Implementierung einer modularen Architektur zur passiven Autokonfiguration mobiler Ad-hoc-Netze (PACMAN) für Linux

Studienarbeit am Institut für Telematik
Prof. Dr. M. Zitterbart
Fakultät für Informatik
Universität Karlsruhe (TH)

von

cand. inform.
Ingmar Baumgart

Betreuer:
Prof. Dr. M. Zitterbart
Dipl.-Ing. K. Weniger

Tag der Anmeldung:	1. Februar 2004
Tag der Abgabe:	30. April 2004

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 30. April 2004

Inhaltsverzeichnis

1. Einleitung

In den letzten Jahren hat die drahtlose Kommunikation stark an Bedeutung gewonnen. Immer häufiger werden mobile Endgeräte wie Laptops, PDAs und Handys mit WLAN oder Bluetooth ausgestattet. Durch diese Entwicklung steigt die Wahrscheinlichkeit, dass mobile Ad-hoc-Netze (sog. MANETs), die in jüngerer Vergangenheit zum Gegenstand intensiver Forschung geworden sind, nun auch im Bereich der Kommunikationselektronik Einzug erhalten.

Mobile Ad-hoc-Netze ermöglichen die Kommunikation zwischen mobilen Endgeräten über Mehr-Hop-Verbindungen ohne auf eine bestehende Infrastruktur zurückgreifen zu müssen. Jedes Endgerät fungiert in diesem Szenario auch als Router und leitet Pakete für andere Knoten weiter. Die Wegewahl erfolgt dabei mittels speziell auf die Eigenschaften solcher Netze zugeschnittener Routing-Algorithmen. Wichtige Grundvoraussetzung für den erfolgreichen Aufbau einer Route sind dabei eindeutige Knotenadressen.

1.1 Zielsetzung der Arbeit

Zielsetzung der Studienarbeit ist die Implementierung und Evaluierung einer modularen Architektur zur Autokonfiguration mobiler Ad-hoc-Netze (PACMAN)[Weni04] für Linux. Unterstützt werden sollen OLSR und FSR aus der Gruppe der proaktive link-state Routingprotokolle.

1.2 Gliederung der Arbeit

Im zweiten Kapitel werden die Grundlagen für diese Studienarbeit erläutert. Dabei wird kurz auf das Thema Autokonfiguration und die Routingprotokolle OLSR und FSR eingegangen.

Im dritten Kapitel wird die Architektur von PACMAN vorgestellt. Ein wichtiger Teil sind hierbei die einzelnen Verfahren zur passiven Erkennung doppelter Adressen (PDAD).

Auf die eigentliche Implementierung wird im vierten Kapitel ausführlich eingegangen. Dort werden auch einzelne Probleme, die während der Implementierungsphase aufgetreten sind, zusammen mit den jeweiligen Lösungsansätzen erörtert.

Abschließend werden im fünften Kapitel noch die Ergebnisse der durchgeführten Funktions- und Leistungsmessungen präsentiert.

2. Grundlagen

Im Folgenden werden die zum Verständnis der Arbeit notwendigen Grundlagen erläutert. Nach der Klärung des Begriffs der „Mobilen Ad-hoc-Netze“ wird kurz auf das Thema Autokonfiguration eingegangen. Abschließend werden zwei in der Arbeit verwendete Ad-hoc Routingprotokolle vorgestellt.

2.1 Mobile Ad-hoc-Netze

Ein mobiles Ad-hoc-Netz (MANET) besteht aus mehreren mobilen, drahtlosen Endgeräten, die ohne das Vorhandensein einer bestehenden Infrastruktur miteinander kommunizieren können. Jedes Endgerät ist in diesem Szenario gleichzeitig auch Router und leitet Pakete für andere Endgeräte, die ihren Kommunikationspartner nicht direkt erreichen können, weiter. Einsatzgebiete für MANETs sind beispielsweise Konferenzen, bei denen die Konferenzteilnehmer untereinander Daten austauschen möchten oder Katastrophengebiete, in denen die komplette Kommunikationsinfrastruktur zerstört wurde.

Da die drahtlose Kommunikation mittels WLAN oder Bluetooth nur eine relativ geringe Datenrate ermöglicht und die Akkukapazitäten der Endgeräte begrenzt sind, muss das Versenden von Signalisierungsdaten auf das Nötigste reduziert werden. Des weiteren ist die drahtlose Kommunikation relativ störanfällig. Daher müssen die verwendeten Protokolle darauf ausgelegt sein, auch bei hohen Paketverlustraten noch zuverlässig zu funktionieren.

Durch die Mobilität der Knoten kommt es bei MANETs zu ständigen Topologieänderungen, die dazu führen können, dass das Netz in verschiedene Partitionen aufgespalten wird. Aus dem selben Grund können auch mehrere Partitionen im Lauf der Zeit zu einem gemeinsamen Netz verschmelzen.

Diese Netzverschmelzungen stellen vor allem bei der Verwendung von Autokonfigurationsverfahren ein Problem dar. Abb. 2.1 zeigt ein Beispiel für eine Netzverschmelzung. In beiden Partitionen wurden vor der Netzverschmelzung durch das Autokonfigurationsverfahren eindeutige[1] Adressen vergeben. Durch die Verschmelzung entsteht jedoch ein gemeinsames Netz mit doppelten Adressen. Daher müssen

[1] innerhalb jeder Partition

Netzverschmelzung durch das Autokonfigurationsverfahren erkannt und die Eindeutigkeit der Adressen erneut geprüft werden.

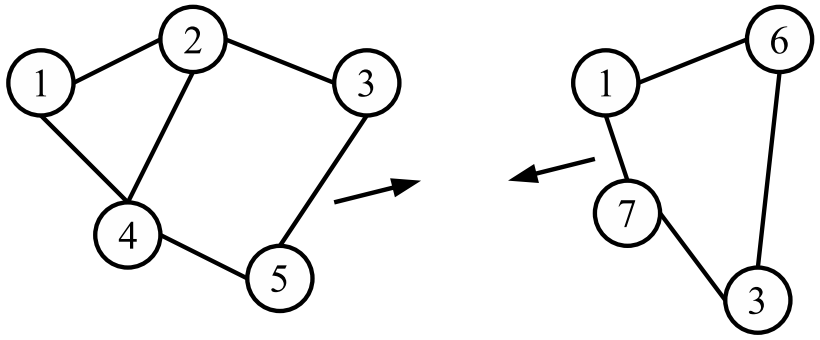

Abbildung 2.1: Problem der Netzverschmelzung bei der Autokonfiguration

Um die spontane Bildung von MANETs möglichst angenehm zu gestalten, sollten diese selbstorganisierend und selbstkonfigurierend sein. Vor allem in größeren Netzen ist eine manuelle Adressvergabe nicht praktikabel. Die Zuweisung einer Adresse muss hierbei ohne die Verwendung zentraler Komponenten erfolgen.

2.2 Autokonfiguration

Um die korrekte Wegewahl in einem Netz zu ermöglichen, muss jeder Knoten eine eindeutige Adresse zugewiesen bekommt. Im Festnetz können zwei Autokonfigurationsansätze unterschieden werden.

Im zustandsbehafteten Fall werden die Adressen von einer zentralen Instanz vergeben. Diese führt eine Liste mit bereits vergebenen Adressen und kann somit eindeutige Adressen garantieren. Ein Beispiel dafür ist das Dynamic Host Configuration Protocol (DHCP) [Drom93]. In MANETs ist dieser Ansatz schwierig zu realisieren, da keine zentralen Komponenten vorhanden sind.

Im zustandslosen Fall wählt jeder Knoten seine Adresse selbst und prüft mittels eines Duplicate Address Detection (DAD) Verfahren die Eindeutigkeit der Adresse. Diese Verfahren wird beispielsweise beim IETF Zeroconf Protokoll [ChAG04] verwendet.

Für MANETs wurden inzwischen mehrere Autokonfigurationsverfahren vorgeschlagen. In [PeRD00] hat Perkins den Zeroconf-Ansatz auf MANETs übertragen: Jeder Knoten flutet das Netz mit einer Nachricht, die seine Adresse enthält. Falls ein Knoten mit der gleichen Adresse eine solche Nachricht empfängt, antwortet er mit einer Unicast-Nachricht. Dieses Verfahren kann jedoch nicht mit Netzpartitionierungen umgehen und verursacht einen hohen Signalisierungsoverhead.

Weak DAD (WDAD) [Vaid02] ist ein weiterer Vorschlag zur Autokonfiguration in MANETs. Hier werden die Routingprotokollnachrichten um einen zufällig gewählten Schlüssel erweitert. Werden von einem Knoten zwei Nachrichten von der gleichen

Adresse mit unterschiedlichen Schlüsseln empfangen, liegt ein Konflikt vor. Allerdings besteht die Möglichkeit, dass zwei Knoten mit der gleichen Adresse zufällig den gleichen Schlüssel wählen und der Konflikt dadurch nicht erkannt werden kann. Außerdem nimmt auch bei diesem Verfahren der Overhead (je nach Schlüssellänge) deutlich zu. Die notwendigen Modifikationen am Routingprotokoll stellen einen weiteren Nachteil dar.

2.3 Routingprotokolle für MANETs

Die in MANETs verwendeten Routingprotokolle können in zwei Klassen aufgeteilt werden: Bei den reaktiven Routingprotokollen wird eine Route nur bei Bedarf, d.h. sobald Datenpakete versendet werden, aufgebaut, während bei proaktiven Protokollen die Routen ständig aktualisiert werden.

Im Folgenden werden nur proaktive Link-State Routingprotokolle behandelt, also Protokolle, bei denen die Knoten periodisch Informationen über den Zustand der Verbindungen zu ihren Nachbarn (sog. Link-States) im kompletten Netz verteilen.

In der Arbeit wurden die Routingprotokolle OLSR und FSR verwendet.

2.3.1 Das Routingprotokoll OLSR

Das Protokoll „Optimised Link State Routing“ (OLSR) [CJLM+01] geht vom klassischen Link-State Routingansatz aus, bei dem die LS[2] im kompletten Netz geflutet werden. Die Weiterleitung der LS-Informationen erfolgt dabei auf Anwendungsschicht durch das Versenden sogenannter TC Nachrichten. Das Auffinden von Nachbarknoten und die Erkennung von Linkbrüchen erfolgt mittels HELLO-Nachrichten. Im Unterschied zu TC-Nachrichten werden diese Nachrichten nicht weitergeleitet. Der komplette Signalisierungsverkehr von OLSR setzt dabei auf UDP auf.

Das wesentliche Konzept von OLSR besteht in der Auswahl sog. Multipoint Relays (MPRs). MPRs sind bestimmte Knoten, die für die Weiterleitung der Link-State Nachrichten zuständig sind. Jeder Knoten wählt[3] anhand der durch HELLO Nachrichten verbreiteten Nachbarschaftsinformationen so viele seiner Nachbarn als MPR, dass über diese Knoten alle 2-Hop-Nachbarn erreicht werden können. Ein Beispiel dafür zeigt Abb. 2.2. Knoten C hat in diesem Szenario die Knoten B und H als MPR gewählt. Diese beiden Knoten leiten daher die LS-Nachrichten von Knoten C weiter.

Zur weiteren Optimierung werden TC Nachrichten nur von MPRs erzeugt. Außerdem enthalten diese Nachrichten nur LS-Informationen über die Knoten, die diesen Knoten als MPR gewählt haben. Durch diese Maßnahmen wird der Overhead im Vergleich zum klassischen Fluten drastisch reduziert.

2.3.2 Das Routingprotokoll FSR

„Fisheye State Routing“ (FSR) [PeGC00] ist ebenfalls ein proaktives Link-State Routingprotokoll, welches für MANETs optimiert wurde. Im Unterschied zu OLSR sammelt jeder Knoten Link-State Informationen für das gesamte Netz und versendet diese aggregierten Daten periodisch per Broadcast an seine Nachbarn.

[2] Link-States
[3] sog. MPR-Selection Algorithmus

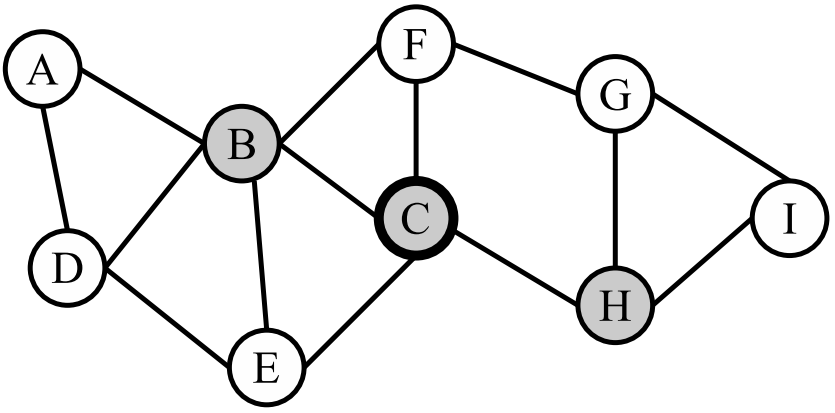

Abbildung 2.2: Beispielszenario für Wahl der MPRs

Um Overhead zu sparen, werden Informationen über weit entfernte Knoten seltener gesendet als Informationen über nahe gelegene Knoten. Hierbei dient die Zahl der Hops als Maß für die Knotenentfernung. Somit sind die Routinginformationen zu nahegelegenen Knoten genauer als Informationen zu weit entfernten Zielen.

Abb. 2.3 zeigt die Unterteilung des Netzes in zwei Reichweitenbereiche. In diesem Fall versendet der schwarze Knoten im Mittelpunkt der Abbildung Informationen über seine direkten Nachbarknoten (innerer Kreis) häufiger als Informationen über weiter entfernt liegende Knoten (äußerer Kreis).

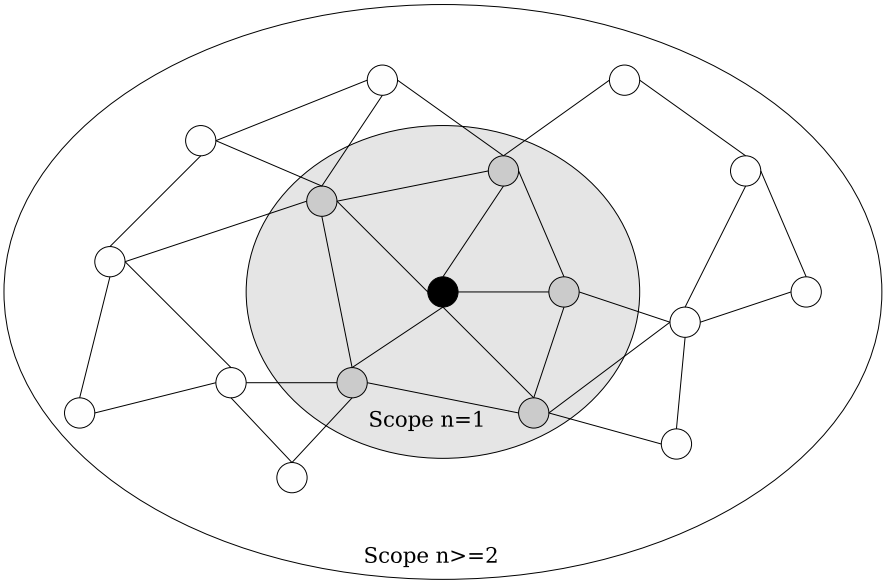

Abbildung 2.3: FSR mit zwei Scopes

Eine weitere Reduktion des Overheads ergibt sich dadurch, dass nur LS-Informationen verbreitet werden, die sich seit dem letzten Sendeintervall geändert haben.

