Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Olaf Christ

Entwurf und Analyse eines sicheren Routingprotokolls für mobile Quellen in Source Specific Multicast Communication

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Olaf Christ

Entwurf und Analyse eines sicheren Routingprotokolls für mobile Quellen in Source Specific Multicast Communication

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Thomas C. Schmidt
Zweitgutachter : Prof. Dr. Bettina Buth

Abgegeben am 19. Oktober 2007

Olaf Christ

Thema der Masterarbeit

Entwurf und Analyse eines sicheren Routingprotokolls für mobile Quellen in Source Specific Multicast Communication

Stichworte

Mobile IPv6, Mobile Source Specific Multicast, Route Optimization, Protocol Security, Cryptographic Identifier, Tree Morphing, OMNeT++, SPIN

Kurzzusammenfassung

Das Problem mobiler Quellen bei Source Specific Multicast Übertragungen ist derzeit weitgehend ungelöst. Die vorliegende Arbeit erweitert den Ansatz des „Tree Morphings“, welches Endgeräten durch die kontinuierliche Adaption von Verteilbäumen einen ununterbrochenen IP-Service bietet. Das eingeführte, leichtgewichtige Protokolldesign wird dabei im Verlauf der Arbeit vollständig systematisch untersucht. Neben der theoretischen Protokoll-Evaluierung und der formalen Verifikation wird eine Protokollimplementierung im Netzwerksimulator OMNeT++ durchgeführt. Diese wird dazu genutzt, die Stärken und Schwächen des Protokolls an bestimmten, vorgegebenen Netzkonstellationen und an realen Topologiedaten zu ermitteln.

Olaf Christ

Title of the paper

Design and analysis of a secure routing protocol for mobile sources in source specific multicast communication

Keywords

Mobile IPv6, Mobile Source Specific Multicast, Route Optimization, Protocol Security, Cryptographic Identifier, Tree Morphing, OMNeT++, SPIN

Abstract

Up until now the problem of mobile sources in Source Specific Multicast communication is largely unsolved. This paper extends the ”Tree Morphing” approach, which provides uninterrupted IP service to nodes by continuously adapting delivery trees. The introduced, lightweight protocol will be systematically and exhaustively examined. In addition to evaluating the protocol theoretically and verifying it formally, a protocol implementation on the network simulator OMNeT++ will be performed. It will be used to determine the strengths and weaknesses of the protocol in special predefined network configurations and by testing it on real topologies.

Inhaltsverzeichnis

1. Einleitung

1.1. Auf dem Weg zur Gruppenkommunikation in einem mobilen Internet

Mobile Internetdienste gewinnen immer mehr an Bedeutung. Nachfragegetrieben ist es heute problemlos möglich, umfangreiche Informationen, z.B. Webseiten, Videos oder Musik aus dem Internet auf ein mobiles Gerät, z.B. ein Handy oder ein Smartphone, zu laden. In Zukunft werden auch Sprach- und Videokonferenzen mit mehreren Teilnehmern möglich sein.

Der mobile Betrieb dieser Geräte wird zur Zeit noch mehrheitlich durch die Telefonnetzbetreiber sichergestellt, welche Handover zwischen Funkzellen innerhalb ihrer Netze bereitstellen. Dabei muss das Netz den Teilnehmer aktiv von einer Funkzelle zur nächsten übergeben. Weiterhin müssen auch Funkzellen, die verschiedene Zugangstechnologien bieten, diese Handover untereinander synchronisieren, damit eine Übergabe zwischen GSM und UMTS stattfinden kann.

Ein Wechsel in ein Netzbetreiber-fremdes Funknetzwerk, z.B. ein privates oder öffentliches Wireless LAN (WLAN), wird durch die aktuelle Technologie allerdings nicht unterstützt, obwohl Handys der neuesten Generation mit WLAN-Schnittstellen ausgestattet sind und den Voice over Internet Protocol (VoIP)-Standard Session Initiation Protocol (SIP) unterstützen. Damit könnten Gespräche vom kostenpflichtigen GSM/UMTS-Netz beim Auffinden eines WLANs in dieses, meist wesentlich günstigere Netz, übernommen werden. Durch das Anmelden am WLAN wird dem Endgerät jedoch eine neue IP-Adresse zugewiesen. Laufende Gespräche oder Datenverbindungen würden somit beim Verlassen des alten Netzes abbrechen. Dieser Adresswechsel findet z.B. auch beim Handover zwischen WLANs unterschiedlicher Betreiber und beim Roaming zwischen GPRS-Operatoren statt. Obwohl GSM und UMTS-basierte Gespräche beim Roaming zwischen Operatoren, die ein Roaming-Abkommen geschlossen haben, weitergeführt werden können, sind GPRS-basierte VoIP-Handover nicht möglich, da dabei der IP-Backbone und damit die IP-Adresse gewechselt wird.

Um den Transportschichten bei einem IP-Adresswechsel einen transparenten IP-Service zu bieten, wurde das Protokoll Mobile IP in [43, 76] standardisiert. Da dieses im Ende-zu-Ende Paradigma arbeitet, ist es Provider-unabhängig. Somit können bestehende Kommunikationsverbindungen beim Wechsel in beliebige IP-Netze weitergeführt werden - unter der Voraussetzung, dass die Netzwerk-Infrastruktur die Mobile-IP Pakete nicht an Firewalls herausfiltert.

Dieses Protokoll beschränkt sich jedoch im Wesentlichen auf Unicast-Verbindungen und bietet für Multicast lediglich rudimentäre Unterstützung. Da nach einem Handover sämtliche Multicastpakete über einen stellvertretenden Rechner gesendet werden müssen (triangluläres Routing), bietet Mobile IP hierfür keine transparente, routenoptimale Lösung.

Multicast ist genau dann die geeignete Übertragungsart, wenn mehrere Empfänger Ziel eines Datenstroms sind. Klassischerweise sind diese Einsatzszenarien die Übertragung von Medieninhalten wie Live-TV und Radio. IPTV ist derzeit ein vielbeachtetes Thema, das insbesondere Internetprovider in sogenannten „Triple-Play“ Angeboten mit einem Internet- und Telefonanschluß (eventuell über VoIP) bündeln. Würden die Streamingserver der Provider jeden Empfänger einzeln (Unicast) bedienen, müssten viele dieser Server vorgehalten und mit steigender Kundenzahl weitere angeschafft werden. Multicast löst dieses Problem, indem die Server (auf Anfrage) alle verfügbaren Streams in das Netzwerk senden. Bei steigender Empfängerzahl sorgt das Netzwerk dafür, dass die Pakete dupliziert werden.

In Funknetzwerken gibt es noch einen wesentlich signifikanteren Grund, Multicast-Übertragungen einzusetzen. Da UMTS z.B. nicht genügend Bandbreite zur Verfügung stellt, um jeden Empfänger mit einem Unicast IPTV-Stream zu versorgen, wird dort die sogenannte Punkt-zu-Mehrpunkt Übertragung eingesetzt. Auch im Backbone kann Multicast das Netz bei der Verteilung von Inhalten entlasten.

Außer bei IPTV wird Multicast in Zukunft bei vielen anderen Anwendungen genutzt werden, wie z.B. Internet Radio, Spiele und Telefon- wie Video-Konferenzen. Teilnehmer an Spielen und Konferenzen, welche selbst der Ursprung von Datenströmen sind, werden dabei auch mobile Geräte nutzen, die beim Handover ihre IP-Adresse ändern können. Um weiterhin den gewünschten Multicast-Service bieten zu können, muss das Netz für ein reibungsloses IP-Layer Handover sorgen. Da Multicast-Übertragungen allerdings auf Verteilbäumen und diese wiederum auf dem topologischen Anschlusspunkt des Gerätes basieren, muss eine Lösung gefunden werden, mobile Geräte in Multicast-Übertragungen einzubinden. Die vorliegende Arbeit betrachtet den speziellen Fall der Mobilität des Senders einer Multicastübertragung, bei dem eine einzelne Quelle an mehrere Empfänger sendet, das sogenannte Source Specific Multicast (SSM).

1.2. Ausgangssituation

Das Problem der mobilen SSM-Quellen ist derzeit weitgehend ungelöst. Ein quellspezifischer Sender, welcher mit dem Netzübergang seine Adresse wechselt, wird von den Quelladressfiltern des bisherigen Multicast-Verteilbaums abgewiesen, ohne welchen er seine Empfänger jedoch nicht mehr erreichen kann. Ein Ansatz diesem Problem zu begegnen, das sogenannte Tree Morphing, wurde von Schmidt und Wählisch erstmals in [83] präsentiert. Die Routerinfrastruktur adaptiert dabei den Verteilbaum bei jedem Handover und sorgt somit dafür, dass die Quelle kontinuierlich Multicastdaten senden kann und diese auch die Empfänger erreichen. Dabei wird - im Gegensatz zu anderen Lösungen - kein neuer Verteilbaum aufgebaut, sondern der alte in einen neuen überführt („gemorphed“).

Da die Empfänger die Gruppe nicht verlassen, können Sie auch unmittelbar nach einem Handover Daten empfangen.

Die bisherige algorithmische Beschreibung des Protokolls stellt das Grundgerüst des Tree Morphings dar. Diese Definition des Grundalgorithmus definiert die Arbeitsweise der Router und der mobilen SSM-Quelle. Des weiteren wurde eine algorithmische und struktuelle Analyse des Algorithmus durchgeführt.

Dabei wurde allerdings noch nicht spezifiziert, wie die für das Tree Morphing nötigen Daten übermittelt werden und welche Sicherheitsmaßnahmen dabei eingesetzt werden sollen. Eine formale Verifikation muss die Fehlerfreiheit des Protokolls nachweisen. Um sicherstellen zu können, dass der Protokollentwurf nicht nur formal sondern auch in einer realen Umgebung korrekt arbeitet, muss eine Implementierung des Protokolls durchgeführt werden. Diese muss weiterhin durch eine strukturierte Analyse auf mögliche Schwachstellen hin überprüft werden, um eine systematisch vollständige Auswertung des Protokolls zu gewährleisten.

1.3. Zielsetzung der Arbeit

Im Rahmen dieser Arbeit soll zunächst ein Übertragungsprotokoll entworfen werden, welches leichtgewichtig ist, also möglichst wenige zusätzlich zu übertragende Pakete einführt. Die mobilitätsbedingt auf die Routerinfrastruktur eingebrachte Last soll dadurch minimal bleiben. Weiterhin sollen Sicherheitsaspekte bereits beim Protokolldesign berücksichtigt werden, um es gegenüber Angreifern und Netzwerkfehlern abzusichern.

Alle weiteren Untersuchungen sollen die Korrektheit und Leistungseigenschaften des Protokolldesigns nachweisen. Dazu soll zunächst eine theoretische Protokoll-Evaluierung erstellt werden, die die Overheads und die Robustheit des Protokolls betrachtet. Um die Fehlerfreiheit des Protokolls sicherzustellen, soll eine formale Verifikation durchgeführt werden. Schließlich wird das Protokoll in einer Simulationsumgebung implementiert, da bestimmte Fehler und Leistungsparameter erst durch eine Implementierung erkannt werden können. Die Implementierung soll einerseits dazu genutzt werden, die korrekte Arbeitsweise des Protokolls an speziell dafür entworfenen Test-Netzwerken mit bestimmten Eigenschaften nachzuweisen, andererseits liefert die Simulation auf realen Topologiedaten wichtige praktische Erkenntnisse über die Leistungsfähigkeit des Protokolls, wie z.B. dessen Konvergenzzeit oder auftretende Paketverluste.

All diese Aufgaben tragen zur vollständigen Untersuchung des Protokolls bei. Die reale Implementierung auf Routerplattformen und der Test im tatsächlichen Internetrouting ist dabei der letzte Schritt, welcher in dieser Arbeit jedoch nicht durchgeführt werden kann.

1.4. Gliederung der Arbeit

Kapitel 2 führt in die Grundlagen der mobilen Kommunikation im Internet der nächsten Generation, der Multicastübertragung, insbesondere SSM, und in weitere für das Protokolldesign benötigte Protokolle ein. Weiterhin werden bestehende Lösungen der Empfänger- und Sendermobilität bei Multicast-Übertragungen vorgestellt. Schließlich wird das Tree Morphing inklusive einer State Machine eingeführt.

Die zur Verifikation und Simulation nötigen Methoden und Programme werden in Kapitel 3 beschrieben.

Kapitel 4 stellt zunächst die Anforderungen an das Protokolldesign dar. Es folgen zwei Protokollentwürfe und eine Diskussion, welcher von diesen besser geeignet ist. Zudem werden die Arbeitsweisen der einzelnen am Tree Morphing beteiligten Knoten beschrieben.

Das Protokolldesign wird in Kapitel 5 auf seine Prozessierungs- und Verarbeitungsaufwände hin untersucht. Des weiteren wird die Robustheit gegenüber Netzwerkfehlern und Angriffen analysiert.

Die formale Verifikation wird in Kapitel 6 behandelt, welches zunächst ein Konzept und den Verifikations-Quelltext darstellt. Die entstandenen Ergebnisse und ein Ausblick auf weitere mögliche formale Verifikationen schließen das Kapitel.

Kapitel 7 beschreibt die benötigten Konzepte des Simulators OMNeT++, wie den Paketfluß und die zur Protokollimplementierung nötigen Änderungen. Dazu zählen die Implementierung des PIM-SSM und MLD-Protokolls und die für das Tree Morphing benötigten Header. Abschließend werden markante Quelltext-Ausschnitte der Implementierung der Tree Morphing Algorithmen vorgestellt.

Die Simulation des Tree Morphing Protokolls und ihre Ergebnisse werden in Kapitel 8 dargestellt. Zunächst wird die Korrektheit der Implementierung überprüft, um dadurch resultierende fehlerhafte Ergebnisse auszuschließen. Anschließend werden die implementierten Analyse-Funktionen und die definierten Test-Netzwerke vorgestellt. Schließlich werden die gewonnenen Ergebnisse analysiert und diskutiert.

Kapitel 9 fasst die durchgeführten Arbeiten zusammen und bietet einen Ausblick auf weitere mögliche Bearbeitungsschritte.

2. Einführung in die mobile Gruppenkommunikation

Dieses Kapitel dient der Einführung in die Techniken und Protokolle, die im Rahmen dieser Arbeit verwendet werden.

2.1. Mobile IPv6

Nicht nur die Anzahl der mit dem Internet verbundenen stationären Rechner, sondern auch die Anzahl der mobilen Geräte mit Internetzugang (Handys, Smartphones, Laptops, etc.) steigt ständig. Um der drohenden Adressknappheit entgegenzuwirken, wurde IPv6 [22] als designierter Nachfolger des aktuell verbreiteten IPv4 [77] eingeführt. Mobile Knoten profitieren damit nicht nur von den vorgesehenen Autokonfigurationsmechanismen [96], sondern auch von Mobile IPv6 [43][1]. Mit Hilfe dieses Protokolls können mobile Knoten ihre bestehenden Kommunikationsverbindungen trotz IP-Adresswechsels, z.B. durch ein Handover von einem Netzwerkzugangspunkt (Access Point, Base Station, etc.) zum nächsten, aufrecht erhalten. Mobile IPv6 bildet die Grundlage des im weiteren Verlauf der Arbeit untersuchten Tree Morphing Protokolls und soll im Folgenden vorgestellt werden.

Ziel des Protokolls ist es, einen für die Transport- und höhere Schichten transparenten Adresswechsel zu bieten. Dazu besitzt ein mobiler Knoten (Mobile Node (MN)) zwei IP-Adressen: die Home Address (HoA) und die Care-of Address (CoA). Eine gültige CoA wird dem MN aus dem jeweils aktuellen Netzwerk zugewiesen und verändert sich damit bei einem Handover. Die sogenannte previous CoA (pCoA) bezeichnet dabei die vor dem Handover benutzte, die next CoA (nCoA) entsprechend die aktuelle CoA. Die HoA ist eine feste Adresse, die der MN im Heimnetzwerk benutzt. Sie verändert sich bei einem Handover nicht. Zusätzlich gibt es noch die Entitäten previous Access Router (pAR), next Access Router (nAR), Correspondent Node (CN) und Home Agent (HA). Der previous bzw. next Access Router bezeichnet analog zu pCoA und nCoA den vorhergehenden bzw. aktuellen Router, der dem MN als First Hop ins Internet dient. Der CN ist ein Kommunikationspartner im Internet, der mit dem MN in einer Kommunikationsbeziehung steht. Ein MN kann also mehrere CNs haben. Der Home Agent schließlich ist ein Dienst, der meist auf einem Router im Heimnetz des MN implementiert ist und der eine besondere Aufgabe übernimmt. Befindet sich der MN nicht im Heimnetz, nimmt der HA Anfragen an die HoA des MN entgegen und leitet sie an die aktuelle CoA des MN weiter. Dazu muss

[1] Autokonfiguration [17] und Mobile IP [76] wurden nachträglich auch für IPv4 entwickelt. Jedoch ist z.B. die automatische Konfiguration von IP-Adressen dabei auf Link-lokale Adressen beschränkt.

der MN seinem HA durch ein sogenanntes Binding Update (BU) zunächst seine aktuelle CoA mitteilen. Dieses BU wird nach jedem Handover gesendet. Des weiteren werden auch alle CNs durch BUs von der neuen CoA in Kenntnis gesetzt. Der HA und alle CNs halten die HoA-CoA Zuordnung in einem sogenannten Binding Cache (BC) fest. Abbildung 2.1 zeigt ein Handover von einem Router (pAR) zum nächsten (nAR). Zusätzlich existiert ein CN. Abbildung 2.2 zeigt den Protokollablauf nach einem Handover zwischen MN, HA

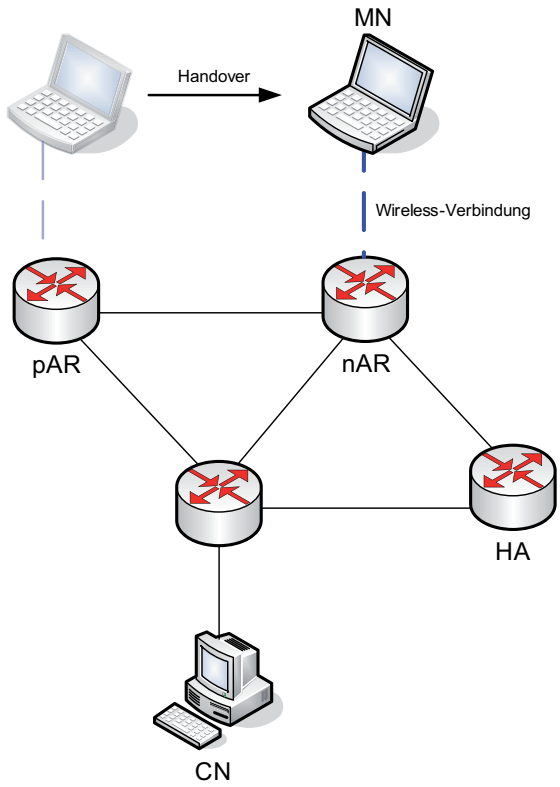


Abbildung 2.1.: Mobile IPv6 Handover

und CN. Im Anfangszustand besteht in diesem Beispiel eine Kommunikationsverbindung zwischen MN und CN. Nach dem Handover bildet der MN zunächst eine neue CoA und sendet dann die BUs an seinen HA und den CN. Von beiden erhält er als Antwort die sogenannte Binding Update Acknowledgement (Back) Nachricht. Die Kommunikationsverbindung kann nun weitergeführt werden.

Um die IP-Layer Mobilität vor höheren Schichten zu verstecken, wird die HoA als feste (und einzige) IP-Adresse an die Transportschicht weitergereicht. Wird ein Datenpaket mit der Zieladresse HoA von der Transportschicht eines CN an den Mobile IP-Layer weitergereicht, überprüft dieser, ob es einen Eintrag im Binding Cache für die HoA gibt. In diesem Fall wird ein Source Routing an die zur HoA passende CoA des MN initiiert. Existiert kein Eintrag im BC, befindet sich der MN in seinem Heimnetz und ist direkt über die HoA erreichbar. An CNs eintreffende Pakete werden ebenfalls vom Mobile IP-Layer untersucht. Enthalten diese einen Mobility Destination Option Header mit der HoA wird dieser Headereintrag zunächst aus Sicherheitsgründen gegen den Eintrag im BC verglichen. Stimmen die Einträge überein, wird der Mobility Destination Option Header entfernt und das Paket an die Transportschicht weitergeleitet. Diese kommuniziert somit

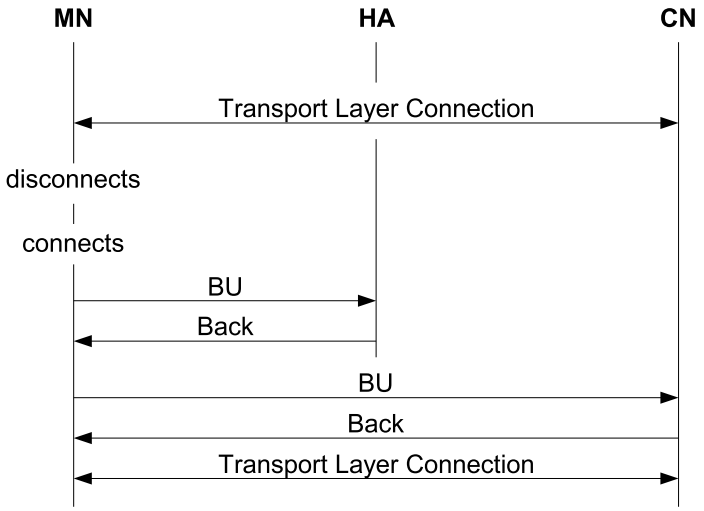


Abbildung 2.2.: Protokollablauf Mobile IPv6 zwischen Mobile Node, Home Agent und Correspondent Node

– auch nach einem Handover – direkt mit der HoA des MN. Der Mobile IP-Layer sorgt dafür, dass die Pakete zur aktuellen CoA gesendet werden.

