des Netzwerks installiert (siehe Abbildung 2.2a). Alle Zugriffe zwischen dem Netzwerk und der Außenwelt laufen über das Gateway. Die

[4]Ein Gateway ist hier eine Netzwerkkomponente, über die der gesamte eingehende und ausgehende Netzwerkverkehr laufen muss.

Firewall kann regulierend in jede Kommunikation eingreifen. Ein Sonderfall stellt die „entmilitarisierte Zone“ (DMZ) dar (siehe Abbildung 2.2b). Die DMZ ist ein äußerer Bereich des zu schützenden Netzwerks [TW10]. Hier wird eine geringere Einschränkung der Zugriffe vorgenommen. Die Firewall wird auf dem inneren Gateway zwischen der DMZ und dem inneren Bereich des Netzwerks installiert. Hier unternimmt sie die gewohnten Einschränkungen. Auf dem äußeren Gateway kann eine zweite Firewall installiert sein, die eine weniger ausgeprägte Form der Zugriffsbeschränkung vornimmt. Diese Architektur eignet sich besonders für Netzwerke, in denen einzelne Rechner (etwa Web-Server) von außen verfügbar sein sollen.



(a) Firewall auf dem Gateway.



(b) Skizze einer Architektur mit DMZ.

Abbildung 2.2 Verschiedene Netzwerkarchitekturen mit Firewalls nach [Bar01, TW10, Fuh00].

Als dedizierte Netzwerkkomponente übernimmt eine Hardware-Firewall lediglich die Aufgabe der Paketfilterung. Sie sorgt nicht für die Verbindung unterschiedlicher Netzwerke, sondern überwacht nur die durchlaufenden Pakete und greift regulierend ein. Eine Firewall kann jedoch auch auf einem Router neben oder als Teil der Router-Software installiert sein. Eine solche Lösung kann als *Firewall-Router* bezeichnet werden. Für den weiteren Verlauf dieser Arbeit soll unter Firewall bzw. Router die integrierte Lösung Firewall-Router (Router mit zusätzlich installierter Firewall-Software) verstanden werden.

2.3 Information Flow Ontology

In diesem Abschnitt wird die in [KS13] beschriebene Information Flow Ontology (InFO) vorgestellt. Es wird ein Überblick über die Ontologie und ihre Funktion gegeben (Abschnitt 2.3.1) und die zugrunde liegende Struktur beschrieben (Abschnitt 2.3.2). Zum Schluss wird die in dieser Arbeit verwendete Router-spezifische InFO-Erweiterung Router-based Flow Control Ontology (RFCO) beschrieben (Abschnitt 2.3.3). Für diese Arbeit wird Version 0.1 der InFO[5] verwendet, da sich Version 0.2 zum Zeitpunkt der Arbeit noch in der Entwicklung befindet. Es wird im folgenden nur eine vereinfachte Sicht auf die InFO geboten. Elemente, die in dieser Arbeit keine Verwendung finden, werden ausgelassen. Die hier beschriebenen Informationen sind [KS13] und [Kas12a] entnommen.

2.3.1 Grundlagen zur Information Flow Ontology

Die Information Flow Ontology (InFO) ist eine Ontologie zur Beschreibung von Maßnahmen zur Regulierung der Kommunikation im Internet. InFO ist in der Web Ontology Language (OWL) [W3C12] verfasst. Die Regulierung erfolgt durch *Regeln*, die jeweils einen bestimmten Informationsfluss erlauben oder verbieten. Ein Informationsfluss ist definiert durch Absender, Empfänger und verwendeten Kanal. Eine verbotener Informationsfluss kann entweder blockiert oder manipuliert werden. So kann eine Regel beispielsweise alle Zugriffe blockieren, die aus dem Netzwerk `10.0.0.0/8` auf die IP-Adresse `141.26.64.1` erfolgen. Die Regeln können mit Informationen verknüpft werden, die die vorgenommene Regulierung begründen. Wird ein Informationsfluss nicht explizit durch eine Regel erlaubt oder verboten, so kann eine Standardregel definiert werden, die dann angewendet wird. Die Standardregel erlaubt oder verbietet in diesem Fall den Informationsfluss.

Zwei oder mehr Regeln können im Konflikt stehen. Dies ist der Fall, wenn es einen Informationsfluss gibt, auf den mehrere Regeln zutreffen. Zum Beispiel kann eine Regel den Zugriff auf das Netzwerk `141.26.0.0/16` verbieten, während eine andere Regel den Zugriff auf die IP-Adresse `141.26.64.1` erlaubt. Um Konflikte aufzulösen, lassen sich mit InFO Priorisierungs- und Konfliktlösungsalgorithmen benennen. Priorisierungsalgorithmen vermeiden Konflikte, in-

[5] http://icp.it-risk.iwvi.uni-koblenz.de/ontologies/0.1/ - Abruf: 22.02.2014

dem sie bestimmen, in welcher Reihenfolge Regeln angewandt werden. Konfliktlösungsalgorithmen lösen Konflikte zwischen Regeln mit gleicher Priorität aktiv auf, indem betroffene Regeln beispielsweise nicht zur Anwendung gebracht werden. Damit die Algorithmen selbst nicht in Konflikt stehen, muss eine eindeutige Reihenfolge für die Abarbeitung der Algorithmen angegeben werden.

Regeln, die den gleichen Zweck verfolgen, werden in einem Regelsatz (*Policy*) zusammengefasst. Eine Policy ordnet ihren Regeln die Hintergrundinformationen für die Regulierung sowie die verantwortlichen Stellen für Regelerstellung und -durchsetzung zu. Sie bestimmt weiter, mit welchem technischen System die Regeln umzusetzen sind. Für Konflikte zwischen den zugeordneten Regeln bestimmt die Policy lokale Konfliktlösungsalgorithmen.

Alle Policys, die auf einem konkreten technischen System angewendet werden, sind einer *Meta Policy* zugeordnet. Die Meta Policy bestimmt im Wesentlichen, welche Regeln der verknüpften Policys angewendet werden und in welcher Reihe sie angewendet werden. Können Regeln auf einem technischen System nicht ausgeführt werden, benennt die Meta Policy Algorithmen, die diese Regeln behandeln. Betroffene Regeln und Policys können durch einen solchen Algorithmus etwa verworfen werden. Für die anwendbaren Policys und Regeln liefert die Meta Policy Priorisierungsalgorithmen, die zunächst die zugeordneten Policys ordnen und anschließend die implizit enthaltenen Regeln. Vorhandene Konflikte werden dann zunächst durch die lokalen Konfliktlösungsalgorithmen der Policy aufgelöst. Für verbleibende Konflikte benennt die Meta Policy globale Konfliktlösungsalgorithmen, die zuletzt angewendet werden. Die Meta Policy benennt außerdem die oben erwähnte Standardregel für Informationsflüsse, die nicht durch die Regeln erfasst sind.

2.3.2 InFO-Datenstruktur

Die grundlegende Struktur besteht aus den Entitäten `Rule`, `Policy` und `MetaPolicy` (siehe Abbildung 2.3). Im folgenden werden diese Entitäten in dem Umfang beschrieben, wie es für das Verständnis dieser Arbeit nötig ist. Eine ausführliche Beschreibung der Entitäten enthalten die zugehörigen *Patterns* unter [Kas12a]. Jeder `Policy` und `MetaPolicy` wird eine technische Einheit (`EnforcingSystem`) zugeordnet, mit der die Vorschrift durchzusetzen ist. Das kann ein Router, ein Nameserver etc. sein. Ebenso wird eine Stelle (`Responsible-`

`Operator`) zugeordnet, die für die Durchsetzung der Vorschrift verantwortlich ist. Dies können Organisationen oder konkrete Personen sein [KS13].

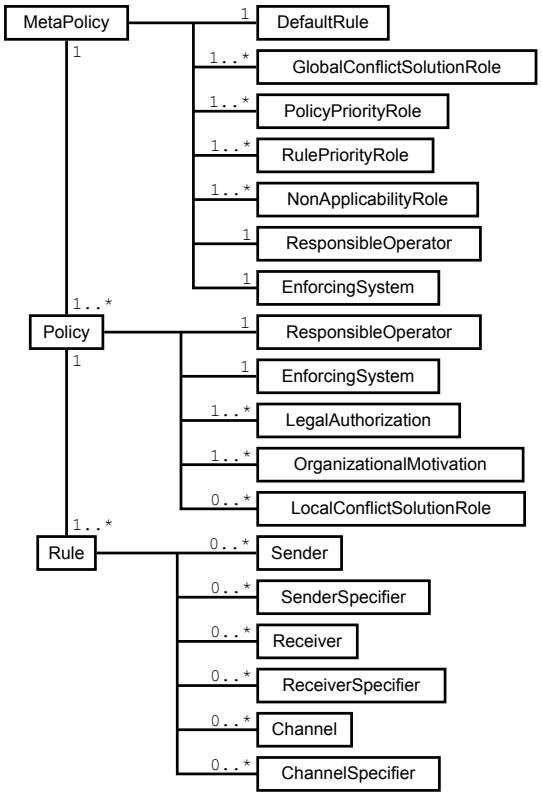


Abbildung 2.3 Vereinfachte Struktur der InFO [KS13, Kas12a].

Die `Rule` bildet die elementarste Einheit. Sie beschreibt die Regulierung eines bestimmten Informationsflusses. Der Informationsfluss wird durch die Quelle (`Sender`), das Ziel (`Receiver`) und den benutzten Kanal (`Channel`) beschrieben. `Sender` und `Receiver` benennen die Endknoten des regulierten Informationsflusses, etwa ein Netzwerk oder ein Web-Server. `SenderSpecifier` und `ReceiverSpecifier` ordnen diesen dann eindeutige Adressen bzw. Adressbereiche zu. `Channel` definiert ein bestimmtes Übertragungsprotokoll, während `ChannelSpecifier` dieses Protokoll noch genauer bestimmen kann, etwa durch eine Versionsnummer. Die Regulierung des Informationsflusses wird durch die entsprechende Erweiterung der `Rule` definiert. So lässt eine `AllowingRule` den Informationsfluss zu, während eine `DenyingRule` ihn untersagt und eine `RedirectingRule` ihn umleitet. Im letzten Fall wird die `Rule` um Elemente erweitert, die die Umleitung genau beschreiben.

Eine Policy kapselt ein oder mehrere Rules, die den selben Zweck verfolgen. Für den Fall, dass Rules der Policy im Konflikt stehen, können Algorithmen zur Konfliktlösung (LocalConflictSolutionRole), benannt werden. Einer Policy wird für die Regulierung zusätzlich der gesetzliche Hintergrund (LegalAuthorization), sowie das Motiv der verantwortlichen Stelle (OrganizationalMotivation) zugeordnet. Der gesetzliche Hintergrund kann ein bestimmter Gesetzesparagraph sein. Ein Beispiel für OrganizationalMotivation ist der „Verhaltenskodex“ (*Code of Conduct*) eines Unternehmens.

Eine MetaPolicy kapselt ein oder mehrere Policys. Sie bietet Informationen zur Anwendung der Policys und Rules. Wird für einen bestimmten Informationsfluss keine zutreffende Rule in den Policys gefunden, definiert DefaultRule eine Standardregel, die auf diesen Informationsfluss angewendet wird. Kann eine Regel vom EnforcingSystem technisch nicht umgesetzt werden, bestimmt NonApplicabilityRole wie mit der Regel zu verfahren ist. Diese kann beispielsweise verworfen werden. Mit PolicyPriorityRole und RulePriorityRole definiert MetaPolicy jeweils ein oder mehrere Algorithmen, die eine Priorisierung der Policys respektive Rules vornehmen. Die Priorisierung kann beispielsweise anhand des Erstellungsdatums der Policys bzw. Rules erfolgen. Anhand der durch die Priorisierung festgelegten Reihenfolge erfolgt später die Auswahl der anzuwendenden Policys bzw. Rules. Durch die Priorisierung sollen bereits Konflikte zwischen Rules vermieden werden, sodass lokale oder globale Konfliktlösungsalgorithmen nicht angewendet werden müssen. Kann ein Konflikt zwischen zwei oder mehreren Rules nicht durch die Priorisierungsalgorithmen und die lokalen Konfliklösungsalgorithmen aufgelöst werden, definiert die MetaPolicy als letzten Schritt globale Konfliktlösungsalgorithmen (GlobalConflictSolutionRole).

In verwandten Arbeiten ist die Umsetzung der InFO für Nameserver [Mos12], Suchmaschine [Rus13] und Proxy-Server [BGSS14] beschrieben. In dieser Arbeit wird die Umsetzung der InFO für einen Router vorgenommen. Die Router-spezifischen Entitäten der InFO-Erweiterung RFCO sind im nächsten Abschnitt beschrieben.

2.3.3 Router-based Flow Control Ontology

Die Router-based Flow Control Ontology (RFCO) erweitert die InFO um Entitäten, die speziell auf Routern Anwendung finden [Kas12b]. Der Arbeitsbereich eines Routers bezieht sich vor allem auf die Internetschicht und die Transportschicht (siehe Abschnitt 2.1). Die RFCO erlaubt hier die Unterscheidung anhand der *IP-Adresse*, der *Portnummer* oder des *Transportprotokolls* (siehe Abbildung 2.4). IP-Adresse und Portnummer lassen sich zudem zur *Socket-Adresse* kombinieren. Die Parameter können genutzt werden, um eine bestimmten Informationsfluss zu erlauben, zu verbieten oder umzuleiten. So kann mit einer `IPAddressDenyingRule` ein Informationsfluss anhand der IP-Adressen verboten werden. Dafür wird als Parameter mindestens eine IP-Adresse für `Sender` und/oder `Receiver` benötigt. Es ist jedoch auch möglich, unter Verwendung von Subnetzmasken ganze Netzwerke zu benennen.

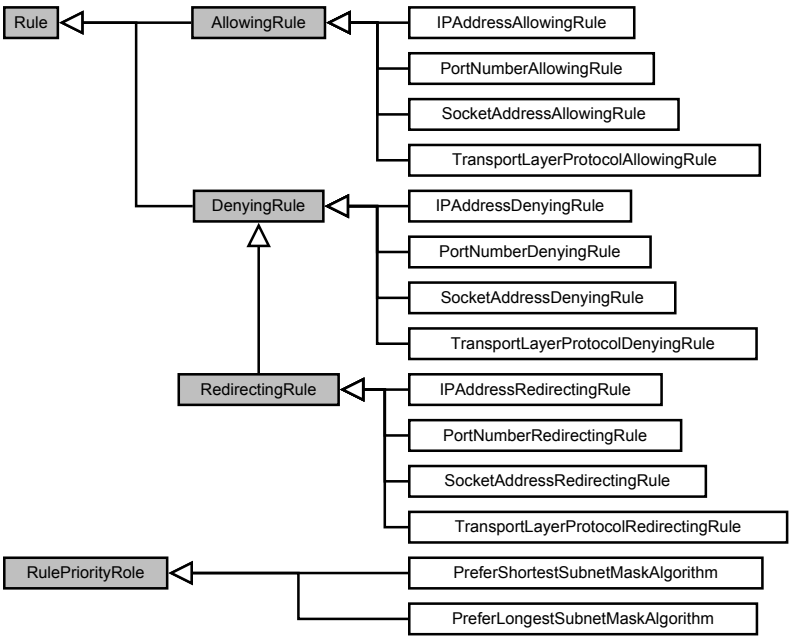


Abbildung 2.4 Erweiterung der InFO durch die RFCO.

Weiter benennt die RFCO mit `PreferShortestSubnetMaskAlgorithm` und `PreferLongestSubnetMaskAlgorithm` konkrete Algorithmen zur Priorisierung von Regeln. So bevorzugt `PreferLongestSubnetMaskAlgorithm` Regeln mit einer längeren Subnetzmaske. Verbietet beispielsweise eine `IPAddressDenyingRule` den Zugriff auf das Netzwerk `141.26.0.0/16` und er-

laubt eine `IPAddressAllowingRule` zugleich den Zugriff auf einen Server innerhalb dieses Netzwerks mit der IP-Adresse `141.26.64.1`, so stehen beide Regeln zunächst im Konflikt. Einerseits wird der Zugriff auf `141.26.64.1` durch die `IPAddressAllowingRule` erlaubt, andererseits wird er durch die `IPAddressDenyingRule` verboten. Unter Anwendung des `PreferLongestSubnetMaskAlgorithm` wird die `IPAddressAllowingRule` in diesem Fall bevorzugt, da `141.26.64.1` implizit die längere Subnetzmaske `255.255.255.255` (Als CIDR-Suffix: `32`) besitzt. `PreferShortestSubnetMaskAlgorithm` bevorzugt entsprechend kürzere Subnetzmasken.

Kapitel 3

Konzeption

Dieses Kapitel umfasst das Konzept der Arbeit. In Abschnitt 3.1 wird kurz die Aufgabenstellung beschrieben. Im darauf folgenden Abschnitt 3.2 wird eine Analyse der Anforderungen an das zu entwickelnde System vorgenommen. Danach wird zunächst in Abschnitt 3.3 ein Überblick über die Lösung gegeben durch die Vorstellung des Systemkonzepts. Die Aktivitäten und Datenflüsse des Systems werden in Abschnitt 3.4 beschrieben und die einzelnen Komponenten des Systems in Abschnitt 3.5 vorgestellt.

3.1 Aufgabenstellung

Ziel der Arbeit ist es, ein System zu erstellen, dass die Router-based Flow Control Ontology (RFCO) (siehe Abschnitt 2.3.3) umsetzt. Das System soll die in OWL-Dateien beschriebene Regulierung auswerten und anwenden. Der IP-Verkehr soll gemäß den Regeln zugelassen, blockiert oder umgeleitet werden. Dabei sollen die Absender stets über die Regulierung und den Grund der Regulierung informiert werden. Es folgt eine Aufstellung der Anforderungen an das System.

3.2 Anforderungsanalyse

In diesem Abschnitt wird eine Analyse der Anforderungen an das zu entwickelnde System vorgenommen. Die Formulierung der Anforderungen ist an [Bra97] angelehnt. „Muss“-Anforderungen sind dabei unbedingt umzusetzen. „Soll“-An-

forderungen sind ebenso umzusetzen, falls nicht schwerwiegende Gründe dagegen sprechen. Diese Gründe sind darzulegen. Die Umsetzung von „Kann“-Anforderungen ist optional. Am Ende des Abschnitts werden die Anforderungen in einer nummerierten Liste zusammengefasst. In der nun folgenden textuellen Beschreibung wird in Klammern jeweils auf diese Listenpunkte verwiesen.

Die Policys liegen im XML-basierten OWL-Format vor. Eine direkte Verarbeitung der Policys durch einen Router ist nativ nicht möglich. Darum muss das System in der Lage sein, die Policys einzulesen und gegebenenfalls in ein Format zu überführen, das eine Weiterverarbeitung gestattet (**A.1**). Damit ein Router die in den Policys beschriebenen Regulierungen umsetzen kann, müssen diese in ein Format übersetzt werden, das der Router interpretieren kann (**A.2**). In Bezug auf die Konzeption ergeben sich damit folgende Alternativen:

1. Es wird eine Erweiterung des Routers vorgenommen, damit dieser selbst Policys verarbeiten und umsetzen kann.

2. Es wird keine direkte Erweiterung des Routers vorgenommen. Das System verarbeitet die Policys vor und nutzt die vorhandenen Funktionen des Routers um die Policys umzusetzen.

3. Es wird ein eigener Software-Router entwickelt, der Policys unterstützt und diese verarbeiten und umsetzen kann.

Bei Alternative 1 wird die Router-Software derart erweitert, dass OWL-Dateien eingelesen und verarbeitet werden können. Dabei ist man durch die Architektur des Routers eingeschränkt. Die Lösung muss auf den Ziel-Router angepasst und speziell für diesen entwickelt werden. Eine Übertragung auf andere Routerarchitekturen ist nicht ohne weiteres möglich. Zudem begrenzen die eingeschränkten Ressourcen eines Routers die Verarbeitung von Ontologien. Aufgrund dieser Schwierigkeiten wird Alternative 1 nicht verfolgt. Die Entwicklung eines eigenen Routers, wie sie in Alternative 3 beschrieben wird, ist sehr umfangreich. Mit Blick auf die beschränkten Zeitvorgaben und die Schwerpunktsetzung dieser Diplomarbeit wird diese Alternative daher ebenfalls ausgeschlossen. Alternative 2 erfordert zwar das Erstellen zusätzlicher Komponenten. Dennoch stellt sie einen vertretbaren Kompromiss zwischen Aufwand und Flexibilität der Lösung dar. Umgesetzt wird daher Alternative 2: Es wird ein Software-Router entwickelt. Zur Regulierung wird eine bereits vorhandene Firewall benutzt. Das

zu entwickelnde System konfiguriert die Firewall, indem es die Policys verarbeitet und ein entsprechendes Skript zur Konfiguration der Firewall erzeugt.

Zwischen den Policys und zwischen den Regeln der InFO können Konflikte auftreten (siehe Abschnitt 2.3.1). Vor der Anwendung der Regeln müssen diese aufgelöst werden. Die Meta Policy und Policy benennen Algorithmen zur Priorisierung und Konfliktlösung. Diese Algorithmen sind in der vorgesehenen Reihenfolge (siehe Abschnitt 2.3.1) anzuwenden, um somit Konflikte zwischen Rules (**A.3**) und Policys (**A.4**) aufzulösen.

Die durch die Policys herbeigeführten Regulierungen sollen für den betroffenen Anwender nachvollziehbar sein. Wenn in eine Kommunikation eingegriffen wird, muss daher eine Rückmeldung an den Initiator der Kommunikation erfolgen (**A.5**). Der Initiator muss diese Rückmeldung der Kommunikation zuordnen können (**A.5.1**). Außerdem muss die Rückmeldung Auskunft darüber geben, wie in die Kommunikation eingegriffen wurde (**A.5.2**). Im Kontext der RFCO wird zwischen der Sperrung oder der Umleitung der Kommunikation unterschieden. Zudem muss der Hintergrund der Regulierung für den betroffenen Initiator einer Kommunikation nachvollziehbar sein (**A.5.3**). In der InFO wird eine Verbindung zwischen der Regulierungsmaßnahme und ihrer rechtlichen und/oder organisatorischen Grundlage hergestellt. Diese Verbindung muss auch in der Benachrichtigung hergestellt werden.

Da im Bereich eines Internet Service Providers (ISP), eines autonomen Systems (AS) oder eines sonstigen größeren Netzwerkverbunds mehrere Router verwaltet werden, ist ein zentraler Administrationsknoten einzurichten (**A.6.1**). Der Administrationsknoten erzeugt aus den Policys eine Router-Konfiguration und verteilt sie an die Router. Dies erleichtert den Administrationsaufwand, weil kein physischer Zugang zu den einzelnen Routern notwendig ist und Aufgaben automatisiert für mehrere Router vorgenommen werden können. Außerdem lässt sich eine einheitliche Sicht auf die technisch unterschiedlichen Routern innerhalb eines Netzwerkverbunds erzeugen. Der Administrationsknoten wird hier kurz als *Admin-Knoten* bezeichnet. Auf dem Router nimmt eine Software (der *Agent-Knoten*) die Router-Konfiguration entgegen und konfiguriert den Router entsprechend dieser (**A.6.2**). Admin-Knoten und Agent-Knoten sind in einer gemeinsamen Architektur zu integrieren (**A.6**).