Problematisch an diesem Verfahren ist die Tatsache, dass sich Informationen über neu hinzugekommene Knoten nur sehr langsam im Netz verbreiten.

3. Die Autokonfigurationsarchitektur PACMAN

Im Folgenden soll kurz die der Implementierung zugrunde liegende Autokonfigurationsarchitektur für mobile Ad-hoc-Netze (PACMAN) [Weni04] vorgestellt werden. Im Gegensatz zu den meisten anderen Autokonfigurationsverfahren kann die Wahl einer Adresse und die Überprüfung deren Eindeutigkeit rein passiv, durch die Analyse von Routingprotokollpaketen, erfolgen.

PACMAN verfolgt einen hybriden Ansatz im Hinblick auf die Zustandshaltung: Zum Einen wird durch die Verwaltung einer Belegungstabelle, die eine Liste der bekannten Adressen des Netzes enthält, ein zustandsbehaftetes Verfahren verwendet. Zusätzlich wird jedoch nach der zufälligen Wahl einer Adresse die Eindeutigkeit dieser Adresse permanent mittels eines „Duplicate Address Detection“ (DAD) Verfahrens überprüft, wie es bei klassischen zustandslosen Autokonfigurationsverfahren der Fall ist. Dadurch können auch bei einer unvollständigen Belegungstablle oder nach Netzverschmelzungen doppelte Adressen erkannt werden. Die Erkennung doppelter Adressen erfolgt ebenfalls rein passiv und wird in Abschnitt 3.2 beschrieben. Somit sind auch keinerlei Modifikationen am verwendeten Routingprotokoll notwendig.

3.1 Systemarchitektur von PACMAN

PACMAN besteht aus einzelnen (z.T. optionalen) Modulen, die voneinander weitgehend unabhängig sind. Einen Überblick über die gesamte PACMAN-Architektur gibt Abb. 3.1. Die empfangenen Routingprotokollpakete werden zuerst von einem routingprotokollspezifischen Modul in ein generisches Format konvertiert und an den **PACMAN Manager** übergeben. Dieser stellt die Schnittstelle zwischen den einzelnen Modulen dar und gibt die empfangenen Daten an die entsprechenden Module weiter.

Das **PDAD**-Modul analysiert die Daten auf Anomalien und erkennt dadurch Adresskonflikte. Dabei kommen je nach Art des Routingprotokolls unterschiedliche Ver-

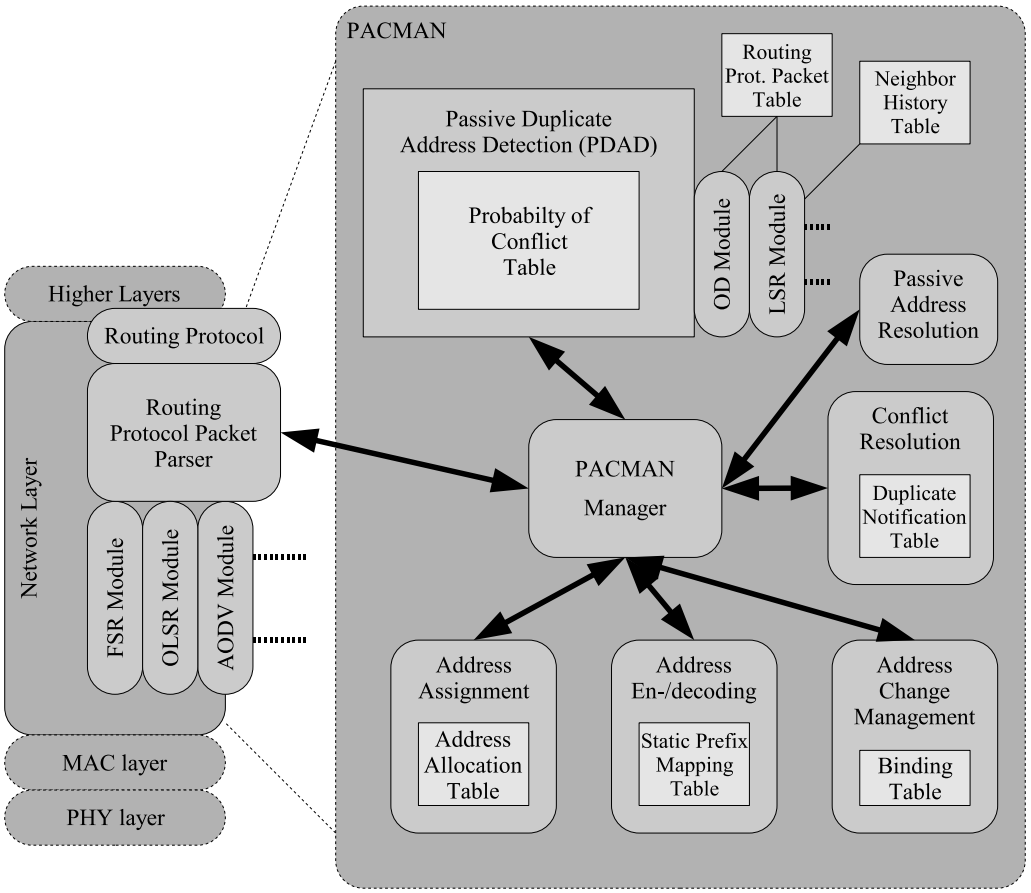

Abbildung 3.1: PACMAN Architektur

fahren für reaktive bzw. proaktive Protokolle zur Anwendung. Einige der PDAD-Verfahren werden in Abschnitt 3.2 vorgestellt.

Im Fall eines Konflikts wird dieser durch das **Conflict Resolution**-Modul aufgelöst. Falls der Konflikt von einem Zwischenknoten erkannt wurde, wird eine Address Conflict Notification (ACN) an die Konfliktadresse geschickt, damit dieser Knoten seine Adresse ändert. Um den Overhead durch ACNs zu reduzieren, wird die Rate der ACNs begrenzt. Dies wird erreicht, indem ACNs von Zwischensystem gegebenenfalls verworfen werden.

Die Wahl einer Adresse (sowohl zur Autokonfiguration wie auch im Fall eines Konflikts) erfolgt unter Verwendung des **Address Assignment**-Moduls durch jeden Knoten selbst. Dieses wählt zufällige eine Adresse innerhalb des vorgegebenen Adressraums und überprüft anhand der Belegungstabelle, ob die Adresse bereits vergeben ist. Die Belegungstabelle wird ständig anhand der Daten empfangener Routingprotokollpakete aktualisiert. Eine genauere Beschreibung des Verfahrens befindet sich in Abschnitt 3.3.

Damit bestehende Verbindungen im Fall eines Adresswechsels nicht abbrechen, sorgt das **Address Change Management**-Modul ähnlich wie bei Mobile IP mittels Tunneln dafür, dass Pakete an die alte Adresse weiterhin beim Zielknoten ankommen.

Das **Address En-/Decoding**-Modul sorgt optional für eine Komprimierung der Adressen in den Routingprotokollnachrichten um Overhead zu sparen. Die Kompri-

mierungsrate hängt dabei von der Größe des verwendeten Adressraums ab. Die Wahl der Adressraumgröße ist somit eine Abwägung zwischen Konfliktwahrscheinlichkeit und Komprimierungsrate.

Eine weitere Reduktion des Overhead kann durch Verwendung des **Passive Address Resolution**-Moduls erreicht werden. Dieses beruht auf der Annahme, dass bei Routingprotokollpakete, die per Broadcast verschickt werden, die IP-Quelladresse stets der Knotenadresse des Senders entspricht. Das Modul übernimmt die Aktualisierung des ARP-Cache durch Informationen aus den erhaltenen Routingprotokollnachrichten und macht somit die Versendung von ARP-Nachrichten überflüssig.

3.2 Einige PDAD-Algorithmen

PDAD stellt die wichtigste Komponente von PACMAN dar, da nur mittels PDAD doppelte Adresse nach einer Netzverschmelzung erkannt werden können. Abhängig von der Art des verwendeten Routingprotokolls kommen hier verschiedene Verfahren zur Anwendung. Da nur Verfahren für „proaktive Link-State“ Routingprotokolle implementiert wurden, werden auch nur diese hier vorgestellt. Für die Verwendung von PACMAN mit „on demand“ Protokollen kommen weitere Algorithmen zur Anwendung. Alle Verfahren nutzen die Tatsache aus, dass LSP nur für begrenzte Zeit im Netz verbreitet werden. Um durch doppelte Adresse hervorgerufene Inkonsistenzen im Routingsprotokollverkehr erkennen zu können, wird das jeweils letzte empfangene LSP jedes Knotens in einer Tabelle gespeichert.

3.2.1 PDAD basierend auf Absenderadressen (PDAD-SA)

Dieses Verfahren nutzt aus, dass die IP-Quelladresse beim Versenden einer Broadcast-Nachricht stets auf die Adresse des sendenden Knotens gesetzt wird. Das ist bei allen gängigen Routingprotokolle der Fall, da dort die Weiterleitung der Routinginformationen auf Anwendungsschicht erfolgt.

Falls ein Knoten eine Routingprotokollnachricht empfängt, die seine eigene Adresse als Absenderadresse enthält, liegt ein Konflikt mit dieser Adresse vor.

3.2.2 PDAD basierend auf Sequenznummern (PDAD-SN)

Hier werden die Sequenznummern der LSP[1] verwendet um Konflikte zu erkennen. Dabei wird ausgenutzt, dass die Sequenznummern kontinuierlich erhöht werden und dieses auch nur durch den Urheber des Pakets. Somit können Konflikte mit folgenden Verfahren erkannt werden:

- **PDAD-SN**: Sofern ein Knoten ein LSP mit seiner Adresse als Ursprungsadresse empfängt und die enthaltene Sequenznummer höher ist, als die aktuell verwendete Sequenznummer, liegt ein Konflikt mit dieser Adresse vor. Im Beispiel in Abb. 3.2 haben Knoten A und D beide die Adresse 1. Knoten A erkennt beim Empfang des LSP von D anhand der zu hohen Sequenznummer einen Konflikt mit Adresse 1.

[1] Link-State Pakete

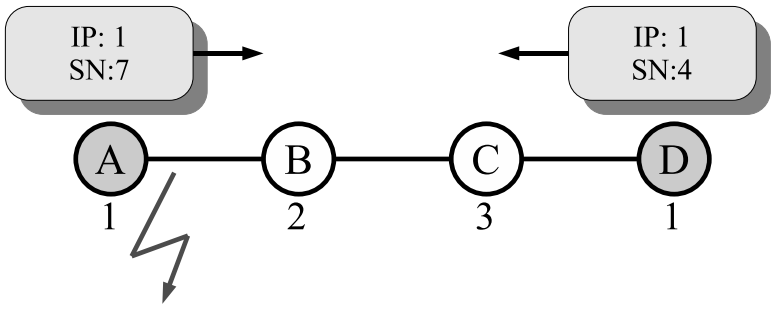

Abbildung 3.2: Beispiel für Konflikterkennung mit PDAD-SN

- **PDAD-SND**: Falls der Sequenznummernunterschied zwischen zwei Knoten mit Konflikt hinreichend groß ist, kann der Konflikt auch von einem Zwischenknoten erkannt werden. Ein Beispiel hierfür ist in Abb. 3.3 zu sehen: Knoten C erkennt nach Empfang der LSP von A und D einen Konflikt mit der Adresse 1 an dem großen Sequenznummernunterschied, welche im Fall von eindeutigen Adressen bei zwei innerhalb kurzer Zeit empfangener LSP nicht auftreten kann.

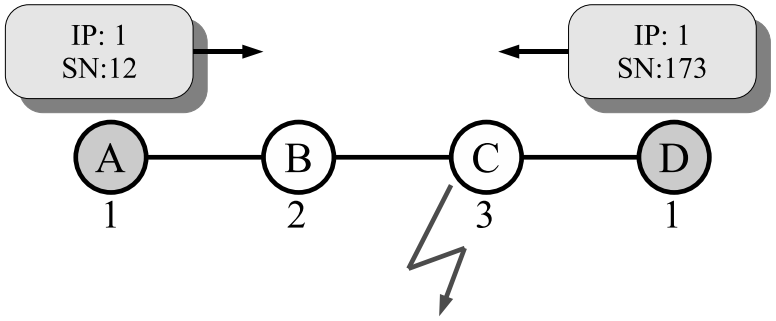

Abbildung 3.3: Beispiel für Konflikterkennung mit PDAD-SND

- **PDAD-SNE**: Eine weitere Möglichkeit zur Konflikterkennung bei Zwischenknoten bietet der Vergleich von LSP mit gleichen Ursprungsadressen und Sequenznummern. In diesem Fall müssten auch die enthaltenen LS identisch sein. Im Beispiel in Abb. 3.4 enthalten die LSP von Knoten A und D jedoch bei gleichen Ursprungsadressen und Sequenznummern unterschiedliche LS. Somit kann Knoten C nach Empfang beider LSP den Konflikt erkennen

3.2.3 PDAD basierend auf Nachbarschaft (PDAD-NH)

Bei PDAD-NH werden bidirektionale LS ausgenutzt. Falls ein Knoten X ein LSP empfängt, das seine Adresse in den LS enthält, muss der Urheber des LSP im Fall

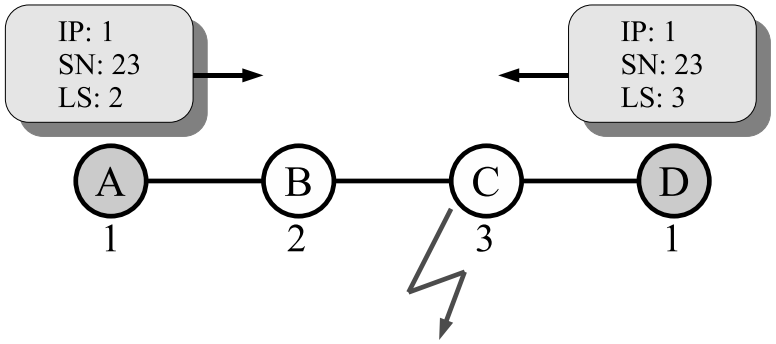

Abbildung 3.4: Beispiel für Konflikterkennung mit PDAD-SNE

von bidirektionalen LS auch in letzter Zeit Nachbar von X gewesen sein. Ist dies nicht der Fall, liegt ein Konflikt vor. Um diese erkennen zu können, müssen alle Knoten eine Tabelle mit ehemaligen Nachbarn verwalten.