Mobile IPv6 ist standardisiert in [43], Ausführungen finden sich in [88].

2.2. Mobile Security

2.2.1. Mobile IPv6 Security Architektur

Die vom MN nach einem Handover gesendeten BU-Pakete eröffnen Angreifern Möglichkeiten, die bestehenden Kommunikationsverbindungen zu übernehmen, sofern diese nicht (auf Transport- oder Anwendungsebene) verschlüsselt oder authentifiziert sind. Verfahren, um die BUs abzusichern, sind im Mobile IPv6-Standard [3, 24, 43] festgelegt und sollen im Folgenden vorgestellt werden.

Um die BUs zwischen MN und HA abzusichern und damit sicherzustellen, dass Angreifer die Verbindungen nicht übernehmen können (Man-in-the-Middle Attacke (MitM)), kann eine IPsec-Verbindung [3, 24, 74] zwischen MN und HA aufgebaut werden. Dies kann entweder durch einen gemeinsamen, geheimen Schlüssel oder durch Zertifikate geschehen. Dadurch kann sichergestellt werden, dass die signierten BUs auf dem Weg zum HA nicht verändert wurden.

Im Gegensatz dazu werden die BUs vom MN zu den CNs grundsätzlich unverschlüsselt und nicht signiert verschickt, da diese spontan in Kontakt treten und sich vorher nicht kennen. Um dennoch ohne kryptographische Mittel nachweisen zu können, dass der MN sowohl Besitzer der HoA als auch der CoA ist, gibt es die Return Routability Procedure [43]. Dabei wird das einfache BU durch eine Folge von Paketen ersetzt. Indem nachgewiesen wird, dass der MN gleichzeitig über HoA und CoA erreichbar ist, wird die Gefahr einer

Man-in-the-Middle Attacke auf ein einziges Subnetz reduziert, was der „üblichen Gefahr“ im Internet entspricht.

Anstelle des BU-Extension Headers werden bei der Return Routability Procedure die Pakete Home-Test Init (HoTi) und Care-of Test Init (CoTi) versendet (siehe Abbildung 2.3). Das CoTi-Paket wird auf direktem Wege an den CN gesendet, das HoTi-Paket nimmt den Umweg über den HA. Beide Pakete sind mit einem Cookie versehen, um deren Beziehung zueinander abzubilden.

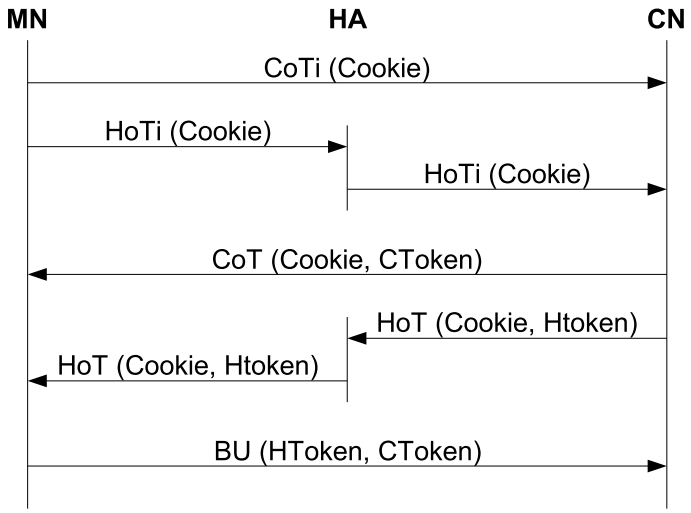


Abbildung 2.3.: Protokollablauf Return Routability Procedure

Der CN sendet als Antwort auf diese Pakete das Home Test (HoT)- und das Care-of Test (CoT)-Paket auf denselben Wegen zurück zum MN. Diese Pakete sind mit dem Home Token (HToken) bzw. Care-of Token (CToken) versehen. Diese Zufallszahlen werden vom MN empfangen und in einem weiteren Paket (BU) auf direktem Wege an den CN gesendet. Durch das gleichzeitige Senden beider Token kann der MN beweisen, dass er sowohl über die CoA-Adresse als auch über die HoA erreichbar ist.

Ein Angreifer müsste nun sowohl das HToken als auch das CToken erlangen, um einen Angriff durchführen zu können. Da diese Beiden meist über unterschiedliche Wege vom CN zum MN geschickt werden, ist es unwahrscheinlich, dass ein Angreifer beide Token abfangen kann. Es bestehen trotzdem noch zwei Angriffspunkte: Beide Token werden im Subnetz des MN (siehe Abbildung 2.4 ①) und des CN (②) auf den gleichen Wegen übertragen. Findet die Übertragung des Tokens zwischen MN und HA jedoch verschlüsselt über IPsec statt, so fällt auch ① als möglicher Angriffspunkt aus. Somit verbleibt die Übertragung der Token im Subnetz des CN als einziger Angriffspunkt. Diese Gefahr ist allerdings nicht größer als bei anderen Übertragungen im Internet und kann damit vernachlässigt werden.

Ein Nachteil dieser Prozedur sind die auftretenden Latenzen, die bei einer Echtzeit-Kommunikation unerwünscht sind. Des weiteren ist die Überprüfung der IP-Adressen lediglich ein Indiz aber kein Beweis dafür, dass das BU legitim ist, da sämtliche Überprüfungspakete nicht-authentifiziert übertragen werden. Ein weiteres Problem ist, dass die Return Routability Procedure nicht mit dem im weiteren Verlauf der Arbeit genutzten

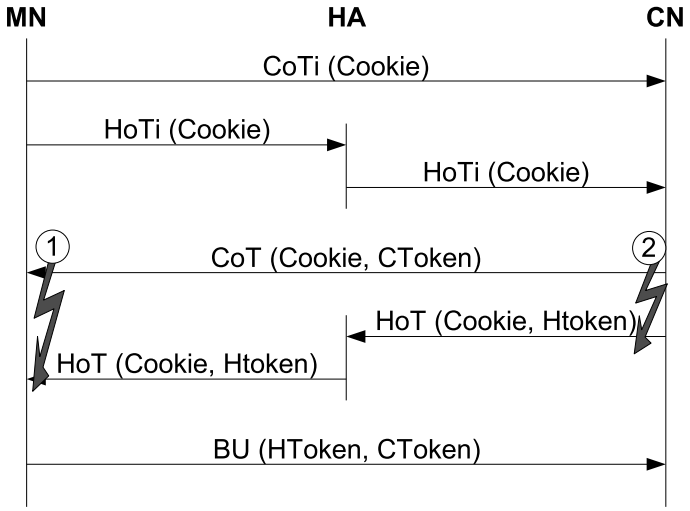


Abbildung 2.4.: Man-in-the-Middle Attacke bei der Return Routability Procedure

Multicast funktioniert, da es dabei keinen Rückkanal vom Empfänger zum Sender gibt. Diese Unzulänglichkeiten können durch den Einsatz von CGAs verbessert werden. Mehr dazu im folgenden Abschnitt.

2.2.2. Cryptographically Generated Addresses (CGAs)

Die Grundidee kryptographisch generierter Adressen (CGAs) [6, 7] ist das Binden einer IP-Adresse an einen Public Key. Da der Sender auch den zu dem Public Key gehörenden Private Key besitzt, kann er mit diesem (Update-) Pakete signieren. Auf der Empfängerseite kann diese Signatur mit Hilfe des an den Interface Identifier gebundenen Public Keys überprüft werden. Pakete können somit selbstkonsistent authentifiziert werden, ohne vorher ein Vertrauensverhältnis durch ein gemeinsames Passwort (shared secret) oder durch einen Public Key-Austausch aufzubauen.

Dazu generiert der Sender den Interface Identifier der IPv6-Adresse, den sogenannten Cryptographically Generated Host Identifier (CGHID), über den im Folgenden vorgestellten Algorithmus. Als Eingabedaten dienen die in der CGA-Parameter Datenstruktur (siehe Abbildung 2.5) hinterlegten Werte. Zusätzlich existiert ein Sicherheitsparameter *Sec*, der die Sicherheitsstufe darstellt. Dieser kann Werte zwischen Null und Sieben annehmen, wobei ein höherer Wert eine größere Sicherheit der CGA bedeutet.

Modifier Der Modifier wird zur Generierung des CGHIDs benutzt. Er wird solange erhöht, bis ein gültiger Hashwert gefunden ist. Mehr dazu im nächsten Abschnitt.

Subnet Prefix Dieses 64 Bit lange Feld enthält das Subnet Prefix der CGA.

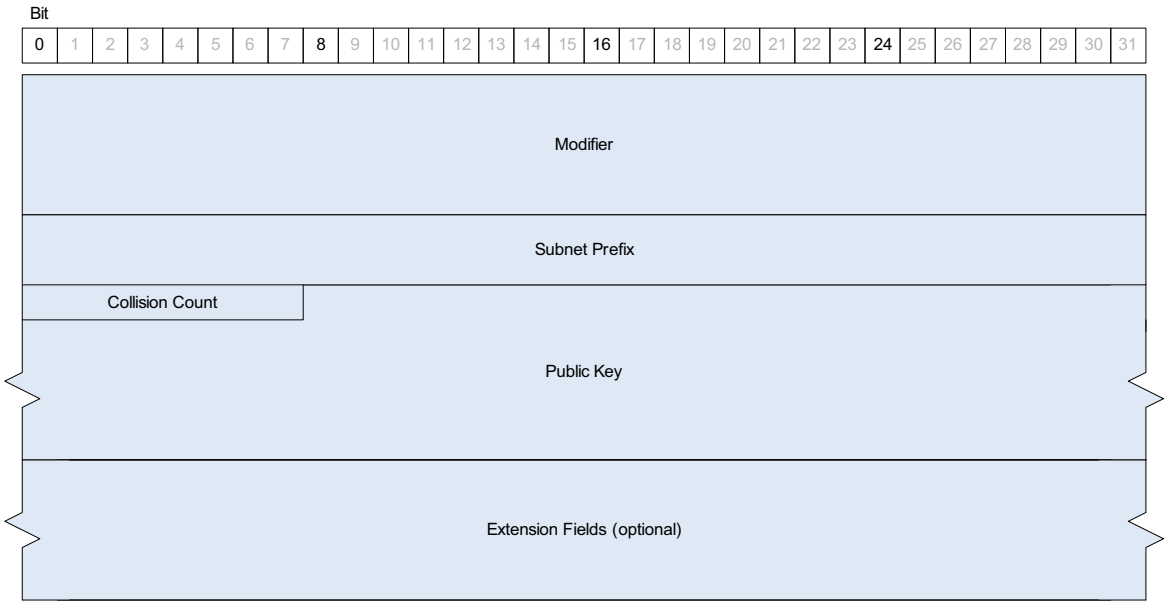


Abbildung 2.5.: CGA-Parameter Datenstruktur (aus [6])

Collision Count Dieses 8 Bit lange Feld darf nur die Werte 0, 1 oder 2 annehmen. Es wird bei der CGA-Generierung erhöht, um Adresskonflikten bei der Duplicate Address Detection (DAD) [66, 96] entgegenzuwirken.

Public Key Das Public Key-Feld hat eine variable Länge und enthält den Public Key des Adressbesitzers. Der Public Key muss als DER-enkodierte [42] ASN.1-Struktur abgelegt werden. Laut [6] sollte ein RSA-Schlüsselpaar mit der minimalen Schlüssellänge von 384 Bits benutzt werden.

Extension Fields Dieses optionale Feld variabler Länge kann mögliche Erweiterungsoptionen [7] aufnehmen.

Das Verfahren zur Generierung von CGAs ist in Abbildung 2.6 dargestellt.

Zur Generierung von CGAs werden folgende Schritte durchgeführt:

1. Der Modifier der CGA-Parameter Datenstruktur wird mit einer Zufallszahl belegt.
2. Der Modifier, 9 Null-Octets[2] und der Public Key werden von links nach rechts aneinandergehängt. Hiervon wird der SHA-1 Hashwert gebildet. Die 112 ersten Bits sind der Zwischenwert Hash2.
3. Die 16*sec ersten Bits des Hash2-Wertes werden mit Null verglichen. Sind alle Null, geht es weiter bei Schritt 4. Ansonsten wird der Modifier um 1 inkrementiert und Schritt 2 wird wiederholt.
4. Der 8 Bit lange Collision Count wird auf Null gesetzt.

[2]Ein Octet ist die in RFCs übliche Bezeichnung für 8 Bit.

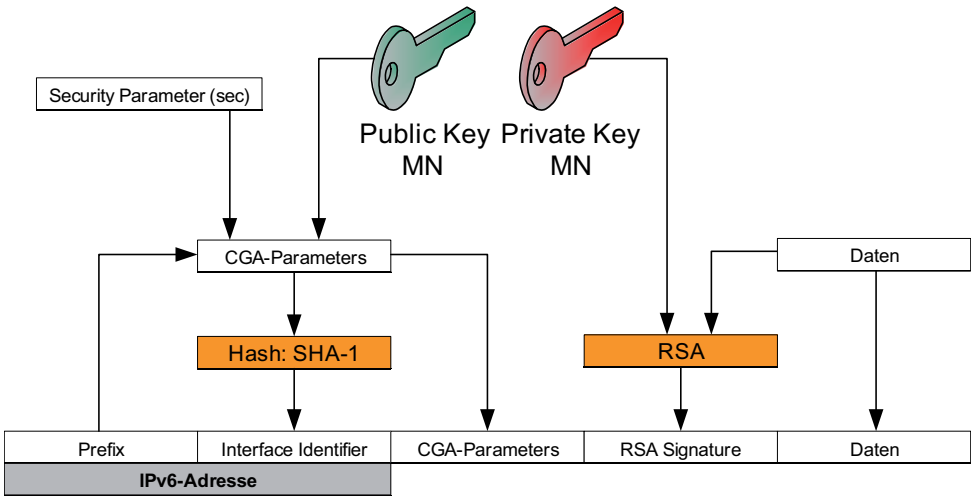


Abbildung 2.6.: Generierung einer CGA

5. Es wird ein SHA-1 Hashwert über die gesamte CGA-Parameter Datenstruktur, einschließlich der Extension Fields, gebildet. Die ersten 64 Bit bilden den Hash1.
6. Der Interface Identifier entspricht dem Hash1, wobei die ersten drei Bits durch den Sicherheits-Parameter *Sec* ersetzt werden. Außerdem müssen die Bits 6 und 7 (also das “u“ und „g“ Bit) auf Null gesetzt werden.
7. Die IPv6-Adresse ergibt sich aus dem Subnet Prefix und dem gerade gebildeten Interface Identifier.
8. Es muss nun geprüft werden, ob die IPv6-Adresse bereits benutzt wird. Dazu wird der Mechanismus der Duplicate Address Detection [66, 96] verwendet. Falls ein Adresskonflikt besteht, wird das Collision Count Feld um 1 erhöht und mit Schritt 5 fortgefahren. Diese Wiederholung darf lediglich 3 Mal durchgeführt werden. Sollte danach immer noch keine eindeutige IPv6-Adresse gefunden worden sein, so ist ein Fehler aufgetreten.

Zur Überprüfung von CGAs werden folgende Schritte durchgeführt (siehe Abbildung 2.7):

1. Zunächst wird ein Sanity Check über die Eingabedaten durchgeführt:
   - Der Wert des Collision Count-Feldes muss zwischen 0 und 2 liegen.
   - Das Subnet Prefix aus der CGA-Parameter Datenstruktur muss mit dem Subnet Prefix der IP-Adresse übereinstimmen.
2. Der SHA-1 Hash wird über die gesamte CGA-Parameter Datenstruktur gebildet. Die ersten 64 Bit ergeben den Hash1.
3. Der Hash1-Wert muss mit dem Interface Identifier der IP-Adresse übereinstimmen. Ausgenommen ist hierbei der Security Parameter *Sec*, der die ersten 3 Bits des Interface Identifier belegt, sowie die o.g. „u“ und „g“ Bits.

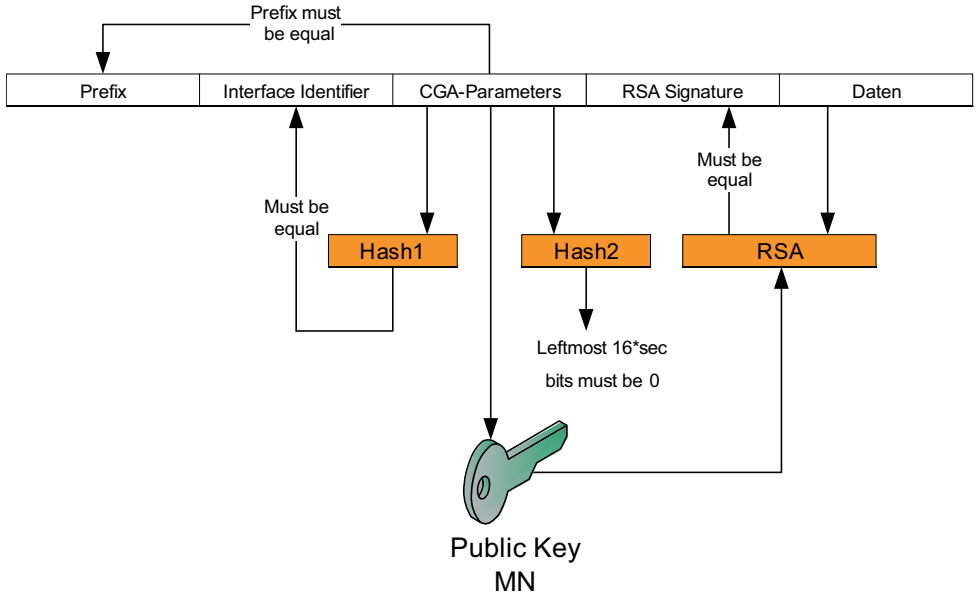


Abbildung 2.7.: Überprüfung einer CGA

4. Der Modifier, 9 Null-Octets, der Public Key und eventuelle Extension Fields werden von links nach rechts aneinandergehängt. Hiervon wird der SHA-1 Hashwert gebildet. Die 112 ersten Bits stellen den Hash2 dar.
5. Die 16*sec ersten Bits des Hash2-Wertes werden mit Null verglichen. Ist dies der Fall, ist die Überprüfung erfolgreich abgeschlossen.

Zur Übertragung der nötigen Werte bestehen die CGA Parameter Option und die CGA Signature Option. Diese werden im Folgenden vorgestellt.

2.2.2.1. CGA Parameters Option

Die CGA-Parameter Datenstruktur wird auf der Empfängerseite benötigt, um die CGA Verifikation durchzuführen. Daher muss sie, in einem Header verpackt, zum Empfänger übertragen werden. Da der CGA-Standard keinen Header vorschreibt, wird im Folgenden der in [5] standardisierte Mobility Header vorgestellt, da er im weiteren Verlauf der Arbeit benutzt wird.

Der Aufbau der CGA Parameters Option ist in Abbildung 2.8 dargestellt.

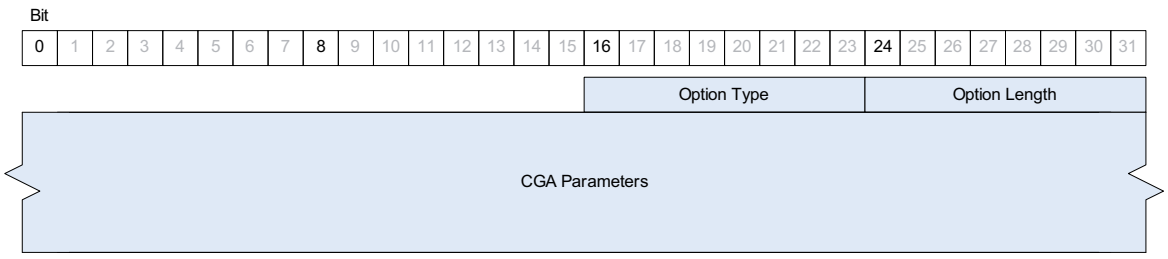


Abbildung 2.8.: CGA Parameters Option

Option Type Das 8 Bit lange *Option Type*-Feld enthält mehrere Optionen: Die höchstwertigsten 2 Bit müssen auf Null gesetzt sein und setzten damit laut [22] fest, dass ein IPv6 Node, der den Optionstyp nicht versteht, die Option überspringen und das restliche Paket weiterverarbeiten muss. Das folgende 3. Bit ist ebenfalls auf Null gesetzt und besagt, dass sich die Daten dieses Headers auf dem Weg nicht verändern dürfen.