Die Regulierung an sich (A.5) und ihr Hintergrund (A.5.3) sollen nachvollziehbar sein. Es soll aber vermieden werden, dass die Regulierungsmaßnahmen

umgegangen oder gestört werden (**A.7**). Bei der Übertragung von Regulierungsinformationen muss daher sichergestellt werden, dass diese nicht verändert werden (**A.7.2**). Es soll auch verhindert, dass gezielt erschlossen werden kann, welche Kommunikation von der Regulierung betroffenen ist. Daher müssen die Details zur technischen Umsetzung vertraulich behandelt werden (**A.7.1**). Es folgt eine zusammenfassende Auflistung der zuvor beschriebenen Anforderungen.

A.1 Das System muss Policys im OWL-Format einlesen können.

A.2 Das System muss eingelesene Policys auf ein natives Router-Format abbilden.

A.3 Das System muss Konflikte zwischen Regeln auflösen können.

- A.3.1 Das System muss Priorisierungsalgorithmen der Meta Policy anwenden können.
- A.3.2 Das System muss Konfliktlösungsalgorithmen der Policy anwenden können.

A.4 Das System muss Konflikte zwischen Policys auflösen können.

- A.4.1 Das System muss Priorisierungsalgorithmen der Meta Policy anwenden können.
- A.4.2 Das System muss Konfliktlösungsalgorithmen der Meta Policy anwenden können.

A.5 Von der Regulierung betroffene Parteien müssen über die Existenz der Regulierung eine Rückmeldung vom Router erhalten.

- A.5.1 Die Rückmeldung muss der regulierten Kommunikation zugeordnet werden können.
- A.5.2 Die Rückmeldung muss Auskunft über die Art der Regulierung geben.
- A.5.3 Die Rückmeldung muss Informationen zum Hintergrund der Regulierung geben.

A.6 Das System muss eine *Admin-Agent-Architektur* verwenden.

A.6.1 Die Verarbeitung der Policys muss auf einem zentralen Admin-Knoten erfolgen

A.6.2 Die Anwendung der Regulierung muss auf einem Agent-Knoten erfolgen.

A.7 Der Austausch von policy-relevanten Informationen zwischen Routern und einem Administrationsknoten muss abgesichert werden.

A.7.1 Die Kommunikation muss vertraulich sein.

A.7.2 Die Kommunikation muss integer sein

3.3 Grobkonzept

Im weiteren Verlauf wird der zentrale Administrationsknoten als *Admin-Knoten* und der einzelne Router als *Agent-Knoten* bezeichnet (siehe Anforderung A.6). In Abbildung 3.1 ist die Verteilung von Admin-Knoten und Agent-Knoten innerhalb eines beispielhaften Netzwerkverbunds veranschaulicht. In den folgenden Ausführungen sind in Klammern jeweils Referenzen zu den Anforderungen aus Abschnitt 3.2 angegeben. Der Admin-Knoten liest die Policys ein und verarbeitet diese (Anforderung A.1). Auf die eingelesen Policys werden Algorithmen zur Priorisierung und Konfliktlösung angewendet (Anforderungen A.4 und A.3). Aus den verarbeiteten Policys wird ein Firewallkonfigurationskript erzeugt, das durch den Agent-Knoten verarbeitet werden kann (Anforderung A.2). Anschließend sendet der Admin-Knoten das erzeugte Skript an den Agent-Knoten. Der Admin-Knoten führt hierzu eine Liste über alle von ihm verwalteten Agents mit Informationen über Verbindungsdetails und die jeweils verwendete Firewall.

Der Agent-Knoten ist für die Anwendung der Policys verantwortlich. Beim Agent-Knoten handelt es sich um einen Software-Router. Durch die Konfiguration einer lokalen Firewall nimmt er die Regulierung des von ihm verwalteten IP-Verkehrs vor. Ist eine Kommunikation von einer Regulierungsmaßnahme betroffen, benachrichtigt der Agent-Knoten den Initiator der Kommunikation (Anforderung A.5). Den Initiator ermittelt er dafür aus der Absenderadresse der regulierten IP-Pakete. Die gesendete Benachrichtigung enthält eine kurze Information über die vorgenommene Regulierung und eine URL, unter der weitere Informationen über den Hintergrund abgerufen werden können (Anforderung A.5.3).

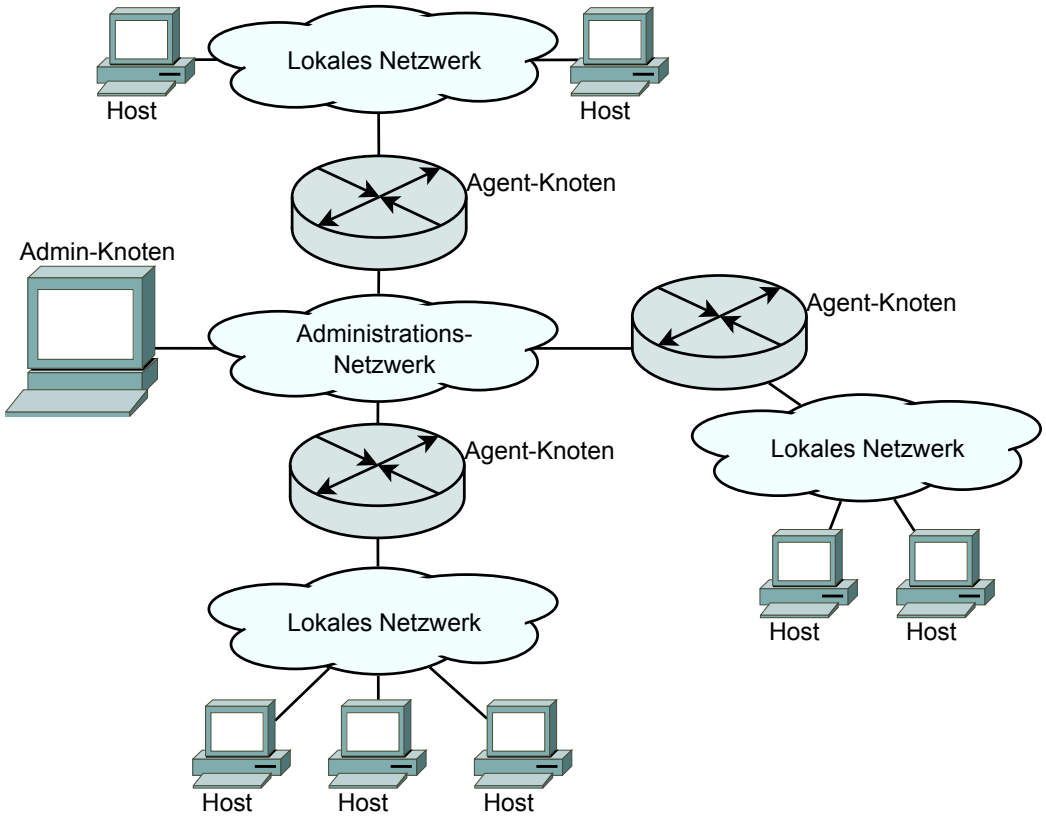


Abbildung 3.1 Skizze eines beispielhaften Netzwerkverbunds mit einem Admin-Knoten und drei Agent-Knoten. Die Agent-Knoten verwalten jeweils ein lokales Netzwerk mit mehreren Hosts. Die Agent-Knoten sind über ein Administrations-Netzwerk mit dem Admin-Knoten und untereinander verbunden. Die Agent-Knoten übernehmen die Regulierung für ihr lokales Netzwerk. Der Admin-Knoten verwaltet die Regulierungsvorschriften für die Agent-Knoten.

3.4 Aktivitäten und Datenflüsse

Während im vorherigen Abschnitt eine eher statische Beschreibung des zu entwickelnden Systems vorgenommen wurde, soll hier auf die Aktivitäten und den Datenfluss näher eingegangen werden. Zur besseren Lesart wird der Agent-Knoten im Singular verwendet. In der Praxis können jedoch mehrere Agent-Knoten von einem Admin-Knoten verwaltet werden.

Ausgangspunkt bildet das Einlesen und Verarbeiten der Policys beim Admin-Knoten. Die von ihm erzeugte Firewallkonfiguration wird dann an den Agent-Knoten übertragen. Der Agent-Knoten pflegt die Firewallkonfiguration ein und wendet sie an. Ein Aktivitätsdiagramm für die Software des Admin-Knoten zeigt

Abbildung 3.2. In Abbildung 3.3 ist ein Aktivitätsdiagramm für die Software des Agent-Knoten vorhanden. Im Text wird jeweils mit Nummern Bezug auf die Nummerierung in den Aktivitätsdiagrammen genommen. Die Nummern ① bis ⑤ referenzieren auf Aktivitäten des Admin-Knoten. Die übrigen Nummern referenzieren auf Aktivitäten des Agent-Knoten.

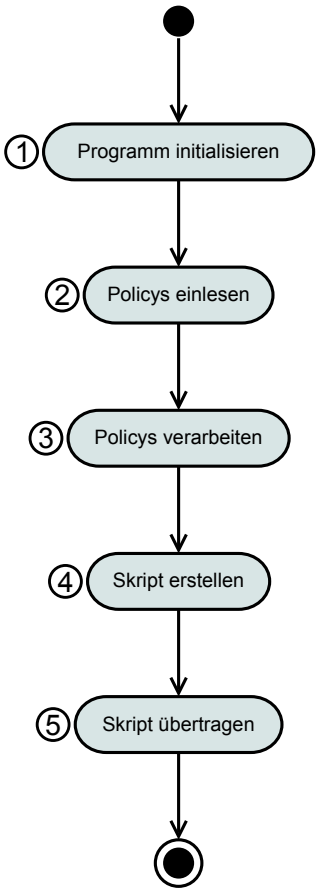


Abbildung 3.2 Aktivitätsdiagramm der Software des Admin-Knoten. Die eingelesenen Policys werden verarbeitet. Anschließend wird ein Konfigurationsskript für die Firewall erzeugt und an den Agent-Knoten übertragen.

3.4.1 Initialisierung der Software

Bei Programmstart werden zunächst etwaige Parameter übergeben und das jeweilige Programm initialisiert. Sowohl auf dem Admin-Knoten ① als auch auf dem Agent-Knoten ⑥ liest das jeweilige Programm die Konfiguration aus ei-

ner Datei ein. Auf dem Admin-Knoten werden dem Programm zusätzlich die zu übersetzenden Policys übergeben.

3.4.2 Einlesen der Policys

Die OWL-Dateien mit den Policys werden von der Software des Admin-Knoten eingelesen ②.

3.4.3 Verarbeiten der Policys

Zur Weiterverarbeitung ③ werden zunächst in einem Parsing-Prozess aus den beschriebenen Entitäten Objekte erzeugt. Extrahiert werden neben den grundlegenden Entitäten der InFO zusätzlich alle Entitäten, die in der RFCO beschrieben sind. Für die Erstellung eines Firewall-Skripts wird nicht gesamte Ausdrucksmächtigkeit einer Ontologie benötigt. Auch sind nicht alle in den Policys enthaltenen Informationen relevant für die technische Regulierung durch die Firewall. Daher werden in einem weiteren Schritt Wrapper-Objekte für die Entitäten der Policys erzeugt. Ein Wrapper-Objekt kapselt das eigentliche Objekt und bietet durch eigene Attribute und Methoden einen indirekten Zugriff auf das Objekt. Es wird somit eine Schnittstelle zum Objekt erzeugt, die den Erfordernissen zur Bearbeitung des Objekts entspricht. Für diese Arbeit wird dadurch eine vereinfachte Darstellung der Policy-Entitäten ermöglicht. Gleichzeitig wird der Anpassungsbedarf gering gehalten, falls in Zukunft Änderungen an InFO vorgenommen werden.

Für die Anwendung der Policy-Regeln innerhalb der Firewall ist es nötig, die Regeln in eine eindeutige Reihenfolge zu bringen, nach der sie später abgearbeitet werden können. Die Meta Policy benennt hierfür Algorithmen zur Priorisierung. Diese Priorisierungsreglen werden interpretiert und die Policys und ihre Regeln in einer Liste angeordnet. Für den Fall, dass Policys oder Regeln im Konflikt stehen (siehe Abschnitt 2.3.1), benennt die Meta Policy Algorithmen zur Konfliktlösung. Diese werden ebenfalls interpretiert, so dass am Ende eine eindeutige Reihenfolge der Regeln steht.

3.4.4 Erzeugen des Firewallskripts

Nachdem die Regeln in eine eindeutige Reihenfolge gebracht wurden, kann ein Skript zur Konfiguration einer Firewall erzeugt werden ④. Die benötigten Funktionen werden durch eine Schnittstelle bereit gestellt, die auf die jeweilige Ziel-Firewall angepasst ist. Diese Schnittstelle bietet Methoden an, um ein Regel-Objekt in eine Firewall-Regel zu transformieren. Zudem bietet sie Methoden zur Erzeugung von Kommentaren und allgemeiner Befehle für die Ziel-Firewall. Alle erzeugten Regeln und Befehle werden zu einem Skript zusammengefügt. Für das spätere Senden von Benachrichtigungen wird zusätzlich eine Informationstextdatei erzeugt. Jeder Policy-Regel/Firewallregel ist ein individueller Informationstext zugeordnet. Die Zuordnung erfolgt über einen eindeutigen Schlüssel, der aus dem URI der Policy-Regel erzeugt wird.

3.4.5 Übertragen des Firewallskripts

Das fertige Skript und die Informationstextdatei werden vom Admin-Knoten zum Agent-Knoten gesendet ⑤. Der Agent-Knoten hält dazu einen Server-Dienst bereit. Der Admin-Knoten führt eine Liste über alle verwalteten Agents und deren Verbindungsinformationen. Die Kommunikation zwischen dem Admin- und dem Agent-Knoten folgt einem festgelegten Übertragungsprotokoll (siehe Abschnitt 3.5.3). Schlägt die Übertragung oder Anwendung des Skripts fehl, wird der Administrator informiert.

Zur Absicherung der Übertragung erfolgt zunächst eine gegenseitige Authentifizierung von Admin- und Agent-Knoten. Die weitere Kommunikation erfolgt verschlüsselt. Dazu verwenden beide Komponenten ein Schlüsselpaar aus privatem und öffentlichen Schlüssel. Die öffentlichen Schlüssel müssen bei der Einrichtung der Software auf Admin und Agent-Knoten verteilt werden. Der Admin-Knoten besitzt die öffentlichen Schlüssel aller verwalteten Agents. Der Agent-Knoten besitzt die öffentlichen Schlüssel aller berechtigten Admins.

3.4.6 Empfangen des Firewallskripts

Auf dem dem Agent-Knoten läuft dauerhaft ein Server-Dienst, der den Empfang von neuen Skripts ermöglicht ⑦. Wurde die Verbindung mit einem Admin-Knoten aufgenommen, sperrt sich der Server-Dient, bis diese Verbindung been-

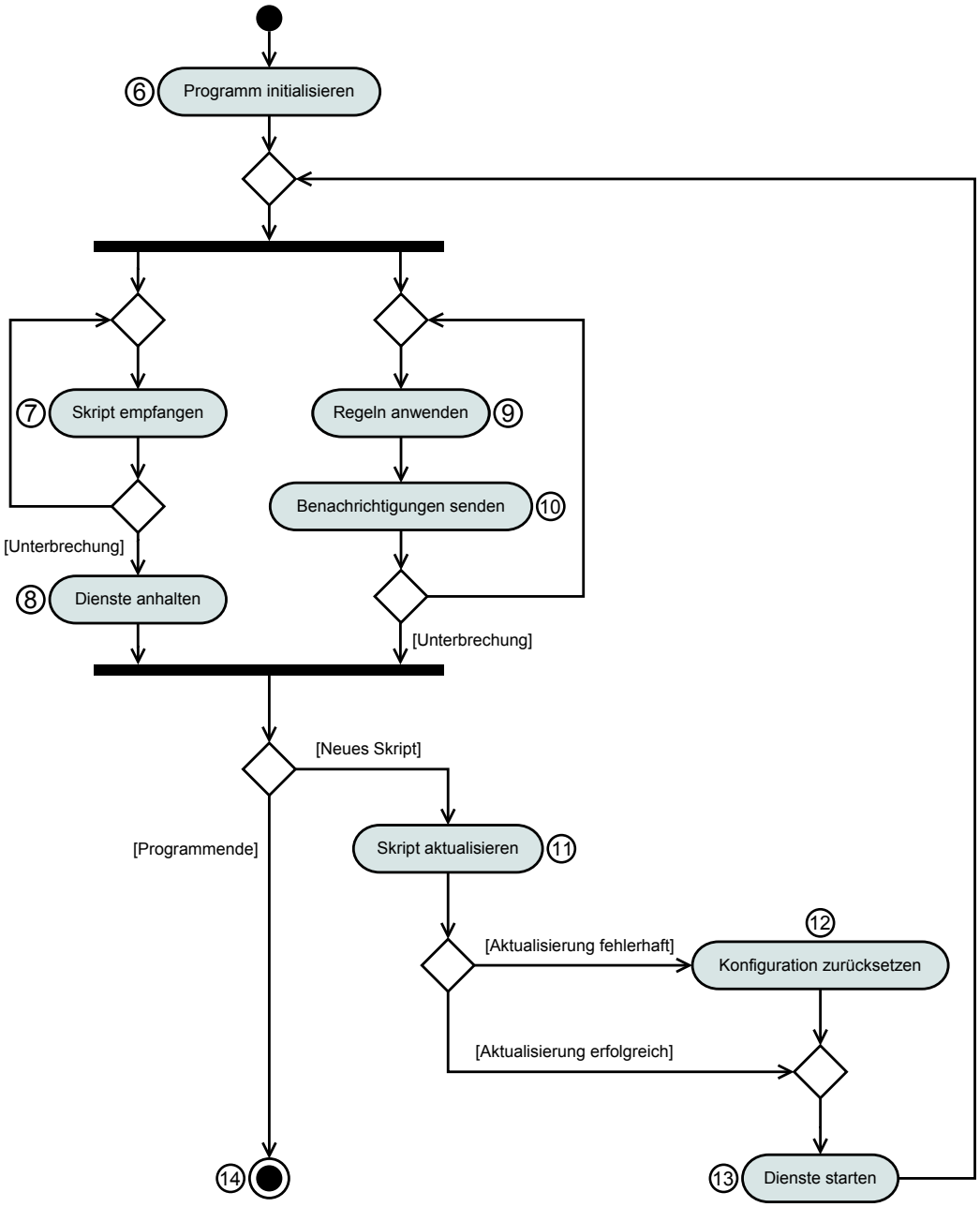


Abbildung 3.3 Aktivitätsdiagramm der Software des Agent-Knoten. Nach der Initialisierung startet die Software einen Server-Dienst zum Empfang neuer Skripte. Parallel dazu werden die Firewallregeln auf die empfangenen Pakete anwendet. Wurde Kommunikation reguliert, wird eine Benachrichtigung gesendet. Zum Einspielen eines neuen Skripts werden alle laufenden Dienste angehalten. Bei Update-Fehlern wird der vorherige Zustand wieder hergestellt. Der Agent muss durch den Administrator explizit beendet werden.

det wurde. Es handelt sich also um einen *iterativen* Server-Dienst. Der Server-Dienst ist maßgeblich für die Einhaltung des Übertragungsprotokolls (siehe Abschnitt 3.5.3) verantwortlich. Widerspricht die Nachricht eines Clients der Protokollsyntax oder -semantik, beendet der Server-Dienst die aktuelle Verbindung. Zur Ausführung wird das empfangene Skript lokal auf dem Agent-Knoten gespeichert. Die lokale Informationstextdatei wird durch die empfangene, aktuellere Datei ersetzt. Tritt ein Fehler während der Aktualisierung auf, wird der Admin-Knoten informiert. Parallel zum Server-Dienst erfolgt auf dem Agent-Knoten die Anwendung der Firewallregeln (siehe Abschnitt 3.4.8) und das Senden der Benachrichtigungen (siehe Abschnitt 3.4.9).

3.4.7 Aktualisierung des Firewallskripts

Bevor auf dem Agent-Knoten ein neues Skript eingespielt wird, werden zunächst alle laufenden, kritischen Dienste beendet ⑧. Die vorhandene Konfiguration wird gespeichert, sodass sie im Fehlerfall wieder eingespielt werden kann. Die Aktualisierung der Konfiguration erfolgt durch Ausführen des neuen Skripts ⑪. Tritt ein Fehler auf, wird die Aktualisierung verworfen und das Sicherungsskript aufgerufen ⑫. In jedem Fall werden im Anschluss die angehaltenen Dienste wieder neu gestartet ⑬.

3.4.8 Anwenden der Firewallregeln

Auf alle empfangenen Paketen werden die Regeln der Firewall-Konfiguration angewendet. ⑨. Treffen die Header-Informationen eines IP-Pakets auf eine InFO-Regel zu, wird das Paket entsprechend der Regel normal weitergeleitet, blockiert oder umgeleitet. Trifft keine Regel auf ein bestimmtes Paket zu, wird das Paket entsprechend *Default Rule* der Meta Policy behandelt (siehe Abschnitt 2.3.1). Falls in eine Kommunikation eingegriffen wurde durch Blockierung oder Umleitung von Paketen, werden Informationen über das Paket gespeichert. Diese Paketinformationen dienen der Benachrichtigung der Absender (siehe nächster Abschnitt).

3.4.9 Senden der Benachrichtigungen

Nach dem Eingriff in eine Kommunikation wird eine Benachrichtigung an den Absender des zugehörigen IP-Pakets gesendet (10). Die Nachricht enthält eine Referenz auf das regulierte Paket. Zusätzlich wird ein Informationstext mitgeliefert, der auf die Existenz und Art der Regulierung hinweist. Zum Inhalt des Textes gehört eine URL, unter der Informationen über den Hintergrund der Regulierung bezogen werden können. Der Informationstext wird der Informationstextdatei entnommen. Die Datei enthält für jeden Text einen eindeutigen Schlüssel, der die Zuordnung zur angewendeten Firewallregel erlaubt (siehe Abschnitt 3.4.4).

3.4.10 Beenden der Software

Die Software auf dem Agent-Knoten wird entweder durch einen Befehl des Admin-Knotens oder lokal durch den Administrator beendet. In diesem Fall werden zunächst die laufenden, kritischen Dienste beendet (8) und anschließend das Programm selbst (14). Die Software auf dem Admin-Knoten dagegen beendet ihre Ausführung automatisch nachdem die Verbindung mit dem Agent-Knoten beendet wurde. Die Ausführung der Software endet mit einer Rückmeldung an den Administrator, ob die Übertragung und Ausführung des Skripts erfolgreich war.

3.5 Architektur

In diesem Abschnitt wird die Architektur des zu entwickelnden Systems erläutert. Das System besteht aus drei Komponenten, die sich auf den Admin-Knoten und den Agent-Knoten verteilen (siehe Abbildung 3.4). Die Komponente *PolicyAdmin* befindet sich auf dem Admin-Knoten. Der PolicyAdmin übersetzt die Policys in ein Skript und überträgt es an die Komponente PolicyAgent. Die Komponente *PolicyAgent* stellt auf dem Agent-Knoten einen Dienst bereit, um das Skript zu empfangen. Mit dem Skript konfiguriert der PolicyAgent die Firewall, die ebenfalls auf dem Agent-Knoten installiert ist. Die *Firewall* wendet die in den Policys respektive im Skript beschriebenen Regeln zur Regulierung an. Die Absender von regulierten Paketen werden vom PolicyAgent benachrichtigt. Informationen zu diesen Paketen werden in einer Datenbank zwischengespeichert[1].

[1]Die Komponente Datenbank wird in Abbildung 3.4 nicht dargestellt.