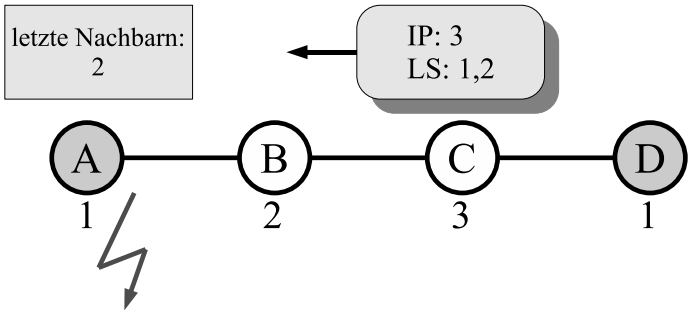

Abbildung 3.5: Beispiel für Konflikterkennung mit PDAD-NH

Im Beispiel in Abb. 3.5 empfängt Knoten A ein LSP von Knoten C mit den LS 1 und 2. Da die Adresse von Knoten A in diesem LSP enthalten ist, müsste im Fall von bidirektionalen LS also C kürzlich ein Nachbar von A gewesen sein. Da C jedoch nicht in der Nachbarschaftstabelle von A vorkommt, liegt ein Konflikt vor.

3.2.4 PDAD basierend auf Lokalität (PDAD-LP)

Hierbei handelt es sich um ein probabilistisches Verfahren, welches ausnutzt, dass die LS von zwei aufeinanderfolgenden LSP des gleichen Knotens in der Regel[2] ähnlich sind. Je unterschiedlicher die beiden LSP sind, desto größer ist die Wahrscheinlichkeit p_i für einen Konflikt. Diese ergibt sich aus Gleichung 3.1:

$$p_i = \frac{|LS_1 \Delta LS_2|}{|LS_1| + |LS_2|} \tag{3.1}$$

[2]das gilt nicht in Szenarien mit im Verhältnis zur LSP-Sendeperiode zu hohen Mobilität

Um die Zuverlässigkeit der Konflikterkennung zu erhöhen, kann p_i mit Hilfe von Gleichung 3.2 geglättet werden:

$$p_i = \alpha p_i + (1 - \alpha) p_{i-1} \tag{3.2}$$

Des weiteren werden nur LSP betrachtet die mindestens zwei LS enthalten.

Überschreitet p_i einen festgelegten Schwellwert, liegt wahrscheinlich ein Konflikt vor.

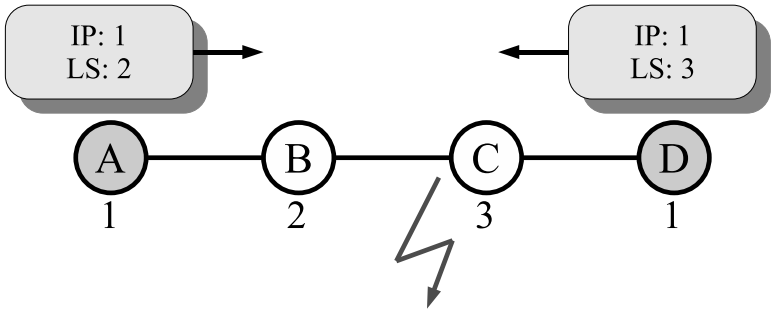

Abbildung 3.6: Beispiel für Konflikterkennung mit PDAD-LP

Im Beispiel in Abb. 3.6 empfängt Knoten C über längere Zeit LSP von Knoten A und D mit der gleichen Ursprungsadresse und unterschiedlichen LS. Somit liegt wahrscheinlich ein Konflikt mit Adresse 1 vor.

3.2.5 PDAD bei Verwendung von MPRs (PDAD-NH+)

Bei der Verwendung von MPR-gestützten Routingprotokollen wie z.B. OLSR kann der MPR-Selection Algorithmus (siehe 2.3.1) beim Netzen mit doppelten Adressen fehlschlagen. Durch wie Wahl von zu wenigen MPRs werden somit LSP nicht mehr im gesamten Netz verbreitet. Da die PDAD von der Analyse dieser LSP abhängt, kann daher in seltenen Fällen die Konflikterkennung fehlschlagen (siehe Abschnitt 4.9.1).

Einige dieser Konfliktszenarien können mittels PDAD-NH+ aufgelöst werden. Der Hinweis auf solch einen Konflikt ergibt sich aus der Inkonsistenzen zwischen zusammengehörigen HELLO und TC Paketen unter der Ausnutzung von bidirektionalen LS.

Beim Empfang eines TC von Knoten X wird für jeden im LS enthaltenen Knoten Y_i durch den empfangenden Knoten Z überprüft, ob kürzlich ein HELLO von Y_i empfangen wurde. In den Regel[3] muss dann X im LS des HELLOs von Y_i enthalten sein. Falls dies nicht der Falls ist, liegt wahrscheinlich ein Konflikt mit der Adresse von Y_i vor. Um einen Fehlerkennung durch hohe Netzauslastung zu vermeiden, meldet Knoten Z den Konflikt nicht direkt, sondern leitet das TC an Knoten Y_i weiter, der den Konflikt mittels PDAD-NH zuverlässig erkennen kann.

[3] Ausnahmen können bei hoher Netzauslastung und Mobilität auftreten

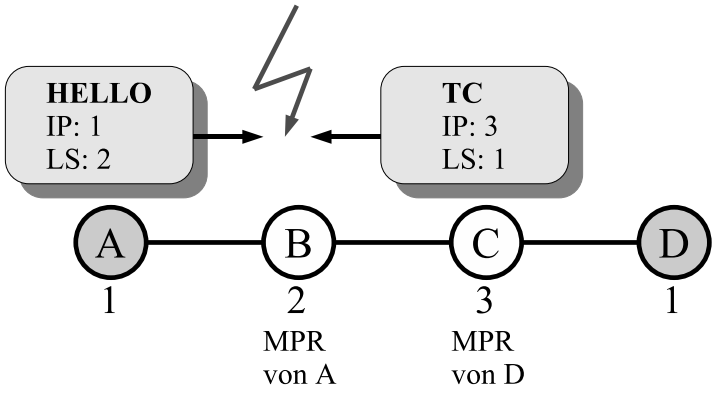

Abbildung 3.7: Beispiel für Konflikterkennung mit PDAD-NH+

Abb. 3.7 zeigt ein Beispiel für diese Szenario. A hat B und D hat C als MPR gewählt. B und C haben aufgrund der doppelten Adresse 1 fälschlicherweise keine MPRs gewählt. Daher leitet B das TC von C in diesem Fall nicht weiter.

Da ein Knoten mit Adresse 1 im LS des TC enthalten ist, prüft B, ob kürzlich ein HELLO mit Ursprungsadresse 1 empfangen wurde. Im LS dieses HELLOs müsste der Urheber des TCs (mit Adresse 3) enthalten sein. Da dies nicht der Fall ist, wird ein Konflikt mit Adresse 1 vermuten und das TC an Knoten A weitergeleitet. Dieser erkennt den Konflikt mittels PDAD-NH.

3.3 Adresszuweisung

Wie bereits erwähnt, erfolgt die Wahl einer neuen Adresse durch jeden Knoten selbstständig. Die Größe des Adressraums, aus dem die Adresse gewählt wird, hängt dabei von der gewünschten Konfliktwahrscheinlichkeit p_c ab. Für eine geschätzte Knotenanzahl n kann diese durch Gleichung 3.3 abgeschätzt werden.

$$p_c \approx 1 - (r \cdot e)^{-j} \frac{(r - n + j)^{r-n+j+\frac{1}{2}}}{(r-n)^{r-n+\frac{1}{2}}} \tag{3.3}$$

Mittels der Belegungstabelle wird überprüft, ob die zufällig gewählte Adresse bereits vergeben ist und gegebenenfalls eine neue Adresse gewählt. Kann im gewünschten Adressraum keine freie Adresse gefunden werden, wird der Adressraum schrittweise vergrößert.

Da die Belegungstabelle eines neuen Knotens anfangs leer ist und sich erst mit der Zeit durch den Empfang von LSPs füllt, kann der Knoten zur Beschleunigung des Autokonfigurationsvorgangs die aktuelle Belegungstabelle seiner Nachbarn anfordern. Dadurch werden in dem häufigen Szenario, dass ein Knoten zu einem bestehenden Netz hinzukommt, die meisten Konflikte bereits vermieden.

4. Implementierung von PACMAN

In diesem Kapitel werden der Aufbau und die Besonderheiten der im Rahmen der Studienarbeit angefertigten PACMAN-Implementierung beschrieben. Nach einem kurzen Überblick über die Struktur der Implementierung werden die wichtigsten Komponenten detaillierter beschrieben. Abschließend werden noch die verwendeten Datenstrukturen und Nachrichtenformate vorgestellt.

4.1 Integration in Linux

PACMAN wurde in Form einer eigenständigen (user level-)Anwendung für Linux unter Kernel 2.4.x implementiert und auf x86- und StrongARM-Architekturen getestet. Ein wichtiges Implementierungsziel war, keine Modifikationen am Routingprotokolldaemon vornehmen zu müssen. Daher musste ein Weg gefunden werden unabhängig vom Routingprotokolldaemon die zur Analyse benötigten LSP mitzuhören.

Die Linux Netfilter-Architektur[Welt] bietet mittels der Bibliothek libipq die Möglichkeit Pakete direkt aus dem Netzwerkstack des Kernels abzufangen und bei Bedarf zu modifizieren. Mittels eines kleinen Kernelmoduls können somit die gewünschten Routingprotokollpakete identifiziert und per libipq an die PACMAN-Anwendung übergeben werden.

Für eingehende Pakete wird der Netfilter-Hook `NF_IP_PRE_ROUTING` verwendet. Diese Stelle wurde gewählt, da hier sowohl Pakete, die für den Knoten selbst bestimmt sind, als auch Pakete, die weitergeleitet werden müssen, gemeinsam abgegriffen werden können. Für ausgehende Pakete wird anstelle von `NF_IP_POST_ROUTING` der Netfilter-Hook `NF_IP_LOCAL_OUT` verwendet, um weitergeleitete Pakete nicht doppelt zu bearbeiten.

Ein Nachteil von libipq ist, dass die Bibliothek momentan nur von einer Anwendung gleichzeitig verwendet werden kann[1]. Es existiert jedoch bereits eine Erweiterung (libxipq) bei welcher diese Einschränkung nicht mehr vorhanden ist.

[1] Beispielsweise verwendet die AODV-Implementierung der Uppsala University ebenfalls libipq

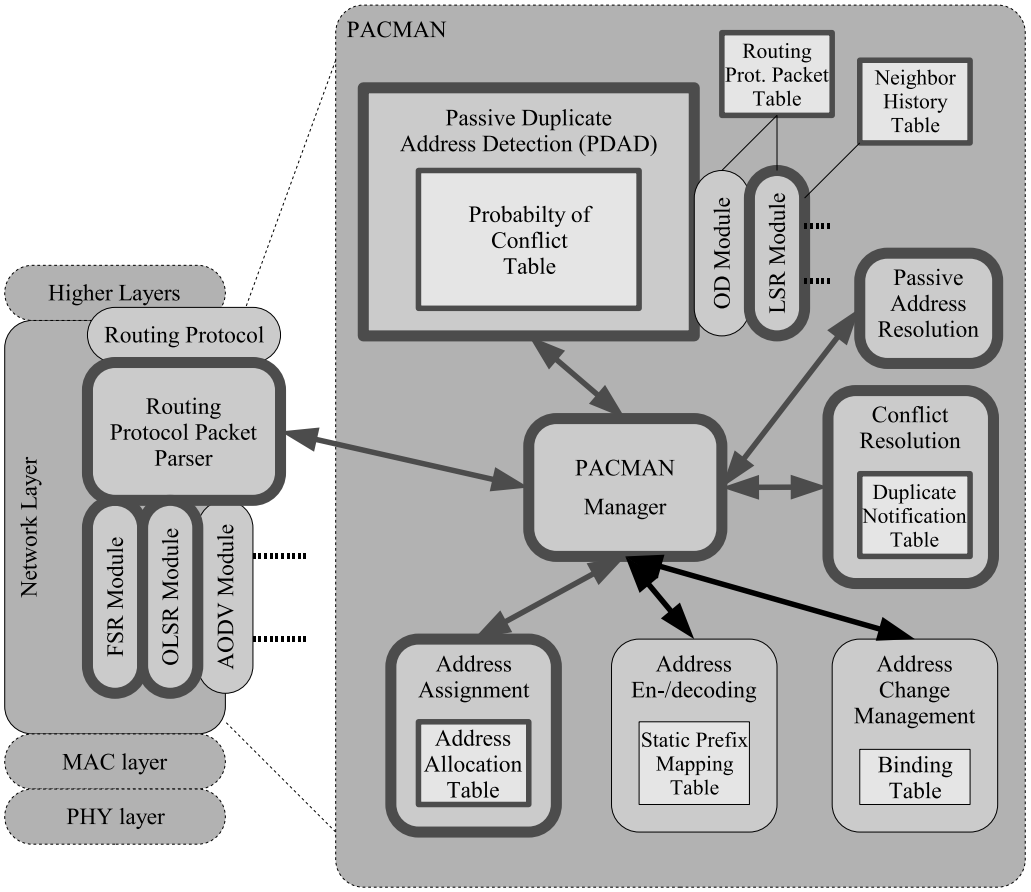

Abbildung 4.1: Implementierte Komponenten der PACMAN Architektur

4.2 Struktur der PACMAN-Implementierung

Bei der Implementierung wurde weitgehend die modulare Struktur des PACMAN Entwurfs übernommen um leichte Erweiterbarkeit zu ermöglichen. Wie bereits erwähnt wurden nur PDAD-Verfahren für proaktive Link-State Routingprotokolle implementiert. Die optionalen Modulen zur Adresskomprimierung und zum Mobilitätsmanagement sollten im Rahmen der Studienarbeit nicht implementiert werden. In Abb. 4.1 sind die Komponenten der PACMAN-Architektur, die in der Arbeit implementiert wurden, hervorgehoben. Die Aufteilung der Implementierung auf die einzelnen Quelldateien ergibt sich aus Tabelle 4.1.

Der Ablauf der Hauptroutine ist in Abb. 4.2 grafisch dargestellt. Nach dem Start beginnt PACMAN in `main.c` mit der Initialisierung der benötigten Datenstrukturen und dem Parsen der Kommandozeile.

Der weitere Ablauf erfolgt rein ereignisgesteuert durch eine Endlosschleife am Ende von `main()`. Ein mögliches Ereignis ist der Ablauf des Timers, der alle `TIMER_INT` Sekunden erfolgt und zum Aufruf von `table_cleanup_timer()` führt, um veraltete Tabelleneinträge zu löschen. Die anderen Ereignisse basieren auf dem Empfang einer Nachricht. Die Registrierung dieser Ereignisfunktionen beim Ereignismanager erfolgt mittels `add_event_func()`. Nach dem Start werden die folgenden Funktionen registriert:

- `packet_input()`: Empfang eines Routingprotokollpakets mittels libipq.