Option Data Len Dieses 8 Bit lange Feld gibt die Länge des Headers (ohne Option Type und Option Data Len) in Octets an.

CGA Parameters Dieses Feld variabler Länge enthält bis zu 255 Byte der CGA Parameter Datenstruktur gemäß [6].

Da die CGA Parameter Datenstruktur größer als 255 Byte sein kann, wird sie in diesem Fall für die Übertragung in mehrere der zuvor vorgestellten CGA Parameter Optionen aufgeteilt. Dabei müssen alle Teile, außer des letzten, genau 255 Byte enthalten. Außerdem müssen die Optionen in der korrekten Reihenfolge im Hop-by-Hop Option Header angeordnet sein, da der Standard kein Sequenzfeld für die Reihenfolge der Daten vorgesehen hat. Zusätzlich darf kein anderer Hop-by-Hop Option Header zwischen zwei CGA Parameter Optionen platziert werden. Die ursprüngliche Datenstruktur kann somit durch Aneinanderhängen der Daten wiederhergestellt werden.

2.2.2.2. CGA Signature Option

Auch die Signatur muss in einem Header zum Empfänger übertragen werden. Hier wird ebenfalls der in [5] standardisierte vorgestellt.

Der Aufbau der CGA Signature Option ist in Abbildung 2.9 dargestellt.

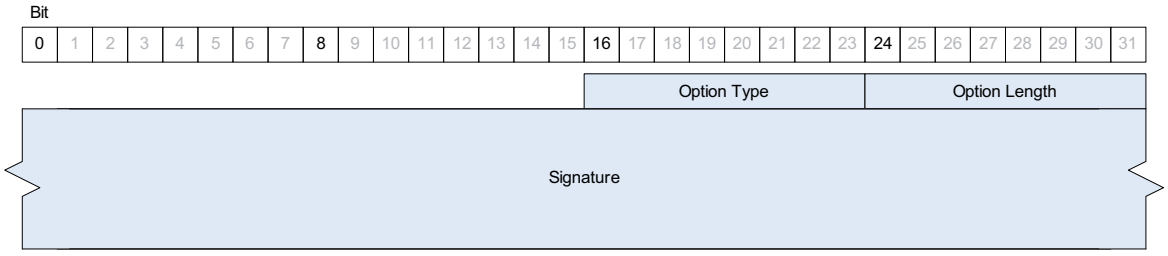


Abbildung 2.9.: CGA Signature Option

Option Type Das 8 Bit lange *Option Type*-Feld enthält mehrere Optionen: Die höchstwertigsten 2 Bit müssen auf Null gesetzt sein und setzten damit fest, dass ein IPv6 Node, der den Optionstyp nicht versteht, die Option überspringen und das restliche Paket weiterverarbeiten muss. Das folgende 3. Bit ist ebenfalls auf Null gesetzt und besagt, dass sich die Daten dieses Headers auf dem Weg nicht verändern dürfen.

Option Data Len Dieses 8 Bit lange Feld gibt die Länge des Headers (ohne Option Type und Option Data Len) in Octets an.

Signature Dieses Feld variabler Länge enthält die Signatur, welche mit Hilfe des privaten Schlüssels der mobilen Quelle erzeugt wird.

2.2.3. Anwendungen von CGAs

CGAs können für eine ganze Reihe von Anwendungen benutzt werden. Dabei schützen sie als einziger zur Zeit bekannter Mechanismus selbstkonsistent und infrastrukturlos vor Man-in-the-Middle (MitM) Attacken.

Das Neighbor Discovery Protocol (NDP) [68, 96] wird bei IPv6 dazu benutzt, Rechner im gleichen Netz zu finden, Link Layer Adressen und Router zu ermitteln und die Duplicate Address Detection (DAD) durchzuführen. Dabei gibt es für Angreifer im lokalen Netz die Möglichkeit, Verbindungen anderer Rechner durch ARP-Spoofing zu übernehmen [69]. Um dieser Schwachstelle entgegenzuwirken, wurde das SEcure Neighbor Discovery (SEND) [4] entwickelt. Dieses nutzt CGAs, um selbstkonsistent zu beweisen, dass der Sender einer Nachricht auch der Eigentümer der benutzten IP-Adresse ist.

Die Autoren in [15] verwenden CGAs, um die Protokolle Multicast Listener Discovery Version 2 (MLDv2) [35, 100] und Internet Group Management Protocol v3 (IGMPv3) [14] gegen Denial of Service Attacken (DoS) abzusichern. Sie nutzen dazu Gruppen-CGAs, um die Multicast-Gruppenzugehörigkeit von Knoten zu überprüfen.

Das Enhanced Route Optimization-Verfahren [5] wird dazu genutzt, Mobile IPv6 Binding Updates zwischen MN und CNs kryptographisch abzusichern und die Return Routability Procedure (siehe Abschnitt 2.2.1) zu vereinfachen. Dabei wird die beim BU übertragene HoA durch eine CGA-Signatur authentifiziert. Des weiteren wird der Besitz der aktuellen CoA durch den Austausch eines sogenannten Care-of Keygen Tokens, einer vom CN generierten Zufallszahl, bestätigt. Dazu sendet der MN dem CN zunächst die early Binding Update Message inklusive der Care-of Test Init Option, auf welche der CN mit dem early Binding Acknowledgement inklusive der Care-of Test Option antwortet (siehe Abbildung 2.10). Diese Option enthält das erwähnte Token. Daraufhin sendet der MN das eigentliche BU inklusive der Care-of Test Option und der signierten HoA. Damit ist sowohl die Erreichbarkeit der CoA, als auch der Besitz der HoA nachgewiesen. Die aus diesem Verfahren resultierenden Vorteile sind somit die Leichtgewichtigkeit und die Unabhängigkeit von einer PKI.

In dieser Arbeit werden CGAs ebenfalls dazu benutzt, das Binding zwischen Multicastquelle und deren Empfängern abzusichern. Dabei wird das zuvor vorgestellte Enhanced Route Optimization Verfahren verwendet. Allerdings wird das Care-of Test Verfahren, mit dem die CoA überprüft wird, nicht genutzt, da es ansonsten zur Überflutung der mobilen Multicastquelle kommen würde. Es wird also lediglich das optimierte BU dazu verwendet, den Besitz der HoA nachzuweisen. Damit wird sichergestellt, dass ein Multicast-Verteilbaum nicht von Angreifern übernommen werden kann. Außerdem setzen auch die

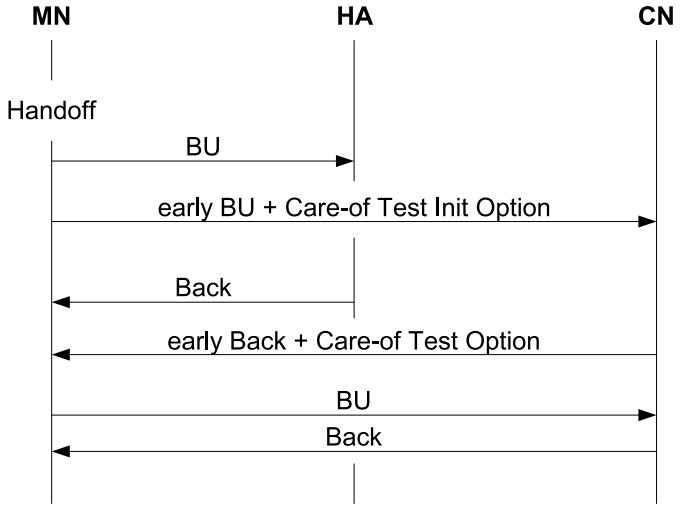


Abbildung 2.10.: Protokollablauf Enhanced Route Optimization zwischen Mobile Node, Home Agent und Correspondent Node (aus [5])

beteiligten Router CGAs ein, um die Multicastquelle nach einem Handover eindeutig identifizieren zu können.

2.3. Multicast

Multicast [23] ist die Realisierung des Host-Group Modells im Internet Protokoll (IP), bei der Pakete an eine Gruppe von Empfängern verteilt werden. IP ist aktuell das einzige Kommunikationsprotokoll, welches einen global routbaren Multicast-Dienst bereitstellt. Dabei wird im Gegensatz zu Broadcast nicht das gesamte Netz geflutet, sondern nur an eine einzelne, spezielle Multicast-Adresse gesendet. In eine Multicastgruppe kann grundsätzlich jeder Host senden. Ob die gesendeten Pakete empfangen werden, hängt davon ab, ob sich die Empfänger an der Gruppe angemeldet haben. Dafür können Empfänger mit sogenannten Join- und Prune-Nachrichten der Gruppe beitreten oder sie verlassen. Die dafür in IPv6 aktuell verwendete Implementierung ist das Multicast Listener Discovery Protocol (MLD) [35][3]. Für die Verteilung von Multicast-Traffic auf im Internet verteilte Empfänger kommen Multicast-Router zum Einsatz. Diese multiplizieren Pakete auf mehrere Interfaces, sofern Empfänger an diesen vorhanden sind. Sender, die nur über eine Internetverbindung mit geringer Bandbreite verfügen, können somit dennoch eine beliebige Anzahl an Empfängern erreichen.

Für Multicast-Übertragungen wurden spezielle IP-Adressräume [23, 33] eingerichtet:

- IPv4: 224.0.0.0/4
- IPv6: FF00:0:0:0:0:0:0:0/8

[3]IPv4 benutzt dazu aktuell das Internet Group Management Protocol (IGMPv3) [14, 35].

In IPv4 steht damit der Class D Adressraum von 224.0.0.0 bis 239.255.255.255 für Multicast-Übertragungen zur Verfügung. Diese Gruppenadressen werden wie in [2] vergeben. Dabei gibt es laut [54] einige besondere Adressen: Es wurde spezifiziert, dass die Adresse 224.0.0.0 keiner Gruppe zugewiesen wird. 224.0.0.1 ist der permanenten Gruppe „alle IP-Hosts“ zugewiesen, auf welche alle Hosts und Router, die IPv4 Multicast unterstützen, hören müssen. Der IPv6-Multicastadressraum wurden in [53] entsprechend ebenso unterteilt.

Es gibt mehrere Multicastverfahren, von denen im Folgenden insbesondere das Source Specific Multicast Routing (SSM) vorgestellt wird, da es die Grundlage für die Protokollerweiterung dieser Arbeit bildet. SSM ist eine Eingrenzung des Host-Group Modells. Dabei muss der Empfänger wissen, von welcher Quelle er Daten empfangen möchte.

Ein generelles Problem der Multi- und Unicastübertragung ist die fehlende Unterstützung mobiler Geräte. Die Topologie der Multicast-Verteilbäume beruht auf topologisch richtigen Adressen, die sich aber bei mobilen Geräten mittels Mobile IP laufend ändern. Bei einem solchen Handover sollten auch Multicastpakete weiterhin eintreffen oder versendet werden können, ohne dass es zu Paketverlusten kommt. Weitere Details zur Multicastmobilität finden sich in Abschnitt 2.4.

2.3.1. Reverse Path Forwarding

Reverse Path Forwarding bezeichnet das Weiterleiten von Paketen auf „umgekehrten“ Unicast Routen. Dies setzt symmetrisches Routing voraus. Das bedeutet, dass die Unicast-Routen zwischen zwei Routern in beide Richtungen gleich sind. Dies ist nötig, da die meisten Multicast-Routingprotokolle RPF-Checks verwenden.

Der Reverse Path Forwarding (RPF)-Check überprüft, ob eingehende Multicastdaten auf dem topologisch korrekten Interface eintreffen. Dazu wird die Unicast-Routingtabelle nach dem korrekten Interface für die Quelladresse durchsucht. Treffen die Multicastdaten auf dem falschen Interface ein, werden sie verworfen. Eine Erweiterung davon ist das in Kapitel 4 vorgestellte Tree Morphing.

2.3.2. Any Source Multicast (ASM)

Any Source Multicast (ASM) bezeichnet die ursprüngliche Form der Implementierung des Host-Group Modells, bei dem beliebige Sender in dieselbe Gruppe sprechen. Dabei gibt es zwei Möglichkeiten, die benötigten Verteilbäume aufzubauen: Sparse Mode (SM) und Dense Mode (DM) Protokolle. Sparse Mode Protokolle sind für weit verteilte Empfänger gedacht, Dense Mode Protokolle für Netzwerke, in denen viele Empfänger sitzen.

Deering [23] schlug für den Aufbau der Verteilbäume das Dense Mode Protokoll Distance Vector Multicast Routing Protocol (DVMRP) [101] vor, welches das Netzwerk mit Paketen flutet. Router, die keine Empfänger besitzen, schneiden nicht benutzte Zweige temporär ab (pruning). Da DVMRP das Netz flutet kann es nicht für weit verteilte Empfänger

eingesetzt werden[4]. Die gleichen Eigenschaften hat auch Protocol Independent Multicast - Dense Mode (PIM-DM) [1].

Daher wurden Shared Tree Protokolle, wie z.B. Protocol Independent Multicast - Sparse Mode (PIM-SM) [27] eingeführt, welche die Verteilzustände erst dann erzeugen, wenn sie benötigt werden. Dazu werden entweder die Unicast-Routinginformationen oder eine separate Multicast Routing Information Base (MRIB) genutzt, um Shared Trees aufzubauen, welche an einem RendezVous Point (RP) wurzeln (siehe Abbildung 2.11). Router besitzen Verteilzustände pro Gruppe, sogenannte $(*, G)$-States. $*$ bedeutet, dass in die Gruppe beliebige Quellen sprechen können. Zusätzlich können vom RP ausgehend sogenannte Short-Cuts durchgeführt werden. Dabei werden die Multicastverteilzustände in SPTs pro Quelle überführt, was den RP entlastet und den Netzwerkverkehr optimiert.

Alle bisher vorgestellten Protokolle sind Intradomain Protokolle [103], d.h. Quellen und Empfänger müssen sich in dem selben Autonomous System (AS) befinden. Da es jedoch wünschenswert ist, auch Multicastdatenströme eines Senders außerhalb des eigenen (Provider-) Netzwerks zu empfangen, gibt es mehrere Möglichkeiten, Multicastdomänen zu verbinden.

Das Multicast Source Discovery Protocol (MSDP) [28] verbindet PIM-SM Domänen miteinander. Jede Domäne benutzt dabei weiterhin ihren eigenen RP. Zum Austausch von Gruppeninformationen stellen MSDP Router TCP-Kontrollverbindungen mit MSDP Routern in anderen Domänen her. Diese MSDP-fähigen Router werden anderen Routern über eine Erweiterung des Border Gateway Protocols (BGP) [78], das sogenannte Border Gateway Protocol with multi-protocol extensions (MBGP) [11], bekannt gemacht. Gibt es Empfänger, die an Quellen außerhalb der eigenen Domäne interessiert sind, so sendet der zugehörige RP eine Join-Anfrage auf direktem Wege an die Quelle. Somit wird ein Verteilzweig für diese Quelle aufgebaut. Außerdem existiert MSDP nur für IPv4. Für IPv6 wurde die Embedded-RP Adressierung eingeführt [82]. Laut [38, 65, 103] ist MSDP jedoch nur eine vorübergehende Lösung, bis BGMP, ein robusteres Interdomainprotokoll, eingesetzt wird.

Mit Hilfe des zur Zeit noch in der Entwicklung befindlichen Border Gateway Multicast Protocol (BGMP) [94] können Multicastdaten über Domaingrenzen hinweg verteilt werden. Basierend auf den RP-Konzepten von CBT [9, 10] und PIM-SM, werden bidirektionale Shared Trees über TCP-Verbindungen zwischen BGMP-Routern aufgebaut. Die Multicastdomänen werden somit miteinander verbunden. Innerhalb der Domänen kann jedes Multicastprotokoll (Sparse und Dense Mode) benutzt werden. Für jede Multicastgruppe muss eine Rootdomäne festgelegt werden, damit die BGMP-Router den Weg zu den einzelnen Gruppen austauschen können. Dazu werden die Multicastadressen dahingehend verändert, dass das Subnet-Prefix in die Adresssen eingefügt wird [31, 95]. Somit können BGMP-Router erkennen, in welchem Netz sich die Rootdomäne der Multicastgruppe befindet. Zusätzlich unterstützt BGMP auch das im nächsten Abschnitt vorgestellte Source Specific Multicast.

[4] Es war auch als Interior Domain Protokoll gedacht.

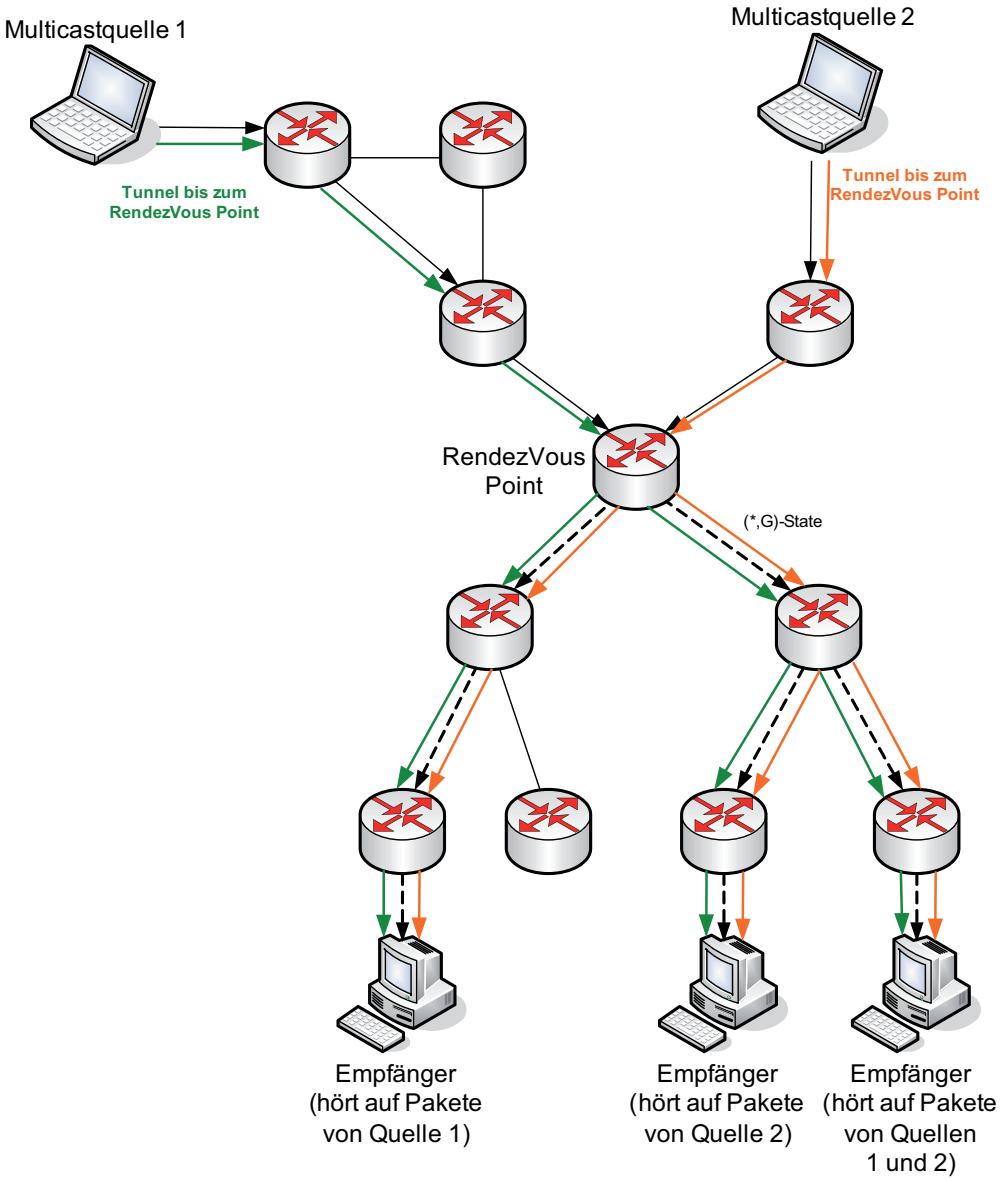


Abbildung 2.11.: Multicastübertragung über einen RendezVous Point

2.3.3. Source Specific Multicast (SSM)

Holbrook [34] hat mit dem Source Specific Multicast (SSM) Modell eine Vereinfachung des Routings geschaffen, die auf „Shared Trees“ und RendezVous Points verzichtet. Ein Empfänger kann eine SSM-Übertragung über die Quelladresse S (Source) und die Gruppenadresse G identifizieren. Er muss also Kenntnis über die Quelladresse des Senders besitzen. Für SSM-Gruppen wurden der IPv4-Adressbereich 232/8 und der IPv6-Adressbereich $FF3x :: /32$ reserviert. Empfänger melden sich an den SSM-Übertragungen durch IGMPv3 oder MLDv2 an. Da die Verteilbäume jeweils (analog zu optimalen SPTs nach PIM-SM short-cuts) von den Quellen ausgehen, benötigt SSM keine Shared Trees und damit auch keine Rendezvous Points. Allerdings ist durch den Wegfall der RPs keine State Aggregation in den Routern möglich.