Im folgenden werden der PolicyAdmin (Abschnitt 3.5.1) und der PolicyAgent (Abschnitt 3.5.2) anhand ihrer Funktionen vorgestellt. Im letzten Abschnitt 3.5.3 wird gesondert auf das verwendete Client/Server-Modell für die Skript-Aktualisierung eingegangen. Auf die implementierungsabhängige Firewall wird hier nicht gesondert eingegangen. Grundlagen zur Firewall bietet Abschnitt 2.2. Die für die Implementierung verwendete Firewall ist in Abschnitt 4.2 beschrieben. Im Anhang B.1 wird auf die hier vorgestellten Klassen detaillierter eingegangen.

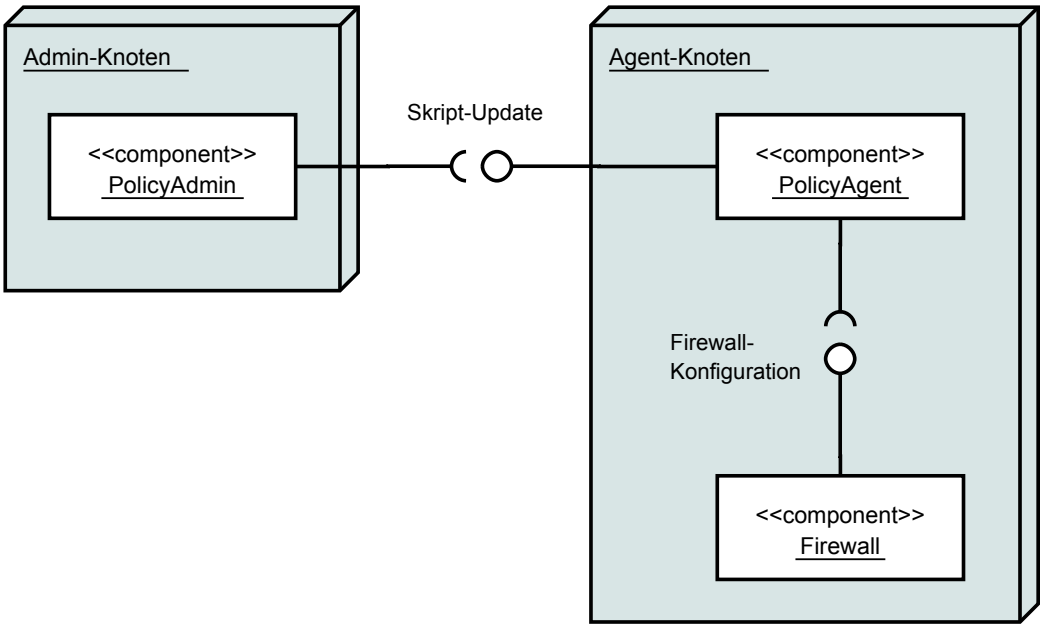


Abbildung 3.4 Komponentendiagramm der Architektur. Auf dem Admin-Knoten ist der PolicyAdmin für die Skripterzeugung und Übertragung zum Agent-Knoten zuständig. Auf dem Agent-Knoten stellt der PolicyAgent einen Dienst zum Empfang des Skripts bereit. Mit dem Skript konfiguriert der PolicyAgent die auf dem Agent-Knoten vorhandene Firewall. Nicht dargestellt ist die Datenbank, über die Informationen über regulierte Pakete von der Firewall zum PolicyAgent geleitet werden.

3.5.1 Die Komponente PolicyAdmin

Die Komponente PolicyAdmin baut sich aus verschiedenen Klassen auf, die sich auf die Pakete `policydistribution`, `policyadmin` und `firewalltranslator` verteilen. Das übergeordnete Paket `policydistribution` enthält Klassen, die gemeinsam von PolicyAdmin und PolicyAgent genutzt werden. Eine Übersicht der Pakete und ihrer Klassen bietet Abbildung 3.5. Ausführlich dar-

gestellt werden die Klassen im Anhang in den Abschnitten B.1.1 ff. Hier wird eine kurze Charakterisierung der Klassen vorgenommen.

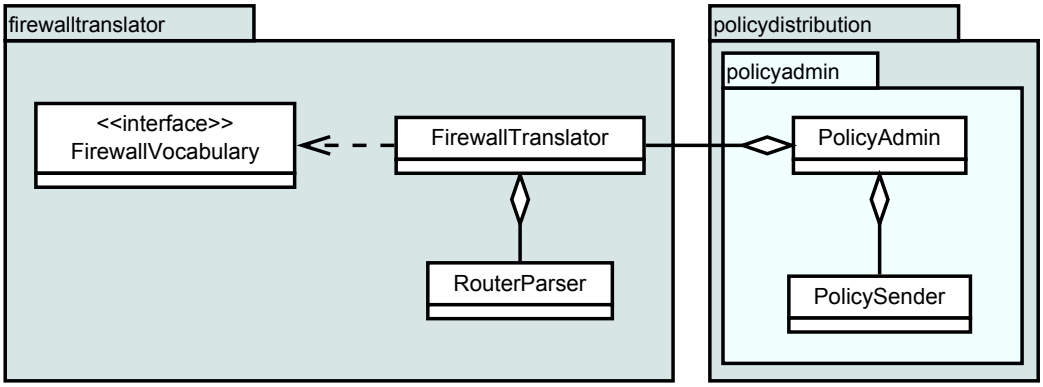


Abbildung 3.5 Vereinfachtes Paketdiagramm für den PolicyAdmin. Das übergeordnete Paket `policydistribution` enthält gemeinsam genutzte Klassen von PolicyAdmin und PolicyAgent. Das Paket `firewalltranslator` enthält Klassen zur Policy-Verarbeitung und Skripterzeugung. Das Paket `policyadmin` besteht aus Klassen zur Skriptübertragung sowie für allgemeine Funktionen.

Die Klasse `PolicyAdmin` bildet die zentrale Klasse der gleichnamigen Komponente PolicyAdmin. Ihr werden bei Programmstart die zu verarbeitenden Policys und eine Konfigurationsdatei übergeben. Die Konfigurationsdatei enthält Parameter zu den verwalteten PolicyAgents und allgemeine Parameter zum Betrieb der Komponente PolicyAdmin. Die Klasse `PolicyAdmin` übergibt die Policys zur Verarbeitung an die Klasse `FirewallTranslator`. Der `FirewallTranslator` übersetzt die Policys in ein Firewallskript. Zur Transformation der Policys in Objekte nutzt der `FirewallTranslator` die Klasse `RouterParser`. Zur Übersetzung der Objekte in ein Firewall-Skript wird die Schnittstelle `FirewallVocabulary` genutzt. Diese Schnittstelle bietet Methoden zur Erzeugung von Firewallregeln. Das vom `FirewallTranslator` erzeugte Skript übergibt der `PolicyAdmin` an die Klasse `PolicySender` zur Übertragung. Der `PolicySender` überträgt das Firewall-Skript zum PolicyAgent. Auf der Seite des PolicyAgent wird der Empfang des Skripts durch die Klasse `PolicyReceiver` geregelt (siehe Abschnitt 3.5.2).

3.5.2 Die Komponente PolicyAgent

Die Komponente PolicyAgent konstituiert sich aus den Klassen des Pakets `policyagent` und den Klassen aus dem gemeinsam genutzten Paket `policydistribution` (vergleiche Abschnitt 3.5.1). Eine Übersicht über die Pakete und Klassen bietet Abbildung 3.6. Im folgenden werden die verschiedenen Klassen charakterisiert. Eine ausführliche Darstellung ausgewählter Klassen bieten die Abschnitte B.1.5 ff. im Anhang.

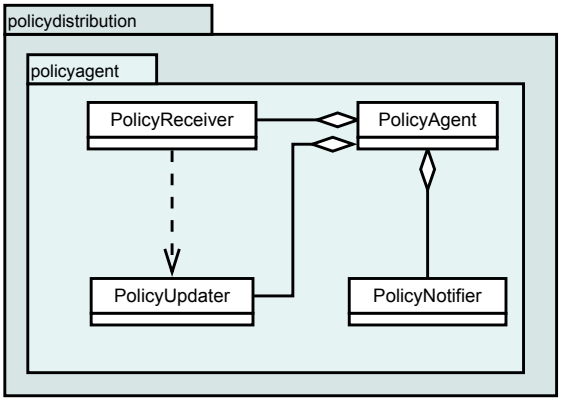


Abbildung 3.6 Vereinfachtes Paketdiagramm für den PolicyAgent. Das übergeordnete Paket `policydistribution` enthält gemeinsam genutzte Klassen von PolicyAdmin und PolicyAgent. Die Klasse `PolicyReceiver` ist für den Empfang der Skripte zuständig. `PolicyUpdater` aktualisiert die Firewallkonfiguration. Das Senden der Benachrichtigungen organisiert die Klasse `PolicyNotifier`.

Die zentrale Klasse für den PolicyAgent ist die gleichnamige Klasse `PolicyAgent`. Ihre Aufgabe ist die Initialisierung bei Programmstart. Die Klasse `PolicyAgent` liest dazu Parameter aus einer Konfigurationsdatei. Sie startet und unterbricht Dienste, die durch die Klassen `PolicyReceiver` und `PolicyNotifier` bereitgestellt werden. Die Klasse `PolicyNotifier` sendet Benachrichtigungen an die Absender von Paketen, die durch die Firewall reguliert wurden. Zur Zwischenpeicherung der regulierten Pakete wird eine Datenbank verwendet[2]. Die Firewall schreibt in diese Datenbank, während `PolicyNotifier` lesend auf diese zugreift. Die Klasse `PolicyReceiver` dient dem Empfang neuer Skripte. Das Senden der Skripte erfolgt durch die Klasse `PolicySender` des PolicyAdmin (siehe Abschnitt 3.5.1). Ein neues Skript leitet der `PolicyReceiver` an die Klasse `PolicyUpdater`. Der `PolicyUpdater` aktualisiert die Firewall-

[2]Die Datenbank wird im Komponentendiagramm in Abbildung 3.4 ausgelassen.

Konfiguration durch Ausführung des Skripts. Zur Durchführung der Aktualisierung unterbricht der `PolicyAgent` die Dienste des `PolicyNotifiers` und der Firewall. Scheitert die Aktualisierung, spielt der `PolicyAgent` die vorherige Konfiguration wieder ein. Außerdem informiert der `PolicySender` den PolicyAdmin, dass das von ihm gesendete Skript nicht eingespielt werden konnte.

3.5.3 Übertragungsprotokoll

Die Konfiguration für die Firewall wird als Skript durch den PolicyAdmin erzeugt und an den PolicyAgent übertragen. Zur Übertragung wird das Client/Server-Modell verwendet [TW10]. Dabei stellt eine Partei einen Server-Dienst bereit. Andere Parteien können sich dann mit dem Server verbinden und nehmen die Rolle des Clients ein. Nach Aufbau der Verbindung zwischen Client und Server senden sich beide Parteien abwechselnd Nachrichten. Die Verbindung kann durch eine Partei beendet werden oder nach Ablauf einer bestimmten Zeit ohne Nachrichten durch eine Partei erfolgen. Ein gemeinsames Protokoll regelt Syntax und Semantik der Kommunikation, also die erlaubten Nachrichten und die erlaubte Abfolge der Nachrichten. Für die Rollenverteilung von Server und Client gibt es zwei Alternativen:

PolicyAdmin als Server: Der PolicyAdmin stellt einen Server-Dienst bereit. Der PolicyAgent verbindet sich in regelmäßigen Abständen mit dem PolicyAdmin und prüft ob ein aktuelleres Skript vorliegt. Diese Option hat den Vorteil, dass der Server an zentraler Stelle liegt. Nachteilig ist, dass sich der PolicyAgent häufig mit dem Server verbinden muss, um nach einem aktuelleren Skript zu fragen. Ein neues Skript kann nicht unmittelbar an den PolicyAgent weitergegeben werden. Der PolicyAdmin muss erkennen, wenn sich ein PolicyAgent über längere Zeit nicht meldet. Zudem entsteht unnötiger Netzwerkverkehr. Diese Technik wird als *Pull* bezeichnet [MF99].

PolicyAgent als Server: Der PolicyAgent stellt einen Server-Dienst bereit, mit dem sich der PolicyAdmin verbindet, falls ein neues Skript vorliegt. Bei dieser Option wird nur im Bedarfsfall eine Verbindung aufgebaut. Ein neues Skript kann unmittelbar an den PolicyAgent gegeben werden. Ist ein PolicyAgent nicht ansprechbar, fällt dies sofort auf. Nachteilig ist hier, dass

jeder PolicyAgent einen Server-Dienst stellen muss. Diese Technik wird als *Push* bezeichnet [MF99].

Für diese Arbeit wurde sich für die zweite Alternative mit dem PolicyAgent als Server entschieden. Der Server-Dienst des PolicyAgent wird als *Receiver* bezeichnet, der Client-Teil beim PolicyAdmin als *Sender*. Ein Zustandsautomat für den Sender ist in Abbildung 3.7 dargestellt. Abbildung 3.8 zeigt den Zustandsautomaten des Receivers. Im Folgenden wird das Protokoll näher erläutert. Der Ablauf des Übertragungsprotokoll besteht aus dem *Verbindungsaufbau*, der anschließenden Durchführung von *Aktionen* und dem *Verbindungsabbau*. Mögliche Aktionen, die der Sender anstoßen kann, sind die *Skriptübertragung*, das *Beenden der Verbindung* oder das *Beenden des PolicyAgents*. Es folgt eine ausführliche Darstellung der Aktionen und der verwendeten Zustände und Nachrichten. Die Implementierung des Übertragungsprotokolls ist detailliert im Anhang in den Abschnitten B.1.4 und B.1.6 dokumentiert.

Wenn noch keine aktive Verbindung besteht, befindet sich der Receiver im Zustand `WAITING`. Der Sender startet den Verbindungsaufbau, indem er ein `HELLO` sendet. Danach geht er in den Zustand `HELLO_SENT` über. Der Receiver quittiert den Empfang der `HELLO`-Nachricht ebenfalls mit einem `HELLO` und geht seinerseits in den Zustand `HELLO_SENT` über. Die Verbindung ist damit aufgebaut. Der Sender kann nun ein neues Skript ankündigen, die Verbindung beenden oder den PolicyAgent beenden. Will der Sender die *Verbindung beenden*, sendet er ein `CLOSE`. Anschließend und geht er in den Zustand `CLOSE_SENT` und wartet auf die Antwort des Receivers. Der Receiver antwortet mit `BYE` und geht wieder in den Zustand `WAITING`. Nach Empfang von `BYE` schließt der Sender die Verbindung. Falls der Sender den *PolicyAgent beenden* soll, sendet er nach Verbindungsaufbau ein `ABORT_SERVICE` und geht in den Zustand `ABORT_SENT`. Der Receiver antwortet in diesem Fall mit `ABORTING_SERVICE`. Anschließend wird die Ausführung des PolicyAgents beendet. Nach Empfang von `ABORTING_SERVICE` beendet auch der Sender die Verbindung. Soll der Sender ein neues *Skript übertragen*, kündigt er dies mit `NEW_SCRIPT` an, nachdem die Verbindung aufgebaut ist. Dann geht der Sender in den Zustand `SCRIPT_ANNOUNCED`. Der Receiver antwortet mit `AWAITING_SCRIPT` und geht in den Zustand `RECEIVING_SCRIPT`. Er ist nun bereit, das neue Skript zu empfangen. Hat der Sender ein `AWATING_SCRIPT` empfangen, geht er in den Zustand `SENDING_SCRIPT`. Nun überträgt er das Skript zeilenweise an den Receiver. Der Receiver speichert

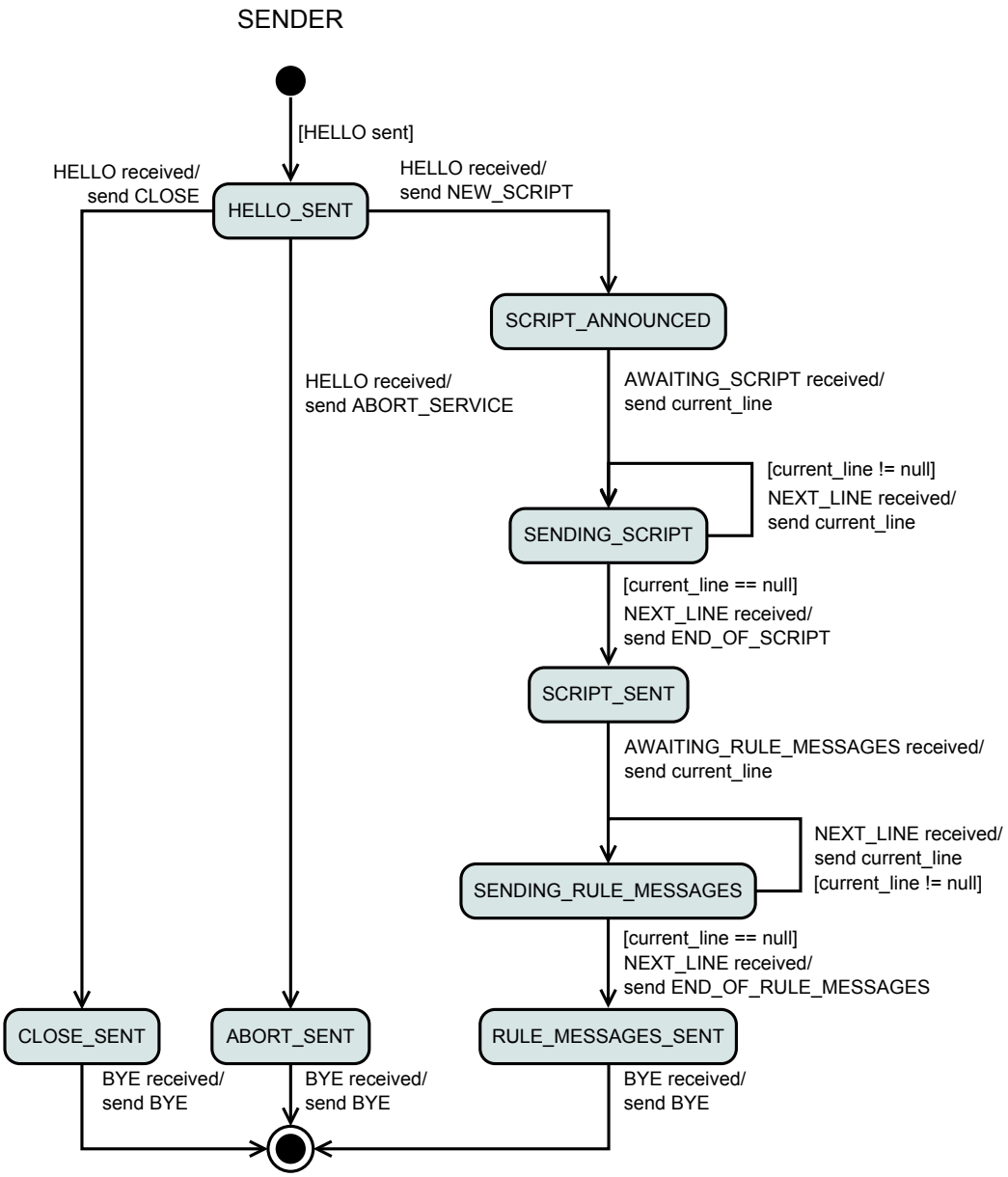


Abbildung 3.7 Zustandsautomat des Senders. Der Sender eröffnet die Verbindung zum Receiver mit `HELLO`. Mit `CLOSE` kann er die Verbindung beenden. `ABORT` beendet die Verbindung und zusätzlich den PolicyAgent. Mit `NEW_SCRIPT` wird die Übertragung eines neuen Skripts eingeleitet. In den Zuständen `SENDING_SCRIPT` und `SENDING_RULE_MESSAGES` erfolgt die Übertragung des Skripts bzw. der Informationstexte für die Firewallregeln. Das Ende der Übertragung wird mit `END_OF_RULE_MESSAGES` bekanntgegeben.

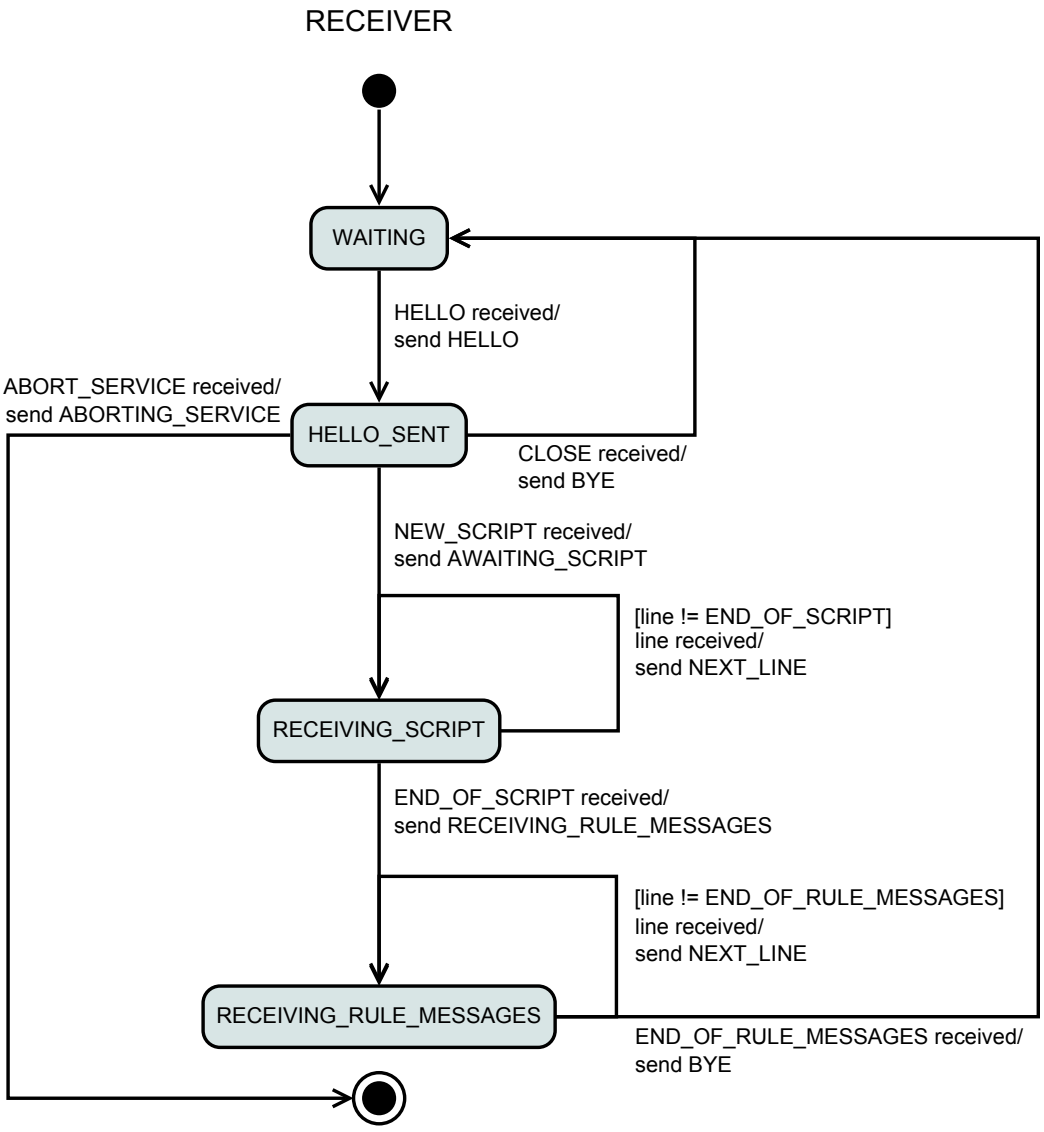


Abbildung 3.8 Zustandsautomat des Receivers. Der Receiver beginnt im Zustand `WAITING` und wartet auf die Verbindungsanfrage eines Senders. Nachdem Verbindungsaufbau ist der Receiver im Zustand `HELLO_SENT`. Im Zustand `RECEIVING_SCRIPT` wird ein neues Skript empfangen. Im Zustand `RECEIVING_RULE_MESSAGES` erfolgt der Empfang der Informationstexte zu den Firewallregeln. Nach Verbindungsende geht der Receiver stets in den Zustand `WAITING` zurück, um neue Verbindungen zuzulassen. Ein empfangenes `ABORT_SERVICE` führt zum Beenden des Receivers und des PolicyAgents.

die empfangenen Zeilen und quittiert sie jedes Mal mit einem NEXT_LINE. Beide Parteien verbleiben in dieser Zeit in den Zuständen SENDING_SCRIPT bzw. RECEIVING_SCRIPT. Ist das Ende des Skripts erreicht, sendet der Sender ein END_OF_SCRIPT und geht in den Zustand SCRIPT_SENT. Der Receiver bestätigt dies mit AWAITING_RULE_MESSAGES. Er signalisiert damit gleichzeitig, dass er nun bereit ist, die Datei mit den Informationstexten für die Firewallregeln zu empfangen. Anschließend geht der Receiver in den Zustand RECEIVING_RULE_MESSAGES. Der Sender geht nach Empfang von AWAITING_RULE_MESSAGES in den Zustand SENDING_RULE_MESSAGES. Es folgt die Übertragung der Informationstexte-Datei. Das Ende der Übertragung signalisiert der Sender mit einem END_OF_RULE_MESSAGES und geht in den Zustand RULE_MESSAGES_SENT. Der Receiver beendet den Empfang, sendet ein BYE und geht in den Zustand WAITING. Nach Empfang von BYE beendet der Sender die Verbindung. Die Verbindung ist abgebaut.