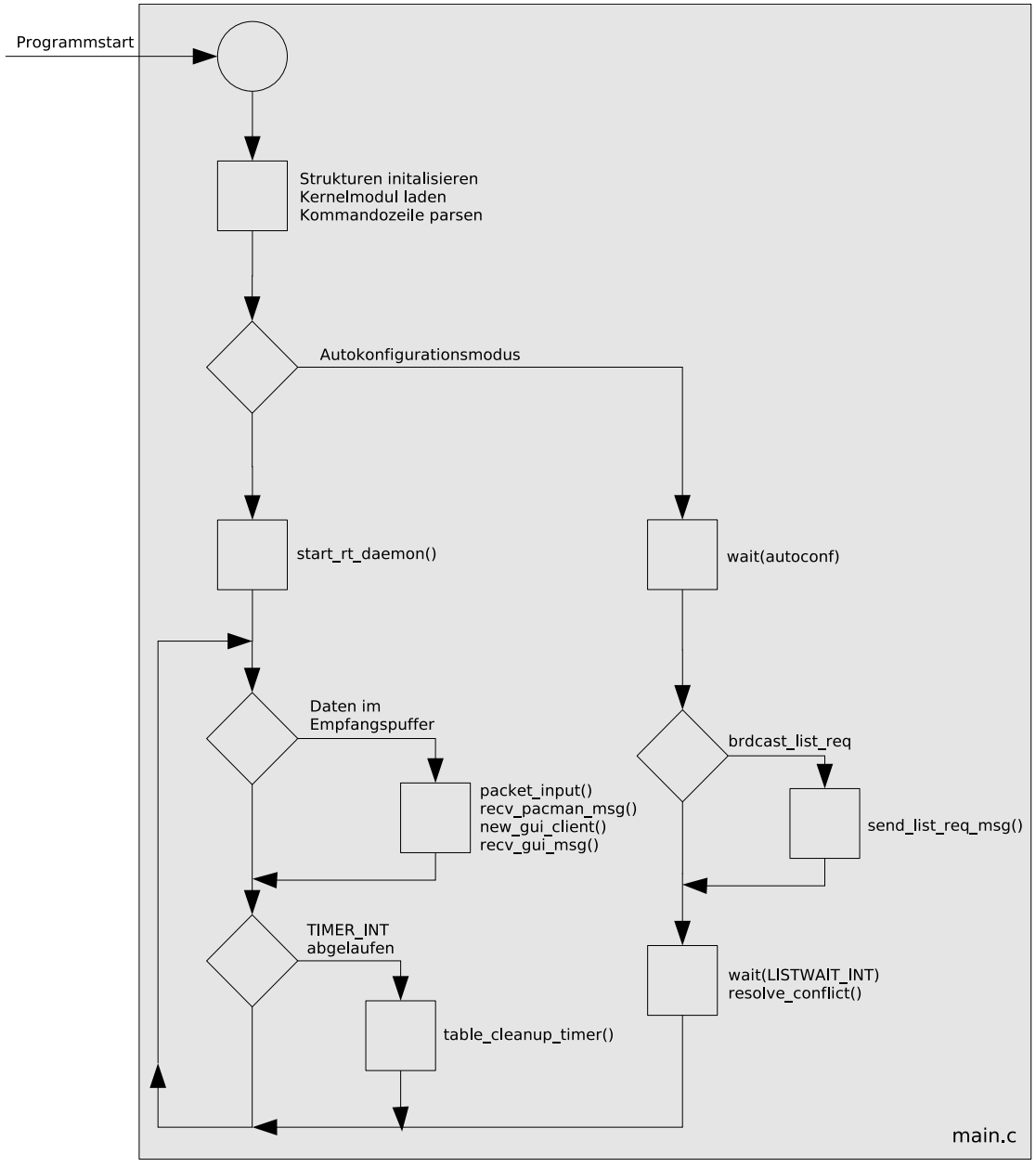

Abbildung 4.2: Ablaufdiagramm Hauptroutine

addr_mgr.[ch]	Adressmanagement und Konfliktauflösung
fsr.[ch]	Routingprotokollparser für FSR
kpacman.c	Kernelmodul mit Netfilterhooks für IP_QUEUE
libipq.[ch]	libipq Netfilter Bibliothek
list.[ch]	Verwaltung von verketteten Listen
main.c	Hauptprogramm mit Initialisierungen
olsr.[ch]	Routingprotokollparser für OLSR
packet_input.[ch]	Paketeingang und Passive Adresserkennung
pacman.h	Allgemeine Konstanten
pdad_algo.[ch]	PDAD Algorithmen
table.[ch]	Verwaltung der Hashtable

Tabelle 4.1: Dateienübersicht

- `recv_pacman_msg()`: Empfang einer PACMAN-Signalisierungsnachricht
- `new_gui_client()`: Signal, wenn neue GUI-Verbindung aufgebaut wurde
- `recv_gui_msg()`: Empfang einer GUI-Nachricht

Zentrale Aufgabe von PACMAN ist die Analyse von LSP mittels PDAD. Die LSP werden mittels libipq in `packet_input.c` durch `packet_input()` empfangen. Dort wird zuerst durch `update_arp()` der zugehörige Eintrag im ARP-Cache des Kernels aktualisiert (Passive Address Resolution). Danach wird das LSP durch Aufruf eines routingprotokollabhängigen Parsers in ein generisches LSP konvertiert (siehe Abschnitt 4.3) und von dort mittels `handle_generic_packet()` weitergereicht.

Dort werden zuerst verschiedene Tabellendaten aktualisiert (siehe Abschnitt 4.6), um dann durch Aufruf von `do_pdad()` die Konflikterkennung durchzuführen (siehe Abschnitt 4.4). Im Fall eines erkannten Konflikts wird die Konfliktauflösung durch `resolve_conflict()` initiiert und das LSP verworfen (`NF_DROP`), um eine weitere Ausbreitung der inkonsistenten Daten zu vermeiden (siehe Abb. 4.4).

Sofern ein Knoten einen Konflikt mit seiner eigenen Adresse erkennt, wählt der Adressmanager (siehe Abschnitt 4.5) in `choose_new_addr()` eine neue Adresse und startet den Routingprotokolldaemon neu. Falls der Konflikt von einem Zwischenknoten erkannt wurden, wird durch `send_acn_msg` eine Nachrichten an den Knoten mit Konflikt gesendet (siehe Abschnitt 4.7).

Parallel hierzu erfolgt die Kommunikation mit anderen Knoten (`send_pacman_msg()` und `recv_pacman_msg()`) sowie mit der GUI (`send_gui_msg` und `recv_gui_msg()`).

4.3 Routingprotokollparser

Da PACMAN auf der Analyse von LSP beruht, muss jedes verwendete Routingprotokoll explizit unterstützt werden. Um das Hinzufügen eines neuen Routingprotokolls so einfach wie möglich zu gestalten, wurde der routingprotokollspezifische Teil auf das Nötigste reduziert und von anderen Komponenten abgekapselt. Die einzige Aufgabe der Routingprotokollparser besteht darin, die LSP in ein generisches Paket vom Typ `ls_entry_t` zu konvertieren. Von 2900 Zeilen Quellcode für die gesamte

Implementierung entfallen daher nur ca. 180 Zeilen auf das OLSR Modul und ca. 75 Zeilen auf das FSR Modul.

Die „Registrierung“ eines Moduls erfolgt durch Erweiterung der Konstante `protocols[]` vom Typ `struct rt_prots` in `main.c`.

```
struct rt_prots {
  char *name;
  u_int16_t port;
  u_int32_t sn_max;
  parse_rt_func_t parse_rt_func;
  float def_par[MAX_PDAD_ALGO];
  int max_ips[LS_TYPE2 + 1];
};
```

Dort muss in `port` der vom Routingprotokoll verwendete UDP-Port und in `sn_max` die Größe des Sequenznummernraums angegeben werden. `def_par[]` ermöglicht die Angabe von auf das Routingprotokoll angepassten PDAD-Standardparametern. `parse_rt_func` ist der Zeiger auf die Parserfunktion, die für die Konvertierung der LSP zuständig ist und `max_ips` das größte Zeitintervall, das zwischen zwei empfangenen LSP liegen darf um eine zuverlässige Konflikterkennung zu ermöglichen.

Der folgende Quellcode zeigt beispielhaft eine solche Parserfunktion:

```
int parse_rt_msg(char *buf, size_t cnt,
                 u_int32_t src_addr, int incoming)
{
  ls_entry_t *e;
  [...]
  e = new_ls_entry(msg->orig_addr, msg->seqn, LS_TYPE1);
  for(i=0; i<ls_count; i++)
    update_ls_neighbor(e, msg->ls_addr[i]);

  return handle_generic_packet(e, src_addr, incoming);
}
```

Aus dem in `buf` übergebenen LSP müssen nun Ursprungsadresse, Sequenznummer und die LS extrahiert werden. Durch Aufruf von `new_ls_entry()` wird ein generisches Paket mit der angegebenen Ursprungsadresse und Sequenznummer erzeugt. Bei Protokollen, die auf MPRs basieren (z. B. OLSR), kann ausserdem die Art des LSP (HELLO oder TC) durch `LS_TYPE1` bzw. `LS_TYPE2` angegeben werden.

In das erzeugte Paket werden durch wiederholten Aufruf von `update_ls_neighbor()` die einzelnen LS eingetragen. Zum Schluß wird das Paket durch den Aufruf von `handle_generic_packet()` weitergereicht.

Der Rückgabewert des Aufrufs enthält die Entscheidung des PDAD-Moduls, ob das Paket akzeptiert (`NF_ACCEPT`) oder verworfen (`NF_DROP`) werden soll. Dieses Ergebnis muss von der Parserfunktion als Rückgabewert weitergeleitet werden.

Sofern das zu parsende Paket aggregierte LSP enthält (z. B. FSR), müssen daraus mehrere generische Pakete erzeugt werden. Falls `handle_generic_packet()` für eines dieser Pakete `NF_DROP` zurückgibt, muss das komplette Paket verworfen werden.

4.4 Passive Duplicate Address Detection

In Abb. 4.3 ist der Ablauf der Konflikterkennung grafisch dargestellt.

Das generische LSP des Routingprotokollparsers wird in `handle_generic_packet()` zur Aktualisierung der Belegungstabelle verwendet. Sofern es sich um ein ausgehendes Paket handelt, wird zusätzlich die eigene Sequenznummer aktualisiert. Im Fall eines eingehenden Pakets folgt nach Auffrischung der Nachbarschaftstabelle die Konflikterkennung.

Die Konflikterkennung durch Anwendung der verschiedenen PDAD-Algorithmen erfolgt durch die Funktion `do_pdad()` in `pdad_algo.c`. Dort wird zuerst überprüft, ob der zeitliche Abstand zwischen dem aktuellen und dem vorherigen (zugehörigen) LSP größer als `max_ips` ist. In diesem Fall ist mit einigen PDAD-Verfahren keine zuverlässige Konflikterkennung möglich, da es sich nicht um aufeinander folgende LSP handelt.

Falls es sich um aufeinander folgende LSP handelt, werden der Reihe nach alle aktiven PDAD-Algorithmen angewendet. Die Auflösung eines erkannten Konflikts erfolgt durch `resolve_conflict()` und ist in Abschnitt 4.5 beschrieben. Jeder Algorithmus wird analog zu den Routingprotokollparsern in `main.c` durch einen Eintrag in der Variablen `pa[]` vom Typ `struct pdad_algo` repräsentiert:

```
struct pdad_algo {
  char *name;                /* algorithm name */
  pdad_func_t pdad_func;     /* pdad function ptr */
  u_int8_t active;           /* active flag */
  float par;                 /* algorithm parameter */
  int stats;                 /* statistics counter */
};
```

`pdad_func` ist ein Funktionszeiger vom Typ `pdad_func_t`:

```
typedef float (*pdad_func_t) (ls_entry_t *e,
           u_int32_t *conf_addr, float *alpha, float par);
```

Als Eingabe erhält die Funktion einen Zeiger auf das zu analysierende LSP `e` und einen algorithmusspezifischen Parameter `par`, der beispielsweise als Schwellwert fungieren und per Kommandozeile geändert werden kann.

Die Funktion gibt nach der Analyse des LSP die Wahrscheinlichkeit für einen Konflikt als Rückgabewert zurück. Abhängig vom Ergebnis der Analyse wird über den Zeiger `alpha` das Gewicht α der Glättungsfunktion 3.2 zurückgegeben. Falls der Algorithmus keine Aussage über die Konfliktwahrscheinlichkeit treffen konnte, ist $\alpha = 0$. Der Zeiger `conf_addr` liefert die schließlich die Adresse zurück, bei der der Konflikt aufgetreten ist.

Das folgende Beispiel zeigt die Implementierung des Verfahrens PDAD-SN:

```
float pdad_sn(ls_entry_t *e, u_int32_t *conf_addr,
                      float *alpha, float sn_thres)
{
    *alpha = 1;
```

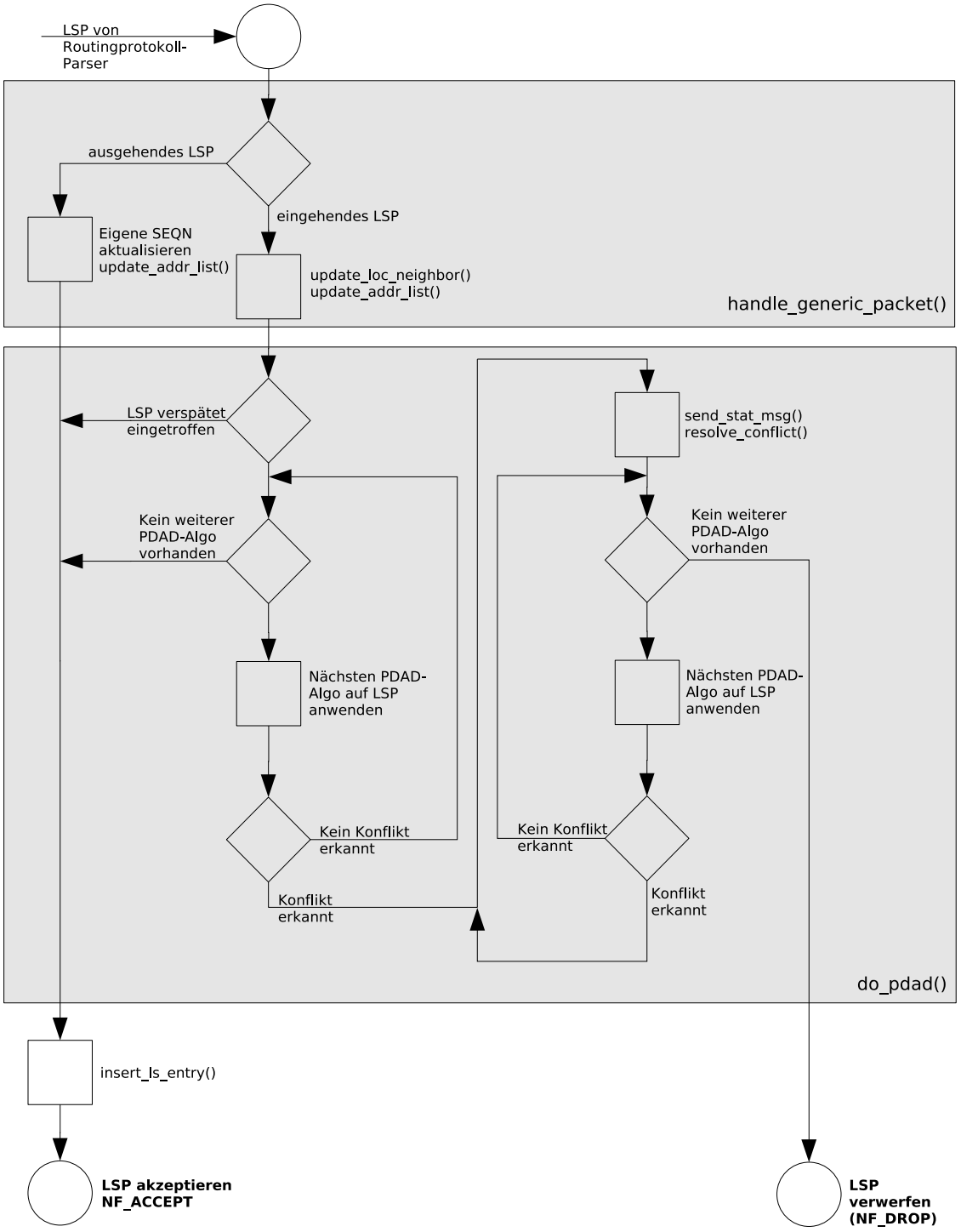

Abbildung 4.3: Ablaufdiagramm Konflikterkennung

```
    if (e->orig_addr == loc_ls_tab.addr) {
        if (e->seqn > loc_ls_tab.seqn[e->type]) {
            /* address conflict detected */
            *conf_addr = e->orig_addr;
            return 1;
        }
    }

    *alpha = 0;
    return 0;
}
```

Sofern die Ursprungsadresse des empfangenen LSP mit der eigenen Adresse identisch sowie die enthaltene Sequenznummer höher als die eigene Sequenznummer ist, liegt mit Sicherheit ein Konflikt vor. In diesem Fall wird sowohl die Konfliktwahrscheinlichkeit als auch der Glättungsfaktor α auf 1 gesetzt und die Konfliktadresse in `*conf_addr` übergeben.