Das Routing bei SSM wird durch eine Untermenge von PIM-SM, dem sogenannten PIM-SSM [27] Routing durchgeführt. Anstelle der (*,G)- werden dabei (S,G)-Joins von den Routern in Richtung der Quelle weitergeleitet. Entlang dieses Weges werden dann in umgekehrter Reihenfolge auch die Multicastdatenpakete verteilt (Reverse Path Forwarding).

Durch den Wegfall des RPs und der „Shared Trees“ treten keine Interdomänenprobleme auf. Die Übertragung kann also über Domänengrenzen hinweg stattfinden. SSM-Übertragungen sind sehr gut für Medienübertragungen aus einer Quelle, wie z.B. IPTV, IP-Radio oder Konferenzen, geeignet.

2.3.4. Layer 2 (Wireless) Multicast

Die vom Netzwerklayer für Multicast-Übertragungen benutzten IP-Adressräume wurden bereits in Abschnitt 2.3 vorgestellt. Es ist wünschenswert, dass Multicast auch direkt auf dem Data Link Layer (Layer 2) unterstützt wird, damit kein Broadcast oder bandbreitenintensive Unicast-Replikation stattfinden muss. Dabei muss zwischen den verbindungslosen Broadcast- und verbindungsorientierten Point-to-Multipoint Protokollen unterschieden werden.

Verbindungslose Broadcast-Protokolle, wie z.B. Wireless LANs (WLANs) (IEEE 802.11) [41], unterstützen Multicast-Datenübertragungen direkt, indem die Multicastdaten vom Sender als Unicastdatenpakete in einem Point-to-Point Kanal an den Access Point (AP) gesendet werden. Dieser bestätigt die Pakete mit einem Acknowledge-Paket (ACK). Mögliche Paketverluste werden vom Sender erkannt und durch erneutes Senden behoben. Erreichen Multicastpakete aus dem kabelgebundenen Netzwerk oder von einer Funkstation einen Access Point, so werden diese in einem in [41] definierten Multicast-Frame versendet. Diese Frames werden von interessierten Empfängern nicht bestätigt. Da Multicast-Paketverluste nicht erkannt werden, ist die Qualität der Multicastdatenübertragung bei ausgelasteten WLAN-Netzen potentiell schlechter als bei Unicast-Übertragungen. Zwar existieren diverse Vorschläge [30, 91, 93], die Multicastdatenübertragung abzusichern, jedoch ist noch keiner davon standardisiert worden.

Das Bilden von Layer 2 Adressen aus IPv6 Multicastadressen geschieht analog zum Verfahren von Ethernet [21, 40, 72]. Ist das niederwertigste Bit des ersten Bytes der 48 Bit langen Adresse auf Eins gesetzt, handelt es sich um eine Multicastadresse. Das Mapping zwischen IPv6-Multicastadressen und Ethernet MAC-Adressen kann durch das Voranstellen des Prefixes 33-33 an die 32 niederwertigsten Bits der IPv6-Multicastadresse hergestellt werden.

Verbindungsorientierte Protokolle, wie z.B. IEEE 802.16 [39], bieten zwei Übertragungsmethoden: Point-to-Point und Point-to-Multipoint. Sender (SS = Subscriber Station) können Multicastdaten grundsätzlich nur über einen Point-to-Point Kanal an ihre Basisstation (BS) senden. Allerdings gibt es durch Erweiterungen auf dem sogenannten *Convergence Sublayer* die Möglichkeit für Teilnehmerstationen, IPv6 Multicastdaten zu senden [44, 75].

In der Verteilrichtung können BS mehrere SS in einer Multicast-Verteilgruppe zusammenfassen. Diese erhalten den selben Channel Identifier (CID). Der Vorschlag, das Mapping von IPv6 Multicastadressen auf CIDs, wie in [44] angegeben, durchzuführen, ist anfällig für Adresskollisionen, da 8 Bits des 16 Bits langen CIDs reserviert sind und somit nur 8 Bit zur Vergabe verbleiben. Diese 8 Bits werden mit den 8 niederwertigsten Bits der IPv6-Multicastadresse gefüllt.

Da BS als Layer 2 Switches arbeiten, können sie MLD Snooping unterstützen, um Multicast-Anfragen der SS mitzuhören und um die SS in die gewünschten Multicast-CID-Gruppen aufzunehmen. SS können beim Empfang solcher Multicast-Pakete diese nicht von (Point-to-Point) Unicast-Paketen unterscheiden.

2.4. Multicastmobilität

Nicht nur der mobile Empfang von Multicastquellen, z.B. auf IPTV-fähigen Smartphones, sondern auch das mobile Versenden von Multicastdaten wird in Zukunft gefordert, da Benutzer immer mehr zu Content-Produzenten werden. Diese könnten z.B. live von einem Ereignis per Multicast an viele (ebenfalls mobile) Empfänger berichten.

Bei beiden Einsatzszenarien tritt das Problem auf, dass die vorhandene Multicastsession an den neuen Standort (Point of Attachment) übernommen werden muss. Abhängig von der Netzwerktopologie und dem benutzten Protokoll benötigt die Netzwerkinfrastruktur von Sekunden bis zu Minuten, um das Multicastrouting an die neuen Verhältnisse anzupassen. Dies ist deutlich zu langsam für eventuelle Medienübertragungen in Echtzeit.

Um mobile Multicastempfänger und -sender zu unterstützen, wurden verschiedene Ansätze entwickelt, die im Folgenden vorgestellt werden sollen. Im Anschluß wird das Tree Morphing Protokoll vorgestellt, welches das Multicastrouting für mobile Multicastquellen bei SSM-Übertragungen anpasst.

Analog zu den in Mobile IP genutzten Begriffen pAR und nAR gibt es bei Multicastmobilität die Konvention, den vorherigen für die Multicastübertragung zuständigen Router als previous Designated Router (pDR) und den aktuellen als next Designated Router (nDR) zu bezeichnen.

Im Mobile IPv6-Standard wurden bereits einfache Möglichkeiten vorgestellt, mobile Multicastempfänger und -sender zu unterstützten. Die einfachste Möglichkeit nach einem Handover auch weiterhin die Daten einer Multicastgruppe zu erhalten, ist es, unter Verwendung der aktuell gültigen CoA eine neue Verbindung zur Multicastgruppe aufzubauen und den alten Verteilzweig am pDR durch ein Timeout terminieren zu lassen. Diese Methode nennt sich Remote Subscription (RS). Das erneute Aufbauen eines Multicast-Verteilbaums dauert, wie bereits erwähnt, zwischen Sekunden und Minuten, was zu erhöhten Latenzen und Paketverlusten führen kann. Schnelle Handover verstärken diesen Effekt weiterhin.

Eine weitere Möglichkeit mobil Multicastdaten zu empfangen und zu senden wurde bereits 1997 von Xylomenos und Polyzos in [105] vorgestellt und ist in den Mobile IPv6-Standard aufgenommen worden: Das sogenannte Bidirektionale Tunneling (BT). Im Fall eines mobilen Multicastempfängers nimmt der HA MLD-Anfragen vom MN über einen Tunnel

entgegen und leitet diese in das lokale Subnetz weiter. Die Multicastdatenpakete treffen somit beim HA ein, der diese über den Tunnel an den MN weiterleitet. Nach einem Handover braucht in diesem Fall kein neuer Verteilbaum aufgebaut werden, da die Pakete weiterhin beim HA eintreffen. Lediglich der Tunnel wird durch das BU an die neue CoA gebunden. Durch Handover verschlechtert sich evtl. die Latenz (durch topologisches Entfernen vom HA), es gehen jedoch keine Pakete verloren, sofern der HA diese während des Handovers zwischenspeichert. Dieses Problem ist als Trianguläres Routing bekannt [84].

Ein weiteres Problem ist das sogenannte Tunnel Convergence Problem. Dabei befinden sich viele MNs in demselben Subnetz (aber nicht im Heimnetz) und fordern Multicastdaten aus derselben Multicastgruppe an. Die Multicastdaten werden dabei – obwohl sich alle MNs in einem Subnetz befinden – vom HA multipliziert und einzeln (unicast) über die Tunnel ausgesendet. Würden diese MNs ihren Wunsch, die Multicastgruppe zu empfangen, nativ senden, müsste nur ein einziger Verteilzweig vom aktuellen AR aufgebaut werden, welcher die Daten dann bereitstellt.

Ein bidirektionaler Tunnel ermöglicht auch das mobile Senden von Multicastdaten. Dabei werden die Daten über den Tunnel in das Subnetz des HA geleitet und von dort aus weiterverteilt. Hierbei kann der Verteilbaum nach einem Handover ebenfalls bestehen bleiben, da sich erneut lediglich der Tunnelendpunkt ändert.

Eine Übersicht über Multicastempfänger- und Sendermobilität findet sich in [80].

2.4.1. Empfängermobilität

Außer den im Mobile IPv6-Standard definierten Ansätzen, mobile Multicastempfänger zu unterstützen, gibt es eine weitere Gruppe an Protokollen, die sogenannten Agent-based Approaches. Dabei baut der mobile Multicastempfänger die Verbindung zum Verteilbaum nicht direkt, sondern über einen Agenten auf. Dieser leitet die Pakete dann an den MN weiter. Die Autoren in [102] haben einen Multicast Agent (MA) vorgeschlagen, der für eine Reihe von Subnetzen zuständig ist und die Multicastpakete auch nach einem Handover des MN entsprechend weiterleitet.

Eine weitere Möglichkeit ist das Verwenden von sogenannten Mobility Support Agents (MSA) [104]. Diese werden analog zu Fast Mobile IP vor einem Handover darüber informiert, dass ein MN in Kürze in ihr Subnetz treten wird, welcher für eine bestimmte Multicastgruppe Daten empfangen möchte. Der MSA sendet somit bereits vor dem Handover IGMP/MLD-Nachrichten, um die Daten aus der Multicastgruppe zu erhalten.

Das Multicast by Multicast Agent Protocol [90] verwendet Multicast Agents (MA) in jedem Subnetz, um Multicastgruppen im Auftrag der MNs beizutreten. Nach einem Handover informiert der MN den neuen MA über den Gruppenwunsch und den alten MA. Dieser tunnelt die Multicastpakete durch eine Anfrage des aktuellen MAs zu diesem, welcher die Multicastpakete somit bereits vor der nativen Gruppenzugehörigkeit bereitstellen kann. Sobald ein nativer Verteilzweig zum aktuellen MA errichtet wurde, informiert dieser den alten MA, dass keine weiteren Pakete getunnelt werden müssen. Besitzt der alte MA keine weiteren Empfänger, baut dieser die Verbindung zur Multicastgruppe ab.

2.4.2. Sendermobilität

Neben den im Mobile IPv6-Standard definierten Ansätzen, mobile Multicastsender zu unterstützen, haben die Autoren in [79] die Erweiterung von RPs zu Mobility-aware RPs (MRP) vorgeschlagen. Diese bei ASM mit PIM-SM einsetzbare Erweiterung hält ähnlich einem HA eine Zuordnung von CoA zu HoA vor. Eine mobile Multicastquelle meldet sich beim MRP durch sogenannte Source Registration (SR) Messages an. Nach einem Handover wird dieser MRP über die neue CoA in Kenntnis gesetzt. Die Multicastpakete werden vom MRP immer mit der HoA des MN verteilt, um Empfängern einen transparenten Service zu bieten.

Der Wechsel der CoA ist ein grunsätzliches Problem, dem alle mobilen Multicastlösungen begegnen müssen. Erhalten die Multicastempfänger Daten von der neuen CoA, können sie ohne weitere Hilfsmittel keine Zuordnung zum Datenstrom von der ursprünglichen Quelladresse herstellen und verwerfen sie. Dies ist das bereits von Mobile IPv6 bekannte Problem der Adressdualität. Eine mobile Quelle besitzt immer zwei Identitäten: eine logische ID (HoA) und eine topologisch aktuelle ID (CoA).

Ein zusätzliches Problem von SSM ist, dass es Quellfilter einführt. Empfänger identifizieren eine Multicastgruppe über eine Gruppenadresse und eine gültige Quelladresse. Ändert eine mobile Multicastquelle ihre CoA, müssen auch alle Empfänger Kenntnis über die neue CoA erhalten, um z.B. ein Multicast-Join zur CoA zu senden.

Weiterhin muss auch die Routinginfrastruktur Kenntnis über die neue CoA erlangen, da sie die Multicastdaten ansonsten nicht ausliefern kann. Werden die Daten nach einem Handover weiterhin mit der HoA als Quelladresse ausgeliefert, müssen die Router diese topologisch falschen Pakete verwerfen (Ingress Filtering [8]). Werden sie mit der CoA als Quelladresse gesendet, kann keine Zuordnung zum Verteilbaum hergestellt werden, da es sich nicht um denselben Baum handelt $(HoA, G) \rightarrow (CoA, G)$. Eine Lösung muss also immer das Tupel (HoA, G) als logischen Gruppenidentifier und die aktuelle CoA als topologischen Identifier unterstützen.

Wenige Untersuchungen liegen bisher für Mechanismen zur Überführung eines bestehenden Verteilbaumes in einen neuen vor. Für DVMRP wird ein Algorithmus in [16] angegeben, der die Wurzel des Baumes, der Senderbewegung folgend, verlängert. Die Autoren sind auf ein komplexes Signalisierungsschema angewiesen, um die DVMRP-Zustände und den RPF-Check anzupassen.

O'Neill [70] umgeht das Fehlverhalten des RPF-Checks, der sich mit dem Wechsel der Senderadresse ergibt, indem er zusätzliche Routing-Informationen im Hop-by-Hop Paketkopf kodiert.

Lee, Han und Hong haben kürzlich einen Mechanismus zur Aktualisierung der Router-Zustände vorgestellt, um Teile des etablierten Multicast-Baumes weiterzuverwenden [49]. Die Autoren beginnen bei einem initialen Verteilbaum, der beim HA angesiedelt ist. Mit dem Verlassen des Heimatnetzes signalisiert die Quelle entlang des Weges zum HA den Aufbau neuer Multicast-Zustände, sodass sich der Baum „künstlich" verlängert. Das Protokoll sieht keine nachfolgenden Optimierungsschritte vor, womit die Multicast-Daten

zwar nativ, aber immer triangulär über den HA geschickt werden. Weiterhin berücksichtigen die Autoren weder das Problem der nun fehlschlagenden RPF-Checks, noch schätzen sie die sich ergebenden Verzögerungen auf der Basis von Messungen oder Simulationen ab.

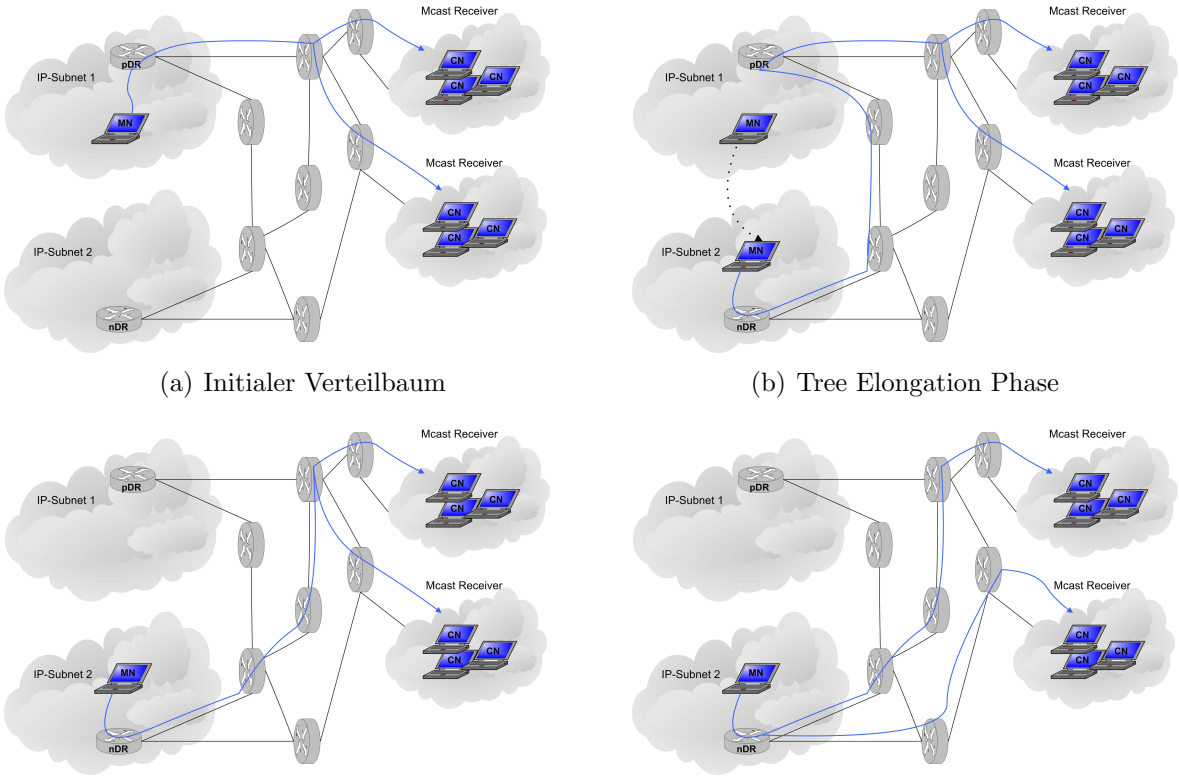
Um das Problem der Sendermobilität bei SSM zu lösen, haben die Autoren in [83, 84] das Tree Morphing Protokoll vorgestellt, mit Hilfe dessen der Multicast-Verteilbaum vom previous Designated Router (pDR) zum next Designated Router (nDR) überführt werden kann.

2.5. Tree Morphing

Ein adaptives SSM Routing-Protokoll, das sogenannte Tree Morphing, ist in Abbildung 2.12 dargestellt und wurde erstmals in [83] präsentiert und ausführlich in [84] evaluiert. Eine mobile Multicastquelle (MS = Mobile Source) außerhalb ihres Heimnetzes sendet unverpackte Daten an eine Multicastgruppe, unter Verwendung der HoA auf dem Application Layer und der aktuellen CoA auf dem Internet Layer, analog zum Versand von Unicast-Paketen in Mobile IPv6. Allerdings besitzt, als Erweiterung zum Unicast-Routing, der gesamte Internet Layer, einschließlich der Router, Kenntnis über die permanente HoA. Das Vorhalten von Adresspaaren in den Routern analog zu Binding Caches erlaubt allen Knoten die Identifizerung von (HoA, G)-basierter Gruppenzugehörigkeit und (CoA, G)-basierter Baumtopologie. Der ursprüngliche Verteilbaum, ausgehend vom pDR, ist in Abbildung 2.12(a) zu sehen, wobei die blauen Pfeile die Verteilwege der Multicastpakete darstellen. Nach einem Handover ändert die MS ihre Adresse von der pCoA zur nCoA und eventuell auch ihren pDR zum nDR. Die MS fährt sofort nach einem Handover fort, Daten entlang einer Verlängerung des vorherigen Verteilbaums zu senden. Dazu wird die Wurzel des vorherigen Verteilbaums vom pDR zum nDR verlängert (siehe Abbildung 2.12(b)). Der pDR sendet die Multicastpakete dabei stellvertretend für den MN in den Verteilbaum hinein. Alle Router entlang dieses Tree Elongation-Pfads und des alten Verteilbaums erlangen somit Kenntnis über die nCoA des MS und etablieren geeignete Weiterleitungszustände.

Router auf dem „alten“ Verteilbaum nutzen den RPF-Check, um potentielle Short-Cuts zu erkennen. Durch das Senden der State Updates mit der nCoA als Quelladresse können Router, die das Update auf dem topologisch falschen Interface erhalten, ein Join in Richtung des kürzesten Pfades in Richtung der Quelle (SPT) senden (siehe Abbildung 2.12(c)). Sobald die Daten auf dem korrekten Interface eintreffen, kann der „alte“ Verteilbaum durch ein Prune abgebaut werden. Alle anderen Router verwenden weiterhin den bestehenden Verteilbaum. Dies betrifft alle Teile des „alten“ Verteilbaums, die mit dem neuen übereinstimmen. Damit müssen nur die Teile des SPTs aufgebaut werden, die zuvor nicht etabliert waren. Der „alte“ Verteilbaum wird damit selbständig und „rechtzeitig“ in einen Verteilbaum kürzester Wege transformiert (siehe Abbildung 2.12(d)). Durch diesen Algorithmus müssen die Daten zu keiner Zeit verpackt oder getunnelt werden.

Das Tree Morphing Protokoll kann prinzipiell in jedes Join- und Prune-fähige Multicast-Protkoll integriert werden. Hier wird PIM-SSM [27] benutzt, um die (S, G)-State Machine



(a) Initialer Verteilbaum

(b) Tree Elongation Phase

(c) Tree Morphing Phase

(d) Konvergierter Verteilbaum

Abbildung 2.12.: Tree Morphing States (aus [86])

um den Tree Morphing-Zustand zu erweitern. Dies geschah, da PIM-SSM das einzige Protokoll ist, welches ohne RendezVous Points und Shared Trees auskommt und welches in Zukunft große Verbreitung finden wird.