Kapitel 4

Implementierung

Das in Kapitel 3 beschriebene Konzept wird in einer Referenz-Implementierung umgesetzt. Die Besonderheiten dieser möglichen Implementierung sind in diesem Kapitel dargestellt. In Abschnitt 4.1 wird die Wahl der Programmiersprache und der Entwicklungsplattform begründet. Darauf folgt in Abschnitt 4.2 die Vorstellung der eingesetzten Firewall. In Abschnitt 4.3 wird die Umsetzung der Benachrichtigungsfunktion dargestellt. Die Absicherung der Skriptübertragung wird in Abschnitt 4.4 erläutert. In Abschnitt 4.5 werden die Details der Implementierung dargestellt. Das Kapitel schließt mit Abschnitt 4.6, in dem die Virtualisierungssoftware VirtualBox vorgestellt wird und ihre Verwendung zur Simulation eines Testnetzwerks beschrieben wird.

4.1 Programmiersprache und Entwicklungsplattform

Voran gegangene Implementierungsarbeiten für die InFO wurden stets in der Programmiersprache Java vorgenommen (siehe Abschnitt 2.3). Dies trifft auch auf den InFOParser zu, der die Grundlage für den hier entwickelten RouterParser stellt (siehe Anhang B.1.2). Zur Integration der verschiedenen Systeme ist es daher sinnvoll, das hier vorgestellte System ebenfalls in Java zu implementieren. Zudem besitzt der Autor dieser Arbeit bereits einige Programmiererfahrung mit Java. Zur Implementierung wird die Java Platform, Standard Edition in der Ver-

sion 6 genutzt. Als Entwicklungsumgebung wird sich für NetBeans IDE[1] in der Version 7 entschieden. Für das Build-Management wird Maven[2] in der Version 3.0.5 genutzt.

4.2 Verwendete Firewall

Das konzipierte System soll mit einer Referenz-Firewall implementiert werden. In Abschnitt 4.2.1 wird die Wahl für iptables als Referenz-Firewall begründet. Im darauf folgenden Abschnitt 4.2.2 wird die Firewallkonfiguration mit iptables beschrieben.

4.2.1 Wahl der Firewall

Für die Implementierung soll keine eigene Firewall entwickelt werden, sondern möglichst auf vorhandene Software zurückgegriffen werden. Die Software soll gut dokumentiert und nach Möglichkeit quelloffen sein, um die Software im Bedarfsfall anzupassen. Es soll möglich sein, die Software über die Kommandozeile zu konfigurieren. Dies ist insbesondere für einen entfernten Funktionsaufruf und/oder Automatisierung relevant.

iptables[3] ist ein verbreitetes Konfigurationsprogramm für den Linux-Kernel. Es ist auf vielen Linux-Distributionen standardmäßig installiert und kann über die Kommandozeile konfiguriert werden. Es bietet einen angemessenen Funktionsumfang zur Konfiguration der Firewall. Die Software ist gut dokumentiert und kann über die Kommandozeile konfiguriert. Daher wird sich für die Referenzimplementierung für iptables entschieden. Im folgenden Abschnitt wird näher auf iptables eingegangen.

4.2.2 Funktionalität der Firewall

iptables dient für den Linux-Kernel lediglich als Konfigurationsprogramm. Die Firewall-Funktionalität selbst wird im Kernel durch eine Reihe von Netfilter-Modulen hergestellt. Das *Netfilter*-Projekt[4] beherbergt verschiedene Software-Modu-

[1] https://netbeans.org/ - Abruf: 25.01.2014
[2] http://maven.apache.org/ - Abruf: 25.01.2014
[3] http://www.netfilter.org/projects/iptables/index.html - Abruf: 25.01.2014
[4] http://www.netfilter.org/ - Abruf: 26.0.2014

le für den Linux-Kernel, die Funktionen wie Paketfilterung und Network Address Translation (NAT) bereit stellen. Netfilter ist seit Kernel-Version 2.4 im Linux-Kernel integriert. iptables ist ebenfalls Teil des Netfilter-Projekts. Intern ist die Netfilter-Firewall in *Tabellen* organisiert [And06]. Eine Tabelle ist einem bestimmten Zweck zugeordnet. So ermöglicht die Tabelle `filter` die Paketfilterung, während die Tabelle `nat` für Network Address Translation (NAT) zuständig ist. Die Namen und eine Beschreibung der in iptables verfügbaren Tabellen liefert Tabelle 4.1. Durch die Definition von Regeln in den Tabellen können Pakete, die die Firewall durchlaufen manipuliert werden. Eine Regel definiert sich aus einem oder mehreren Parametern, die auf ein Paket zutreffen müssen (*Match*). Zudem definiert die Regel eine Aktion, die im Falle eines Matchs ausgeführt wird. Eine zulässige Regel für die Tabelle `filter` wäre zum Beispiel das Verwerfen aller Pakete, die `141.26.64.1` als Ziel-IP-Adresse besitzen.

Tabelle	Aufgabe
raw	Spezielle Tabelle zur Verbindungsverfolgung (*connection tracking*).
mangle	Änderung bestimmter IP-Header-Daten wie TTL oder TOS.
nat	Anwendung der NAT (siehe Abschnitt 2.1.2).
filter	Paketfilterung.

Tabelle 4.1 Namen und Beschreibung der Tabellen in iptables.

Bei der Anwendung der Firewallregeln ist unter Umständen der Zeitpunkt entscheidend. So kann die obige Paketfilterregel unterschiedliche Effekte haben, wenn davor oder danach eine NAT-Regel angewendet wird. Zur Festlegung, wann eine Regel angewendet wird, sind die Regeln einer Tabelle in vordefinierten *Ketten* organisiert. So werden die Regeln der Kette PREROUTING auf alle eingehenden Pakete angewendet, während die Kette INPUT nur auf eingehende Pakete angewendet wird, deren Ziel ein lokaler Prozess ist. Die Ketten „verankern“ somit ihre Regeln im Paketverarbeitungsprozess des Kernel. Eine Übersicht der in iptables verfügbaren Ketten gibt Tabelle 4.2. Die Regeln einer Kette werden in der Reihenfolge abgearbeitet, wie sie in der Kette eingetragen sind. Trifft eine Regel zu, wird die Kette anschließend verlassen.

Für die Regulierung mit der RFCO sind die Tabellen `nat` (Network Address Translation) und `filter` (Paketfilterung), sowie die Ketten `FORWARD` (weitergeleitete Pakete), `PREROUTING` (eingehende Pakete) und `POSTROUTING` (ausgehen-

Kette	Anwendungsbereich
PREROUTING	Alle eingehenden Pakete.
INPUT	Alle eingehenden Pakete, deren Ziel ein lokaler Prozess ist.
FORWARD	Alle eingehenden Pakete, deren Ziel ein anderer Host ist.
OUTPUT	Alle ausgehenden Pakete, die von einem lokalen Prozess stammen.
POSTROUTING	Alle ausgehenden Pakete.

Tabelle 4.2 Namen und Beschreibung der Ketten in iptables.

de Pakete) relevant. Eingehende Pakete durchlaufen die Ketten in einer festgelegten Reihenfolge [And06], die in Abbildung 4.1 dargestellt ist. Ein empfangenes Paket wird zuerst von der Kette `PREROUTING` (Tabelle `nat`) behandelt. Ist ein lokaler Prozess das Ziel des Pakets, durchläuft es anschließend die Kette `INPUT` (Tabelle `filter`) und wird an den Prozess weitergeleitet. Ist ein entfernter Host das Ziel des Pakets, durchläuft es zunächst die Kette `FORWARD` (Tabelle `filter`) und anschließend die Kette `POSTROUTING` (Tabelle `nat`). Danach wird es an die entsprechende Netzwerkschnittstelle weitergeleitet.

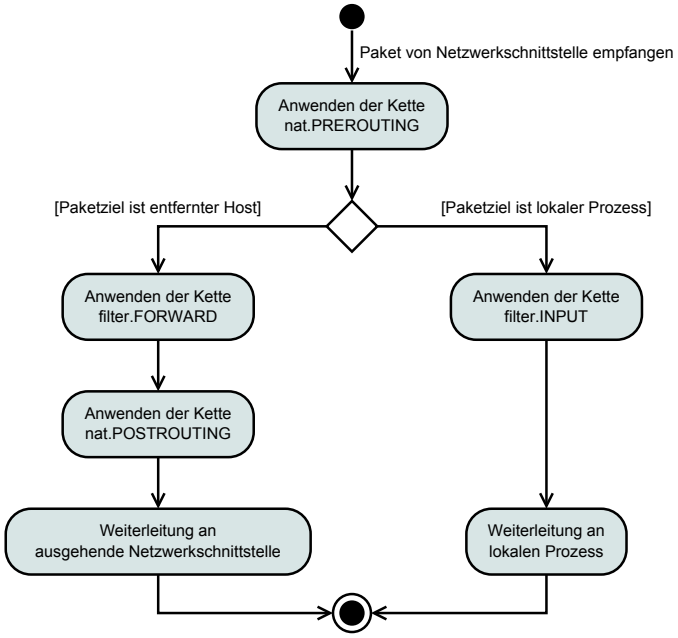


Abbildung 4.1 Übersicht der Kettenabarbeitung innerhalb der Linux-Firewall nach [And06]. Ein eingehendes Paket durchläuft die Ketten in der angegebenen Reihenfolge.

Um eine Regel auf eine Kommunikation anzuwenden, muss diese mit iptables zur entsprechenden Tabelle und Kette hinzugefügt werden. Sollen zum Beispiel alle Pakete verworfen werden, deren Ziel im Netzwerk `10.0.0.0/8` liegt und die nicht von einem lokalen Prozess stammen, so muss eine solche Regel zur Paketfilterungs-Tabelle `filter` und dort zur Kette für weitergeleitete Pakete `FORWARD` hinzugefügt werden. Dies lässt sich erreichen, indem iptables mit den entsprechenden Parametern aufgerufen wird. Listing 4.1 zeigt den entsprechenden Befehl.

```
iptables --append FORWARD --destination 10.0.0.0/8 --jump DROP
```

Listing 4.1 Hinzufügen einer Filterregel zu iptables.

Hier gibt der Parameter `append` die Kette an, der die Regel hinzu gefügt werden soll. Falls wie hier keine Tabelle mit dem Parameter `table` benannt ist, wird der Standardwert `filter` genutzt. `destination` bestimmt das Ziel der Pakete. Der Parameter `jump` gibt an, wie weiter mit dem Paket verfahren werden soll. Der Wert `DROP` legt fest, dass es verworfen wird. Eine ausführliche Beschreibung der verfügbaren Parameter bietet [And06]. Mehrere solcher Konfigurationsbefehle für iptables wie in Listing 4.1 können in einem Kommandozeilenskript zusammengefasst werden. Ein kommentiertes Bash-Skript zur offenen Initialisierung von iptables zeigt Listing B.2 im Anhang B.3.

Die Anforderung A.5 verlangt, dass bei regulierter Kommunikation eine Nachricht an den Initiator der Kommunikation gesendet wird. iptables ermöglicht es mit dem Parameter `jump␣REJECT` eine ICMP-Nachricht als Rückmeldung zu senden. Listing 4.2 zeigt einen Konfigurationsbefehl für iptables, durch den dem Absender mitgeteilt wird, dass der Zugang zum Netzwerk `10.0.0.0/8` administrativ unterbunden ist.

```
iptables --append FORWARD --destination 10.0.0.0/8
         --jump REJECT --icmp-net-prohibited
```

Listing 4.2 Filterregel mit Benachrichtigung über ICMP.

4.3 Senden von Benachrichtigungen

Es ist über iptables nicht direkt möglich, eine eigene Nachricht mit Hintergrundinformationen (siehe Anforderung A.5.3) in eine ICMP-Nachricht einzubetten (siehe vorherigen Abschnitt 4.2.2). Diese Funktion muss daher extern bereitgestellt werden. Dieser Abschnitt erläutert den Vorgang zur Protokollierung eines regulierten IP-Paket und der Erstellung eines Benachrichtigungspakets. In Abschnitt 4.3.1 wird zunächst der allgemeine Ablauf vorgestellt. Die Funktionen der zur Protokollierung benötigten Hilfsprogramme werden dann in den darauf folgenden Abschnitten 4.3.2 und 4.3.3 erklärt. Die eigentliche Erstellung des Benachrichtigungspakets wird später in Abschnitt 4.5.3 erläutert.

4.3.1 Allgemeiner Ablauf

Das Modul *PolicyNotifier* im PolicyAgent implementiert das Senden von Benachrichtigungspaketen (siehe Abschnitte 3.5.2 und 4.5.3). Zur Benachrichtigung von betroffenen Hosts benötigt der PolicyNotifier zeitnah Informationen über regulierte Pakete. Mit iptables kann das Protokollieren von Paketinformationen konfiguriert werden. Die Pakete werden dazu an den Logging-Daemon *ulogd* weitergeleitet. Mit ulogd ist es möglich die Paketinformationen zu extrahieren und in verschiedenen Formaten auszugeben. Für die Ausarbeitung werden die Paketinformationen durch den ulogd in eine Datenbank ausgegeben. Die Datenbank dient dabei zur Zwischenspeicherung der Paketinformationen. Die Paketinformationen werden anschließend vom PolicyNotifier aus der Datenbank ausgelesen. Der PolicyNotifier erstellt auf Basis der Paketinformationen ein ICMP-Paket, dass er an den zu benachrichtigenden Host sendet. Die Interaktion zwischen den einzelnen Komponenten ist in Abbildung 4.2 dargestellt. ulogd und die eingesetzte Datenbank sind in den folgenden Abschnitten näher beschrieben. Die Erweiterungen am PolicyNotifier zur Benachrichtigung werden in Abschnitt 4.5.3 erläutert.

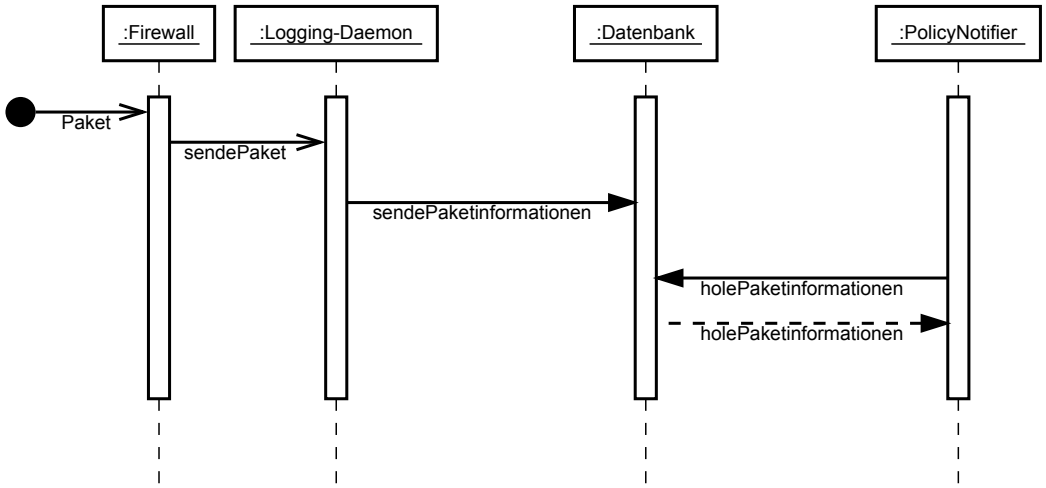


Abbildung 4.2 Sequenzdiagramm zur Protokollierung der Paketinformationen. Das Diagramm zeigt die an der Paketprotokollierung beteiligten Komponenten und die zwischen ihnen gesendeten Nachrichten/Daten. Ein von der Firewall reguliertes Paket wird an den Logging-Daemon weitergeleitet. Der Logging-Daemon extrahiert die Paketinformationen und legt sie in der Datenbank ab. Der PolicyNotifier liest die Paketinformationen aus der Datenbank. Mit den Paketinformationen erstellt der PolicyNotifier ein Benachrichtigungspaket.

4.3.2 Logging-Daemon

ulogd[5] ist ein allgemeiner Logging-Daemon des Netfilter-Projekts, der ein benutzerdefinierte Protokollierung von Paketen für Netfilter-Module und damit auch iptables ermöglicht. Damit können Pakete erfasst werden, die durch die Firewall reguliert wurden. Die Pakete werden dazu von der Firewall an ulogd weitergeleitet mit Nennung einer bestimmten *Multicast*-Gruppennummer. Anhand dieser Nummer kann innerhalb von ulogd unterschieden werden, von welchem Netfilter-Modul das Paket stammt und wie es weiter bearbeitet werden soll. Die Festlegung der Weiterverarbeitung und Speicherung muss dabei in der Konfigurationsdatei von ulogd (s.u.) vorgenommen werden. Listing 4.3 zeigt einen Konfigurationsbefehl für iptables, bei dem alle an das Netzwerk 10.0.0.0/8 gerichteten Pakete mit ulogd unter der Gruppe 1 protokolliert werden. Die Protokollierung wird mit dem Parametern `jump␣ULOG` und `ulog-nlgroup␣1` bestimmt.

Es ist bei jeder Logging-Regel möglich, ein Präfix mitzugeben, das mitgeloggt wird. Anhand des Präfix lässt sich ein reguliertes Paket später leicht einer

[5] http://www.netfilter.org/projects/ulogd/index.html - Abruf: 27.0.1.2014

```
iptables --append FORWARD --destination 10.0.0.0/8
         --jump ULOG --ulog-nlgroup 1
```

Listing 4.3 Protokollieren von Paketen mit ulogd.

Firewall-Regel zuordnen. Das Präfix ist jedoch auf 31 Zeichen beschränkt. In Listing 4.4 wird die Regel aus Listing 4.3 um das Präfix `rule-foo` ergänzt.

```
iptables --append FORWARD --destination 10.0.0.0/8
         --jump ULOG --ulog-nlgroup 1 --ulog-prefix rule-foo
```

Listing 4.4 Loggen mit Regel-Präfix

Innerhalb von ulogd werden die protokollierten Pakete durch Plugins verarbeitet und ausgegeben. Die zu verwendenden Plugins und die Reihenfolge ihrer Abarbeitung werden in einer Konfigurationsdatei für ulogd bestimmt. ulogd arbeitet mit drei Arten von Plugins: *Input-Plugins* formatieren eingehende Daten in ein internes Format, indem die Inhalte extrahiert und den spezifischen Schlüsseln des Logging-Daemon zugeordnet werden. Je nach Wahl des Input-Plugin werden unterschiedliche Daten extrahiert. *Filter-Plugins* bestimmen im weiteren Schritt die interne Verarbeitung der Daten. Diese können etwa der Konvertierung von Schnittstellennummern in Schnittstellennamen dienen oder der isolierten Extraktion bestimmter Daten. Die Verwendung von Filter-Plugins ist optional. Es können mehrere Filter-Plugins in Reihe geschaltet werden. *Output-Plugins* bestimmen Ausgabeformat und -ort. Die Ausgabe kann in eine Textdatei erfolgen. Es existieren aber auch Output-Plugins für verschiedene Datenbanksysteme. Für die Referenzimplementierung wird das Standard-Input-Plugin für IP-Pakete `ulogd_BASE.so` verwendet. Die Ausgabe der verarbeiteten Daten erfolgt in eine SQLite-Datenbank (siehe Abschnitt 4.3.3) mit dem Output-Plugin `ulogd_SQLITE3.so`. Die kommentierte Konfigurationsdatei für die Implementierung enthält Listing B.3 in Anhang B.4.

Nach der Verarbeitung der Pakete durch ulogd stehen nicht mehr alle Paketdaten zur Verfügung. So geht durch die Anwendung der Plugins der genaue Inhalt der Payload des IP-Pakets verloren. Zur Erhaltung der Payload müssten die Pakete im Klartext gespeichert und anschließend weiterverarbeitet werden. Da diese Alternative erheblich aufwändiger ist, wurde sich für oben erwähnten Weg entschieden.

4.3.3 Datenbank

Zur Zwischenspeicherung der Paketinformationen wird eine SQLite[6]-Datenbank verwendet. SQLite ist ein relationales Datenbankmanagementsystem (DBMS), das als Programmbibliothek implementiert ist. Dies ermöglicht die direkte Integration des DBMS in ein Programm. Die Daten der Datenbank werden in einer Datei abgelegt. Die Datenbank für die Paketinformationen besteht aus einem einfachen Schema mit einer Tabelle mit dem Namen *ulog*. Das Tabellenschema (Spaltennamen und -typen) wird dabei durch die gewählten Plugins von ulogd (siehe Abschnitt 4.3.2) vorgegeben: Als Spaltennamen müssen die Namen der ulogd-Schlüssel verwendet werden. Beim Start von ulogd interpretiert ulogd seine Konfigurationsdatei. Anschließend wertet ulogd das Datenbankschema der dort referenzierten Datenbank aus. Anhand des Datenbankschemas, erkennt ulogd welche Paketinformationen in der Datenbank gespeichert werden sollen. Ist für eine später verfügbare Paketinformation kein entsprechende Spalte in der Datenbank vorhanden, wird die Information verworfen.

Die Spalten der Datenbanktabelle ulog sind in Tabelle 4.3 aufgelistet. Die Spalte `id` enthält eine eindeutige Nummer zur Unterscheidung der verschiedenen Einträge und wird automatisch durch die Datenbank erzeugt. Aus `oob_time_sec` und `oob_time_usec` ergibt sich der Zeitstempel des Empfangszeitpunkt des Pakets. Bei `oob_prefix` handelt es sich um das Präfix mit dem die angewendet Firewallregel und damit die InFO- eindeutig identifiziert wird. Die restlichen Spalten beherbergen Paketinformationen der Netzzugangs-, Internet- oder Transportschicht.