Im anderen Fall kann das Verfahren keine Aussage über die Konfliktwahrscheinlichkeit treffen und setzt daher $\alpha = 0$. Der Paramter `sn_thres` wird nur für die Übergabe des Schwellwerts bei PDAD-SND benötigt und hat im Fall von PDAD-SN keine Verwendung.

4.5 Adressmanager und Konfliktauflösung

Das Adressmanager-Modul in `addr_mgr.c` dient primär der Konfliktauflösung und der Wahl einer neuen Adresse. Außerdem wurde dort die Kommunikation zwischen den einzelnen PACMAN-Knoten durch den Versand von Nachrichten realisiert. Diese Entscheidung wurde getroffen, da Nachrichten in PACMAN in der Regel zur Konfliktauflösung und zum Anfordern der Belegungstabelle (siehe Abschnitt 4.7) verwendet werden und somit ein enger Bezug zur Adressverwaltung gegeben ist.

Wie bereits erwähnt (siehe Abschnitt 3.3) wird im Adressmanager-Modul eine Belegungstabelle verwaltet, in der alle kürzlich gehörten Adressen gespeichert werden. Die Aktualisierung der Tabelle erfolgt durch Aufruf von `update_addr_list()` mit der jeweiligen Adresse. Dies passiert für alle in einem empfangenen LSP enthaltenen Adressen, sobald das Parsermodul das Paket an `handle_generic_packet()` weitergereicht hat. Des weiteren kann die Belegungsliste auch durch den Empfang einer `LIST_REP` Nachricht aktualisiert werden (siehe Abschnitt 4.7.3). Falls eine Adresseintrag nicht innerhalb der Zeit `HTABLE_TIMEOUT` aktualisiert wird, wird er automatisch aus der Belegungstabelle gelöscht.

Die Auflösung eines Adresskonflikts ist in Abb. 4.4 dargestellt. Diese wird durch Aufruf von `resolve_conflict()` initiiert und ist wie folgt deklariert:

```
void resolve_conflict(u_int32_t addr, u_int32_t src_addr,
                      char* reason, u_int32_t new_addr);
```

Der Parameter `addr` enthält die Konfliktadresse. Durch `src_addr` wird die IP-Quelladresse des LSP, dessen Analyse zur Konflikterkennung geführt hat, übergeben. Der Zeichenkette `reason` enthält den Grund für den Adresswechsel (z. B. den Namen

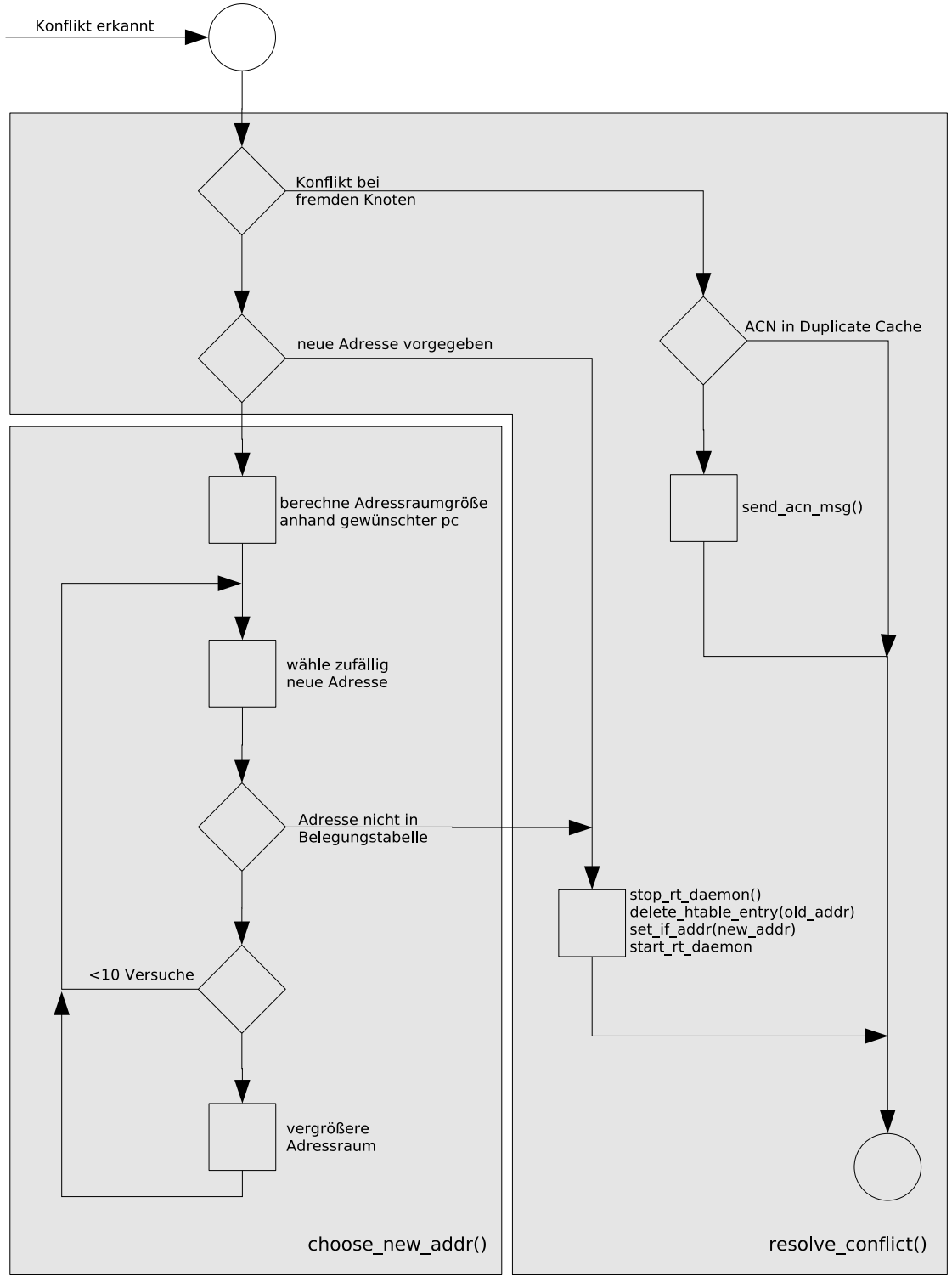

Abbildung 4.4: Ablaufdiagramm Konfliktauflösung

des PDAD-Algorithmus), der somit durch die GUI angezeigt werden kann. Durch `new_addr` kann optional die neue Adresse des Knotens vorgegeben werden. In der Regel (falls `new_addr = 0`) wird die neue Adresse automatisch gewählt.

Falls es sich bei der Konfliktadresse `addr` nicht um die eigene Knotenadresse handelt, wurde der Konflikt von einem Zwischenknoten erkannt. Um den Konflikt aufzulösen wird dieser durch den Versand einer sogenannten `ACN`[2] Nachricht aufgefordert seine Adresse zu ändern. Da die Nachricht derjenige Knoten erhalten soll, der im LSP, das von `src_addr` gesendet wurde, enthalten ist, wird die Nachricht über den Knoten `src_addr` verschickt.

Handelt es sich bei der Konfliktadresse jedoch um die eigene Knotenadresse, wird durch die Funktion `choose_new_addr()` zufällig eine neue Knotenadresse gewählt. Die Größe des Adressraums, aus dem die neue Adresse gewählt wird, richtet sich dabei nach der gewünschten Konfliktwahrscheinlichkeit `target_prob_conf` und wird mit Hilfe von Gleichung 3.3 bestimmt. Falls die gewählte Adresse bereits in der Belegungsliste enthalten ist, wird solange eine neue Adresse gewählt (ggf. unter schrittweiser Vergrößerung des Adressraums) bis dies nicht mehr der Fall ist. Nach der Wahl wird die entsprechende Netzwerkschnittstelle mit der neuen Adresse konfiguriert.

4.6 Verwendete Datenstrukturen

Die Belegungstabelle, die Tabelle mit generischen LSP sowie die „Duplicate Notification Table“[3] wurden aus Performancegründen als Hashtabelle realisiert. Die Funktionen zur Hashtabelle-Verwaltung befinden sich gemeinsam mit den Funktionen zur Verwaltung der Nachbarschaftstabelle, die in Form einer verketteten Liste implementiert wurde, in der Datei `table.c`. Um die Implementierung übersichtlich zu halten, wurden die Einträge der verschiedenen Tabellen (mit Ausnahme der Nachbarschaftstabelle) in einer gemeinsamen Hashtabelle abgelegt. In `list.c` befinden sich zusätzlich noch Hilfsfunktionen zur Verwaltung verketteter Listen.

4.6.1 Gemeinsame Hashtabelle

Da in der gemeinsamen Hashtabelle unterschiedliche (aber ähnliche) Typen von Einträgen gespeichert werden, kommen dort drei verschiedene Strukturen zum Einsatz: Die Grundstruktur `generic_entry_t` wird für Belegungstabelleneinträge, die Struktur `conf_entry_t` für Konflikteinträge und die Struktur `ls_entry_t` für generische LSP-Einträge verwendet:

```
typedef struct {
  list_t l;
  u_int32_t orig_addr;
  int type;
  struct timeval ts;
} generic_entry_t;
```

Die Komponente `l` der Grundstruktur `generic_entry_t` enthält Zeiger zur Verwaltung der doppelt verketteten Liste, die im Fall von (Hashtabellen-)Kollisionen

[2] Address Conflict Notification

[3] enthält kürzlich weitergeleitete ACNs (siehe Abschnitt 4.7.1)

Typ	verwendete Struktur	Verwendung
LS_TYPE1	ls_entry_t	generisches LSP Typ 1 (bei OLSR: HELLO)
LS_TYPE2	ls_entry_t	generisches LSP Typ 2 (bei OLSR: TC)
SA_RCVD	ls_entry_t	Pseudo-LSP für PDAD-SA
CONF_TYPE	conf_entry_t	Konfliktwahrscheinlichkeit und ACN Cache
ADDR_LIST_TYPE	generic_entry_t	Belegungstabelle

Tabelle 4.2: Mögliche Hashtabelleneinträge

benötigt wird. Die Adresse `orig_addr` stellt den Schlüssel der Hashtabelle dar. Das folgende Typfeld `type` dient zur Unterscheidung der verschiedenen Typen von Eintragen. Der Zeitstempel `ts` enthält den Zeitpunkt der Erzeugung des Eintrags.

```
typedef struct {
  list_t l;
  u_int32_t orig_addr;
  int type;
  struct timeval ts;
  float conf_p;
  struct timeval acn_msg_ts[2];
} conf_entry_t;
```

Die Struktur `conf_entry_t` dient zur Verwaltung von Konfliktwahrscheinlichkeiten und ACNs. Zusätzlich zu den Komponenten der Grundstruktur sind `conf_p` und `acn_msg_ts[]` enthalten. Die Variable `conf_p` gibt die momentane Wahrscheinlichkeit für einen Konflikt mit der Adresse `orig_addr` an. In `acn_msg_ts` wird der Zeitpunkt der letzten gesendeten bzw. weitergeleiteten `ACN` für diesen Knoten für beide möglichen Senderichtungen[4] gespeichert.

```
typedef struct {
  list_t l;
  u_int32_t orig_addr;
  int type;
  struct timeval ts;
  u_int32_t seqn;
  int n_cnt;
  list_t neighbors;
} ls_entry_t;
```

Die Struktur `ls_entry_t` dient zur Verwaltung von generischen LSP. Zu den Komponenten der Grundstruktur kommen hier noch die Sequenznummer `seqn`, die Zahl der LS `n_cnt` und die eigentliche Liste der LS `neighbors` hinzu. Die Liste ist aufsteigend nach Adressen sortiert um zwei LS effizient vergleichen zu können.

Eine Zusammenfassung über die verschiedenen Hashtabelleneinträge und Strukturen gibt Tabelle 4.2.

[4]Erläuterungen der Problematik unter Abschnitt 4.9.3

4.6.2 Nachbarschaftstabelle

Das PDAD-Verfahren NH benötigt eine Nachbarschaftstabelle, die alle Adressen kürzlicher Nachbarn eines Knotens mit Zeitstempeln enthält. Diese Tabelle wurde als doppelt verkettete Liste vom Typ `loc_neigh_t` realisiert:

```
typedef struct {
  list_t l;
  u_int32_t addr;
  struct timeval ts;
} loc_neigh_t;
```

Die Tabelle wird beim Empfang jedes LSP durch Aufruf von `update_loc_neighbor()` aktualisiert. Veraltete Einträgen werden nach Ablauf der Zeit `LOC_TIMEOUT` gelöscht.

4.7 Verwendete Nachrichtenformate

Obwohl PACMAN einen passiven Ansatz zur Autokonfiguration verfolgt, können zur Beschleunigung der Autokonfiguration und Konflikterkennung Nachrichten zwischen den Knoten verschickt werden. In Szenarien, in denen der Signalisierungsoverhead auf ein absolutes Minimum reduziert werden muss, können diese Nachrichten weggelassen werden.

Im Folgenden werden die vier verwendeten Nachrichtenformate kurz vorgestellt. Alle Nachrichten beginnen einheitlich mit einem 1 Byte langen Typfeld gefolgt von einer 3 Byte langen „Magic Sequence“ bestehend aus den 3 ASCII-Zeichen “PAC”.