2.5.1. State Machine

Die Abbildung 2.14 zeigt die State Machine eines Downstream-Interfaces eines Tree Morphing fähigen Routers. Downstream-Interfaces sind die Interfaces eines Routers, welche in Richtung der Multicastempfänger zeigen. Das Interface, auf dem die Daten der Multicastquelle eintreffen, nennt sich Upstream-Interface.

Die hier vorgestellte State Machine basiert auf der regulären, statischen PIM-SSM State Machine, wie in [27] in Kapitel 4.5.3 (Receiving (S,G) Join/Prune Messages) standardisiert. Die States No Info (NI), Join (J) und Prune-Pending (PP) sind daraus übernommen. Neu hinzugekommen ist der Tree Morphing (TM)-Zustand. In diesen Zustand gelangt ein Routerinterface, wenn ein sogenanntes State Update, also ein Multicastdatenpaket mit Updateinformationen, empfangen wird. Der Übergang in diesen Zustand kann aus allen der drei anderen Zustände geschehen. Als Besonderheit kann die State Machine im Join-State verbleiben, obwohl ein State Update empfangen wurde. Dies ist an die Bedingung geknüpft, dass das Update den RPF-Check bestanden hat. Das ist dann der Fall,

sobald ein Router trotz Handover und veränderter CoA die Multicastdaten auch weiterhin über den Pfad des Updatepakets aus einem für ihn optimalen Multicast-Verteilbaum enthält. Der Verteilzustand wird somit nur überschrieben (State Override). Schlägt der RPF-Check fehl, so findet ein Übergang in den TM-State statt. Abbildung 2.13 zeigt ein Beispielnetzwerk in dem ein MN ein Handover vom pDR zum nDR durchführt. Der am pDR wurzelnde Verteilbaum ist durch grüne Pfeile dargestellt, die die Verteilrichtung symbolisieren. Der „neue“, optimale Verteilbaum ist durch orangene Pfeilen dargestellt. Die schraffierten Router führen das zuvor beschriebene „State Override“ durch.

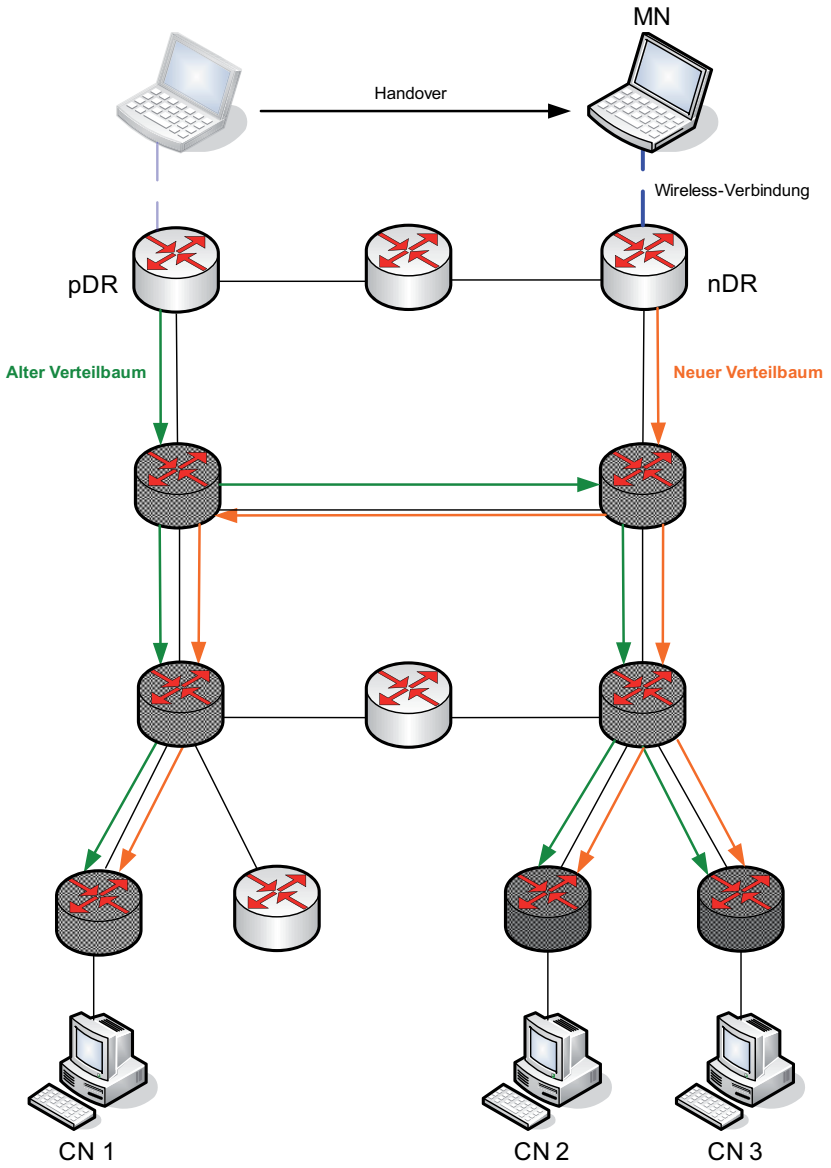


Abbildung 2.13.: Alter und neuer Multicast-Verteilbaum nach einem Handover. In den schraffierten Routern findet ein State Override statt. Dabei beiben die (HoA, G)-Forwarding States erhalten. Lediglich die CoA wird überschrieben.

Eine weitere Besonderheit ist die Zustandsänderung der State Machine beim Eintreffen von Multicastpaketen ohne Updateinformationen. Dies wird für den Fall benötigt, dass

Pakete in der Tree Optimization Phase auf dem neuen, topologisch korrekten Interface eintreffen. Damit ist die Tree Optimization Phase beendet und der Router wechselt in den J-State. Ansonsten verbleibt er im TM-State.

In Abbildung 2.14 sind der Join- und der No Info-State durch einen doppelten Kreis dargestellt. Dies soll verdeutlichen, dass es sich dabei um diejenigen States handelt, in denen sich ein Interface üblicherweise befindet. Alle anderen States werden lediglich temporär angesprungen, um den Verteilbaum zu überführen oder Teile dessen abzubauen.

Die State Machine hat folgende Zustände:

No Info (NI) Es bestehen keine (CoA, HoA, G) Join-Zustände auf dem aktuellen Interface, d.h. es werden keine Multicastdaten weitergeleitet und es laufen keine Timer.

Join (J) Das Interface ist im Join-Zustand, was eine Weiterleitung der Multicastpakete für (CoA, HoA, G) bewirkt.

Tree Morphing (TM) Das Interface befindet sich im Tree Morphing-Zustand und arbeitet gemäß der in [84] vorgestellen Algorithmen. Während der Tree Optimization Phase kann es mehrere Weiterleitungszustände für verschiedene CoAs (z.B. bei schnellem Handover) geben. Diese sind in der State Machine durch ihre Anzahl abgebildet: *Count(·, HoA, G)*[5] bezeichnet die Anzahl der für den logischen Gruppenbezeichner (HoA, G) vorhandenen Zustände. Diese Information wird nicht nur für das *Pruning* von alten Verteilzuständen, sondern auch für die Timeouts von Verteilzuständen pro pCoA genutzt. Dafür existieren die CoA-Expiry Timer (CoA-ET). Diese werden benutzt, um einzelne alte Verteilzustände nach ihrer maximalen Gültigkeitsdauer zu verwerfen. Läuft der letzte dieser CoA-ETs im TM-State aus, wechselt die State Machine in den NI-Zustand.

Prune-Pending (PP) Der Router hat ein Prune(CoA, HoA, G) empfangen und wartet, ob ein Downstream Router weiterhin Daten empfangen möchte. Dieses würde durch ein Join signalisiert werden. In diesem Zustand werden die Multicast-Daten ebenso wie im Join-Zustand weitergeleitet.

In der State Machine gibt es außerdem drei Timer:

Expiry Timer (ET) Der Expiry Timer wird gestartet, wenn eine gültige Join- oder State Update-Nachricht empfangen wird. Läuft der ET aus, wird die State Machine in den No Info-Zustand überführt.

[5](·, HoA, G) bezeichnet im Folgenden irgendeinen Zustand mit der Gruppenadresse G und der Home Address HoA. (*, HoA, G) bezeichnet dagegen alle dieser Zustände.

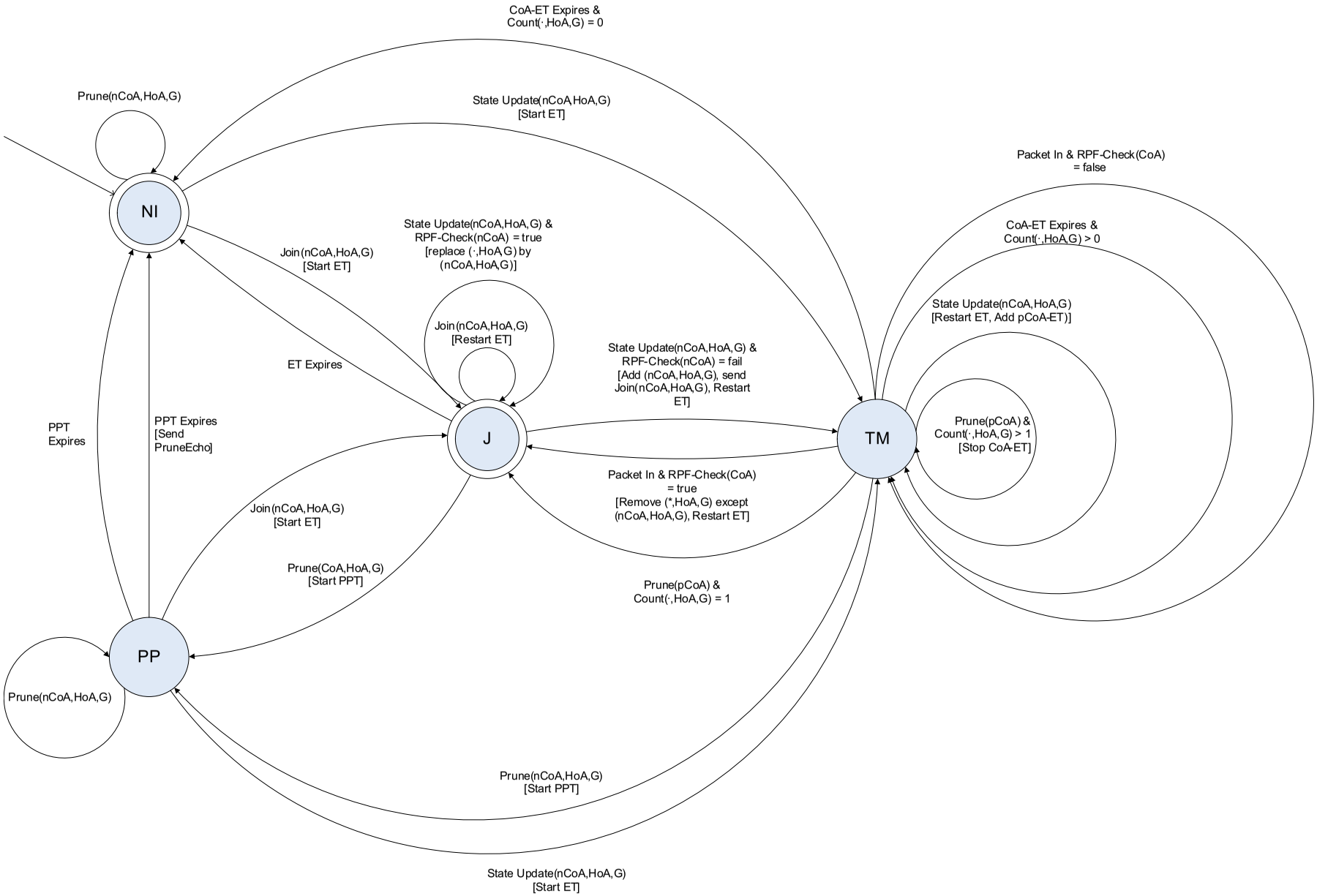


Abbildung 2.14.: Per-Interface Downstream State Machine

CoA Expiry Timer (CoA-ET) Ähnlich dem ET wird auch dieser Timer dazu benutzt, Forwarding States nach einer bestimmten ungenutzten Zeit verfallen zu lassen. Dieser Timer ist jedoch speziell für einen (CoA, HoA, G)-Zustand zuständig. Es kann also mehrere dieser Timer geben.

Prune-Pending Timer (PPT) Bei Eintritt in den Prune Pending-Zustand wird der Prune Pending Timer gesetzt. Läuft dieser aus, so wird die State Machine in den No Info-Zustand überführt.

3. Methodische Einführung

Dieses Kapitel führt in die zum Verifizieren und Simulieren benötigten Methoden und Programme ein.

3.1. Verifikation

Verifikation bezeichnet im Allgemeinen die Überprüfung des Wahrheitsgehaltes einer Behauptung. Die Protokollverifikation dient analog dazu, eine Protokollspezifikation formal auf ihre logische Korrektheit und fehlerfreie Funktionsweise zu überprüfen. Dazu wird meist eine Erreichbarkeitsanalyse verwendet, da sie sich laut [46] am besten für Finite State Machine (FSM)-basierte Protokollbeschreibungen eignet. Im vorherigen Kapitel wurde bereits die State Machine des Tree Morphing Protokolls vorgestellt, welche in Kapitel 6 verifiziert werden soll.

Zur Verifikation wird der vollständige Zustandsraum des Protokolls durch eine Erreichbarkeitsanalyse untersucht. Dazu werden, ausgehend vom Initialzustand des Systems, alle Kombinationen erreichbarer Zustände und deren Übergänge in einem Erreichbarkeitsgraphen, einem zyklenfreien, gerichteten Graphen, gespeichert. Mit Hilfe dessen ist es möglich, das Protokoll auf bestimmte Eigenschaften zu untersuchen:

- **Deadlock-Freiheit** Ein Protokoll ist Deadlock-frei, wenn kein Zustand existiert, der keinen Folgezustand besitzt. Ausgenommen hiervon sind die explizit definierten Endzustände. Ein Deadlock entsteht laut [92] bei kommunizierenden Prozessen meist beim Wettbewerb um Ressourcen, wobei zwei Prozesse darauf warten, dass der jeweils andere die gewünschte Ressource frei gibt. Nichtdeterministische Deadlocks sind laut [81] die häufigste und am meisten gefürchtete Gefahr in parallelen Systemen.
- **Livelock-Freiheit** Ein Protokoll ist Livelock-frei, wenn es keinen Zustand gibt, der - obwohl noch aktiv - nicht mehr oder nur auf nichtdeterministische Weise mit anderen Prozessen kommuniziert. Livelocks verhindern somit ähnlich Deadlocks das Fortschreiten des Gesamtsystems und sind besonders trügerisch, da sie von außen betrachtet, aufgrund der Ausgaben weiterhin fehlerfrei aussehen können.
- **Terminierung** Ein Protokoll terminiert korrekt, wenn die definierten Endzustände ohne Verklemmungen erreicht werden können.
- **Vollständigkeit** Die Vollständigkeit eines Protokolls ist dann nachgewiesen, wenn alle Zustände des Erreichbarkeitsgraphen erreicht werden können.

Um den für die Verifikation nötigen Erreichbarkeitsgraphen zu erzeugen, gibt es im wesentlichen zwei Möglichkeiten: die Breiten- und die Tiefensuche.

Die Breitensuche (Breadth-First Search (BFS)) generiert, ausgehend vom Initalzustand, zunächst alle direkt folgenden Zustände (siehe Abbildung 3.1(b)). Für jeden dieser Zustände werden danach die jeweils direkt folgenden Zustände generiert (Abbildungen 3.1(c) und 3.1(d)). Dieser Algorithmus wird ausgeführt, bis alle Zustände gefunden wurden.

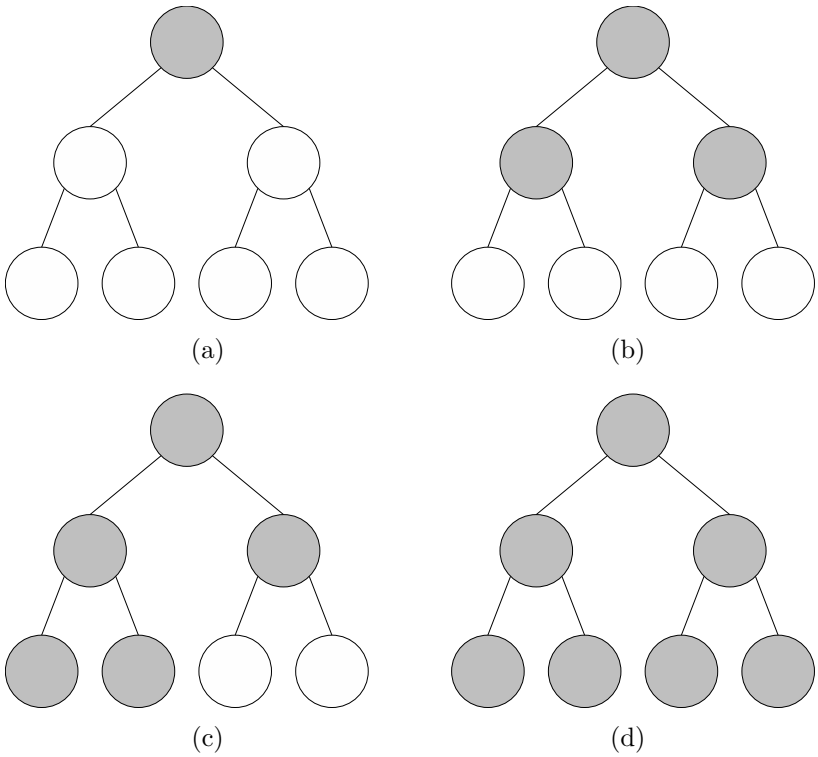


Abbildung 3.1.: Breitensuche

Die Tiefensuche (Depth-First Search) dagegen durchsucht zunächst den jeweils ersten gefundenen Zustand und entwickelt diesen bis ein Endzustand gefunden wurde (siehe Abbildung 3.2(c)). Dann kommt das Verfahren des Backtrackings zum Einsatz. Ist ein Zweig bis zum Ende entwickelt, wird so lange zum vorherigen Zustand zurückgesprungen, bis dieser einen noch nicht entwickelten Zustand enthält. Dieser wird dann wiederum bis zum Ende entwickelt (Abbildung 3.2(d)). Der Algorithmus endet, wenn keine weiteren Zweige zum Entwickeln vorhanden sind.

Beide Verfahren zum Erstellen des Erreichbarkeitsgraphen überprüfen bei der Generierung eines neuen Zustands außerdem, ob dieser bereits über einen anderen Weg erreicht wurde. Dadurch werden doppelte Zustände vermieden und der Erreichbarkeitsgraph klein gehalten.

Ein Problem, das bei der Analyse des Erreichbarkeitsgraphen auftritt, ist die Zustandsraumexplosion (State Space Explosion). Zustände werden bei der Analyse nicht nur abstrakt betrachtet, sondern auch die Belegung der Variablen und der Eingangswarteschlangen

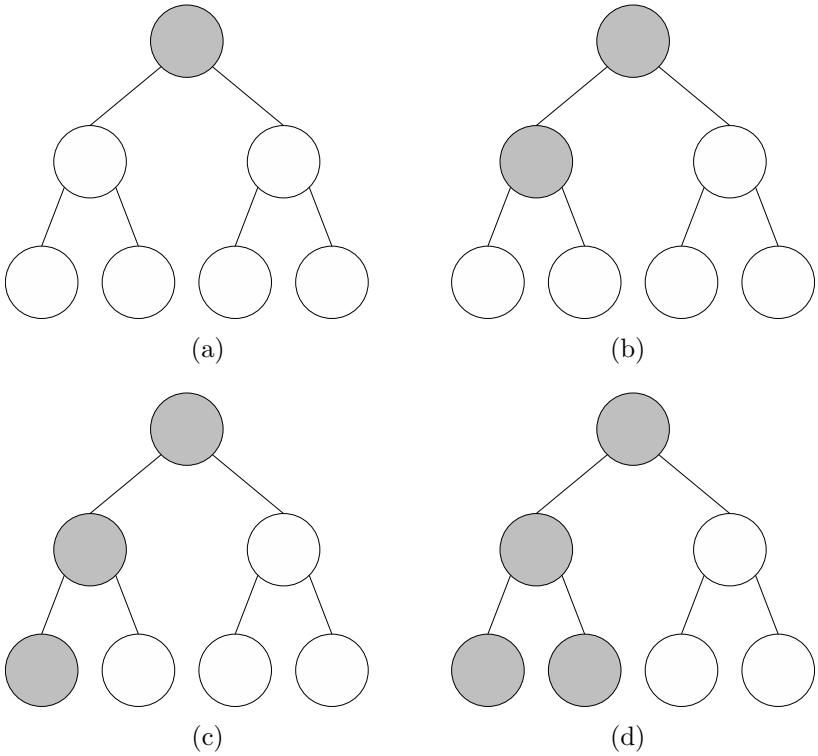


Abbildung 3.2.: Tiefensuche

wird beachtet. Interagieren mehrere Automaten, kann durch Interleaving eine große Anzahl an Ablauf-Kombinationen erstellt werden, was laut [36] selbst bei relativ kleinen Problemen schnell zur Zustandsraumexplosion führen kann. Deser kann durch diverse Verfahren entegengewirkt werden. Der Autor in [37] stellt mehrere dieser Verfahren vor. Hier soll jedoch nur die Partial Order Reduction vorgestellt werden. Diese nutzt den Umstand aus, dass es für die Ausführung des Gesamtsystems unerheblich ist, in welcher Reihenfolge zwei voneinander unabhängige Prozesse abgearbeitet werden. Dadurch können die Permutationen dieser Transitionen zu einer einzigen zusammengefasst werden, was die Anzahl der States deutlich verringert.