4.4 Absicherung der Übertragung

Die Übertragung des Skripts vom PolicyAdmin zum PolicyAgent soll integer und vertraulich sein (siehe Anforderung A.7). Zur Gewährleistung dieser Anforderung wird das Secure-Sockets-Layer-(SSL)[7]-Protokoll benutzt [DA99]. Bei Verbindungsaufbau erfolgt eine gegenseitige Authentifizierung von PolicyAdmin und PolicyAgent. Umgesetzt wird das Protokoll mit Klassen der Java Secure Socket

[6] http://www.sqlite.org/ - Abruf: 29.01.2014
[7] SSL ist das Vorgänger-Protokoll von TLS.

Spaltenname	Beschreibung
id	Automatisch vergebene ID der Datenbank
oob_time_sec	Empfangszeitpunkt in Sekunden
oob_time_usec	Mikrosekundenteil des Empfangszeitpunkts
oob_prefix	Verwendetes Präfix der Firewall-Regel
oob_in	Eingehende Netzwerkschnittstelle
raw_mac	MAC-Adressen des Ethernet-Frame
ip_protocol	IP-Protokollnummer
ip_id	IP-Identification (bei Fragmentierung)
ip_totlen	IP-Paketlänge
ip_saddr	IP-Absenderadresse
ip_daddr	IP-Zieladresse
tcp_sport	TCP-Quelladresse
tcp_dport	TCP-Zieladresse
udp_sport	UDP-Quelladresse
udp_dport	UDP-Zieladresse

Tabelle 4.3 Namen und Beschreibung der Spalten für die Datenbanktabelle ulog.

Extension[8]. Der PolicyAgent setzt dafür statt der Klasse *java.net.ServerSocket* die Klasse *javax.net.ssl.SSLServerSocket* ein. Ebenso nutzt der PolicyAdmin zum Verbindungsaufbau die Klasse *javax.net.ssl.SSLSocket*.

Für das SSL-Protokoll wird von jedem Teilnehmer ein Schlüsselpaar aus privatem und öffentlichem Schlüssel benötigt. Das eigene Schlüsselpaar wird von jedem Teilnehmer in einer Datei gespeichert. Diese Datei wird als *Keystore* bezeichnet. Die öffentlichen Schlüssel sind untereinander auszutauschen. Fremde öffentliche Schlüssel werden ebenfalls in einer Datei gespeichert, die als *Truststore* bezeichnet wird. PolicyAdmin und PolicyAgent geben jeweils bei Programmstart dem JRE den Keystore und Truststore bekannt. Nur so können die Schlüssel für eine SSL-Verbindung genutzt werden. Keystore und Truststore können zusammengelegt werden. Für die Simulation (siehe Abschnitt 4.6) werden Schlüssel mit RSA [RSA78] als Verschlüsselungsalgorithmus und einer Schlüssellänge von 3072 Bits benutzt. Als Hash-Algorithmus wird SHA-256 [Nat12] verwendet. Dies entspricht den Empfehlungen der European Network and Information Security Agency (ENISA) zum Erstellungszeitpunkt der Arbeit [Eur13]. Schlüsselerzeugung und -verteilung sind im Anhang A.2 beschrieben. Aus Gründen

[8] http://docs.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html - Abruf: 25.01.2014

der Einfachheit werden die Passwörter für den Keystore und Truststore in der Referenzimplementierung im Klartext in den Konfigurationsdateien von PolicyAdmin und PolicyAdmin gespeichert. Für eine sichere Implementierung ist jedoch beispielsweise die Abfrage der Passwörter an den Administrator beim Starten des PolicyAdmins/PolicyAgents nötig.

4.5 Details der Referenzimplementierung

Um das entwickelte System praktisch einsetzen zu können, ist es nötig, die Architektur zu erweitern und die in der Konzeption beschriebenen Schnittstellen zu implementieren. In diesem Abschnitt werden die vorgenommenen Erweiterungen und Änderungen für die Referenzimplementierung dokumentiert, sofern diese noch nicht in den vorangegangenen Abschnitten beschrieben wurden.

4.5.1 Erzeugung des URI-Hashs

Um eine Rule eindeutig zu unterscheiden, wird ein Hash-Wert des zugehörigen URIs gebildet. Die Hash-Erzeugung wird beim Konstruktor-Aufruf von `Rule` vorgenommen. Als Hash-Algorithmus wird *MD5* verwendet [Riv92]. Das Verfahren MD5 ist nicht kollisionsfrei [WFLY04]. Jedoch wird der Algorithmus hier lediglich zur Identifizierung und nicht zur Signierung verwendet. Eine zufällige Hash-Kollision von zwei unterschiedlichen URIs wird ausgeschlossen.

4.5.2 Realisierung der Schnittstelle FirewallVocabulary

Die Übersetzung von Rules in Firewallregeln wird durch FirewallTranslator vorgenommen. Dazu benötigt der FirewallTranslator eine Realisierung der Schnittstelle FirewallVocabulary, die auf die Ziel-Firewall angepasst ist (siehe Abschnitte 3.4.4 und 3.5.2). Dies wird für die gewählte Referenz-Firewall iptables (siehe Abschnitt 4.2.2) durch die Klasse `IPTablesVocabulary` bewerkstelligt. Für jede zu übersetzende Rule erzeugt `IPTablesVocabulary` zwei Firewallregeln. Die erste Firewallregel veranlasst die Firewall zum Protokollieren des regulierten Pakets mit ulogd (siehe Abschnitt 4.3.2). Dies dient der späteren Benachrichtigung durch den PolicyNotifier (siehe Anhang B.1.7). Damit ulogd die Pakete

protokollieren kann, muss in der Firewallregel als Parameter die gleiche Multicast-Gruppe[9] angegeben werden, die in der ulogd-Konfiguration für iptables verwendet wurde (siehe Abschnitt 4.3.2). Als Regel-Präfix wird der Hash-Wert der URI der Rule verwendet. Dieser dient damit zugleich der Zuordnung einer Firewallregel zum Informationstext durch den PolicyNotifier. Die zweite Firewallregel enthält die eigentliche Regulierung, also die Blockierung oder Umleitung des Pakets. Für Rules, die Kommunikation zulassen, wird kein Log-Befehl erzeugt durch `IPTablesVocabulary`.

4.5.3 Erweiterungen am PolicyNotifier

Der PolicyNotifier sendet Benachrichtigungen an die Absender der durch die Firewall regulierten Pakete (siehe Abschnitt 3.5.2). Die dafür nötigen Paketinformationen werden in einer SQLite-Datenbank zwischengespeichert (siehe Abschnitt 4.3.3). Der PolicyNotifier liest die Datenbank aus und erstellt aus den Paketinformationen ein Benachrichtigungspaket (s.u.). Zur Interaktion mit der Datenbank verwendet der PolicyNotifier die Klasse `DatabaseReader`. Der DatabaseReader selbst nutzt zum Stellen von Anfragen an die SQLite-Datenbank einen entsprechenden *JDBC*-Treiber[10]. Die Datenbank soll als Zwischenspeicher nur Informationen über Pakete vorhalten, für die noch kein Benachrichtigungspaket gesendet wurde Deshalb löscht der DatabaseReader vor dem Abrufen neuer Einträge die bereits gelesenen Einträge aus der Datenbank. Dazu führt der DatabaseReader die Variablen `timeSecondsMarker` und `timeUsecondsMarker`. Aus beiden Variablen ergibt sich der Zeitstempel des jüngsten, bereits gelesenen Eintrags in der Datenbank. Der PolicyNotifier setzt nach Auslesen des einer Datenbankabfrage die neuen Werte für `timeSecondsMarker` und `timeUsecondsMarker`.

Die Benachrichtigung des Absenders eines regulierten Pakets erfolgt durch Senden einer ICMP-Nachricht. ICMP-Nachrichten dienen zur Fehlerdiagnose im Netzwerk [Pos81a] und werden von allen Netzwerkkomponenten verstanden. Dadurch ist ICMP zum Senden der Benachrichtigungen besser geeignet als ein eigenes Nachrichtenformat. Jedes ICMP-Paket enthält im Nachrichtenkopf (*Header*) ein *Type*-Feld und ein *Code*-Feld. Das Type-Feld gibt die Fehlerkategorie an

[9] Für die Implementierung wird durchgängig die Gruppennummer 1 verwendet.

[10] http://www.oracle.com/technetwork/java/overview-141217.html - Abruf: 10.02.2014

und bestimmten den weiteren Aufbau des ICMP-Pakets. Das Code-Feld dient einer genaueren Einteilung des Fehlers. ICMP besitzt vordefinierte Fehlerkategorien. Es ist theoretisch aber auch möglich, für die Regulierungsbenachrichtigungen eigene Kategorien anhand unbenutzter Type-Code-Kombinationen zu definieren. Diese Nachrichten können jedoch nicht nativ vom Ziel-Host verarbeitet werden und werden dort verworfen. Eine solche Lösung würde eine Anpassung des ICMP-Protokolls beim Ziel-Host erforderlich machen. Zielführender ist es, die vorhandenen Fehlerkategorien von ICMP zu nutzen und um einen Informationstext für die Regulierung zu erweitern. Die modifizierte ICMP-Nachricht kann somit durch entsprechende Software auf dem Ziel-Host ausgewertet werden. Gleichzeitig wird eine Abwärtskompatibilität sichergestellt für Hosts, die die enthaltenen zusätzlichen Informationen nicht auswerten können. Die Umsetzung dieser Lösung wird im folgenden erläutert.

Die vom PolicyNotifier gesendeten Benachrichtigungen nutzen die ICMP-Fehlerkategorie *Destination Unreachable* (Type 3). Als ICMP-Code wird der Wert 13 (*Communication administratively prohibited*) gesetzt. Als Empfängeradresse wird die Absenderadresse des regulierten Pakets genutzt. Die Nutzdaten (*Payload*) sollen nach RFC-Definition [Pos81a] den Header und die ersten 64 Bits der Payload des regulierten Pakets enthalten. Da die Payload des regulierten Pakets nach der Verarbeitung durch ulogd (siehe Abschnitt 4.3.2) nicht mehr vorliegt, werden die 64 Bits mit Nullen aufgefüllt. Anschließend folgt in ASCII-Kodierung eine kurze Nachricht, die auf die Regulierung hinweist und auf die URI der angewendeten Regel verweist. Der Einfachheit halber wurde sich hier für die URI der Regel entschieden. Es ist jedoch möglich, hier eine andere individuelle URL anzugeben. Ein Schema des ICMP-Pakets zeigt Abbildung 4.3. Das ICMP-Paket wird über die Netzwerkschnittstelle gesendet, von der das regulierte Paket empfangen wurde.

Zur Erzeugung des ICMP-Pakets durch den PolicyNotifier wird die freie Java-Bibliothek *Pcap4J*[11] genutzt. Da es sich bei Pcap4J um einen Wrapper für native Packet-Capture-Bibliotheken handelt, ist die Debian-Bibliothek *libpcap* (siehe Anhang A.1) eine Installationsvoraussetzung für den PolicyAgent .

[11] http://www.pcap4j.org/ - Abruf: 10.02.2014

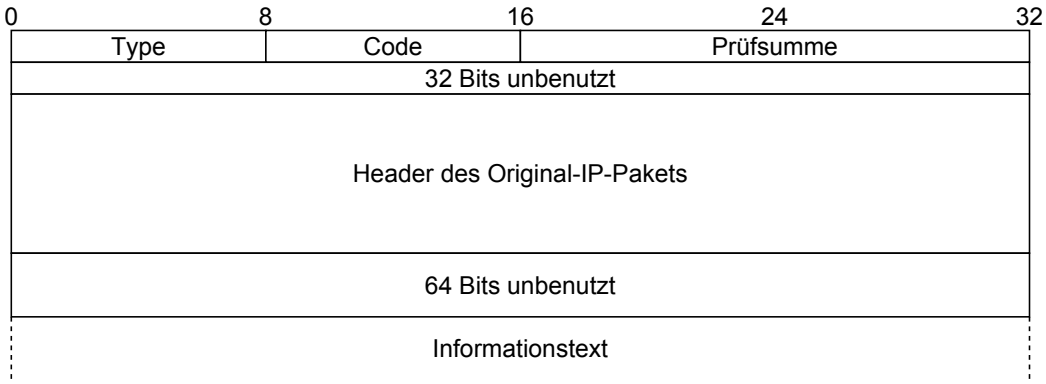


Abbildung 4.3 Schema des ICMP-Pakets zur Regulierungs-Benachrichtigung. Das Type-Feld hat den Wert 3, das Code-Feld den Wert 13.

4.6 Simulation in VirtualBox

Das entwickelte System soll exemplarisch in einem Netzwerk getestet werden. Um den Aufbau eines Testnetzwerk zu erleichtern, wird die Virtualisierungssoftware VirtualBox eingesetzt. In Abschnitt 4.6.1 wird VirtualBox kurz vorgestellt und seine Möglichkeiten zur Netzwerk-Simulation beschrieben. In Abschnitt 4.6.2 wird dann der Aufbau eines Testnetzwerks beschrieben. Die Informationen sind [Ora13] entnommen.

4.6.1 Grundlagen zu VirtualBox

VirtualBox[12] ist eine Virtualisierungssoftware unter Open Source-Lizenz. Mit VirtualBox ist es möglich, ein oder mehrere Gast-Betriebssysteme (*VM-Guests*) auf einem Wirts-Rechner (*VM-Host*) auszuführen. Ein VM-Guest muss dafür die Architektur des VM-Host unterstützen. Beispielsweise kann ein VM-Guest mit 32-Bit-Betriebssystem nur auf einem VM-Host mit 32-Bit-Prozessor betrieben werden. VirtualBox bietet eine grafische Benutzeroberfläche zur Erzeugung, Verwaltung und Ausführung der VM-Guests. Alternativ kann VirtualBox über das Frontend *VBoxManage* komplett über die Kommandozeile gesteuert und damit automatisiert ausgeführt werden. Durch Installation der Zusatz-Software *Guest Additions* auf dem VM-Guest, können gemeisam genutzte Verzeichnisse (*shared folders*) eingerichtet werden. Damit ist eine einfache Übertragung von Dateien zwischen VM-Guest und VM-Host möglich. Die VM-Guests können über virtuelle Netz-

[12] https://www.virtualbox.org/ - Abruf: 31.01.2014

werke miteinander verbunden werden, sowie Netzwerkverbindungen des VM-Hosts nutzen. Hierfür bietet VirtualBox verschiedene Netzwerkmodi. Für die Simulation werden die Modi *NAT* und *internal network* verwendet. *NAT* ermöglicht das Nutzen der Internetverbindung des VM-Host durch den VM-Guest. Wie der Name andeutet, wird dazu nach außen Network Address Translation (NAT) angewendet. Ein *internal network* erlaubt das Verbinden der VM-Guests untereinander, ohne Internetzugriff.

4.6.2 Aufsetzen der Netzwerks

Der Aufbau des Testnetzwerks in VirtualBox ist in Abbildung 4.4 dargestellt. Für den Admin-Knoten und den Agent-Knoten wird jeweils ein VM-Guest mit dem Betriebssystem *Debian*[13] in der Version 6.0.8 aufgesetzt. Debian ist hier zum Test geeignet, da es sich um eine sehr stabile Distribution handelt und zudem eine umfangreiche Sammlung an Anwendungssoftware enthält. Dies erleichtert die Installation der benötigten Drittsoftware (siehe Anhang A). Admin-Knoten und Agent-Knoten sind über ein *internal network* (siehe Abschnitt 4.6.1) miteinander verbunden. Der Agent-Knoten nutzt über eine weitere virtuelle Netzwerkschnittstelle die Internetverbindung des VM-Hosts (Netzwerkmodus *NAT*, siehe Abschnitt 4.6.1). Damit fungiert der Agent-Knoten innerhalb dieser Simulation als Internet-Gateway für die restlichen virtuellen Maschinen. Drei weitere VM-Guests mit *Tiny Core Linux*[14] als Betriebssystem simulieren die Hosts der Endanwender. Tiny Core Linux ist eine Linux-Distribution mit geringem Ressourcenverbrauch, die komplett im Arbeitsspeicher läuft. Dennoch bietet sie eine grafische Benutzeroberfläche und ist daher gut zur Veranschaulichung der Regulierungsmaßnahmen geeignet. Die Hosts sind über ein weiteres *internal network* mit dem Agent-Knoten verbunden.

4.6.3 Durchführen der Simulation

Zur exemplarischen Veranschaulichung der Regulierung sollen Zugriffe auf bestimmte Systeme des Netzwerks der Universität Koblenz (Netzwerkadresse `141.26.0.0/16`) für die Hosts `TC_Host_1`, `TC_Host_2` und `TC_Host_3` reguliert

[13] http://www.debian.org/ - Abruf: 25.01.14
[14] http://www.tinycorelinux.net/ - Abruf: 01.02.2014

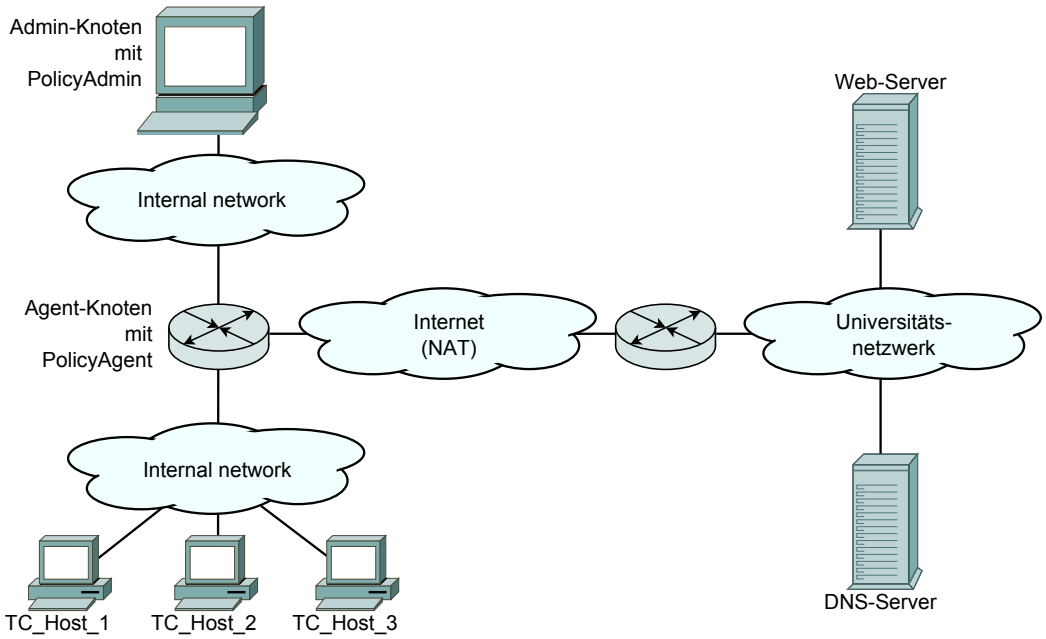


Abbildung 4.4 Skizze vom Aufbau des Testnetzwerks. Die Hosts `TC_Host_1` etc. sind über ein *internal network* mit dem *PolicyAgent* verbunden. Der PolicyAgent fungiert als Internet-Gateway für die Hosts. Der *PolicyAdmin* ist über ein separates *internal network* mit dem PolicyAgent verbunden. Über ein *NAT*-Netzwerk nutzt der PolicyAgent die Internetverbindung des VM-Hosts. Für die Simulation soll der Zugriff von den Hosts auf das Universitätsnetzwerk mit *Web-Server* blockiert werden. Der *DNS-Server* der Universität soll erreichbar bleiben.

werden (siehe Abbildung 4.4). Der Zugriff auf den DNS-Server der Universität *ns1.uni-koblenz.de* (IP-Adresse `141.26.64.1`) soll weiter möglich sein. Der Web-Server der Universität *www.uni-koblenz.de* (IP-Adresse `141.26.64.18`) wird dagegen nicht erreichbar sein für die Hosts. Von den Hosts an den Webserver gerichtete Pakete sollen durch den PolicyAgent abgewiesen werden. Der PolicyAgent sendet dem betroffenen Host ein Benachrichtigungspaket über die Regulierung. Dazu wird eine InFO-Policy mit zwei Regeln erstellt: Eine `IPAddressDenyingRule` (siehe Abschnitt 2.3.3) sperrt den Zugriff aus dem Netzwerk von `TC_Host_1` etc. (`10.0.0.0/8`) auf das Universität-Netzwerk. Eine `IPAddressAllowingRule` (siehe Abschnitt 2.3.3) erlaubt den Zugriff aus dem Netzwerk `10.0.0.0/8` auf den DNS-Server. Die Policy wird mit einer Meta Policy verknüpft, die als `RulePriorityRole` den `PreferLongestSubnetMaskAlgorithm` definiert (siehe Abschnitt 2.3.2 f.). Damit wird sichergestellt, dass die oben

genannte `IPAddressAllowingRule` bevorzugt wird. Die in OWL-Dateien verfasste Policy und Meta Policy werden später dem PolicyAdmin übergeben.

Der PolicyAgent wird mit einer „offenen“ Konfiguration gestartet (siehe Listing B.2 in Anhang B.3). Alle empfangenen Pakete werden durch den Agent-Knoten durchgeleitet. Die Netzwerkadressen der Hosts werden nach außen mit NAT maskiert (siehe Abschnitt 2.1.2). Es wird nicht in die Kommunikation der Hosts eingegriffen. Ein Ping auf dem Host `TC_Host_1` zeigt, dass benachbarte und entfernte Hosts erreichbar sind. Der Web-Server und der DNS-Server der Universität können angesprochen werden. Auf dem Admin-Knoten wird nun der PolicyAdmin gestartet. Dem PolicyAdmin werden die Policys mit der beabsichtigten Regulierung und die UID des PolicyAgents übergeben. Der PolicyAdmin übersetzt die Policys in ein Skript und sendet es an den PolicyAgent auf dem Agent-Knoten. Der PolicyAgent spielt das neue Skript ein. Ein erneuter Ping-Versuch auf `TC_Host_1` ergibt, dass der DNS-Server weiterhin erreichbar ist. Der Agent-Knoten und der Nachbar-Host `TC_Host_2` sind ebenfalls erreichbar. Ein Ping des Web-Servers scheitert jedoch.

Durch einen Mitschnitt mit *tcpdump*[15] auf `TC_Host_1` (siehe Listing 4.5) lässt sich die Regulierung durch den PolicyAgent nachvollziehen. Das Listing 4.5 zeigt den Ping-Versuch von `TC_Host_1` und (unerwartete) Antwort des Admin-Knoten. Dabei sind jeweils die gesamten IP-Pakete in Hex-Kodierung (links) und in ASCII-Kodierung (rechts) dargestellt. Für den Ping-Versuch wird ein ICMP-Echo-Request-Paket von `TC_Host_1` an den Webserver gesendet (Zeile 1 ff.). Daraufhin antwortet der Agent-Knoten bzw. der PolicyAgent mit einem ICMP-Destination-Unreachable-Paket (Zeile 10 ff.). In der Payload des ICMP-Pakets steht eine kurze Information über die vorgenommene Regulierung (Zeile 15 ff.). Außerdem wird auf die URI der Regel[16] (Zeile 17 ff.) verwiesen. Die beabsichtigte Regulierung wurde somit vollständig umgesetzt.