Der Versand der Nachrichten erfolgt als UDP-Nutzlast an Port `PACMAN_PORT`[5]. Falls der Knoten noch keine Adresse gewählt hat, wird diese mit der Absenderadresse 0.0.0.0 verschickt. Für Broadcast-Nachrichten wird die Zieladresse 255.255.255.255 verwendet. Die Abarbeitung der empfangenen PACMAN-Nachrichten wird vom Ereignismanager durch Aufruf der Funktion `recv_pacman_msg()` in `addr_mgr.c` veranlasst.

4.7.1 ACN

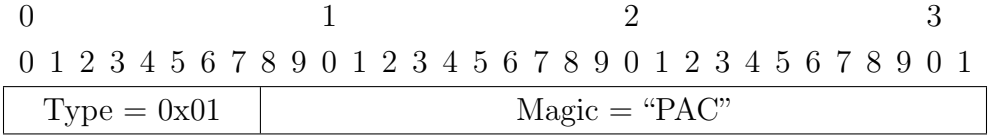

Abbildung 4.5: Paketformat `ACN`

Mit Hilfe einer `ACN` Nachricht wird der Empfänger der Nachricht von einem Zwischenknoten über einen bestehenden Adresskonflikt mit seiner Adresse informiert und aufgefordert diese zu ändern. Falls derselbe Konflikt zeitnah von mehreren Knoten erkannt wird, kann dies zu einer unnötigen Belastung des Netzes durch `ACN` Nachrichten führen. Um dies zu vermeiden, verwaltet jeder Knoten eine Liste mit kürzlich weitergeleiteten `ACNs` und verwirft innerhalb des Intervalls `ACN_TIME`[6] nach dem Empfang einer `ACN` weitere dieser Nachrichten an den gleichen Empfänger.

[5] hier: 10099
[6] in dieser Implementierung 500ms

4.7.2 LIST_REQ

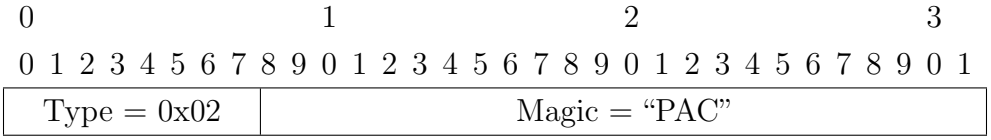

Abbildung 4.6: Paketformat `LIST_REQ`

Ein häufiges Autokonfigurationsszenario besteht darin, dass ein neuer Knoten zu einem bestehenden Netz hinzukommt. Da die Belegungstabelle dieses Knotens anfangs leer ist, wäre bei kleinem Adressraum und zufälliger Wahl der Adresse die Wahrscheinlichkeit für einen Adresskonflikt relativ hoch. Um diese zu reduzieren, könnte der Knoten die Wahl einer Adresse solange hinauszögern, bis sich die Belegungstabelle durch das Mithören von LSP gefüllt hätte.

Um diese Wartezeit zu vermeiden, kann ein Knoten die Belegungstabelle seiner Nachbarknoten durch das Verschicken einer `LIST_REQ` Broadcast-Nachricht anfordern.

4.7.3 LIST_REP

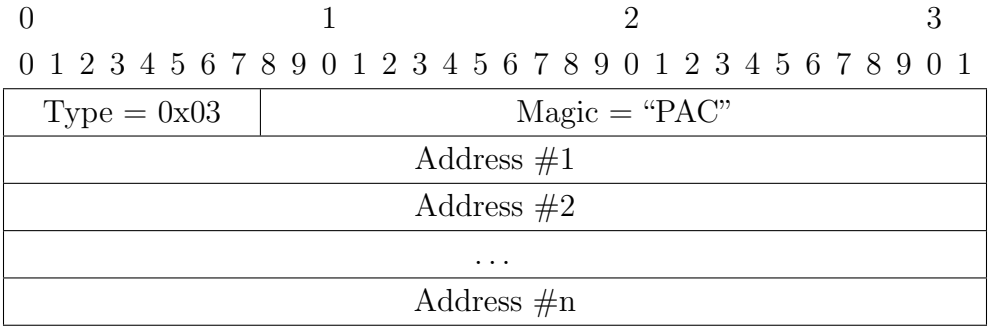

Abbildung 4.7: Paketformat `LIST_REP`

Als Antwort auf eine `LIST_REQ` Nachricht erhält ein Knoten per Unicast von allen Nachbarknoten deren Belegungstabelle durch ein `LIST_REP` Nachricht. Um die Wahrscheinlichkeit einer Kollision mit anderen `LIST_REP` Nachrichten zu reduzieren, wird diese mit einer zufälligen Verzögerung[7] verschickt.

4.7.4 HINT_MSG

Das PDAD-Verfahren NH+ (siehe 3.2.5) basiert auf der Weiterleitung von TC Nachrichten durch PACMAN, falls diese aufgrund doppelter Adressen nicht vom Routingprotokoll selbst weitergeleitet werden. Sobald dieses Ereignis von PDAD-NH+ erkannt wird, erfolgt die Weiterleitung des generischen LSP durch Versand einer sog. `HINT_MSG`. Diese enthält neben dem üblichen PACMAN Nachrichtenkopf alle Felder einer `ls_entry_t` Struktur. Beim Empfänger wird das erhaltene LSP direkt über `handle_generic_packet()` an das PDAD-Modul zu weiteren Analyse vorgelegt. Im Falle eines Konflikts kann dieser nur mittels PDAD-NH erkannt und aufgelöst werden.

[7] zwischen 0 und 20ms

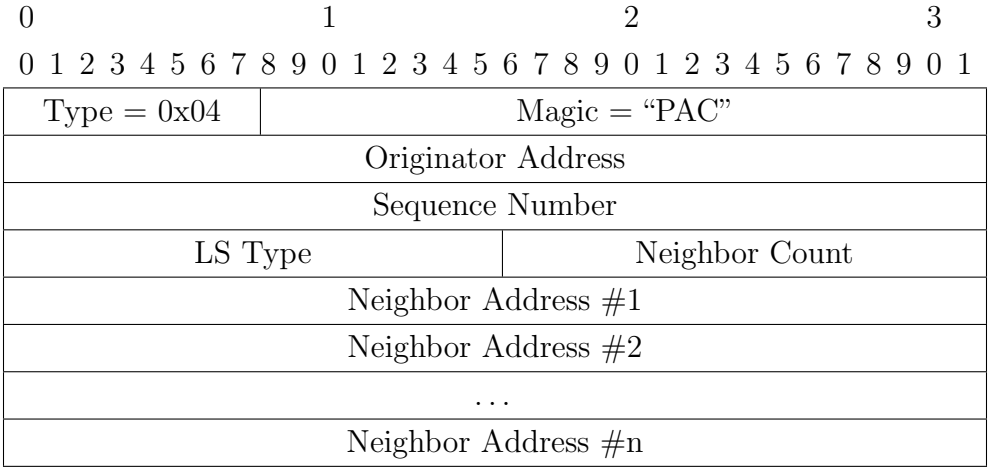

Abbildung 4.8: Paketformat `HINT_MSG`

4.8 GUI-Schnittstelle

Um die einzelnen Ereignisse und Statistikdaten der Autokonfiguration übersichtlich darstellen zu können, wurde die Möglichkeit geschaffen, eine grafische Oberfläche anzubinden. Die Anbindung erfolgt über den UNIX-Socket `GUI_SOCKETNAME` unter Verwendung eines ASCII- und zeilenorientierten Protokolls.

Nach dem Verbindungsaufbau der GUI sowie im Fall der Konflikterkennung und des Adresswechsels werden unaufgefordert alle Statisikdaten an die GUI geschickt. Die Daten haben die folgende Form:

```
STAT
A:169.254.0.5:AUTOCONF:0:04.7
SA:0
SN:1
LP:0
NH:2
MPR:0
ADDR_CHANGES:7
ADDR_BITS:11
PC:50
```

Der Datensatz beginnt mit dem Schlüsselwort `STAT`. Die folgende Zeile beginnt mit einem `A` oder einem `C`, abhängig davon ob es sich bei dem Ereignis um einen Adresswechsel (`A`) oder eine Konflikterkennung (`C`) handelt. Bei der folgenden IP-Adresse handelt es sich um die Adresse, zu der gewechselt wurde bzw. bei der ein Konflikt erkannt wurde. Das nächste Feld enthält den Grund des Adresswechsels bzw. der Konflikterkennung im Klartext. Die Zeile endet mit einem Zeitstempel in der Form `Minute:Sekunde.Zehntelsekunde` relativ zum PACMAN-Start. In den weiteren Zeilen folgt die Zahl der Konflikte, die mit dem jeweiligen PDAD-Verfahren bereits erkannt wurden. Zum Schluss kommen noch die Zahl der Adresswechsel, die Adressraumgröße in Bits sowie die momentan gewünschte Konfliktwahrscheinlichkeit in Prozent (siehe Abschnitt 4.5).

Zusätzlich kann die GUI folgende Befehle geschicken:

- `"KILL"` : Beendet PACMAN
- `"AUTO"` : Wahl einer neuen Adresse (Autokonfiguration).
- `"CONF␣192.168.1.12"` : Manuelles Setzen der neuen Adresse 192.168.1.12.
- `"SETC␣30"` : Stellt 30% Konfliktwahrscheinlichkeit ein
- `"LIST"` : Fordert die aktuelle Belegungstabelle an. Als Antwort erhält die GUI ein Paket mit diesem Format:

  ```
  ADDR_LIST
  169.254.3.29
  169.254.2.65
  ...
  ```

  Nach dem Schlüsselwort `ADDR_LIST` folgenden alle Adressen der Belegungstabelle zeilenweise.

4.9 Aufgetretene Probleme

In diesem Abschnitt wird eine Auswahl der Probleme, die während der Implementierung aufgetreten sind, vorgestellt und der gewählte Lösungsansatz dokumentiert.

4.9.1 OLSR MPR-Selection-Algorithmus

In Abschnitt 3.2.5 wurde bereits angesprochen, dass der MPR-Selection-Algorithmus von OLSR im Fall von doppelten Adressen fehlschlagen kann. In Abb. 4.9 ist nochmal ein solches Szenario dargestellt.

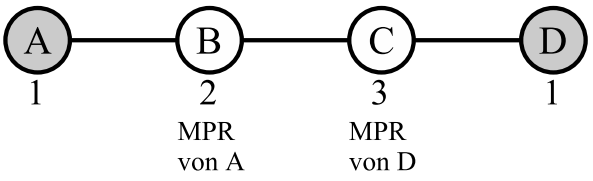

Abbildung 4.9: Konflikterkennung mit PDAD-NH+ möglich

A hat B und D hat C als MPR gewählt. B und C haben aufgrund der doppelten Adresse 1 fälschlicherweise keine MPRs gewählt. Daher leitet B das TC von C in diesem Fall nicht weiter. Analog dazu wird auch das TC von B von C nicht weitergeleitet. Dadurch kommt es zu einer Netzpartitionierung auf Schicht 3. Mit den in [Weni03] vorgestellten PDAD Algorithmen kann dieser Konflikt nicht erkannt werden.

Die Auflösung der Konflikts kann daher nur bei Topologieänderungen durch Mobilität der Knoten erfolgen. In größeren, dicht gepackten Netzen ist aufgrund des höheren Knotengrades die Wahrscheinlichkeit für das Auftreten dieses Konfliktszenario relativ gering. In der in dieser Arbeit verwendeten Testumgebung (siehe Kap.

5) trat der Konflikt aufgrund der verwendeten Kettentopologie des öfteren auf und konnte aufgrund der verwendeten statischen Topologie auch nicht durch Mobilität aufgelöst werden.

Daher wurde ein zusätzliches PDAD-Verfahren (PDAD-NH+, siehe Abschnitt 3.2.5) entworfen und implementiert, das Konflikte dieser Art anhand von Inkonsistenzen zwischen empfangenen HELLO und TC Paketen vermuten[8] kann und daraufhin die Weiterleitung des LSP durch PACMAN veranlasst. Die eigentliche Erkennung und Auflösung des Konflikts erfolgt dann durch den Knoten, an den das LSP weitergeleitet wurde mittels PDAD-NH.

Allerdings können auch mit Hilfe von PDAD-NH+ nicht alle Konflikte dieser Art erkannt werden. Ein Beispiel für einen Konflikt, der nicht mittels PDAD-NH+ aufgelöst werden kann, zeigt Abb. 4.10.

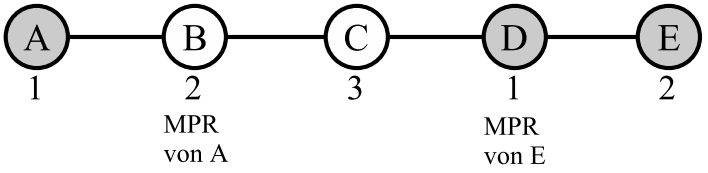

Abbildung 4.10: Konflikterkennug mit PDAD-NH+ nicht möglich

Ein solcher Konflikt ist jedoch während der ca. 150 Autokonfigurationsdurchläufe, die im Rahmen der Leistungsmessungen für diese Studienarbeit durchgeführt wurden, nie aufgetreten.

4.9.2 Ungültige Einträge im ARP-Cache

Bei der praktischen Überprüfung der Implementierung kam die in Kap. 5 beschriebene Testumgebung zum Einsatz. Teil der Testumgebung war der Netzwerktopologieemulator `wnte`. Damit konnten die gewünschten Netzwerktopologien über iptables MAC-Filter realisiert werden.

Da iptables nur zur Manipulation von IP-Paketen geeignet ist, können ARP Anfragen auf diese Weise nicht unterdrückt werden. Sofern alle IP-Adressen des Netzes eindeutig sind, hat diese Einschränkung auch keine weiteren Konsequenzen. Sofern jedoch eine Adresse doppelt vorkommt, kann es wie in Abb. 4.11 zu ungültigen Einträgen im ARP-Cache kommen.

In diesem Beispiel wird das ARP Reply des Knoten D von allen Knoten empfangen, da sich allen Knoten in der Sendereichweite von D befinden. Die dargestellte Topolgie wird nur für Schicht 3 Pakete mittels iptables MAC-Filter emuliert. Falls Knoten B jetzt ein IP-Paket an seinen Nachbarn A schickt, wird dieses aufgrund des ARP Reply von D an die MAC-Adresse von D adressiert. Da Knoten D jedoch auf Schicht 3 kein direkter Nachbar von Knoten B ist, geht das Paket verloren.

[8] die gleichen Inkonsistenzen können auch durch hohe Netzauslastung hervorgerufen werden

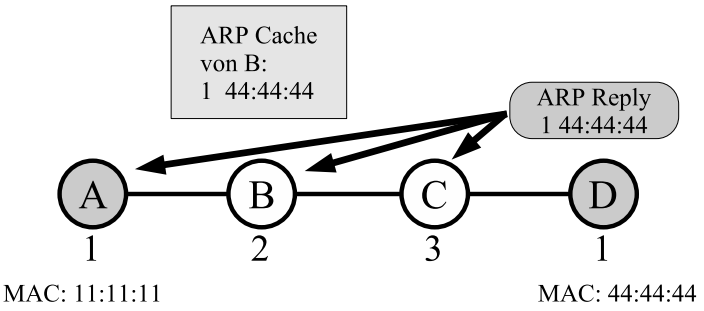

Abbildung 4.11: Ungültiger ARP-Eintrag bei B durch iptables MAC-Filter

Das Problem wurde durch die Implementierung des „Passive Address Resolution"-Moduls in `packet_input.c` gelöst. Durch den Empfang von (Broadcast-)LSPs erhält ein Knoten somit ständig aktuelle MAC und IP-Adressen Tupel über seine Nachbarknoten mit denen der ARP-Cache aktualisiert werden kann. Für jedes empfangene LSP erfolgt die Aktualisierung der ARP-Cache durch Aufruf der Funktion `update_arp()`.