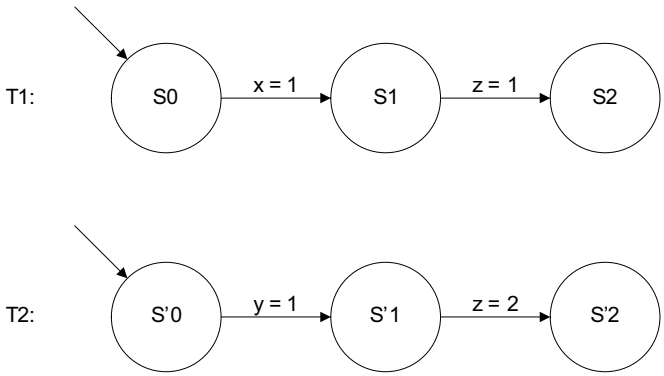


Abbildung 3.3.: State Machines der Prozesse T1 und T2 (angelehnt an [37])

Abbildung 3.3 zeigt beispielhaft die Zustandsautomaten der zwei Prozesse T1 und T2.

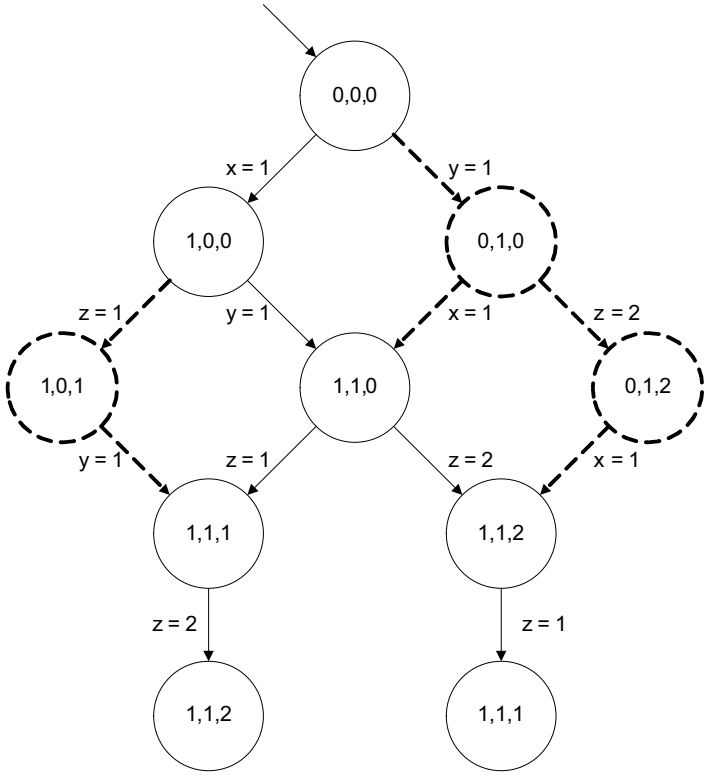


Abbildung 3.4.: Asynchrones Produkt der Prozesse T1 und T2 (angelehnt an [37])

Diese greifen auf die gleichen Variablen zu und verändern deren Werte. Wird das asynchrone Produkt dieser Prozesse gebildet, entsteht der in Abbildung 3.4 dargestellte Erreichbarkeitsgraph. Die Werte in den Zuständen repräsentieren dabei die aktuellen Werte der Variablen x, y und z, wobei diese mit dem Wert 0 initalisiert werden. Die gestrichelt gezeichneten Zustände können aus dem Erreichbarkeitsgraphen entfernt werden, da es für die Verifikation unerheblich ist, in welcher Reihenfolge die Werte der Variablen x und y verändert werden (mutually independent). Daher wird nur ein möglicher Weg vorgegeben. Ebenso ist es unerheblich, in welcher Reihenfolge die Variablen x und z, bzw. y und z verändert werden. Es ist lediglich zu beachten, in welcher Reihenfolge die Variable z verändert wird, da der Zuweisungsoperator (=) nicht kommutativ ist und beide Prozesse auf dieselbe Variable (z) zugreifen. Der Wert dieser Variablen ist also nach der Durchführung beider Zuweisungen entweder 1 oder 2. Daher sind nach der Partial Order Reduction noch immer zwei Endzustände mit verschiedenen z-Werten vorhanden.

Es existieren unzählige Model-Checking Tools, mit deren Hilfe Protokolle auf die zuvor beschriebenen Eigenschaften überprüft werden können. [13] stellt einige Tools und deren Merkmale vor. In Kapitel 6 wird der frei verfügbare Model-Checker Spin [37] genutzt, um die Verifikation durchzuführen. Spin eignet sich sehr gut, um asynchrone Prozesse und endliche Zustandsautomaten (FSM) durch Guarded Commands zu modellieren. Die Verifikation kann randomisiert, interaktiv oder geführt inklusive Sequenzdiagramm (siehe Abbildung 3.5) ausgeführt werden, was bei der Identifikation von Fehlern hilfreich ist. Spin nutzt die Sprache PROMELA (Process Meta-Language), um sowohl die Prozesse als auch die Prüfbedingungen zu beschreiben.

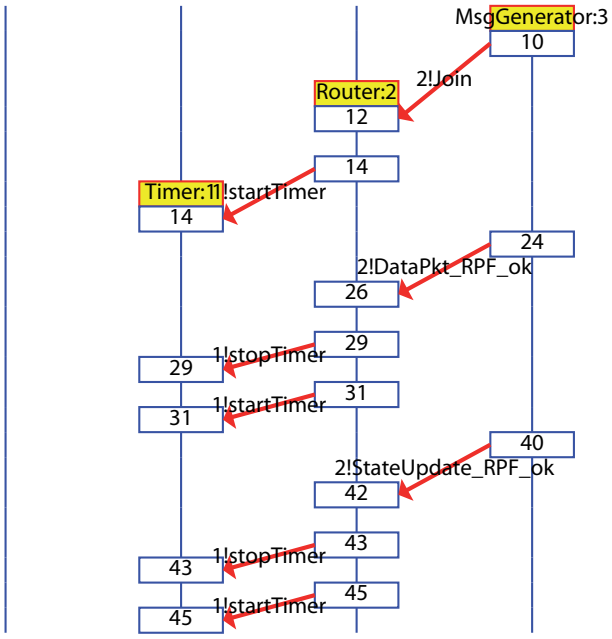


Abbildung 3.5.: Spin Sequenzdiagramm (Auszug)

Als Prüfbedingungen kommen, zusätzlich zu den zuvor vorgestellten Eigenschaften des Erreichbarkeitsgraphen, weitere in Betracht. Dabei können z.B. Variablen auf bestimmte Werte oder Wertebereiche überprüft (*asserts*) oder anhand von sogenannten *Progress Labels* die Abarbeitung des Prozesses kontrolliert werden. Letzteres kann dazu genutzt werden, zu überprüfen, ob ein Prozess wichtige Ausführungspunkte erreicht oder ob er in einer Idle-Loop wartet.

3.2. Simulation

Im Rahmen der Protokollentwicklung soll zusätzlich zur Verifikation eine Protokollimplementierung und deren Simulation durchgeführt werden, da bestimmte Probleme erst offensichtlich werden, wenn eine realitätsnahe Umsetzung durchgeführt wird. Das Protokoll soll daher auf folgende Aspekte hin untersucht werden:

- **Delay Stretch** Der Delay Stretch bezeichnet den Faktor, um den sich die Paketauslieferung während des Tree Morphings im Vergleich zur optimalen Auslieferungsdauer maximal verzögert. Ein Faktor von 2 bedeutet beispielsweise, dass die Übertragungszeit doppelt so hoch ist. Dieser Wert ist wichtig für zeitkritische Anwendungen, wie VoIP, bei der eine komfortable Benutzung nur bis zu einer bestimmten Übertragungszeit möglich ist.
- **Konvergenzzeit** Die Konvergenzzeit beschreibt die Zeit, die die Routerinfrastruktur benötigt, um optimale Weiterleitungszustände zu etablieren.

- **Paketduplikationen** Da bei Multicast-Übertragungen die Pakete dupliziert und über verschiedene Interfaces gesendet werden, können Paketduplikationen auftreten, wenn die States noch nicht konvergiert sind.
- **Paketüberholungen** Durch das Überführen des Verteilbaums kann es zu Paketüberholungen kommen, wenn die „neuen“ Pakete einen Empfänger durch eine kürzere Übertragungsstrecke früher erreichen als die noch auf dem alten Baum befindlichen.
- **Paketverluste** Ein nicht zu unterschätzender Faktor ist die Untersuchung auf mögliche Paketverluste, da diese sich auf die Qualität der Verbindung wesentlich stärker auswirken als die vorher aufgeführten Probleme.
- **Routing-Loops** Entstehen durch die Überführung des Verteilbaums Routing-Loops, werden Pakete in diesem Loop so lange übertragen bis das Hop Count-Feld des IPv6-Headers auf 0 steht. Dies führt zu unnötiger Belastung der Router, was andere Übertragungen beeinflussen kann und somit absolut unerwünscht ist. Zudem können Paketduplikationen durch fehlerhafte States eines Routers entstehen.

Die Implementierung könnte auf einem echten System, wie z.B. Linux oder XORP [64], einer speziellen Router-Plattform, durchgeführt werden. Um Messergebnisse zu erhalten, muss dazu allerdings eine echte Test-Umgebung geschaffen und für verschiedene Tests umkonfiguriert werden. Da Multicast-Routingeffekte jedoch nur in großen Netzen zuverlässig erkennbar und außderdem stark topolgieabhängig sind, ist es im Rahmen dieser Arbeit nicht möglich, diese aufzubauen und für verschiedene Tests umzukonfigurieren. Alternativ könnte diese echte Implementierung auf dem Overlay-Netzwerk PlanetLab [63] ausgeführt werden, welches durch in der Welt verteilte Knoten eine virtuelle Netzwerkumgebung bietet. Da diese allerdings keine realistische Internet-Topologie darstellt und zudem starr ist, können keine Tests für unterschiedliche Topologien durchgeführt werden. Aus diesem Grund soll stattdessen eine Simulation durchgeführt werden, die reale und tagesaktuelle Topologiedaten [56, 89] nutzen kann. Durch die visuelle Darstellung des Paketflusses in einem Simulator kann zusätzlich leicht überprüft werden, ob das Protokoll gemäß der Spezifikation arbeitet.

Simulatoren bilden einen Ausschnitt der Realität möglichst genau nach und werden meist zur Analyse eines Systems genutzt, wenn dessen theoretische Untersuchung zu komplex ist. Das Systemverhalten großer Modelle kann somit experimentell untersucht werden. Aus den Simulationsergebnissen können Rückschlüsse auf das reale System gezogen und dieses verbessert werden.

Diskrete Ereignissimulatoren (Discrete Event Simulators) berechnen Zustände, im Gegensatz zu kontinuierlichen Simulatoren, nur zu bestimmten Zeitpunkten. Dazu werden anstehende Ereignisse in einer Warteschlange mit aufsteigendem Ausführungszeitpunkt gespeichert und nacheinander ausgeführt. Um bei jedem Simulationslauf die gleichen Ergebnisse zu liefern, wird der Pseudo-Zufallszahlengenerator mit dem gleichen (konfigurierbaren) Wert initialisiert (Seed-Value), welcher z.B. für die zufälligen Back-off Zeiten der Netzinterfaces verwendet wird.

Durch die Simulation soll nicht nur die Korrektheit der Implementierung nachgewiesen, sondern insbesondere der Zeitabschnitt, in dem das Tree Morhping stattfindet, näher untersucht werden. Dazu werden Simulationen mit vorgegebenen Parametern durchgeführt. Die wichtigsten dabei sind die Topologie des Netzes, die Entfernung der Router pDR und nDR zueinander und die Frequenz mit der die mobile Multicastquelle Daten versendet. Dabei wird erwartet, dass sich die Konvergenzzeit und der Delay Stretch mit steigender pDR-nDR Distanz erhöhen. Paketduplikationen, -überholungen und -verluste sollten dagegen nur bei bestimmten, unwahrscheinlichen Netzkonstellationen auftreten. Bei korrekter Implementierung sind Routing-Loops hingegen auszuschließen. Durch die gewonnenen Ergebnisse können Schlüsse auf die Leistungsfähigkeit des Tree Morphing Protokolls in realen Umgebungen gezogen, eventuelle Fehler korrigiert und ggf. Optimierungspotentiale erkannt werden.

Um ein Protokoll zu simulieren, bietet es sich an, einen existierenden Netzwerksimulator um die Protokollimplementierung zu erweitern und die vorhandenen Funktionen des Simulators zu benutzen. Idealerweise bietet der Simulator bereits die für die Implementierung benötigten Netzwerkprotokolle IPv6, UDP, sowie MLD, welche somit nicht selbst implementiert werden müssen. Es existieren diverse Netzwerksimulatoren, von denen im folgenden nur NS-2 [59] und OMNeT++ (Objective Modular Network Testbed in C++) [60] verglichen werden sollen, da sie die bekanntesten frei[1] verfügbaren Tools sind. Die Autoren in [12] vergleichen die verfügbaren Netzwerksimulatoren und kommen zu dem Ergebnis, dass sowohl NS-2 als auch OMNeT++ sehr gut geeignet sind, um eigene Protokolle in diesen zu implementieren. Bei OMNeT++ wird zusätzlich die saubere Trennung von Modulen (programmiert in C++) und Simulations-Modellen (geschrieben in der „NEtwork Description language“ (NED) [99]) erwähnt. Weiterhin sind in OMNeT++ die einzelnen Module nicht voneinander abhängig, sondern werden erst zur Laufzeit zu größeren Einheiten zusammengesetzt. Dadurch können diese, sauber voneinander getrennt, implementiert werden.

Da für OMNeT++ ein leistungsstarkes Protokollframework (IPv6SuiteWithINET) [48] inklusive MIPv6 existiert und in einer Projektgruppe an der HAW Hamburg ebenfalls OMNeT++ eingesetzt wird, soll dieser Simulator für die Implementierung und Simulation des Tree Morphing Protokolls genutzt werden.

[1] OMNeT++ ist frei verfügbar für akademische und nicht kommerzielle Einsatzzwecke, NS-2 ist verfügbar unter der GNU General Public License Version 2 (GPLv2) [52].

4. Design des Tree Morphing Protokolls

In diesem Kapitel wird das Design des Tree Morphing Protokolls vorgestellt. Es wurde vereinfacht bereits in [18] präsentiert.

Zunächst werden die Anforderungen an das Protokoll aufgezeigt. Danach werden zwei Protokollentwürfe vorgestellt, von denen eines für die weiteren Betrachtungen ausgewählt wird. Anschließend wird der Ablauf des Protokolls dargestellt. Das Kapitel schließt mit einer Diskussion.

4.1. Anforderungen

Das Tree Morphing Protokoll erfordert, dass die Routerinfrastrukur nach einem Handover der Multicastquelle mit Hilfe der Paketverarbeitung ihre Weiterleitungszustände aktualisiert. Insbesondere muss nach einem Mobile IPv6-Handover der von der vorhergehenden Quelladresse pCoA ausgehende Multicast-Verteilbaum zur nachfolgenden nCoA übertragen werden (siehe Abbildung 2.12). Hierzu müssen Pakete, die von den Routern inhaltlich ausgewertet werden, sowohl die kontextuellen als auch die topologischen Verteilbauminformationen enthalten. Dabei werden die kontextuellen Informationen, gegeben durch das Tupel (HoA, G), dazu verwendet, den zu überführenden Verteilbaum zu identifizieren. Die topologischen Informationen (nCoA, G) hingegen werden für das Routing benötigt. Diese drei IP-Adressen müssen unmittelbar nach dem Handover allen Routern des vorherigen und – soweit möglich – des neuen Verteilbaums bekannt gemacht werden.

Multicastpakete müssen dem Tree Morphing Protokoll folgend, unmittelbar nach dem Handover einer mobilen Quelle vom next Designated Router (nDR) zum previous Designated Router (pDR) weitergeleitet werden. Dazu werden sie per Source Routing [22] zum pDR gesendet. Die beiden im nächsten Abschnitt vorgestellten Protokollentwürfe verwenden verschiedene Möglichkeiten, um dieses Source Routing zu realisieren.

Da die Aktualisierungspakete von den Internet Core Routern verarbeitet werden, muss beim Protokollentwurf eine leichtgewichtige Lösung gefunden werden, die nur einen geringen Verarbeitungsaufwand zur Folge hat. Daher werden die Aktualisierungsinformationen mit den unmittelbar nach dem Handover fliessenden Nutzdatenpaketen verschickt. Durch diesen Mechanismus des „Piggybacking“ werden unerwünschte Komplikationen, wie z.B. Paketüberholungen vermieden. Außerdem müssen dadurch keine zusätzlichen Signalisierungspakete übertragen werden, was unerwünschten Zusatzaufwand durch die Übertragung vermeidet. Der Aufwand, Pakete mit zusätzlichen Headern auszustatten

und diese wieder zu extrahieren ist im Gegensatz zum Versenden zusätzlicher Paketen mit verschwindend geringem Prozessierungs-Aufwand verbunden, da Implementierungen dafür üblicherweise Pointerarithmetik verwenden. Selbstständige Update-Pakete müssen zusätzlich zur üblichen Paketauswertung eventuelle Firewallregeln durchlaufen, was mit weiterem Verarbeitungsaufwand verbunden ist. Obwohl auch weiterhin Nutzdatenpakete in falscher Reihenfolge eintreffen können, enthalten zumindest die ersten Pakete die Updateinstruktionen, um robust gegenüber Störungen zu sein. Somit kann das Update durch jedes der mit dem Update versehenen Pakete durchgeführt werden.

Schließlich muss beim Protokollentwurf darauf geachtet werden, es gegenüber Angreifern abzusichern. Um fehlerhafte Pakete erkennen und verwerfen zu können, wird die im Mobile IPv6 Binding Update genutzte Sequenznummer verwendet. Diese Sequenznummer wird nach jedem Handover erhöht. Das mehrfach in den ersten Multicastdatenpaketen nach dem Handover gesendete Update enthält somit dieselbe Sequenznummer. Einerseits können damit bereits verarbeitete Updates ohne weiteren Aufwand verworfen, andererseits auch Updatepakete mit gefälschten Sequenznummern leicht erkannt und verworfen werden.

Die vorzunehmenden Zustandsaktualiserungen in der Vermittlungsinfrastruktur sind ebenso anfällig für einen Identitätsdiebstahl wie z.B. Mobile IPv6 oder das Neighbor Discovery Protocol (NDP). Daher ist eine robuste, kryptographisch starke Authentifizierung der Signalisierung notwendig, welche allerdings ohne Rückkommunikation erfolgen muss. Diese Einwegauthentifizierung muss selbstkonsistent beweisen, dass die Aktualisierungspakete tatsächlich von der mobilen Multicastquelle, also der Besitzerin der Home Adresse stammen und kann analog zu [5] und [4] mit Hilfe kryptographisch generierter Adressen (CGAs) [6] erreicht werden. Dazu werden mit der State Update Message in demselben Paket auch die für die CGA-Authentifizierung nötigen Informationen übermittelt. Im Einzelnen sind dies die CGA Parameter und CGA Signature Option. Damit können Sender den Besitz der Home Address ohne weitere Infrastruktur kryptographisch stark nachweisen. Außerdem wird das Paket durch eine RSA-Signatur authentifiziert, was eine fälschungsfreie Übertragung sicherstellt. Veränderte Pakete können somit erkannt und verworfen werden. Details zu CGAs finden sich in [6, 43] und Abschnitt 2.2.2.

Eine Evaluierung der hier spezifizierten Anforderungen und eine Risiko-Abschätzung folgen in Abschnitt 5.

4.2. Zwei Protokollentwürfe

Im Folgenden werden zwei Ansätze zum Design des Tree Morphing Protokolls vorgestellt. In Abschnitt 4.2.1 wird eine neue Hop-by-Hop Option präsentiert, die alle nötigen Informationen zum Überführen des Baumes beinhaltet. Abschnitt 4.2.2 stellt eine Lösung vor, die lediglich bestehende IPv6 Header verwendet. Dafür werden lediglich neue Headertypen definiert. Schließlich werden die Vor- und Nachteile der Ansätze in Kapitel 4.2.3 diskutiert und der geeignetere Ansatz für weitere Analysen ausgewählt.