[15] http://www.tcpdump.org/ - Abruf: 16.02.2014

[16] `http://icp.it-risk.iwvi.uni-koblenz.de/policies/ipPolicy01.owl#r-1`

```
12:00:00.433867 IP TC_Host_1 > www.uni-koblenz.de:
 ICMP echo request, id 1579, seq 1, length 64
0x0000:  4500 0054 0000 4000 4001 6379 0a00 0004  E..T..@.@.cy....
0x0010:  8d1a 4012 0800 36e8 062b 0001 30f7 fd52  ..@...6..+..0..R
0x0020:  9b9e 0600 0809 0a0b 0c0d 0e0f 1011 1213  ................
0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  .............!"#
0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
0x0050:  3435 3637                                4567

12:00:01.822227 IP Agent-Knoten > TC_Host_1:
 ICMP host www.uni-koblenz.de unreachable - admin prohibited filter, length 160
0x0000:  4500 00b4 0064 0000 6401 41d9 0a00 0009  E....d..d.A.....
0x0010:  0a00 0004 030d 8d44 0000 0000 4500 0098  .......D....E...
0x0020:  0000 0000 6401 7f35 0a00 0004 8d1a 4012  ....d..5......@.
0x0030:  0000 0000 0000 0000 436f 6d6d 756e 6963  ........Communic
0x0040:  6174 696f 6e20 6861 7320 6265 656e 2062  ation.has.been.b
0x0050:  6c6f 636b 6564 2e20 5365 6520 6874 7470  locked..See.http
0x0060:  3a2f 2f69 6370 2e69 742d 7269 736b 2e69  ://icp.it-risk.i
0x0070:  7776 692e 756e 692d 6b6f 626c 656e 7a2e  wvi.uni-koblenz.
0x0080:  6465 2f70 6f6c 6963 6965 732f 6970 506f  de/policies/ipPo
0x0090:  6c69 6379 3031 2e6f 776c 2372 2d31 2066  licy01.owl#r-1.f
0x00a0:  6f72 206d 6f72 6520 696e 666f 726d 6174  or.more.informat
0x00b0:  696f 6e2e                                ion.
```

Listing 4.5 tcpdump-Mitschnitt auf Host *TC_Host_1*. Der Mitschnitt zeigt zwei Pakete: Einen Ping-Versuch von TC_Host_1 an *www.uni-koblenz.de* (Zeilen 1 bis 8) und die Benachrichtigung des PolicyAgents (Zeilen 10 bis 23). Der Inhalt der Pakete wird links in Hex-Kodierung und rechts in ASCII-Kodierung dargestellt. In der Payload des Benachrichtigungspakets ist der Informationstext enthalten (Zeilen 15 bis 23). Die IP-Adressen wurden durch die Bezeichner *TC_Host_1* und *Admin-Knoten* ersetzt.

Kapitel 5

Fazit und Ausblick

Diesel Kapitel fasst die Arbeit zusammen und bietet einen Ausblick auf zukünftige Arbeiten. In Abschnitt 5.1 werden die erreichten Ziele kurz zusammengefasst. Im darauf folgenden Abschnitt 5.2 erfolgt eine Bewertung der Arbeit durch den Vergleich der Anforderungen mit der vorgenommenen Konzeption und Implementierung. Das Kapitel schließt in Abschnitt 5.3 mit einer Diskussion über mögliche Verbesserungen und Weiterentwicklungen im Kontext dieser Arbeit.

5.1 Zusammenfassung

Mit dem in dieser Arbeit entwickelten System wird eine Möglichkeit zur Anwendung der mit RFCO beschreibbaren Policys aufgezeigt. Damit wurde erstmals die Router-spezifische Erweiterung der InFO technisch umgesetzt. Zusammen mit dem entwickelten DNS-Server [Mos12], der Suchmaschine [Rus13] und dem Proxy-Server [BGSS14] komplettiert sie die in [KS13] beschriebenen Domänenspezifischen Erweiterungen der InFO.

Das entwickelte System ermöglicht die für betroffene Nutzer transparente Gestaltung von Regulierungsmaßnahmen: Durch den Einsatz von ICMP-Nachrichten mit menschenlesbaren Informationstexten sind vorgenommene Regulierungsmaßnahmen auf IP-Ebene als solche erkennbar. Durch die Aufteilung in die Komponenten PolicyAdmin und PolicyAgent kann das entwickelte System flexibel an das zu administrierende Netzwerk angepasst werden. Auch die verwendete Schnittstelle FirewallVocabulary erlaubt eine flexible Anpassung auf die die eingesetzten Firewalls.

5.2 Bewertung

In diesem Abschnitt werden die gestellten Anforderungen (siehe Abschnitt 3.2) mit Konzept (siehe Abschnitt 3.3 ff.) und Implementierung (siehe Kapitel 4) verglichen. Es wird dabei jeweils in Klammern auf die Anforderungen referenziert. Überblickend wird festgestellt, dass alle Anforderungen umgesetzt wurden. Lediglich Anforderung A.5.3 wurde nur teilweise umgesetzt.

Es wurde ein System entwickelt, das die Umsetzung von InFO-Policy-Dateien auf einem Router erlaubt (Anforderung A.2). Die Policys werden dabei durch das Modul *RouterParser* eingelesen (Anforderung A.1) und in Java-Objekte übersetzt. Diese werden dann durch das Modul *FirewallTranslator* weiterverarbeitet und in Firewallregeln übersetzt. Die zur Übersetzung verwendete Schnittstelle *FirewallVocabulary* erlaubt eine Anpassung auf die jeweilige Ziel-Firewall. Auf einem Firewall-Router können die Regeln dann zur Anwendung gebracht werden. Bei der Weiterverabeitung durch den FirewallTranslator werden die Priorisierungs- und Konfliktlösungsalgorithmen der Meta Policy bzw. der Policys angewendet. Dadurch werden Konflikte zwischen Regeln (Anforderung A.3) und zwischen Policys (Anforderung A.4) aufgelöst.

Die Regulierung wird für die betroffene Parteien transparent dargestellt (Anforderung A.5), indem ICMP-Nachrichten an diese gesendet werden, die Auskunft über die Regulierung geben. Die Benachrichtigungsfunktion wird durch das Modul *PolicyNotifier* umgesetzt. Die ICMP-Nachricht referenziert auf das regulierte Paket (Anforderung A.5.1) und enthält einen Informationstext. Der Informationstext beschreibt, ob ein Paket blockiert oder umgeleitet wurde (Anforderung A.5.2) und referenziert auf die URI der angewendeten InFO-Regel. Eine Verbindung zum Regulierungshintergrund ist damit nur teilweise vorhanden (Anforderung A.5.3). Es fehlt eine Verknüpfung der URI mit Gesetzestexten etc., welche beispielsweise über einen Online-Dienst umgesetzt werden kann.

Das entwickelte System verwendet die geforderte Admin-Agent-Architektur (Anforderung A.6). Die Vorverarbeitung der Policys findet durch die Software *PolicyAdmin* auf einem Admin-Knoten statt (Anforderung A.6.1). Die erzeugten Firewallregeln werden auf einem Agent-Knoten durch eine Firewall angewendet (Anforderung A.6.2). Für die Konfiguration der Firewall auf dem Agent-Knoten ist die Software *PolicyAgent* zuständig.

Die Firewallregeln werden als Konfigurationsskript vom PolicyAdmin zum PolicyAgent übertragen. Durch eine SSL-Verbindung wird die Vertraulichkeit (Anforderung A.7.1) und Integrität (Anforderung A.7.2) der Verbindung sichergestellt. Zur Herstellung der SSL-Verbindung erfolgt eine gegenseitige Authentifizierung von PolicyAdmin und PolicyAgent.

Das konzipierte System wurde für einen Firewall-Router praktisch umgesetzt. Als Firewall-Router wird ein *Software-Router* genutzt, bestehend aus einem *Debian*-Betriebssystem und *iptables* als Firewallkonfigurationsprogramm. Zur Benachrichtigung von betroffenen Parteien werden die regulierten Pakete mit Hilfe des Logging-Daemon *ulogd* aufgezeichnet und in einer *Sqlite*-Datenbank zwischengespeichert. Die Funktion des Systems wurde mit *VirtualBox* in einer virtuellen Netzwerkumgebung exemplarisch getestet.

5.3 Ausblick

Das entwickelte System ist noch verbesserungsfähig und erweiterbar. Wie im vorherigen Abschnitt 5.2 beschrieben, enthält der Informationstext der Benachrichtigung die URI der InFO-Regel. Diese könnte durch eine URL ersetzt werden, die auf eine individuelle Informationsseite verweist. Hier wäre beispielsweise ein Online-Dienst vorstellbar, der die Hinterlegung von Informationstexten für InFO-Regeln ermöglicht. Die InFO-Regel kann wie in dieser Arbeit über die URI oder einen zugehörigen Hash identifiziert werden. Die Hinterlegung der Informationstexte könnte schon bei Erzeugung der Regeln erfolgen und beispielsweise als Funktion in den vorhandenen InFO-Editor [Tis13] integriert werden.

In der Praxis werden ICMP-Nachrichten aus Sicherheitsgründen in Netzwerken häufig geblockt [SFSG09]. Es kann somit hier nicht sichergestellt werden, dass die Nachrichten ihren Empfänger auch erreichen. Dieses Problem konnte im Rahmen dieser Arbeit nicht untersucht werden. Eine Untersuchung wäre jedoch hilfreich, um zu klären, ob die Verwendung von ICMP-Nachrichten eine praxistaugliche Lösung ist, oder ob hier eigene Protokolle geeigneter sind. Dennoch stellt aus der momentanen Sicht ICMP in seiner Eigenschaft als verbreitetes Diagnose-Protokoll [Pos81a] die bessere Alternative dar.

Der mitgelieferte Informationstext der ICMP-Nachricht wird durch die vorhandenen Netzwerkanwendungen beim betroffenen Nutzer nicht angezeigt. Aktuell sind hierzu Packet-Sniffer wie tcpdump notwendig. Nutzer-Anwendungen

wie Browser zeigen hier nur allgemeine Informationstexte, basierend auf der ICMP-Fehlerkategorie. Hier wäre die Entwicklung einer Browsererweiterung denkbar, die auch den eingebetteten Informationstext interpretiert und dem Nutzer dezidierte Informationen zur Regulierung anzeigt. Eine solche Lösung könnte mit dem oben beschriebenen Online-Dienst kombiniert werden.

Mit VirtualBox wurde in dieser Arbeit eine einfache und anschauliche Möglichkeit zur Netzwerksimulation aufgezeigt. Die hier beschriebene Simulation ist weiter ausbaufähig. So könnte sie um andere InFO-Implementierungen (DNS-Server, Suchmaschine, Proxy-Server) erweitert werden. Die verschiedenen Regulierungskomponenten könnte man u.a. auch auf ihre Wechselwirkung testen.

Die bereits entwickelten Regulierungskomponenten könnten zusammen mit dem InFO-Editor zu einer Software-Suite zusammengefasst werden. Ziel einer solchen Software-Suite könnte es sein, dem Nutzer das Erstellen von Regulierungsmaßnahmen unabhängig vom umsetzenden System zu ermöglichen. Die Erstellung der Domänen-spezifischen Regeln und deren Umsetzung würde dabei automatisiert durch die Software-Suite respektive durch deren Module vorgenommen.

Anhang A

Installation & Konfiguration

In diesem Kapitel wird die Einrichtung des Systems erläutert. In Abschnitt A.1 werden die Systemvoraussetzungen beschrieben. Abschnitt A.2 führt durch die Schlüsselerstellung und -verteilung mit Keytool. In Abschnitt A.3 wird die Einrichtung des PolicyAdmins beschrieben. Die Einrichtung des PolicyAgents beschreibt Abschnitt A.4. Die hier beschriebene Anleitung wurde auf einem Debian-Betriebssystem in der Version 6.0.8 mit Hilfe der Virtualisierungssoftware VirtualBox vollzogen.

A.1 Systemvoraussetzungen

PolicyAdmin und PolicyAgent benötigen ein Java Runtime Environment (JRE) in Version 6 oder höher. Die hier verwendete Debian-Version 6.0.8 benutzt die freie Implementierung OpenJDK[1] in der Version 6 als Standard-JRE. Sie kann über den in Listing A.1 angegebenen Bash-Befehl installiert werden.

```
apt-get install default-jre
```

Listing A.1 Installation der Standard-JRE unter Debian.

Der PolicyAgent benötigt zum Betrieb zusätzliche Software. Dazu zählen der Logging-Daemon ulogd, die Datenbank-Software SQLite und das zur Erstellung der Benachrichtigungen benötigte Paket *libpcap*[2]. Die zugehörigen Pakete können mit dem in Listing A.2 dargestellten Bash-Befehl installiert werden.

[1] http://openjdk.java.net/ - Abruf: 27.02.2014
[2] http://packages.debian.org/wheezy/libpcap0.8 - Abruf: 01.02.2014

```
apt-get install sqlite3 sqlite3-doc ulogd ulogd-sqlite3 libcap
```

Listing A.2 Installation der Drittsoftware für den PolicyAgent.

A.2 Erzeugen und Verteilen der Schlüssel

Zur verschlüsselten Übertragung des Konfigurationsskripts vom PolicyAdmin zum PolicyAgent müssen beide Komponenten über je ein Schlüsselpaar verfügen. Ein Schlüsselpaar besteht aus einem öffentlichen Schlüssel und einem geheimen privaten Schlüssel. Der öffentliche Schlüssel wird an andere Parteien verteilt, mit denen kommuniziert werden soll. Er dient den anderen Parteien zum Verifizieren von Signaturen des Schlüsselinhabers und zum Verschlüsseln von Nachrichten, die an den Schlüsselinhaber gesendet werden. Der private Schlüssel verbleibt beim Schlüsselinhaber. Er dient zum Signieren von Nachrichten und zum Entschlüsseln von Nachrichten, die mit dem öffentlichen Schlüssel verschlüsselt wurden.

Für die Schlüsselererzeugung für PolicyAdmin und PolicyAgent wird das im Java Runtime Environment (JRE) enthaltene Programm *keytool*[3] genutzt. Es wird hier exemplarisch die Schlüsselerzeugung und -verteilung beschrieben für den PolicyAdmin. Für den PolicyAgent funktioniert sie entsprechend. Das Programm keytool befindet sich im Unterordner *bin* des JRE und kann über die Kommandozeile aufgerufen werden. Listing A.3 zeigt den Befehl zur Erzeugung eines Schlüsselpaars mit entsprechenden Parametern.

```
keytool -genkeypair -alias PolicyAdmin -dname "cn=PolicyAdmin"
        -keyalg RSA -keysize 3072 -sigalg SHA256withRSA
        -keypass adminkeypassword -validity 180
        -keystore adminkeystore -storepass adminstorepassword
```

Listing A.3 Erzeugung eines Schlüsselpaars mit keytool.

Mit diesem Befehl erzeugt keytool ein Schlüsselpaar für den Alias `PolicyAdmin` und speichert ihn im Keystore `adminkeystore`. Der Keystore ist eine Datei zur gesicherten Aufbewahrung der eigenen Schlüssel. Falls kein Keystore mit

[3] http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html - Abruf: 23.02.2014

dem angegebenen Dateinamen existiert, wird einer erzeugt. Der Schlüssel wird mit dem Algorithmus RSA erzeugt und besitzt eine Schlüssellänge von 3072 Bits. Als Signaturalgorithmus wird SHA-256 ausgewählt. Der Schlüssel ist 180 Tage gültig. Die Passwörter für Schlüssel und Keystore sind hier im Klartext angegeben. Werden sie nicht beim Aufruf übergeben, fordert keytool den Nutzer auf, sie über die Eingabeaufforderung einzugeben.

Der öffentliche Schlüssel für den PolicyAdmin kann anschließend exportiert werden. Dazu weist man keytool an, ein Zertifikat für den entsprechenden Alias zu exportieren. Listing A.4 zeigt den benötigten Befehl.

```
keytool -exportcert -alias PolicyAdmin -file policyadmin.cer
        -keystore adminkeystore
```

Listing A.4 Export eines Zertifikats mit keytool.

Nach Eingabe des Passworts für den Keystore wird ein Zertifikat mit dem öffentlichen Schlüssel in der Datei `policyadmin.cer` gespeichert. Das Zertifikat kann dann in den Keystore des PolicyAgents (`agentkeystore`) importiert werden. Den entsprechenden Befehl zeigt Listing A.5.

```
keytool -importcert -alias PolicyAdmin -file policyadmin.cer
        -keystore agentkeystore
```

Listing A.5 Import eines Zertifikats mit keytool.

Es erfolgt eine Abfrage von keytool, ob diesem Zertifikat vertraut wird. Nach der Bestätigung ist das Zertifikat und damit der öffentliche Schlüssel des PolicyAdmins im Keystore des PolicyAgents vorhanden. Für den PolicyAgent muss auf gleichem Weg ein Schlüsselpaar erzeugt werden und ein Zertifikat dessen in den Keystore des PolicyAdmin importiert werden. Für beide Komponenten wird in der jeweiligen Konfigurationsdatei der Pfad zur eigenen Keystore-Datei benannt. Ist dies geschehen, können die privaten und öffentlichen Schlüssel zur vertraulichen und integrem Datenaustausch verwendet werden.

A.3 Einrichten des PolicyAdmins

Zur Einrichtung des PolicyAdmins auf dem Admin-Knoten muss die JAR-Datei `PolicyAdmin.jar`, die zugehörige Konfigurationsdatei `policyadmin.properties.xml` und der Keystore des PolicyAdmins (siehe Abschnitt A.2) in ei-

nen beliebigen Ordner auf dem Rechner kopiert werden. Die Konfigurationsdatei enthält die UID des PolicyAdmin und Angaben zum Keystore. Diese Angaben sind gegebenenfalls anzupassen. Weiter sind für die verwalteten PolicyAgents jeweils die UID, Socket-Adresse und der zu verwendende Schlüssel-Alias angegeben. Die Schlüssel der einzelnen Einträge sind in dem Format `agent.«UID».«Schlüsselname»` verfasst. Auch diese Angaben sind auf die jeweiligen Erfordernisse anzupassen.

Der PolicyAdmin kann über den in Listing A.6 dargestellten Bash-Befehl ausgeführt werden. Hinter dem Parameter `-i` werden die zu übersetzenden Policy-Dateien übergeben. Der Parameter `-a` bestimmt die UIDs der PolicyAgents, an die das spätere Skript übertragen werden soll. Entfällt dieser Parameter überträgt der PolicyAdmin das Skript an alle in der Konfigurationsdatei gelisteten PolicyAgents. Der Parameter `-v` bestimmt das zu verwendende FirewallVocabulary zur Übersetzung der Policys. Entfällt dieser Parameter wird als Standardwert `iptables` benutzt.

```
java -jar  PolicyAdmin.jar -i MetaPolicy-1.owl Policy-1.owl
     -a 1234 -v iptables
```

Listing A.6 Ausführungsbefehl für den PolicyAdmin.

A.4 Einrichten des PolicyAgents

Zur Einrichtung des PolicyAgents auf dem Admin-Knoten muss analog die JAR-Datei `PolicyAgent.jar`, die zugehörige Konfigurationsdatei `policyagent.properties.xml`, der Keystore des PolicyAgents (siehe Abschnitt A.2) und die Datei mit den Regel-Informationstexten `inforulemessages.properties.xml` in einen beliebigen Ordner auf dem Rechner kopiert werden. Die Konfigurationsdatei des PolicyAgent enthält die UID des PolicyAgent und Angaben zum Keystore. Für den berechtigten PolicyAdmin enthält die Datei die UID und den zu verwendenden Schlüssel-Alias. Weiter enthält die Datei Pfadangaben und Konfigurationsbefehle zur verwendeten Datenbank, der Firewall, der Informationstextdatei und zur temporären Speicherung von empfangenen Dateien. Auch diese Angaben sind auf die jeweiligen Erfordernisse anzupassen.

Die SQLite-Datenbank kann unter Verwendung des in Listing B.1 dokumentierten Datenbankskript eingerichtet werden. Ein entsprechender Bash-Befehl zur Einrichtung ist in Listing A.7 beschrieben.

```
sqlite3 -init ulog.sql ulog.db .quit
```

Listing A.7 Bash-Befehl zur Einrichtung der SQLite-Datenbank.

Zur Einrichtung des Logging-Daemon ulogd muss die Datei `/etc/ulogd.conf` angepasst werden. Dazu kann etwa die unter Listing B.2 dokumentierte Konfigurationsdatei verwendet und auf die lokalen Gegebenheiten angepasst werden. Mit dem Bash-Befehl in Listing A.8 wird ulogd als Hintergrunddienst gestartet.

```
ulogd -d
```

Listing A.8 Bash-Befehl zum Starten von ulogd als Hintergrunddienst.

Vor dem Start des PolicyAgents wird die Firewall mit iptables in einer „offenen“ Konfiguration mit Network Address Translation (NAT) und Forwarding aller Pakete gestartet. Dazu kann das in Listing B.2 angegebene Bash-Skript verwendet werden. Die Bezeichner der Netzwerkschnittstellen sind dabei gegebenenfalls anzupassen. Anschließend kann der PolicyAgent mit dem in Listing A.9 angegebenen Bash-Befehl gestartet werden.

```
java -jar PolicyAgent.jar
```

Listing A.9 Ausführungsbefehl für den PolicyAdmin.

Anhang B

Dokumente

In diesem Kapitel werden zusätzliche Dokumente und Artefakte aufgeführt.

B.1 Klassendiagramme

In diesem Abschnitt werden die in Abschnitt 3.5 eingeführten Klassen und Schnittstellen der Komponenten PolicyAdmin und PolicyAgent im Detail erläutert. Zunächst werden die werden die Klassen des PolicyAdmins vorgestellt, danach folgen die Klassen des PolicyAgents.

B.1.1 PolicyAdmin

Die Klasse `PolicyAdmin` bündelt die restlichen Klassen der Komponente PolicyAdmin. Eine Übersicht bietet das Klassendiagramm B.1. `PolicyAdmin` selbst ist Teil einer `Main`-Klasse. Die Konfiguration des PolicyAdmin ist in der Konfigurationsdatei `policyadmin.properties.xml` abgelegt. Sie enthält Informationen zum PolicyAdmin und den verwalteten PolicyAgents sowie zu den privaten und öffentlichen Schlüsseln für die Skript-Übertragung. Die Angaben zum PolicyAdmin selbst sind in der Klasse `AdminDescriptor` beschrieben und im Attribut `adminDescriptor` abgelegt. Analog dazu beschreibt die Klasse `Agent-Descriptor` einen PolicyAgent. Die verwalteten PolicyAgents werden von der Klasse `PolicyAdmin` in der HashMap-Datenstruktur `agentDescriptorHash-Map` geführt, die über die UID der PolicyAgents indexiert ist. Instanzen der Klas-

sen `PolicySender` und `FirewallTranslator` enthalten die Attribute `policysender` bzw. `firewalltranlator`.