4.9.3 Routing der ACN-Nachrichten

Wird ein Konflikt von einem Zwischenknoten erkannt, schickt dieser eine `ACN` an die Konfliktadresse. Um bestehende Routen zu schützen, sollte die Nachricht an den Knoten geschickt werden, der die Adresse kürzere Zeit verwendet hat, also über den Knoten, von dem das konfliktbehaftete LSP empfangen wurde.

Falls die Nachricht einfach nur an die Konfliktadresse adressieren würde, gelänge diese jedoch (unter Verwendung der Routingtabelle des Kernels) zu dem Knoten, der seine Adresse schon längere Zeit besitzt.

Eine einfache Möglichkeit die Routingtabelle zu umgehen, besteht in der Verwendung von „Source Routing". Es können zwei Arten von Source Routing unterschieden werden. Beim „Strict Source Routing" (SSRR) muss jeder einzelne Zwischenknoten der Route vom Sender spezifiziert werden, wogegen beim „Loose Source Routing" (LSRR) nur einzelne Zwischenknoten, die auf der Route liegen müssen, angegeben werden.

Abhängig von der Entfernung des Zielknoten kommen in der Implementierung beide Arten zur Anwendung. Falls der Zielknoten ein direkter Nachbar ist, wird die Nachricht mit SSRR versendet und als einziger Zwischenknoten die Zieladresse selbst angegeben.

Sollte der Zielknoten mehrere Hops entfernt sein, wird die Nachricht mir LSRR unter der Angabe des ersten Zwischenknoten verschickt. Die Adresse des Zwischenknoten entspricht dabei der Absenderadresse des letzten empfangenen LSP.

Wie bereits erwähnt können, um eine unnötige Netzbelastung durch Signalisierungsverkehr zu vermeiden, `ACNs` von Zwischenknoten verworfen werden, falls vor kurzem bereits ein `ACN` mit der gleichen Zieladresse weitergeleitet wurde. Das Verwerfen

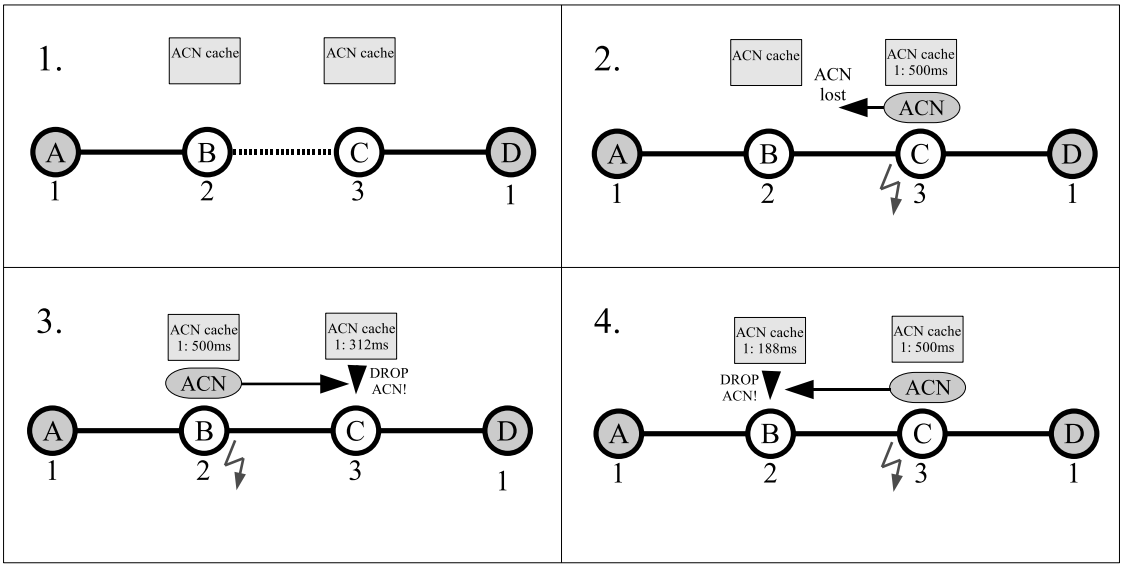

Abbildung 4.12: Gegenseitige Blockierung beim Verwerfen von `ACNs`

von `ACNs` kann aber, wie im Beispiel in Abb. 4.12 gezeigt, zu einer gegenseitigen Blockierung führen.

Dort kommt es im ersten Schritt zu einer Netzverschmelzung die zu einem Konflikt mit Adresse 1 führt. Dieser wird in Schritt zwei von Zwischenknoten C erkannt, der daraufhin ein `ACN` über Knoten B[9] versendet wird. Diese Nachricht geht jedoch auf dem Weg zu Knoten B verloren. In Schritt drei wird der Konflikt jetzt auch von Zwischenknoten B erkannt. Dieser schickt nun ebenfalls ein `ACN` über Knoten C. Diese Nachricht wird jedoch von C aufgrund des Eintrags im Duplicate Cache verworfen. Im weiteren Verlauf verwerfen Knoten B und C jeweils gegenseitig Ihre `ACNs` – es kommt zur Verklemmung.

In der Implementierung wurde das Problem durch einen zusätzlichen Eintrag im Duplicate Cache vermieden, der eine Unterscheidung zwischen gesendeten und weitergeleiteten `ACNs` ermöglicht.

Im dargestellten Beispiel würde Knoten C im dritten Schritt erkennen, dass das `ACN` von Knoten B in entgegengesetzter Richtung zum eigenen `ACN` weitergeleitet wird und somit nicht verworfen werden darf.

4.9.4 Empfang eigener Broadcastpakete

Die Linux Netzwerk-Architektur sieht vor, dass Pakete an eine Broadcast-Adresse auch von denjenigen Knoten empfangen werden, die das Paket selbst gesendet haben. Dies würde in der PACMAN-Implementierung zu Problemen führen, da der Empfang eines solchen Pakets fälschlicherweise zu einer Konflikterkennung durch PDAD-SA führen würde. Ein weiteres Problem wäre der Empfang eigener `LIST_REQ` Nachrichten, der unnötigerweise zum Versand der eigenen Belegungstabelle führen würde.

Um dieses Problem zu umgehen, werden im Kernelmodul `kpacman.c` alle eigenen PACMAN- und Routingprotokollpakete, unter Zuhilfenahme der Netfilter-Architektur

[9] da der Konflikt durch ein von Knoten B weitergeleitetes LSP erkannt wurde

markiert (`(*skb)->nfmark = 1`). Ein so markiertes Paket wird beim Empfang durch das Kernelmodul direkt verworfen.

4.9.5 Linux Source Address Selection

Ein weiteres Problem stellte der „Source Address Selection"-Algorithmus des Linux Kernels dar. Alle PACMAN-Nachrichten sollten als Absenderadresse die Adresse der per Kommandozeile spezifizierten Netzwerkschnittstelle tragen. Falls noch keine Adresse konfiguriert wurde, sollte `0.0.0.0` als Absenderadresse verwendet werden. Insbesondere im Fall das noch keine Adresse konfiguriert wurde, wählte der Kernel des öfteren die Adresse einer anderen Netzwerkschnittstelle als Absenderadresse.

Um dies zu umgehen, wird für jede gesendete PACMAN-Nachricht die Absenderadresse unter Verwendung von `sendmsg()` und `IP_PKTINFO` explizit angegeben.

4.9.6 Vorgehen bei Adresswechseln

Zur Auflösung eines Adresskonflikts muss die Adresse der Netzwerkschnittstelle im laufenden Betrieb geändert werden. Dies stellt vor allem ein Problem für die verwendeten Routingdaemon dar, da diese die konfigurierte Adresse in der Regel nur einmal beim Start des Daemon auslesen.

Daher wurde die Möglichkeit geschaffen den zu verwendenden Routingdaemon per Kommandozeile (siehe Anhang A) an PACMAN zu übergeben. In diesem Fall übernimmt PACMAN den Neustart des Routingdaemon nach jedem Adresswechsel. Da viele Routingsdaemon beim Neustart eine leere Routingtabelle erwarten, wird die Routingtabelle von PACMAN vor dem Start des Daemon gelöscht.

5. Evaluierung der Implementierung

In diesem Kapitel werden die verwendete Testumgebung sowie die durchgeführten Funktionstests beschrieben. Abschließend werden die Ergebnisse der durchgeführten Leistungsmessungen präsentiert und erläutert.

5.1 Testumgebung

Die Implementierung wurde in einem Testnetz bestehend aus sechs IPAQ Pocket PCs und 802.11b WLAN-Karten praktisch getestet. Als Ad-hoc Routingprotokolle wurden die OLSR[1] Implementierung von INRIA sowie die FSR[2] Implementierung von ATR verwendet.

Für die Messungen mit OLSR wurde eine Kombination der PDAD-Verfahren SA, SN, SND, SNE und NH verwendet. Aufgrund unidirektionaler Links konnte bei FSR das PDAD-Verfahren NH nicht verwendet werden, wodurch dort nur PDAD-SN, SND und SNE zum Einsatz kamen. Die verwendeten PACMAN-Parameter und Schwellwerte sind in Tabelle 5.1 dargestellt. Bei den Routingprotokolldaemonen wurden die Standardparameter der jeweiligen Implementierung verwendet. Das Sendeintervall bei OLSR betrug für HELLOs 2s und für TCs 6s. Bei FSR wurden zwei Reichweitenbereiche verwendet. Dort wurden die Sendeintervalle 5s für nahe gelegene und 15s für entfernte Knoten verwendet. Bei beiden Protokollen wurde die initiale Sequenznummer zufällig gewählt.

Die Realisierung unterschiedlicher Netztopologien erfolgte mit Hilfe des „Wireless Network Topology Emulator“. Das Werkzeug `wnte`[BaIW] emuliert die gewünschte Topologie unter Verwendung von iptables MAC-Filtern und stellt diese grafisch dar.

5.2 Funktionale Analyse

Um die korrekte Implementierung der einzelnen PDAD Verfahren überprüfen zu können, wurde jeweils nur das zu prüfende Verfahren aktiviert und manuell ein Adress-

[1] OLSRv3 von http://hipercom.inria.fr/olsr/

[2] Version 0.3.6 von http://www.acr.atr.co.jp/acr/general/product/GSRFSR/

Parameter	Wert	Erläuterung
`LISTWAIT_INT`	200ms	Wartezeit nach dem Senden eines `LIST_REQ`
`ACN_TIME`	500ms	Haltezeit in der Duplicate Notification Table
OLSR `sn_thres`	50	Schwellwert für PDAD-SND
OLSR `nh_thres`	15s	Schwellwert für PDAD-NH
OLSR `max_ips` TC	6s	max. Intervall zwischen zwei TCs
OLSR `max_ips` HELLO	3s	max. Intervall zwischen zwei HELLOs
FSR `sn_thres`	50	Schwellwert für PDAD-SND
FSR `max_ips`	16s	max. Intervall zwischen zwei LSP

Tabelle 5.1: In der Testumgebung verwendete PACMAN-Parameter

konflikt erzeugt, der von diesem Verfahren erkannt werden konnte. Des weiteren wurde geprüft,ob in konfliktfreien Szenarien auch bei häufigen Topologieänderungen nicht fälschlicherweise Konflikte erkannt wurden.

Der Versand von Signalisierungsnachrichten wurde durch Mithören des Datenverkehrs mittels `ethereal` überprüft und nachvollzogen. Das häufige Szenario, dass ein neuer Knoten zu einem bestehenden Netz hinzukommt, wurde ausgiebig getestet und hierbei die korrekte Übertragung der Belegungstabelle verifiziert.

In allen Fällen erfolgte die Konflikterkennung und Auflösung fehlerfrei.

5.3 Leistungsmessungen

Zur Beurteilung der Leistungsfähigkeit wurden mit der obigen Testumgebung mehrere Messungen der Konfliktauflösezeit durchgeführt. Dabei wurde die Zeit und die Zahl der Adresswechsel gemessen, die benötigt wurden bis alle Knoten mit einer eindeutigen Adresse konfiguriert waren. Die Autokonfiguration wurde dabei sowohl bei allen Knoten gleichzeitig als auch zufällig nacheinander in einem Intervall von 15s durchgeführt.

Zum Einsatz kamen die beiden statischen Topologien aus Abb. 5.1.

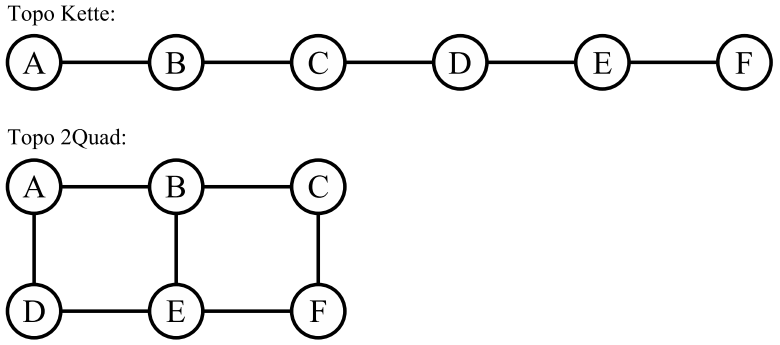

Abbildung 5.1: Eingesetzte Topologien

Die Messungen wurden jeweils mit den Konfliktwahrscheinlichkeiten $p_c = 0.5$ und $p_c = 1.0$ durchgeführt (siehe auch Gleichung 3.3). Die folgenden Schaubilder zeigen

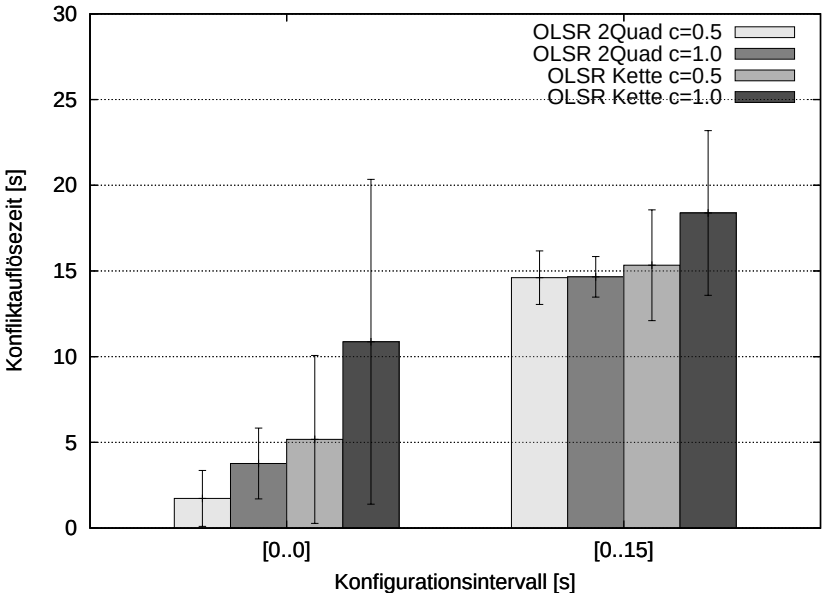

Abbildung 5.2: Messergebnisse Konfliktauflösezeit mit OLSR

den Durchschnittswert und das 90%-Konfidenzintervall von je fünf Messungen mit unterschiedlichen Startwerten der Zufallsgeneratoren.