4.2.1. State Update Message als Hop-by-Hop Option

Für das Tree Morphing sind folgende Werte nötig:

- Care-of Address (CoA)
- Home Address (HoA)
- Gruppenadresse (G)

Da sowohl die Routinginfrastruktur, als auch die Multicastempfänger diese Nachricht erhalten und verarbeiten müssen, werden diese Daten in einem Hop-by-Hop Header verpackt, der laut [22] von jedem Rechner ausgewertet werden muss.

Der Header ist in Abbildung 4.1 dargestellt.

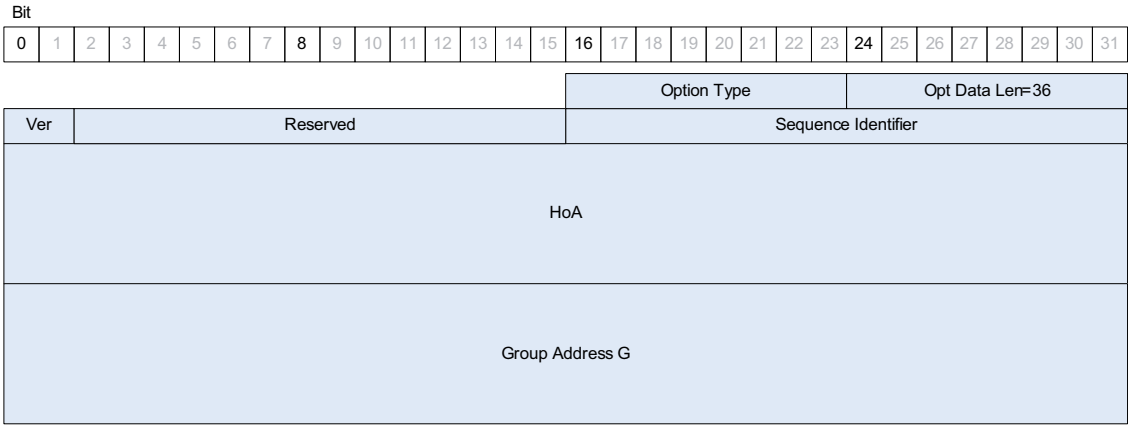


Abbildung 4.1.: Hop-By-Hop Option Header: State Update Message

Option Type Das 8 Bit lange *Option Type*-Feld enthält mehrere Optionen: Die höchstwertigsten 2 Bit müssen auf Null gesetzt sein und legen damit fest, dass ein IPv6 Node, der den Optionstyp nicht versteht, die Option überspringen und das Paket weiterverarbeiten muss. Das folgende 3. Bit ist ebenfalls auf Null gesetzt und besagt, dass die Daten dieses Headers sich während der Übertragung nicht verändern dürfen. Die restlichen Bits identifizieren diesen Header als State Update Message und müssen von der IANA zugewiesen werden.

Option Data Len Dieses 8 Bit lange Feld gibt die Länge des Headers (ohne Option Type und Option Data Len) in Octets an. Dieser Header ist 68 Octets lang.

Version Das 2 Bit lange *Version*-Feld gibt die Versionsnummer des Headers an. Die Versionsnummer dieses Headers ist 1. Die Werte 0, 2 und 3 sind reserviert.

Sequence Identifier Dieses 16 Bit lange Feld enthält die Sequenznummer der Nachricht. Mehrfach empfangene, gleiche Nachrichten haben die selbe Sequenznummer und können somit leicht von Routern und Multicast-Empfängern erkannt und ignoriert werden, ohne den gesamten Datensatz (pCoA, HoA, G) zu vergleichen. Auch können Pakete mit einer ungültigen Sequenznummer verworfen werden. Dies kann allerdings erst ab dem zweiten Paket geschehen.

HoA Dieses Feld enthält die 128 Bit lange Home Address, welche den HoA-Identifier des SSM-Kanals darstellt.

G Dieses Feld enthält den 128 Bit langen Gruppen-Identifier des SSM-Kanals.

4.2.1.1. State Update Message: Paketaufbau

Der Paketaufbau der State Update Message (siehe Abbildung 4.2) ist in allen Phasen des Tree Morphing Protokolls gleich, lediglich die Inhalte der Header unterscheiden sich dabei.

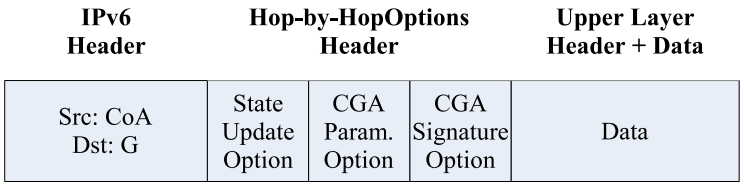


Abbildung 4.2.: IPv6-Paketaufbau mit State Update Message als Hop-by-Hop Option

Tree Elongation

Das Paket wird vom MN mit der aktuell gültigen CoA an den previous Designated Router (pDR) gesendet. Auf den IPv6 Header folgt der Hop-by-Hop Option Header mit der State Update Nachricht und den CGA Parameter und CGA Signature Optionen (siehe Abschnitt 2.2.2). Diese drei Header enthalten alle für das Update nötigen Informationen. Da das Update in ein Multicastdatenpaket eingefügt wird, enthält es schließlich auch Nutzdaten.

Router, die das Paket erhalten, müssen die Hop-by-Hop Optionen auswerten und erkennen das Update anhand der State Update Nachricht. Die zum Routing benötigten Informationen werden den Headern gemäß Tabelle 4.1 entnommen.

Reguläre Multicastübertragung

Erreicht das Paket den pDR, wird die im State Update Header gespeicherte Gruppenadresse als Zieladresse in den IPv6 Header eingefügt und das Paket an den vor dem Handover etablierten Verteilbaum weitergeleitet. Dadurch, dass die Gruppenadresse in der State Update Option gespeichert ist, wird kein weiterer Routing Header benötigt, um

IP-Adresse	Speicherort
HoA	State Update Option: HoA-Feld
CoA	IPv6 Header: Source Address
G	State Update Option: G-Feld

Tabelle 4.1.: Headerposition der HoA, CoA und G IP-Adressen beim Tree Elongation (State Update Message als Hop-by-Hop Option)

das Paket vom MN zum pDR zu leiten. Alle weiteren Router und die Empfänger erhalten die zum Update nötigen Daten aus den Headern wie in Tabelle 4.2 angegeben.

IP-Adresse	Speicherort
HoA	State Update Option: HoA-Feld
CoA	IPv6 Header: Source Address
G	IPv6 Header: Destination Address (und State Update Option: G-Feld)

Tabelle 4.2.: Headerposition der HoA, CoA und G IP-Adressen bei der Multicastverteilung (State Update Message als Hop-by-Hop Option)

Um Paketverlusten im Netzwerk entgegenzuwirken, wird die State Update Hop-by-Hop Option in die erste Gruppe der Nutzdatenpakete eingefügt. Dadurch erhalten die Empfänger meist mehrere State Updates. Um gleiche Nachrichten zu erkennen, wird die Sequenznummer in der State Update Nachricht nach jedem Handover erhöht. Bereits empfangene und verarbeitete Updates können somit erkannt und ohne weiteren Verarbeitungsaufwand verworfen werden.

4.2.2. State Update Message aus bestehenden IPv6 Headern

Ein alternativer Ansatz, das Tree Morphing Protokoll zu implementieren, ist es, die zum Überführen des Verteilbaums nötigen Informationen durch das Übertragen einer aus bestehenden Headern zusammengesetzten Nachricht zu übermitteln. Dazu werden minimale Erweiterungen an der bestehenden Mobilitätssignalisierung durchgeführt, um ein Höchstmaß an Einfachheit und Standardkonformität zu gewährleisten. Die nachfolgend vorgestellte Implementierung des Tree Morphings ist deshalb allein durch die zweckorientierte Kombination bestehender Protokollstrukturen und unter minimaler Ergänzung einer vorhandenen Hop-by-Hop Option realisiert. Sie kommt ohne Änderungen etablierter Header aus. Vorhandene Protokollimplementierungen wie Mobile IPv6 [43], welches für die Aktualisierung der Multicastempfänger benötigt wird, und PIM-SSM [27] können somit leicht angepasst werden, da alle Verarbeitungsfunktionen bereits vorhanden sind. Weiterhin bringt dieser leichtgewichtige Ansatz Vorteile für die Robustheit des Protokolls mit sich, da die vorhandenen, standardisierten Header und Protokolle bereits genauen Analysen und praktischen Einsatzerfahrungen unterzogen wurden.

Im Folgenden werden zunächst die neuen Headertypen der „Router Alert Option“ und des „Routing Headers“ vorgestellt, anschließend wird der Paketaufbau in den unterschiedlichen Phasen des Tree Morphings dargestellt.

4.2.2.1. Router Alert Option

Die Signalisierung einer neuen Verteilbaumquelle nach dem Mobile IPv6 Handover erfolgt auf der Netzwerkschicht durch Zusatzinformationen in den Datenpaketen. Die benötigten Informationen, Gruppenadresse, Home Address und Care-Of Address, sowie die Authentifizierungsnachweise, sind bereits Bestandteil der Binding Update Messages, wie sie mobile Teilnehmer – zumindest an ihre Unicast-Empfänger – nach jedem Netzwechsel verschicken. Die State Update Message kann deshalb aus verschiedenen Headern der Mobile IPv6 Netzwerkschicht zusammengesetzt werden und erfordert keine Neudefinition von Datenstrukturen. Multicast Tree Morphing Signalisierungen können so prinzipiell in transparenter Weise mit regulären, CGA-authentifizierten [5] Binding Updates prozessiert werden. Sie müssen dabei allerdings von jedem Router entlang des Paketweges interpretiert werden.

Um eine solchermaßen transparente Multicast-Mobilitätssignalisierung zu ermöglichen, werden die Pakete um eine Router Alert Option im Hop-by-Hop Header [73] ergänzt. Dieses Format wird benutzt, um Routern zu signalisieren, weitere Paketverarbeitungen zu veranlassen. Durch die Platzierung der Option im Hop-by-Hop Header müssen weitergehende Instruktionen von jedem Router auf dem Weg zum Ziel prozessiert werden.

Die Router Alert Option besitzt das in Abbildung 4.3 dargestellte Format.

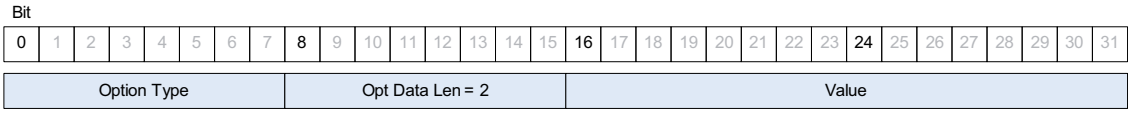


Abbildung 4.3.: Router Alert Option

Option Type Das 8 Bit lange *Option Type*-Feld besitzt den Wert 5. Die ersten beiden Null-Bits spezifizieren laut [22], dass Router, die die Option nicht verstehen, diese überspringen und mit der Verarbeitung des Pakets fortfahren müssen. Das folgende Null-Bit definiert, dass sich die Option während der Übertragung zum Ziel nicht ändern darf.

Opt Data Len Die Datenlänge der Option in Octets ohne die Felder *Option Type* und *Opt Data Len* wird in dem 8 Bit langen Feld *Option Data Length* gespeichert. Der Wert dieses Feldes beträgt 2.

Value Dieser von der IANA zuzuweisende Wert für diesen speziellen Header spezifiziert die Semantik der State Update Message. Die weitere Paketverarbeitung der nachfolgend beschriebenen Optionsheader wird durch diesen Wert eindeutig festgelegt, sodass ihre Verarbeitung mit dem Ziel, die Forwarding States der Router zu aktualisieren, wohldefiniert erfolgen kann.

4.2.2.2. Routing Header Type 7

Für das Tree Morphing wird im Folgenden ein eigener Routing Header Typ definiert[1]. Der genaue Inhalt des Headers wird in Abschnitt 4.2.2.3 beschrieben. Die Type-Nummer des Routing Headers wurde durch Auffinden bereits vorhandener Routing Header ermittelt. Eine Übersicht über die Routing Header Typen 0-6 findet sich in Anhang A. Routing Header Type 7 ist frei für den hier vorgestellten Header.

Der Routing Header Type 7 besitzt das in Abbildung 4.4 dargestellte Format.

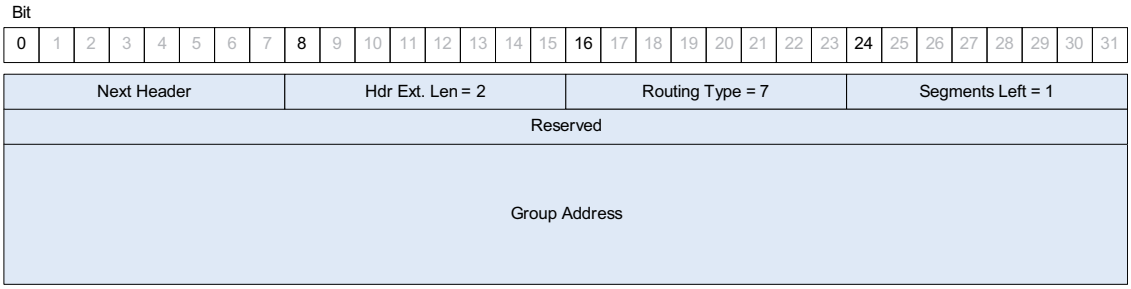


Abbildung 4.4.: Routing Header Type 7

Next Header Das 8 Bit lange *Next Header*-Feld identifiziert den Header, der unmittelbar auf den Routing Header folgt.

Hdr Ext Len Der Wert des 8 Bit langen *Hdr Ext Len*-Feldes muss auf 2 gesetzt sein. Es beschreibt die Länge des Routing Headers in 8 Octet-Einheiten exklusive der ersten 8 Octets.

Routing Type Dieser von der IANA zuzuweisende Wert für diesen speziellen Header spezifiziert den Typ des Routing Headers. Der nächste aktuell freie Wert für dieses Feld ist 7. In Anhang A werden die bereits vergebenen Header aufgeführt.

Segments Left Der Wert dieses 8 Bit langen Feldes muss auf 1 gesetzt sein.

[1] Der in Mobile IPv6 definierte Routing Header Type 2 kann, obwohl er den gleichen Aufbau wie der vorgestellte besitzt, nicht genutzt werden, da das vorhandene IPv6-Adressfeld laut [43] nur eine Unicast-Adresse enthalten darf, hier allerdings eine Multicast-Gruppenadresse hinterlegt werden muss.

Reserved Dieses 32 Bit lange Feld ist für zukünftige Erweiterungen reserviert und muss vom Sender mit Nullen initialisiert werden. Der Empfänger muss dieses Feld ignorieren.

Group Address Dieses 128 Bit lange Feld enthält die SSM-Gruppenadresse.

4.2.2.3. State Update Message: Paketaufbau

Der vollständige Paketaufbau unterscheidet sich nach den Phasen des Tree Morphing Protokolls und wird in den folgenden Abschnitten beschrieben.

Tree Elongation

Abbildung 4.5 zeigt den Paketaufbau in der Phase des Tree Elongation. Dabei wird das Paket vom MN mit der aktuell gültigen CoA an den previous Designated Router (pDR) gesendet. Gemäß der Headeranordnung im IPv6 Standard [22] folgt darauf der Hop-by-Hop Option Header mit der Router Alert Option, welche im Abschnitt 4.2.2.1 erläutert wurde. Anschließend folgt der Destination Options Header, der die Home Address Option gemäß [43] enthält, mit der den Empfängern die HoA mitgeteilt wird. Im Mobility Header [43] werden die CGA Parameter Option und die CGA Signature Option hinterlegt. Sie übertragen die für die CGA-Authentifizierung nötigen Daten. Dabei ist zu beachten, dass mehrere CGA Parameter Optionen aufeinander folgen können. Als letzter Header folgt der Routing Header vom Typ 7, dessen Funktion in der Weiterleitung des Pakets an die Gruppe nach seinem Eintreffen am pDR besteht. Dazu wird die Gruppenadresse G in das erste Adress-Feld eingetragen. Schließlich folgen der Nutzdatenheader (Upper Layer Header), sowie die Nutzdaten. Die zum Routing benötigten Informationen werden den Headern gemäß Tabelle 4.3 entnommen.

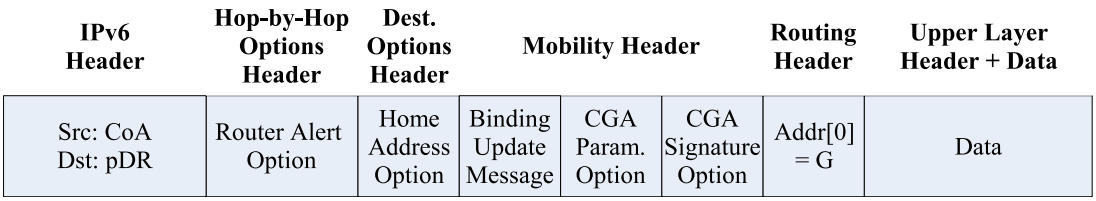


Abbildung 4.5.: IPv6-Paketaufbau mit State Update Message aus bestehenden IPv6 Headern beim Tree Elongation vom Next Designated Router zum previous Designated Router

IP-Adresse	Speicherort
HoA	Destination Options Header: Home Address Option
CoA	IPv6 Header: Source Address
G	Type 7 Routing Header

Tabelle 4.3.: Headerposition der HoA, CoA und G IP-Adressen beim Tree Elongation (State Update Message aus bestehenden IPv6 Headern))

Reguläre Multicastübertragung

Die weitere Multicastübertragung wurzelt am pDR und findet über den vor dem Handover etablierten Verteilbaum statt. Im Paketaufbau (siehe Abbildung 4.6) entfällt der Routing Header, nachdem der Source Routing Übergangspunkt pDR erreicht ist. Hierbei wurde entsprechend die Zieladresse im IPv6 Header mit der Gruppenadresse G des Routing Headers ersetzt (siehe Tabelle 4.4).

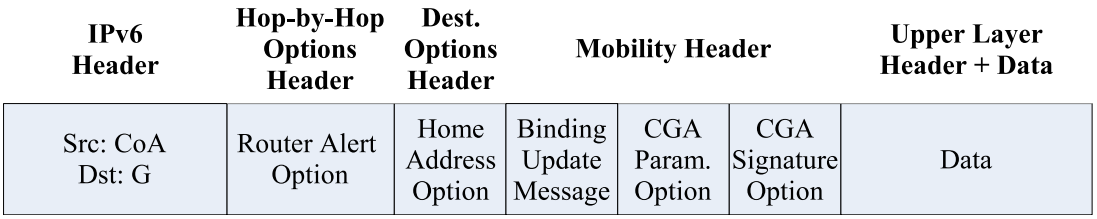


Abbildung 4.6.: IPv6-Paketaufbau mit State Update Message aus bestehenden IPv6 Headern vom previous Designated Router zur Multicastgruppe

IP-Adresse	Speicherort
HoA	Destination Options Header: Home Address Option
CoA	IPv6 Header: Source Address
G	IPv6 Header: Destination Address

Tabelle 4.4.: Headerposition der HoA, CoA und G IP-Adressen bei der Multicastverteilung (State Update Message aus bestehenden IPv6 Headern)

4.2.3. Vergleich und Auswahl des geeigneteren Entwurfs

Im Folgenden werden die beiden Ansätze, die für das Tree Morphing nötigen Daten per *State Update Message als Hop-by-Hop Option* (vgl. Kapitel 4.2.1) und die *State Update Message aus bestehenden IPv6 Headern* (vgl. Kapitel 4.2.2) zu verschicken, untersucht.

Lediglich einen neuen Header in den Nutzdatenstrom einzufügen, wie es der erste Ansatz (*State Update Message als Hop-by-Hop Option*) vorschlägt, hat den Vorteil, dass alle Daten in einem einzigen Header hinterlegt werden können. Zusätzlich müssen für eine vollständige State Update Message nur noch die beiden CGA-Header (CGA Parameter Option und CGA Signature) eingefügt werden. Dies ermöglicht auf den ersten Blick eine einfache Implementierung, jedoch muss auch der Mobile IPv6-Stack jedes beteiligten Knotens (Router und Empfänger) angepasst werden, um das Update mit den Daten des neuen Headers auszuführen. Dies führt zu unerwünschten Änderungen an bestehendem, gut getesteten Code.

Der Entwurf, die *State Update Message aus bestehenden IPv6 Headern* zusammenzusetzen, verwendet dagegen die standardisierten Mobile IPv6-Header (Home Address Option und Binding Update Message), um das Binding Update durchzuführen. Es sind also nur minimale Änderung an bestehendem Code nötig. Dies ist wünschenswert, da somit

Implementierungsfehler seltener auftreten und robuste, getestete Protokolle und deren Quellcode weiterverwendet werden können. Da das Binding Update nach einem Handover bei beiden Entwürfen gesendet werden muss, bringt der zweite Entwurf den Vorteil, dass das Update bereits in der Multicastsignalisierung enthalten ist. Weiterhin wurde das in [5] standardisierte Verfahren, CGAs zum Sichern der Binding Updates zu benutzen, eingesetzt.