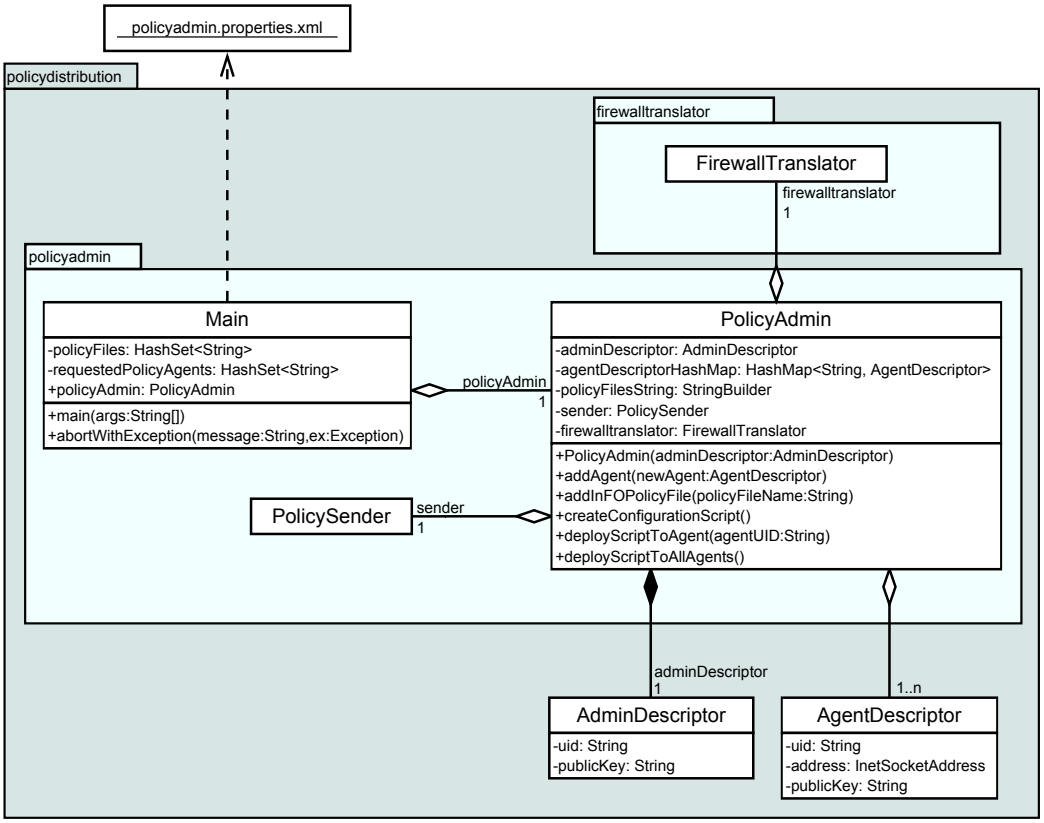


Abbildung B.1 Klassendiagramm des PolicyAdmins. `PolicyAdmin` kapselt Klassen und Funktionalität der Komponente PolicyAdmin. `PolicyAdmin` ist Teil einer `Main`-Klasse. `AdminDescriptor` und `AgentDescriptor` enthalten Informationen zum PolicyAdmin bzw. den verwalteten PolicyAgents.

B.1.2 RouterParser

Der RouterParser ist ein Modul zur Überführung von OWL-Dateien in Objekte. Abbildung B.2 zeigt ein Klassendiagramm für dieses Modul. Umgesetzt wird die Funktionalität durch die Klasse `TechnicalRouterParser`. `TechnicalRouterParser` wird abgeleitet von der in [BGSS14] beschriebenen Klasse `TechnicalParser`. `TechnicalParser` ist für die Übersetzung der grundlegenden Entitäten der InFO zuständig, während `TechnicalRouterParser` die in der RFCO beschriebenen Erweiterungen abdeckt. Die für die RFCO spezifischen Ob-

jekt-Klassen sind im Unterpaket `model.technical.rule` definiert. Die aus den OWL-Dateien eingelesenen Entitäten werden in einem Triplestore zwischengespeichert.

B.1.3 FirewallTranslator

Das Modul FirewallTranslator ermöglicht die Übersetzung der RFCO-Objekte (siehe vorherigen Abschnitt B.1.2) in ein Firewallskript. In einem Zwischenschritt werden dazu Wrapper-Objekte der Klassen `MetaPolicy`, `Policy`, `Rule` und `Algorithm` genutzt. `MetaPolicy` kapselt als Wurzel-Klasse alle übrigen Klassen. Eine Übersicht der Pakete und Klassen bietet Abbildung B.3. `Rule` und `Algorithm` stellen abstrakte Klassen dar. Die konkreten Rules und Algorithms der RFCO sind in den Unterpaketen `rules` und `algorithms` definiert. Die Erzeugung der Objekte der beiden abstrakten Klassen übernehmen die Factory-Klassen `RuleFactory` respektive `AlgorithmFactory`. Die Klasse `ConflictSolver` kapselt die Anwendung der Konfliktlösungsalgorithmen. Die Übersetzung der Wrapper-Objekte ist durch die Schnittstelle `FirewallVocabulary` beschrieben. Eine Referenz zu dieser Schnittstelle hält `FirewallTranslator` im Attribut `vocabulary`.

B.1.4 PolicySender

Die Klasse `PolicySender` kapselt die Funktionalität für die Übertragung des Skripts zum PolicyAgent (siehe Abbildung B.4). Das Übertragungsprotokoll ist nach dem *State Pattern* realisiert. Im State Pattern ist das Verhalten eines Objekts abhängig von seinem Zustand. Das Objekt bildet den *context* und hält eine Referenz auf eine Schnittstelle als *state*. Das Verhalten wird in den implementierenden Klassen der Schnittstelle *state* realisiert. Hier bildet für das Übertragungsprotokoll die Klasse `SenderProtocol` den *context*. Sie hält das Attribut `state`, das vom Typ `StateLike` ist. `StateLike` ist eine Schnittstellenklasse. Die implementierenden Klassen der Schnittstelle `StateLike` korrespondieren mit den in Abschnitt 3.5.3 beschriebenen Zuständen des Übertragungsprotokolls.

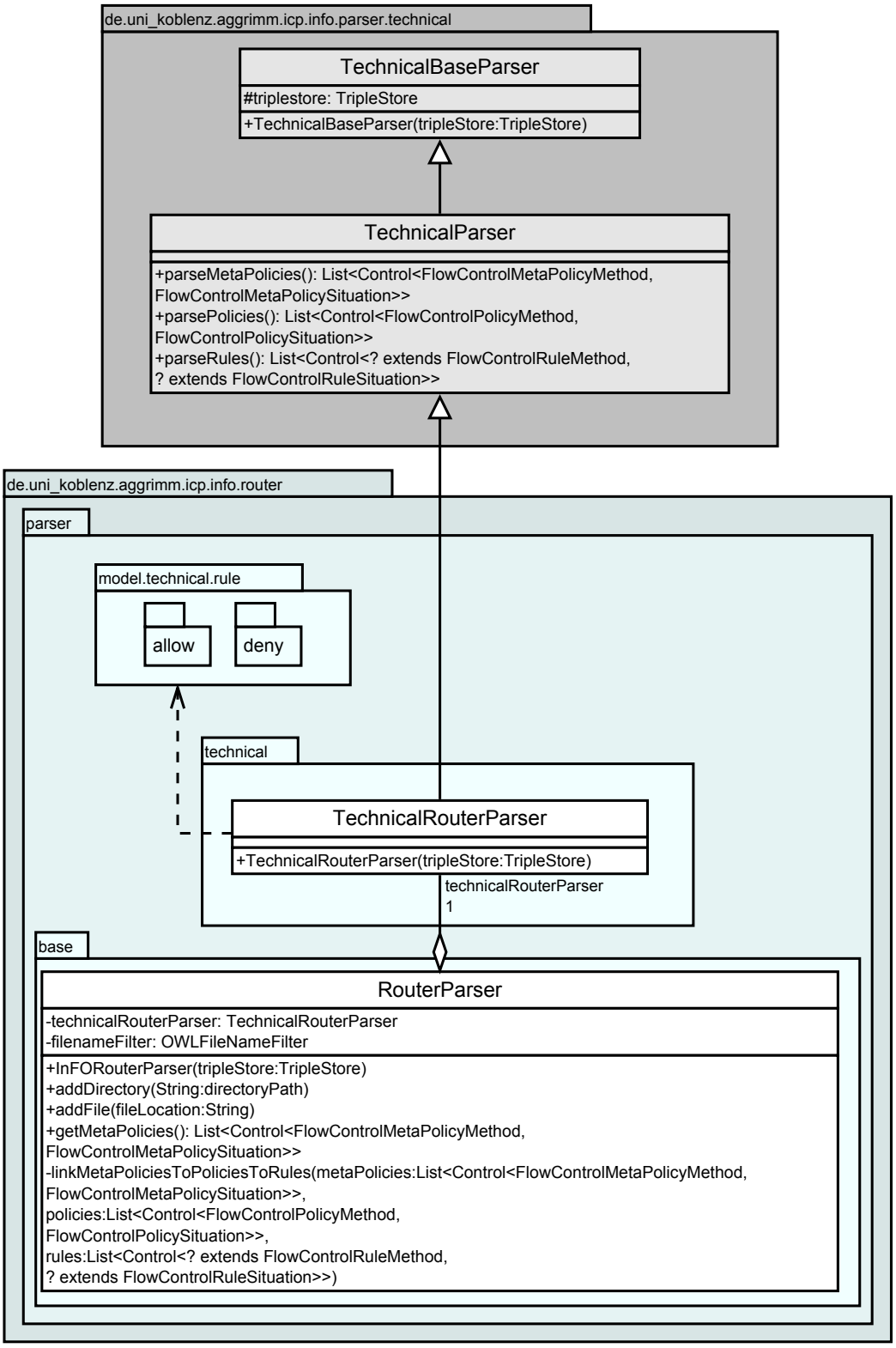


Abbildung B.2 Klassendiagramm des RouterParsers. Der RouterParser nutzt den TechnicalRouterParser, der eine Erweiterung des TechnicalParser ist. Die spezifischen Entitäten der RFCO sind im Unterpaket *model.technical.rule* als Klassen definiert.

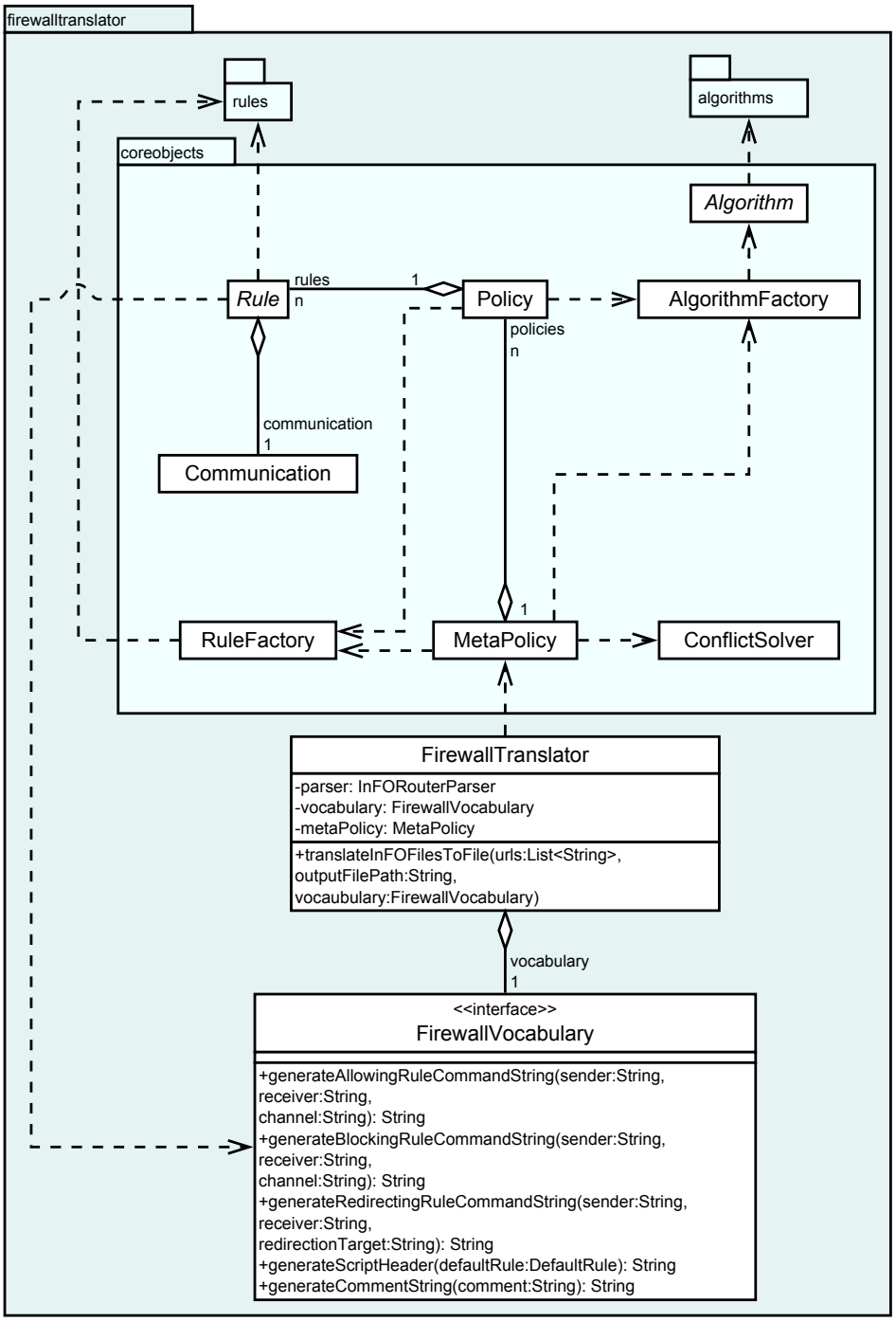


Abbildung B.3 Vereinfachtes Klassendiagramm des FirewallTranslators. Der FirewallTranslator nutzt Wrapper-Klassen für MetaPolicy, Policy, Rule und Algorithm. `Rule`- und `Algorithm`-Objekte werden durch die gleichnamigen Factory-Klassen erzeugt. Die Klasse `ConflictSolver` bietet die Funktionalität zur Konfliktlösung zwischen Policys und Rules. Die Schnittstelle `FirewallVocabulary` dient der Erzeugung des Firewallskripts aus den Wrapper-Klassen.

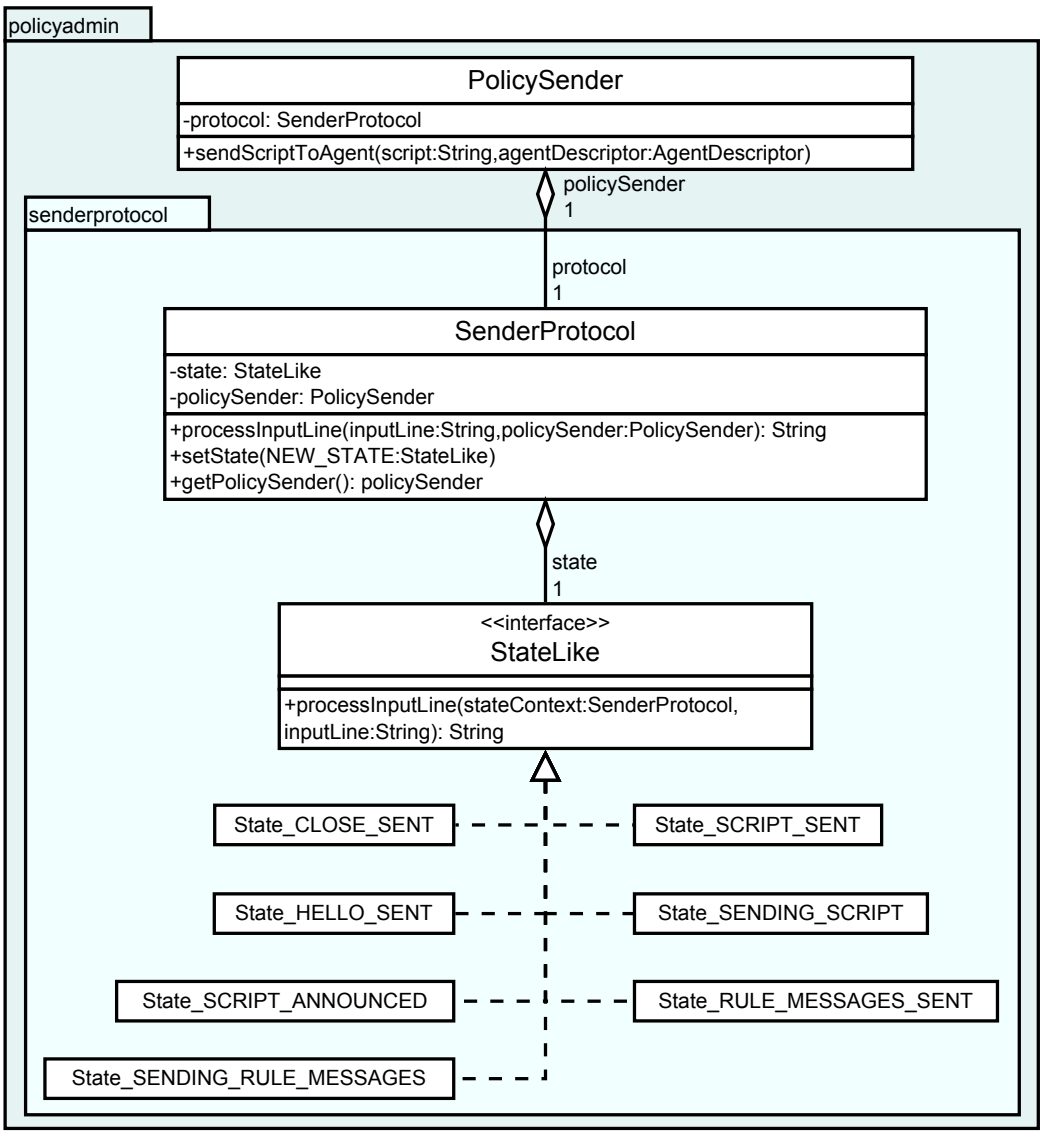


Abbildung B.4 Klassendiagramm für den PolicySender. Die Organisation des Übertragungsprotokolls erfolgt nach dem *State Pattern* [GHJV95]. Die Schnittstelle `StateLike` repräsentiert den aktuellen Zustand des Moduls PolicySender. Das Verhalten wird von den implementierenden Klassen realisiert. Der aktuelle Zustand wird im Attribut `state` der Klasse `SenderProtocol` festgehalten. Es referenziert auf die aktuelle Zustandklasse.

B.1.5 PolicyAgent

Die Klasse `PolicyAgent` kapselt die Klassen der Komponente PolicyAgent. Eine Übersicht über `PolicyAgent` und die verbundenen Klassen bietet Abbildung B.4. Wie die Klasse `PolicyAdmin` (siehe Anhang B.1.1) wird `PolicyAgent` über eine `Main`-Klasse initialisiert. Die Datei `policyagent.properties.xml` enthält die Konfiguration für die Komponente PolicyAdmin. Inhalt der Datei sind folgende Informationen:

- Einstellungen des Server-Dienstes
- Konfigurationsbefehle für die Firewall
- Verbindungsinformationen für die Log-Datenbank
- Informationen über die öffentlichen und privaten Schlüssel
- Dateipfadangaben für die temporäre Sicherung der Konfiguration

Im übergeordneten Paket `policydistribution` sind die mit `PolicyAdmin` gemeinsam genutzten Klassen `AgentDescriptor` und `AdminDescriptor` enthalten. Diese Klassen enthalten zur Laufzeit die Informationen zum PolicyAgent und dem PolicyAdmin (vergleiche Anhang B.1.1). `PolicyAgent` hält in den Attributen `policyNotifier`, `policyReceiver` und `policyUpdater` Referenzen zu Objekten der entsprechenden Klassen.

B.1.6 PolicyReceiver

Die Klasse `PolicyReceiver` kapselt die Funktionalität für den Empfang eines Skripts (siehe Abbildung B.6). Die Komposition der einzelnen Klassen orientiert sich wie bei der Klasse `PolicySender` am *State Pattern* (siehe Abschnitt B.1.5 und [GHJV95]). Das *State Pattern* wird für das Modul PolicyReceiver durch die Klassen der Unterpakets `receiverprotocol` realisiert. Das boolesche Attribut `abortRequested` hält zusätzlich fest, ob ein Aufruf zur Unterbrechung der Ausführung durch den PolicyAgent erfolgt ist.

B.1.7 PolicyNotifier

Die Klasse `PolicyNotifier` kapselt die Funktionalität zur Benachrichtigung der Absender von regulierten Paketen. Abbildung B.7 bietet eine Übersicht des

Moduls. `PolicyNotifier` hält im Attribut `ruleMessageFile` eine Referenz auf die Datei mit den Informationstexten zu den angewendeten Firewallregeln. Das boolesche Attribut `canceled` signalisiert, ob von `PolicyAgent` eine Unterbrechung der Ausführung verlangt wurde. Die durch die Firewall regulierten Pakete sind in einer Log-Datenbank abgelegt. Das Auslesen der Log-Datenbank wird durch die Klasse `DatabaseReader` des gleichnamigen Unterpakets bewerkstelligt. Die aus der Datenbank ausgelesenen Paketinformationen werden in Objekten der Klasse `RegulatedPacket` gespeichert. Das Attribut `rulePrefix` der Klasse `RegulatedPacket` wird zur Zuordnung des Informationstextes zum regulierten Paket verwendet.

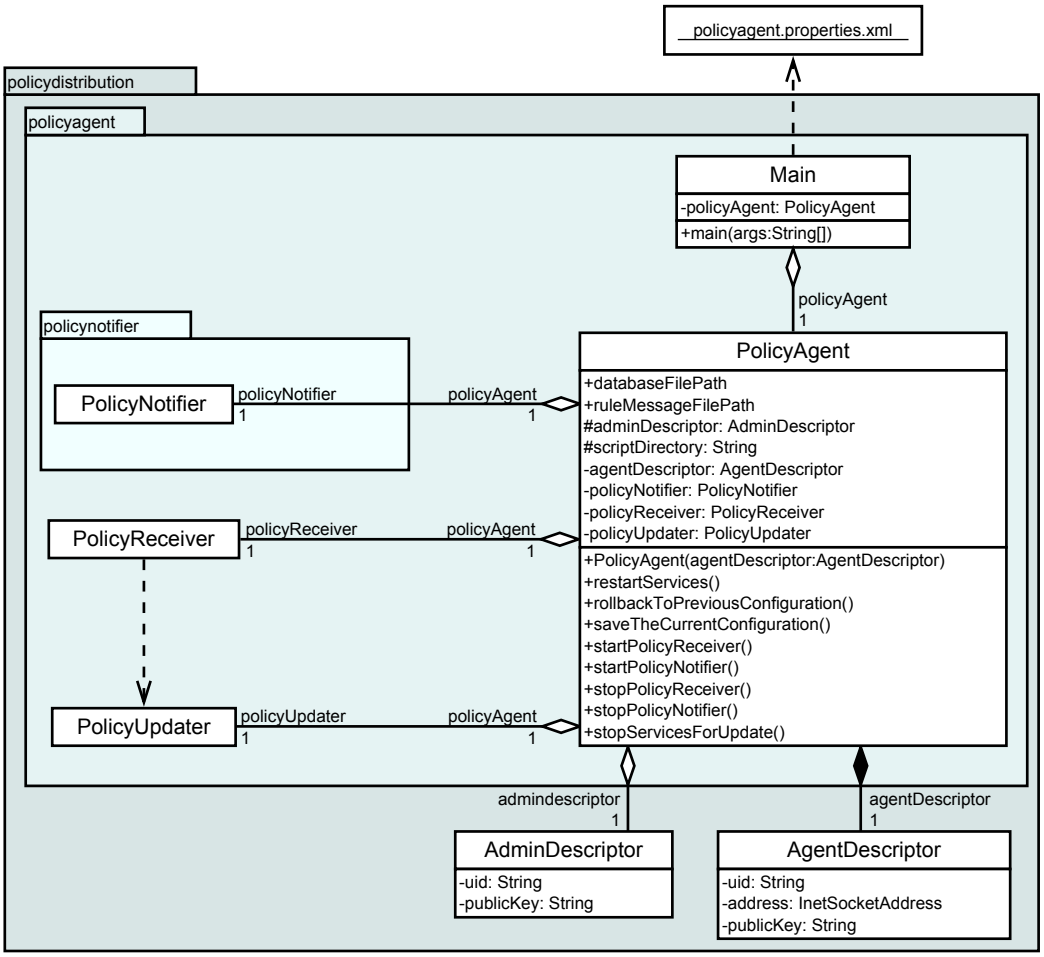


Abbildung B.5 Klassendiagramm des PolicyAgents. `PolicyAgent` kapselt Klassen und Funktionalität der Komponente PolicyAgent. `PolicyAgent` ist Teil einer `Main`-Klasse. Referenzen auf Objekte der Klassen `PolicyNotifier`, `PolicyReceiver` und `PolicyUpdater` sind in den gleichnamigen Attributen enthalten. `AgentDescriptor` und `AdminDescriptor` enthalten Informationen zum PolicyAgent und PolicyAdmin.