Die beiden Abb. 5.2 und 5.3 zeigen die Ergebnisse der Messungen mit OLSR. Der linke Teil des Schaubildes zeigt die Versuche, bei denen alle Knoten gleichzeitig gestartet wurden. Deutlich erkennbar ist der direkte Zusammenhang zwischen geringerer Konfliktwahrscheinlichkeit und kürzerer Konfliktauflösezeit. Eine geringere Konfliktwahrscheinlichkeit führt jedoch auch zu einem weniger dicht gepackten Adressraum und verhindert somit eine effiziente Adresskomprimierung.

Des weiteren führt die Topologie „Kette“ im Vergleich mit „2Quad“ meist zu schlechteren Ergebnissen. Ein Grund hierfür dürfte im geringeren Durchmesser des Graphen „2Quad“ und in der damit verbundenen schnelleren Ausbreitung von Routinginformationen liegen. Des weiteren werden Konflikte mit Hilfe von PDAD-NH aufgrund der größeren Menge an Nachbarn schneller erkannt. Sofern die Knoten nacheinander konfiguriert werden, ist der Graph in diesem Fall auch zu einem früheren Zeitpunkt verbunden. Aufgrund des höheren Knotengrads erhält ein neuer Knoten hier auch mit höherer Wahrscheinlichkeit eine vollständige Belegungstabelle durch die `LIST_REP` Nachrichten seiner Nachbarn. Ein Beispiel hierfür zeigt der rechte Teil von Abb. 5.3: Bei $p_c = 1.0$ treten im Fall von Topologie „2Quad“ deutlich weniger Konflikte auf.

Beim Vergleich der beiden Teilblöcke von Abb. 5.2 scheint die Konfliktauflösung in den Szenarien, in denen die Knoten nacheinander gestartet werden, wesentlich länger zu dauern. Da das Netz aber frühestens vollständig (und konfliktfrei) konfiguriert sein kann, wenn auch der letzte Knoten gestartet wurde, hängt die dargestelle Konfliktauflösezeit insbesondere vom Startzeitpunkt des letzten Knoten ab.

Die durchschnittliche Zeitdauer bis alle Knoten gestartet wurden ist somit der Erwartungswert über das Maximum der einzelnen Startzeitpunkte. Im unserem Fall mit $n = 6$ Knoten und einer Intervallgröße von $b = 15s$ ergibt sich mit Gleichung 5.1 eine durchschnittliche Dauer von $12,86s$.

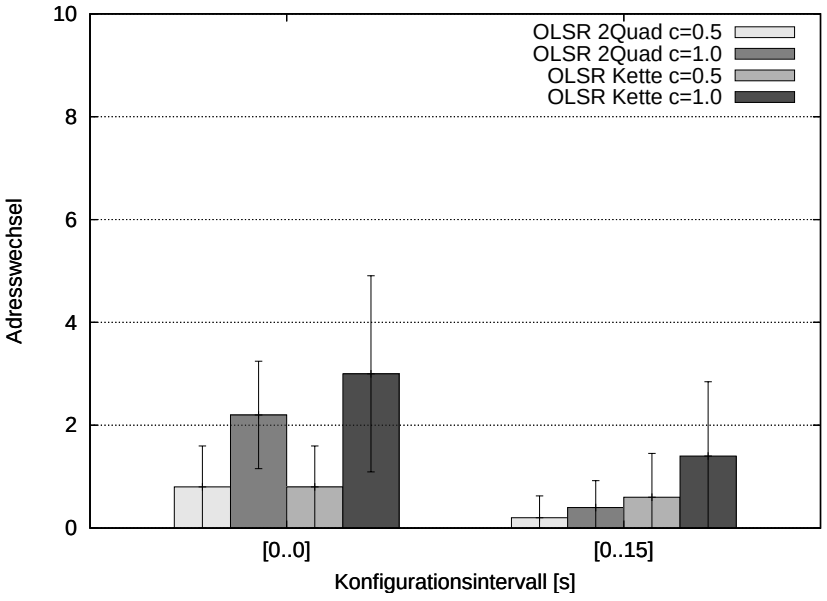

Abbildung 5.3: Messergebnisse Adresswechsel mit OLSR

$$E(X) = \frac{b \cdot n}{n+1} \tag{5.1}$$

Um also die reine Konfliktauflösezeiten der beiden Teilblöcke von Abb. 5.2 miteinander vergleichen zu können, müssen daher vom rechten Teilblock 12.86s abgezogen werden. Die Konfliktauflösezeit ist hier also wesentlich kürzer. Dies legt auch Abb. 5.3 nahe, da im Fall, dass die Knoten nacheinander gestartet wurden, die Zahl der Adresswechsel relativ gering ist. Dies liegt daran, dass ein neu gestarteter Knoten hier in der Regel auf die Belegungstabelle seiner Nachbarn zurückgreifen kann und somit Adresskonflikte gar nicht erst auftreten.

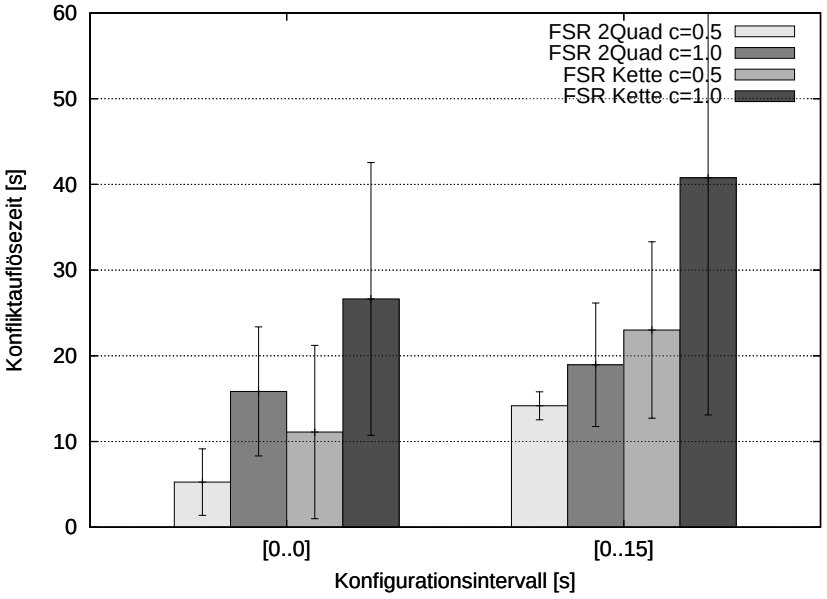

Abbildung 5.4: Messergebnisse Konfliktauflösezeit mit FSR

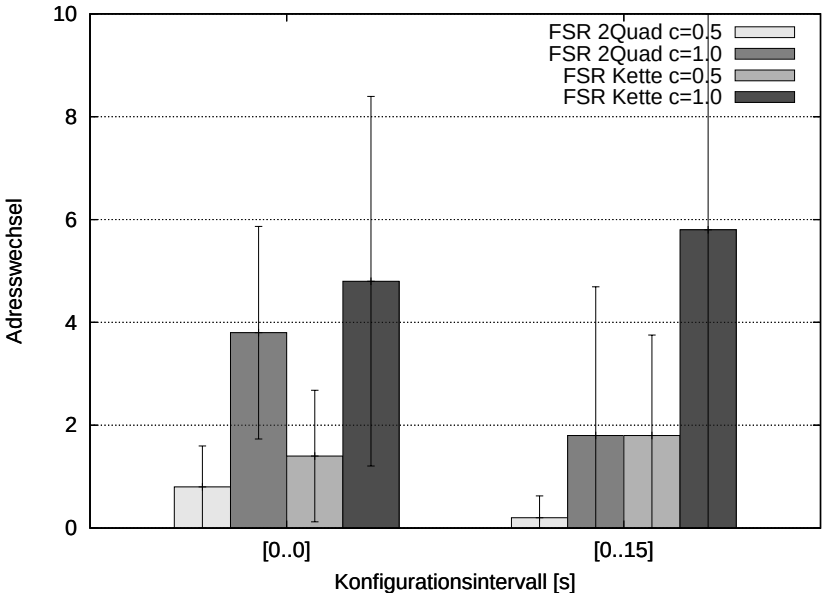

Abbildung 5.5: Messergebnisse Adresswechsel mit FSR

Die Abbildungen 5.4 und 5.5 zeigen die entsprechenden Messergebnisse bei der Verwendung von FSR. Die Ergebnisse sind den OLSR-Ergebnissen relativ ähnlich. Allerdings fällt auf, dass die Konfliktauflösung bei FSR zum Teil wesentlich länger dauert als bei OLSR. Dies liegt an der langsameren Ausbreitungsgeschwindigkeit der Routingprotokollinformationen bei FSR, da LS aggregiert und Informationen über weit entfernte Knoten nur selten versendet werden (siehe 2.3.2), sowie an der Tatsache, dass PDAD-NH aufgrund der Möglichkeit unidirektionaler Links nicht verwendet werden kann.

Zum Teil führt die langsame Ausbreitungsgeschwindigkeit auch dazu, dass im Fall eines Adresskonflikts lange Zeit nachdem einer der beiden Knoten seine Adresse bereits geändert hat und auch der zweite Knoten seine Adresse noch ändert, wodurch erneut ein Konflikt erzeugt werden kann.

Wichtigstes Ergebnis der Messungen ist jedoch die Tatsache, dass in jedem Versuchsdurchlauf alle Konflikte erkannt und aufgelöst wurden. Angesichts der Tatsache, dass es sich bei den getesteten Szenarien um „worst-case“ Szenarien handelt (insbesondere: Zeitgleiche Autokonfiguration des gesamten Netzes bei minimalem Adressraum bei Kettentopologie) sind auch die gemessenen Konfliktauflösezeiten ein gutes Ergebnis und sind oft nahe an der (ausbreitungsbedingt) minimal möglichen Konfliktauflösezeiten.

6. Zusammenfassung und Ausblick

Bei PACMAN handelt es sich um eine modulare Architektur zur passiven Autokonfiguration mobiler Ad-hoc-Netze. Im Gegensatz zu herkömmlichen Autokonfigurationverfahren kann die Autokonfiguration bei PACMAN rein passiv durch Analyse von Routingprotokollpaketen erfolgen. Eine hervorstechende Eigenschaft der PACMAN-Architektur stellt der reibungslose Umgang mit Netzverschmelzungen dar.

Im Rahmen der Studienarbeit wurden diejenigen Komponenten der PACMAN-Architektur implementiert, die für den praktischen Einsatz zur Autokonfiguration bei Verwendung der Protokolle OLSR oder FSR benötigt werden. Da in seltenen Fällen bei der Verwendung von OLSR Konflikte mit den in [Weni03] vorgestellten PDAD-Verfahren nicht aufgelöst werden konnte, wurde noch ein weiteres PDAD-Verfahren[1] entworfen und implementiert. Theoretisch sind aber auch dann noch in sehr seltenen Fällen Konflikte denkbar, die nur durch Mobilität aufgelöst werden können. Dieser Fall ist jedoch während der durchgeführten Leistungsmessungen nicht aufgetreten.

Die Messungen in der Testumgebung haben gezeigt, dass PACMAN unter den getesteten Bedingungen problemlos die Autokonfiguration des kompletten Netzes in kurzer Zeit ermöglicht und somit einen vielversprechenden Ansatz zur Autokonfiguration mobiler Ad-hoc-Netze darstellt. Als nächster Schritt folgt die Wiederholung der Messungen in einer größeren Testumgebung.

In weiteren Arbeiten kann dann noch die Erweiterung der Implementierung um eine Unterstützung von „on demand“ Routingprotokollen sowie um Module zur Adresskomprimierung und zum Adresswechselmanagement erfolgen.

[1] PDAD-NH+

A. Kommandozeilenparameter

```
Usage: ./pacman [OPTIONS] [/path/rt_daemon [rt_opt1] [rt_opt2]...]

-a, --enable-sa          enable PDAD-SA

-b, --brdcst-list-req    send broadcast LIST_REQ
                         message (implies -u)

-c, --probconf=FLOAT     probability that an address
                         conflict occurs

-d, --debug=MASK         set debug level to MASK

-e, --est-node-code      estimated number of nodes
                         in the network

-h, --help               display this help and exit

-i, --device=DEV         set network device to DEV

-l, --enable-lp[=VAL]    enable PDAD-LP with parameter VAL

-m, --enable-mpr         enable PDAD-NH+

-n, --enable-nh[=VAL]    enable PDAD-NH with parameter VAL

-p, --protocol=PROTO     set routing protocol to PROTO
                         valid protcols: OLSR, FSR

-r, --seed=VAL           set random number generator
                         seed to VAL (for debugging)

-s, --enable-sn[=VAL]    enable PDAD-SN with parameter VAL
```

```
--max-ips-t1=VAL          set maximum inter-packet spacing
                          for HELLOs to VAL

--max-ips-t2=VAL          set maximum inter-packet spacing
                          for TCs to VAL

-u, --autoconf[=VAL]      perfom autoconfiguration
                          (after listening for VAL ms)

-V, --version             print version information and exit
```

Literatur

[BaIW] I. Baumgart, Y. Iskenderoglu und K. Weniger. The Wireless Network Topology Emulator wnte. http://sourceforge.net/projects/wnte/.

[ChAG04] S. Cheshire, B. Aboba und E. Guttman. Dynamic Configuration of Link-Local IPv4 Addresses. Draft, IETF, 2004.

[CJLM+01] T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, A. Qayyum und L. Viennot. Optimized Link State Routing Protocol. In *IEEE INMIC Pakistan 2001*, 2001.

[Drom93] R. Droms. Dynamic Host Configuration Protocol. RFC, IETF, 1993.

[PeGC00] G. Pei, M. Gerla und T.-W. Chen. Fisheye State Routing: A Routing Scheme for Ad Hoc Wireless Networks. In *Proceedings of ICC 2000*, 2000.

[PeRD00] C. Perkins, E. Royer und S. Das. IP Address Autoconfiguration for Ad Hoc Networks. Draft, IETF, 2000.

[Vaid02] N. H. Vaidya. Weak duplicate address detection in mobile ad hoc networks. In *Proceedings of ACM MobiHoc 2002*, 2002.

[Welt] H. Welte. The Linux Netfilter Project. http://www.netfilter.org/.

[Weni03] K. Weniger. Passive Duplicate Address Detection in Mobile Ad Hoc Networks. WCNC 2003, New Orleans, USA, Mar 2003. IEEE.

[Weni04] K. Weniger. Passive Autoconfiguration of Mobile Ad Hoc Networks. Telematics Technical Reports TM-2004-1, 2004.