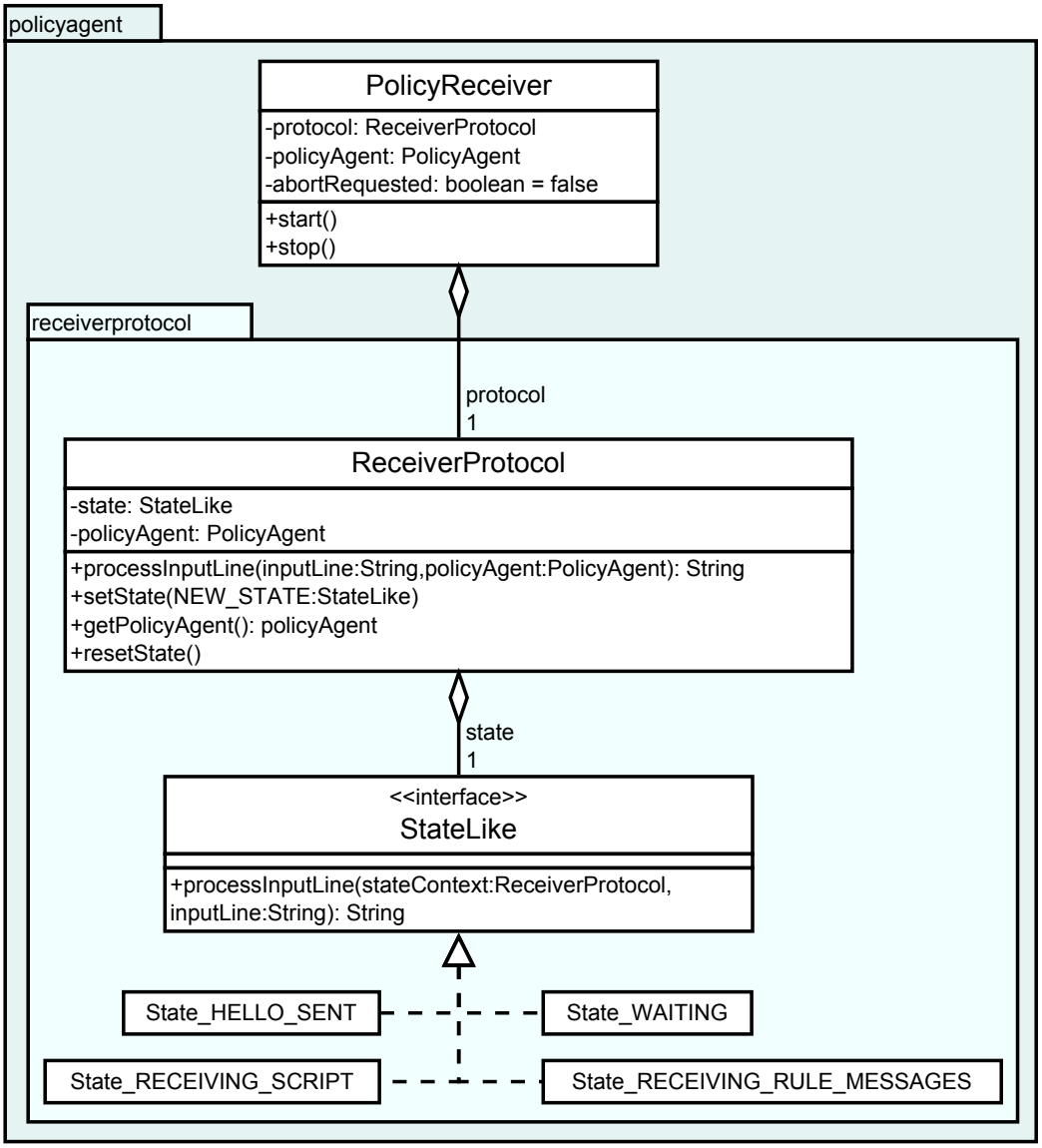


Abbildung B.6 Klassendiagramm für den PolicyReceiver. Analog zum PolicySender (siehe Anhang B.1.4) wird auch beim PolicyReceiver das *State Pattern* umgesetzt. Die Schnittstelle `StateLike` repräsentiert den aktuellen Zustand des Moduls PolicyReceiver. Das Verhalten wird von den implementierenden Klassen realisiert. Der aktuelle Zustand wird im Attribut `state` der Klasse `ReceiverProtocol` festgehalten. Es referenziert auf die aktuelle Zustandklasse.

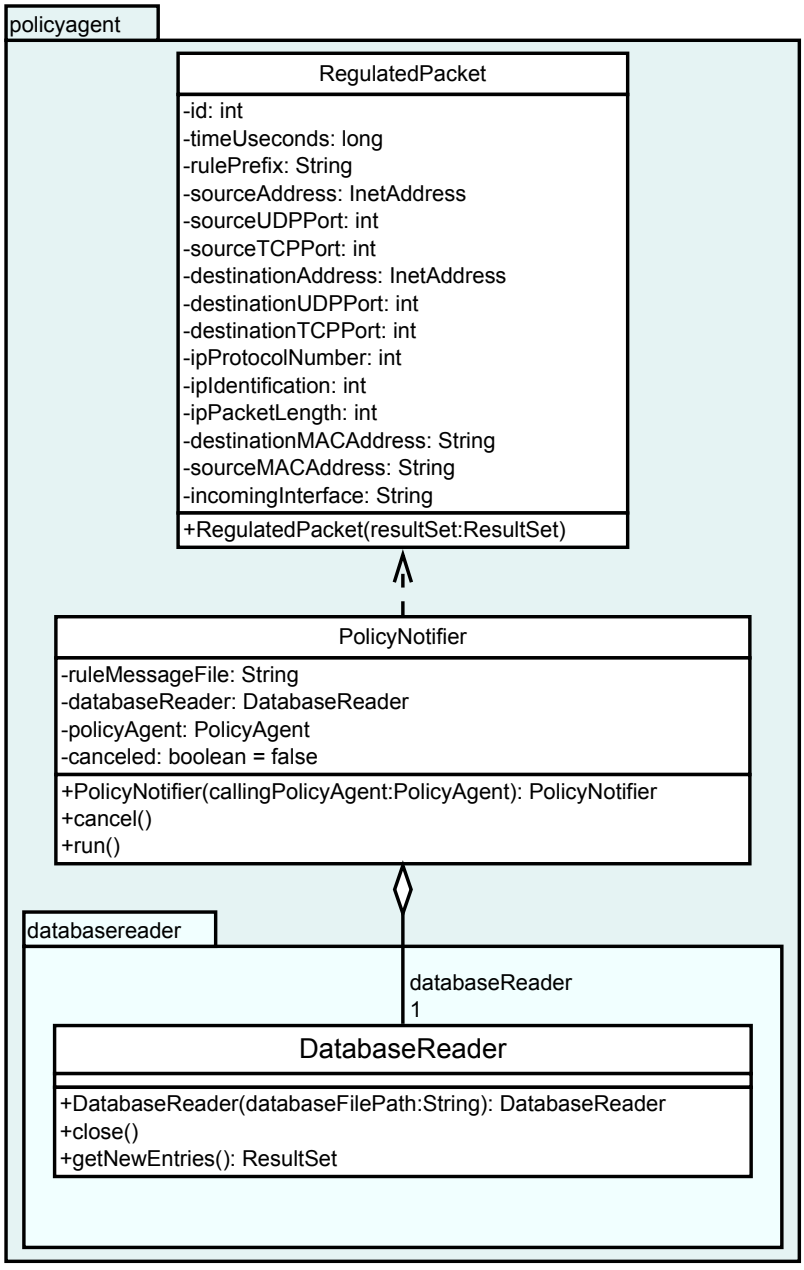


Abbildung B.7 Klassendiagramm des PolicyNotifiers. Die `PolicyNotifier`-Klasse kapselt die Funktionalität zum Senden der Regulierungsbenachrichtigungen. `DatabaseReader` dient zum Auslesen der Paketinformationen der Log-Datenbank. Die Klasse `RegulatedPacket` kapselt alle Informationen über ein reguliertes Paket. Die Informationstexte der Benachrichtigungen sind in einer Datei hinterlegt Sie wird über das Attribut `ruleMessageFile` der Klasse `PolicyNotifier` referenziert.

B.2 Datenbankskript

Listing B.1 enthält das SQL-Skript zur Erstellung des Datenbankschemas.

```
BEGIN TRANSACTION;
CREATE TABLE ulog(
id INTEGER PRIMARY KEY AUTOINCREMENT,
oob_time_sec INT UNSIGNED,
oob_time_usec INT UNSIGNED,
oob_prefix TEXT,
oob_in TEXT,
raw_mac TEXT,
ip_protocol INT UNSIGNED,
ip_id INT UNSIGNED,
ip_totlen INT UNSIGNED,
ip_saddr INT UNSIGNED,
ip_daddr INT UNSIGNED,
tcp_sport INT UNSIGNED,
tcp_dport INT UNSIGNED,
udp_sport INT UNSIGNED,
udp_dport INT UNSIGNED);
COMMIT;
```

Listing B.1 SQL-Skript zur Erstellung des Datenbankschemas. Das Skript erstellt die Tabelle *ulog* mit den in Tabelle 4.3 beschriebenen Spalten.

B.3 Bash-Skript

In Listing B.2 wird ein Bash-Skript gezeigt, mit dem ein offenes Internet-Gateway konfiguriert wird.

```
#!/bin/sh
#
# Dieses Skript erlaubt die Weiterleitung
# aller eingehenden Pakete

# Variable für die ausgehende Netzwerkschnittstelle.
EXTDEV="eth0"

# Variablen für Pfadangaben.
```

```
ECHO="/bin/echo"
SYSCTL="/sbin/sysctl"
IPTABLES="/sbin/iptables"

# Standardmäßig werden alle Pakete durchgelassen.
$IPTABLES -P INPUT ACCEPT
$IPTABLES -P OUTPUT ACCEPT
$IPTABLES -P FORWARD ACCEPT

# Alle Regeln in den Tabellen filter und nat werden entfernt.
$IPTABLES -t filter -F
$IPTABLES -t nat -F

# Maskierung aller ausgehenden Pakete (NAT)
$IPTABLES -t nat -A POSTROUTING -o $EXTDEV -j MASQUERADE

# Das Weiterleiten von Paketen in den Systemeinstellungen
# aktivieren.
$SYSCTL -w net.ipv4.ip_forward=1

# Alternativer Befehl
# zur Aktivierung der Paketweiterleitung.
# $ECHO "1" > /proc/sys/net/ipv4/ip_forward
```

Listing B.2 Bash-Skript für iptables. Das Skript bewirkt eine offene Konfiguration der Firewall. Alle Pakete werden empfangen bzw. weitergeleitet.

B.4 Konfigurationsdatei für den Logging-Daemon

Listing B.3 zeigt die verwendete Konfigurationsdatei für den Daemon *ulogd* (siehe Abschnitt 4.3.2).

```
# Konfiguration für ulogd

# Globale Einstellungen.
[global]
#
# Netlink Multicast-Gruppe.
nlgroup=1
```

```
#
# Log-Datei und Log-Level für ulogd
logfile="/var/log/ulog/ulogd.log"
loglevel=5
#
# Buffer-Größen
rmem=131071
bufsize=150000

# Plugins
#
# Input Plugins
#
# Interpreter Plugin für IPv4 Header-Informationen
plugin="/usr/lib/ulogd/ulogd_BASE.so"
#
# Output Plugins
#
# Output Plugin für SQLite
plugin="/usr/lib/ulogd/ulogd_SQLITE3.so"

# Plugin-Einstellungen
[SQLITE3]
table="ulog"
db="/home/rainer/ulog.db"
buffer=1
```

Listing B.3 Konfigurationsdatei für ulogd.

Literaturverzeichnis

[And06] ANDREASSON, Oskar: *Iptables Tutorial 1.2.2*. Online. https://www.frozentux.net/iptables-tutorial/iptables-tutorial.html. Version: November 2006, Abruf: 26.01.2014

[Bar01] BARTH, Wolfgang: *Das Firewall Buch*. SuSE Press, 2001

[BFG+05] BIANCO, Andrea ; FINOCHIETTO, Jorge M. ; GALANTE, Giulio ; MELLIA, Marco ; NERI, Fabio: Open-Source PC-Based Software Routers: A Viable Approach to High-Performance Packet Switching. In: *Third Internation Workshop on QoS in Multiservice IP Networks* Bd. 1, Springer-Verlag, Februar 2005, S. 353–366

[BGSS14] BALKE, Alexander ; GORBULSKI, Felix ; SCHENS, Artur ; SCHENS, Erwin: *Projektpraktikum Phi: Implementation eines InFO-konformen Application-Level Proxy-Servers*. 2014

[Bra89] BRADEN, R.: *Requirements for Internet Hosts - Communication Layers*. RFC 1122 (INTERNET STANDARD). http://www.ietf.org/rfc/rfc1122.txt. Version: Oktober 1989, Abruf: 21.02.2014

[Bra97] BRADNER, S.: *Key words for use in RFCs to Indicate Requirement Levels*. RFC 2119 (Best Current Practice). http://www.ietf.org/rfc/rfc2119.txt. Version: März 1997, Abruf: 02.02.2014

[DA99] DIERKS, T. ; ALLEN, C.: *The TLS Protocol Version 1.0*. RFC 2246 (Proposed Standard). http://www.ietf.org/rfc/rfc2246.txt. Version: Januar 1999, Abruf: 25.01.2014

[DPRZ08] DEIBERT, Ronald (Hrsg.) ; PALFREY, John (Hrsg.) ; ROHOZINSKI, Rafal (Hrsg.) ; ZITTRAIN, Jonathan (Hrsg.): *Access Denied: The Practice and Po-*

licy of Global Internet Filtering (Information Revolution and Global Politics). The MIT Press, 2008. – ISBN 0262541963

[DPRZ10] DEIBERT, Ronald (Hrsg.) ; PALFREY, John (Hrsg.) ; ROHOZINSKI, Rafal (Hrsg.) ; ZITTRAIN, Jonathan (Hrsg.): *Access Controlled: The Shaping of Power, Rights, and Rule in Cyberspace (Information Revolution and Global Politics)*. The MIT Press, 2010. – ISBN 0262514354

[Eur13] EUROPEAN NETWORK AND INFORMATION SECURITY AGENCY: *Algorithms, Key Sizes and Parameters Report - 2013 recommendations*. http://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/algorithms-key-sizes-and-parameters-report. Version: Oktober 2013, Abruf: 23.02.2014

[Fre00] FREED, N.: *Behavior of and Requirements for Internet Firewalls*. RFC 2979 (Informational). http://www.ietf.org/rfc/rfc2979.txt. Version: Oktober 2000, Abruf: 17.02.2014

[Fuh00] FUHRBERG, Kai: *Internet- Sicherheit*. 2. Auflage. Carl Hanser Verlag, 2000

[FV08] FARIS, Robert ; VILLENEUVE, Nart: Measuring Global Internet Filtering. Version: 2008. http://access.opennet.net/wp-content/uploads/2011/12/accessdenied-chapter-1.pdf, Abruf: 24.02.2014. In: [DPRZ08], Kapitel 1, 5-27

[GHJV95] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts : Addison-Wesley, 1995. ISSN 0201633612 9780201633610

[Hun02] HUNT, Craig: *TCP/IP Network Administration*. 3. Auflage. O'Reilly Media, 2002. – 748 S.

[Int94] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO/IEC 7498-1: 1994(E): Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*. http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip. Version: November 1994, Abruf: 19.02.2014. – International Standard

[Int13] INTERNET & JURISDICTION PROJECT: *Synthesis 3/2013*. Online. http://www.internetjurisdiction.net/wp-content/uploads/2013/08/Internet-Jurisdiction-SYNTHESIS-3-July-2013.pdf. Version: Juli 2013, Abruf: 24.02.2014

[Kas12a] KASTEN, Andreas: *ICP Wiki: Flow Control Ontology*. Online. http://icp.it-risk.iwvi.uni-koblenz.de/wiki/index.php?title=Flow_Control_Ontology&oldid=125. Version: Oktober 2012, Abruf: 24.10.2013

[Kas12b] KASTEN, Andreas: *ICP Wiki: Router-based Flow Control Ontology*. Online. http://icp.it-risk.iwvi.uni-koblenz.de/mediawiki/index.php?title=Router-based_Flow_Control_Ontology&oldid=57. Version: September 2012, Abruf: 21.02.2014

[KS13] KASTEN, Andreas ; SCHERP, Ansgar: *Ontology-Based Information Flow Control of Network-Level Internet Communication*. November 2013. – Unpublished

[MA08] MURDOCH, Steven J. ; ANDERSON, Ross: Tools and Technology of Internet Filtering. Version: 2008. http://access.opennet.net/wp-content/uploads/2011/12/accessdenied-chapter-3.pdf, Abruf: 24.02.2014. In: [DPRZ08], Kapitel 3, 57-72

[MF99] MARTIN-FLATIN, J. P.: Push vs. pull in Web-based network management. In: *Integrated Network Management, 1999. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management*, 1999, S. 3–18

[Mos12] MOSEN, Dominik: *Erweiterung eines Nameservers zur policy-basierten Internetregulierung*, Universität Koblenz-Landau, Bachelorarbeit, September 2012

[Nat12] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY: *FIPS PUB 180-4: Secure Hash Standard (SHS)*. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf. Version: März 2012, Abruf: 29.01.14

[Ora13] ORACLE CORPORATION: *Oracle VM VirtualBox User Manual*. https://www.virtualbox.org/manual/UserManual.html. Version: 2013, Abruf: 31.01.2014

[PD11] PETERSON, Larry L. ; DAVIE, Bruce S.: *Computer Networks: A Systems Approach*. 5. Auflage. Morgan Kaufmann, 2011. – 920 S.

[Per99] PERLMAN, Radia: *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols*. 2. Auflage. Addison-Wesley, 1999

[Pos80] POSTEL, J.: *User Datagram Protocol*. RFC 768 (INTERNET STANDARD). http://www.ietf.org/rfc/rfc768.txt. Version: August 1980, Abruf: 25.02.2014

[Pos81a] POSTEL, J.: *Internet Control Message Protocol*. RFC 792 (INTERNET STANDARD). http://www.ietf.org/rfc/rfc792.txt. Version: September 1981, Abruf: 24.01.2014

[Pos81b] POSTEL, J.: *Internet Protocol*. RFC 791 (INTERNET STANDARD). http://www.ietf.org/rfc/rfc791.txt. Version: September 1981, Abruf: 21.02.2014

[Pos81c] POSTEL, J.: *Transmission Control Protocol*. RFC 793 (INTERNET STANDARD). http://www.ietf.org/rfc/rfc793.txt. Version: September 1981, Abruf: 25.02.2014

[RFB01] RAMAKRISHNAN, K. ; FLOYD, S. ; BLACK, D.: *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168 (Proposed Standard). http://www.ietf.org/rfc/rfc3168.txt. Version: September 2001, Abruf: 21.02.2014

[RIP08] RIPE NETWORK COORDINATION CENTRE: *YouTube Hijacking: A RIPE NCC RIS case study*. Online. http://www.ripe.net/internet-coordination/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study. Version: Februar 2008, Abruf: 24.02.2014

[Riv92] RIVEST, R.: *The MD5 Message-Digest Algorithm*. RFC 1321 (Informational). http://www.ietf.org/rfc/rfc1321.txt. Version: April 1992, Abruf: 10.02.2014 (Request for Comments)

[RSA78] RIVEST, R. L. ; SHAMIR, A. ; ADLEMAN, L.: A Method for Obtaining Digital Signatures and Public-key Cryptosystems. In: *Communications*

of the ACM 21 (1978), Februar, Nr. 2, 120–126. http://doi.acm.org/10.1145/359340.359342, Abruf: 25.01.2014. – ISSN 0001–0782

[Rus13] RUSTER, Michael: *Polsearchine: Implementierung einer Policy-basierten Suchmaschine zur Internetregulierung*, Universität Koblenz-Landau, Bachelorarbeit, September 2013. http://kola.opus.hbz-nrw.de/volltexte/2013/934/pdf/2013_09_25_pls_hires.pdf, Abruf: 21.02.2014

[SFSG09] SRISURESH, P. ; FORD, B. ; SIVAKUMAR, S. ; GUHA, S.: *NAT Behavioral Requirements for ICMP*. RFC 5508 (Best Current Practice). http://www.ietf.org/rfc/rfc5508.txt. Version: April 2009, Abruf: 26.02.2014

[SH99] SRISURESH, P. ; HOLDREGE, M.: *IP Network Address Translator (NAT) Terminology and Considerations*. RFC 2663 (Informational). http://www.ietf.org/rfc/rfc2663.txt. Version: August 1999, Abruf: 20.02.2014

[Tim03] TIMOFEEVA, Yulia A.: Hate Speech Online: Restricted or Protected? Comparison of Regulations in the United States and Germany. In: *Journal of Transnational Law & Policy* 12 (2003), Frühjahr, Nr. 2, 253-286. http://www.law.fsu.edu/journals/transnational/vol12_2/timofeeva.pdf, Abruf: 25.02.2014

[Tis13] TISSEN, Johann: *Entwicklung und Evaluierung eines Editors zur policy-basierten Internetregulierung*, Universität Koblenz-Landau, Bachelorarbeit, Juli 2013

[TW10] TANENBAUM, Andrew S. ; WETHERALL, David J.: *Computer Networks:*. 5. Auflage, International Edition. Prentice-Hall, 2010

[W3C12] W3C OWL WORKING GROUP: *OWL 2 Web Ontology Language Document Overview (Second Edition)*. http://www.w3.org/TR/owl2-overview/. Version: Dezember 2012, Abruf: 21.02.2014. – W3C Recommendation

[WFLY04] WANG, Xiaoyun ; FENG, Dengguo ; LAI, Xuejia ; YU, Hongbo: *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*. Cryptology ePrint Archive: Report 2004/199. http://eprint.iacr.org/2004/199.pdf. Version: August 2004, Abruf: 10.02.2014

[ZCC00] ZWICKY, Elizabeth D. ; COOPER, Simon ; CHAPMAN, D. B.: *Building Internet Firewalls*. 2. Auflage. O'Reilly Media, 2000. – 896 S.

[ZP08] ZITTRAIN, Jonathan ; PALFREY, John: Internet Filtering: The Politics and Mechanisms of Control. Version: 2008. http://access.opennet.net/wp-content/uploads/2011/12/accessdenied-chapter-2.pdf, Abruf: 24.02.2014. In: [DPRZ08], Kapitel 2, 29-56