Zusätzlich zu den regulären Mobile IPv6-Headern werden lediglich die ebenfalls standardisierten IPv6 Header Router Alert Option und Routing Header genutzt. Dies ermöglicht eine wesentlich einfachere Implementierung, da vorhandener Code weiterbenutzt werden kann und lediglich die Semantik der Headerzusammensetzung integriert werden muss.

Da die Vorteile des zweiten Entwurfs (*State Update Message aus bestehenden IPv6 Headern*), wie Weiterverwendung des Mobile IPv6-Codes und die Nutzung vorhandener IPv6 Header, überwiegen, wird im weiteren Verlauf der Arbeit mit diesem Entwurf weitergearbeitet.

4.3. Protokollablauf

4.3.1. Arbeitsweise der mobilen Multicastquelle

Nach einem Layer 2 Handover und der darauf folgenden Adresskonfiguration sendet die mobile Multicastquelle ihre Nutzdaten per Source Routing an den previous Designated Router (pDR). Zusätzlich werden die Home Address Option im IPv6 Destination Option Header und die State Update Message im Hop-by-Hop Option Header gesendet, um die für das (Binding-) Update nötigen Informationen zu übertragen. Außerdem werden die CGA-Optionen eingefügt (siehe Abbildung 4.5), um das Updatepaket zu authentifizieren.

Die mobile Multicastquelle muss die Nutzdatenpakete solange mit den Update-Headern versehen, bis sie ein Binding Acknowledgement vom pDR erhält. Dadurch ist der Baumanschluß sichergestellt. Trotzdem können noch weitere Pakete mit den zusätzlichen Headern versehen werden. Dies obliegt jedoch der Verantwortung der mobilen Multicastquelle und kann von Faktoren wie z.B. Robustheitsanforderungen und Paketaufkommen abhängen.

Parallel zu dem vorhergehend beschriebenen Verhalten führt die mobile Multicastquelle ein reguläres Binding Update mit ihrem Home Agent durch.

4.3.2. Arbeitsweise der Router

Die Router auf dem Weg zu den Empfängern erhalten das State Update Paket und müssen laut [22] den Hop-by-Hop Option Header analysieren, dessen erste Option die in Abschnitt 4.2.2.1 definierte Router Alert Option ist. Das *Value*-Feld dieser Option spezifiziert, dass es sich um eine State Update Message handelt. Daher müssen auch die im Protokoll definierten weiteren Headerbestandteile des Pakets untersucht werden, welche ohne diese Option von Routern ignoriert werden. Bei dem darauffolgenden Header handelt es sich

um den Destination Option Header. Die darin hinterlegte Option ist die Home Address Option, aus welcher der Router die HoA des Senders entnimmt.

Hiernach folgt der Mobility Header mit der Binding Update Message sowie den CGA-Optionen. Aus der Binding Update Message wird u.a. die Sequenznummer entnommen. Ist diese gültig, wird aus den CGA Parameter Optionen die CGA Parameter Datenstruktur entnommen. Mit dieser wird die in [6] beschriebene CGA-Überprüfung durchgeführt. Dies beinhaltet u.a. den Vergleich des in der Datenstruktur hinterlegten Subnet Prefixes mit dem der Source Address des IPv6 Header. Des weiteren wird ein SHA-1 Hash [26] über die Datenstruktur gebildet und die 64 ersten Bits des Ergebnisses mit dem Interface Identifier der Source Address verglichen. Nach erfolgreicher Überprüfung kann die aktuelle CoA des Senders eindeutig mit dem enthaltenen Public Key verbunden werden. Die folgende CGA Signature Option enthält eine kryptographische Signatur über die Home Address Option. Diese Signatur wird mit Hilfe des RSA-Algorithmus überprüft. Als Eingabeparameter dient hierbei der zuvor entnommene Public Key. Ist die Signatur gültig, kann davon ausgegangen werden, dass die HoA mit der aktuellen CoA verbunden ist, der Sender also Besitzer der HoA ist. Somit ist die Aktualisierung des Verteilbaums kryptographisch stark authentifiziert. Sollte an einer beliebigen Stelle des Ablaufs eine Überprüfung fehlschlagen, so ist die Paketverarbeitung unverzüglich abzubrechen und das Paket sofort zu verwerfen.

Die weitere Paketverarbeitung hängt davon ab, ob der Router auf dem Pfad der Tree Elongation oder im regulären Multicast-Verteilbaum liegt. Bei ersterem entnimmt er die Gruppenadresse G dem Type 7 Routing Header (siehe Tabelle 4.3), implementiert den (nCoA, HoA, G) Multicast-Forwardingstate und leitet das Paket weiter. Ist der Router im regulären Multicast-Verteilbaum, arbeitet er wie in den *State Injection* und *Extended Forwarding* Algorithmen in [84] beschrieben.

4.3.3. Arbeitsweise des pDR

Erreicht das Paket den pDR, wird das Paket zunächst gemäß der im vorherigen Abschnitt besprochenen Weise abgearbeitet. Danach überprüft der pDR, ob ein entsprechender (·, HoA, G) Zustand in seiner Multicast-Weiterleitungstabelle existiert und damit ein Mutlicastverteilbaum vor dem Handover vorhanden war. Ist dies der Fall, wird der Type 7 Routing Header entfernt und die Multicastadresse G als Destination Address in den IPv6 Header eingesetzt. Dann aktualisiert auch der pDR seine Multicast Forwarding States und leitet das Paket an die Multicast Gruppe weiter.

Um das Protokoll gegenüber möglichen Paketverlusten abzusichern, wird vom pDR verlangt, das erste eintreffende Updatepaket mit einem in Mobile IPv6 standardisierten Binding Acknowledgement zu bestätigen. Der pDR ist dabei der letzte sinnvolle Router, der die erfolgreiche Paketübertragung bestätigen kann, bevor es auf dem Multicast-Verteilbaum (möglicherweise) multipliziert wird.

Besteht kein (·, HoA, G)-Zustand, werden die Pakete verworfen, da es sich um gefälschte Pakete handeln könnte. Es wird dann auch kein Binding Acknowledgement Paket gesendet.

4.3.4. Arbeitsweise der Multicastempfänger

Multicastempfänger erhalten die Pakete und analysieren sie nach vorgenanntem Algorithmus. Ist die CGA-Überprüfung erfolgreich, interpretieren sie die Home Address Option im Destination Option Header. Daraufhin wird nicht nur der zur HoA gehörige Mobile IPv6 Binding Cache-Eintrag mit der CoA, sondern auch ebenso der Multicast Binding Cache-Eintrag aktualisiert. Schließlich werden die Daten mit der korrekten HoA und G an die Transportschicht weitergereicht, um eine störungsfreie Multicastkommunikation auf der Anwendungsschicht zu gewährleisten.

4.4. Diskussion: Data Plane, Control Plane

Das Einfügen der State Updates (Control Plane Daten) in den laufenden Nutzdatenstrom (Data Plane) birgt einige Probleme. Durch die Verwendung der Router Alert Option müssen alle Router die Header des u.U. großen Pakets untersuchen, was nicht nur die Anforderungen an die Prozessierungsgeschwindigkeit, sondern auch an den Speicherplatz erhöht. Das Einfügen und Auswerten von CGAs erhöht den Aufwand noch zusätzlich. Im Gegensatz dazu sind reine Control Plane Nachrichten kurze Datenpakete, die schnell ausgewertet werden können. Multicastempfänger müssen die Update-Pakete zunächst auf der Control Plane auswerten und die Inhalte dann weiter an die Data Plane leiten. Dies kann zu unsauberen und fehleranfälligen Implementierungen führen.

Aus diesem Vorgehen resultieren jedoch einige Vorteile. So werden die State Updates durch die enthaltenen CGAs nicht nur selbstkonsistent authentifiziert, die Binding Update Message enthält außerdem eine Sequenznummer, welche doppelte State Update-Verarbeitung verhindert. Außerdem können Paketduplikationen, Paketverlust und Pakete, die in falscher Reihenfolge eintreffen, erkannt werden.

5. Protokoll-Evaluierung

Im Folgenden soll eine Evaluierung des zuvor vorgestellten Tree Morphing Protokolls präsentiert werden. Eine Evaluierung der algortihmischen Effizienz kann in [84] gefunden werden. Die Qualität der vorgeschlagenen Realisierung kann unter Bezug auf Overheads, die durch Signalisierung, Paketverarbeitung und Implementationskomplexität auftreten, und durch ihre Robustheit gegenüber gestörten Netzwerkbedingungen oder Sicherheitsangriffen, beurteilt werden.

5.1. Protokoll-Overhead

Das Tree Morphing Protokoll ist durch das Einfügen zweier Header, den Router Alert Option und den Type 7 Routing Header, in die Binding Update Nachricht implementiert, welche mit den ersten regulären Multicast-Nutzdatenpaketen gesendet wird[1]. Daher müssen keine zusätzlichen Signalisierungspakete übertragen werden. Stattdessen werden alle nötigen Informationen in der Mobile IPv6 Binding Update Nachricht und der Home Address Option gesendet. Diese sind durch die CGA Header, ebenfalls im Mobility Header eingebettet, authentifiziert. Diese beiden Header fügen einen zusätzlichen Overhead von 256 Bit in das Paket ein.

Das Protokolldesign, welches für das Tree Morphing Protokoll eingeführt wurde, hat nur minimale Veränderungen an vorhandenen Kommunikationsprotokollen zur Folge. Es verwendet die Router Alert Option, um die State Update Message zu definieren, welche lediglich einen neuen Wert für das „Value“-Feld des Router Alerts benötigt, um den Typ der State Update Message darzustellen. Alle anderen Operationen basieren auf existierenden Protokollen wie z.B. IPv6 Source Routing oder Mobile IPv6. Dies schließt die Binding Update Message und die CGA Parameter und CGA Signature Optionen im Mobilitiy Header, wie in [5] definiert, ein. Durch die Wiederverwendung existierender Header und Protokolle können Implementationen auf leichte und zuverlässige Art realisiert werden.

5.2. Verarbeitungs-Overhead

Die State Update Pakete lösen eine Paketverarbeitung in jedem Router entlang des Pfades aus. Während der algorithmische Aufwand der SSM Sendermobilitätsverwaltung unter dem State-Management Aufwand für ASM in PIM-SM durch die Wiederverwendung von

[1] Zum jetzigen Zeitpunkt [85] kann davon ausgegangen werden, dass ein Binding Update immer ein Bestandteil der zukünftigen Multicast Quellen-Mobilitätslösungen sein wird.

States ohne weitere Signalisierungen bleibt, führt die kryptographische Verifikation der CGA Home Adressen einen Berechnungsaufwand ein. Gemäß [6] werden die hereinkommenden Daten zunächst einem Sanity Check unterzogen. Dabei wird überprüft, ob die Eingabeparameter aus der CGA Parameter Option gültige Werte enthalten. Das „Collision Count"-Feld darf beispielsweise nur die Werte 0, 1 und 2 annehmen. Pakete, die diese Überprüfung nicht bestehen, müssen sofort verworfen werden. Außerdem können Pakete mit einer ungültigen Sequenznummer ebenso direkt verworfen werden. Diese Sequenznummer ist in der Binding Update Message hinterlegt.

Während diese Überprüfungen leicht durchgeführt werden können, sind die folgenden Berechnungen deutlich rechenintensiver. Nun wird der SHA-1 Hashwert über die gesamte CGA Parameter Option generiert und die 64 höchstwertigen Bits mit dem Interface Identifier verglichen. Da diese Generierung von der CGA Parameter Option abhängt und diese neben Feldern mit fester auch den Public Key und optionale Erweiterungsfelder mit variabler Länge enthält, hängt die für die SHA-1-Generierung nötige Zeit linear von diesen Eingabeparametern ab. Gefälschte Pakete werden wiederum verworfen. Anschließend wird ein weiterer SHA-1 Hashwert – ebenfalls über die gesamte CGA Parameter Option – generiert und abhängig vom Security-Parameter *sec* die $16 * sec$ höchstwertigen Bits mit Null verglichen. Hierbei werden jedoch das „Subnet Prefix"-, sowie das „Collision Count"-Feld auf Null gesetzt, was sich allerdings nicht auf Berechnungsdauer des Hashwertes auswirkt. Sollten diese Überprüfungen erfolgreich sein, wird die CGA-Signatur des Pakets mit Hilfe des vorher überprüften Public Keys durch den RSA-Algorithmus verifiziert. Diese Berechnung ist rechenaufwändig und besitzt die komplexität $O(k^2)$, wobei k die Länge des Schlüsselmoduls ist [20].

Dennoch legt die Absicherung der Updates durch CGAs und damit die Überprüfung der RSA-Signatur einen nicht unerheblichen, zusätzlichen Berechnungsaufwand auf die Tree Morphing Router. Allerdings müssen die State Updates von jedem beteiligten Router aus den vorgenannten Gründen nur einmal pro Handover verarbeitet werden. Nimmt man eine mittlere Handoverfrequenz von einigen Updates pro Minute an, so liegt der Berechnungsaufwand immer noch deutlich unter dem Aufwand, den Protokolle wie z.B. SEND [4] auf die Router legen. SEND sichert die ARP-Requests eines Netzwerks ab, welche in größeren Netzwerken mehrfach pro Sekunde auftreten.

5.3. Robustheit

5.3.1. Robustheit gegenüber Netzwerkfehlern

In ungestörten Netzen ohne Paketverluste muss das State Update nur ein einziges Mal mit dem ersten Paket nach einem Multicast Source Handover gesendet werden, da die notwendigen Zustände in den Routern dadurch bereits etabliert werden. Um jedoch möglichem Paketverlust in unzuverlässigen oder verstopften (Funk-) Netzwerken entgegenzuwirken, muss die Übertragungsstrecke der State Update Pakete abgesichert werden. Dabei sollte nicht nur die unsichere Übertragung über (Funk-) Netzwerke, sondern auch der Weg vom nDR zum pDR abgesichert werden, um den Baumanschluss sicherzustellen. Der auf dem Weg zur Multicastverteilung letzte sinnvolle Knoten, der Empfangsbestätigungen auf eingehende State Update Pakete senden kann, ist der pDR. Es wird deshalb verlangt, dass der pDR auf die erste eingehende State Update Nachricht mit einer Bestätigung antwortet. Hierzu bietet sich die in Mobile IPv6 vorhandene Binding Update Acknowledgement Nachricht an. Wird diese Bestätigung von der mobilen Multicastquelle empfangen, ist der Anschluss an den Multicast-Verteilbaum sichergestellt. Dennoch können weitere State Updates gesendet werden, um Paketverlusten im Verteilbaum entgegenzuwirken. Somit kann sichergestellt werden, dass alle Multicastempfänger das Update erhalten.

Ein weiteres Problem, das in Netzwerken auftreten kann, sind Paketüberholungen. Dabei erhält der Empfänger der Pakete diese nicht in der Reihenfolge, in der sie vom Sender abgeschickt worden sind. Da die State Updates in die Nutzdatenpakete eingebettet werden, können sich lediglich Pakete mit State Updates überholen. Jedes dieser Pakete kann ein Update in den Routern auslösen. Daher ist die Reihenfolge der Pakete *eines* Handovers für die Aktualisierung der Router irrelevant. Auch hier kann wieder die Sequenznummer der Pakete benutzt werden, um die Updates – selbst in falscher Reihenfolge – nur einmal pro Handover zu verarbeiten. Die Update-Sequenznummer ändert sich lediglich bei jedem Handover.

In dem Beispielnetzwerk in Abbildung 5.1 können nach dem Handover der mobilen Multicastquelle Paketverluste am nDR auftreten, da im nDR nach einem State Update kein State mehr für die Paketverteilung der aus Richtung pDR eintreffenden Pakete vorhanden ist. Obwohl es Topologien gibt, in denen dieses Worst-Case Szenario auftreten kann, tritt dieser Fehler bei üblichen Handoverzeiten (Layer 2 Handover, IPv6 Adresskonfiguration und Mobile IPv6 Binding Update) nicht auf, da nach einem Handover alle „alten“ Multicastpakete, einschließlich vorheriger Updates, bereits ausgeliefert worden sind. Beim Einsatz von Beschleuningungsprotokollen, wie z.B. Fast Handovers for Mobile IPv6 [47] oder Hierarchical Mobile IPv6 Mobility Management [87], können solche Effekte aber durchaus auftreten. Um die Auslieferung „alter“ Multicastpakete zu garantieren, sollten alle mobilen Multicastquellen sicherheitshalber eine Back-Off Zeit einhalten, in der nach einem Handover keine Multicastpakete gesendet werden dürfen.

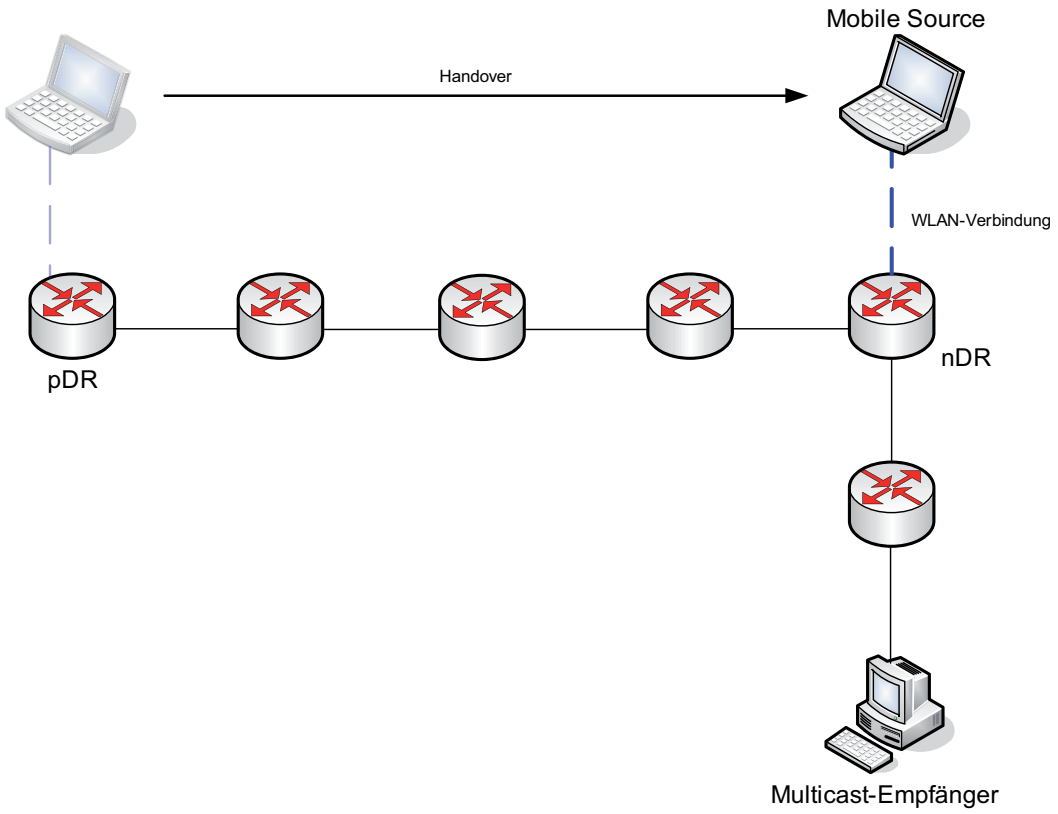


Abbildung 5.1.: Beispiel möglichen Paketverlusts am nDR

5.3.2. Robustheit gegenüber Angriffen

Das Protokoll muss diversen bekannten Attacken standhalten. Durch das erneute Senden von abgefangenen Paketen könnte ein Angreifer der Routerinfrastruktur zusätzlichen Verarbeitungsaufwand aufzwingen. Das Ziel einer sogenannten „Replay Attacke“ müsste die CGA jedes Mal überprüfen, wenn ein Paket eintrifft. Das Protokoll widersteht diesen Attacken durch die Verwendung der Sequenznummer in der Binding Update Nachricht, welche durch die Paketsignatur geschützt ist. Pakete mit ungültigen Paketnummern werden bereits beim vorher erwähnten Sanity Check herausgefiltert. Das Tree Morphing Protokoll ist deshalb im Rahmen der wohlbekannten Sicherungsmechanismen lediglich so anfällig für Attacken wie standardisierte, wohlbekannte Protokolle, wie SEND [4] und führt keine neuen Sicherheitsprobleme ein. Daher müssen neue Nachrichten von den Routern lediglich einmal kryptographisch verarbeitet werden.

Des weiteren könnte ein Angreifer seine eigene valide Home Address benutzen, um State Updates im Netzwerk auszulösen. Da solche Pakete bei Erreichen des previous Designated Routers (pDR) durch SSM Quellenfilterung verworfen werden, führt eine solche Attacke nicht dazu, dass das Netzwerk ungültige Pakete in den Multicast-Verteilbaum sendet, sondern lediglich dazu, dass sie entlang der initialen Unicast Source Route gesendet werden. Folglich führt die vorgestellte Tree Morphing Implementation im Gegensatz zu ASM nicht die Möglichkeit von netzwerkunterstützten, verteilten Denial of Service Attacken ein.

Das Ableiten von CGAs für eine Vielzahl von Interface Identifiern ist eine zeitaufwändige Aufgabe, besonders, wenn das Angriffsziel einen hohen Security Parameter *sec* verlangt (vgl. [6]). Um die Komplexität, gültige CGAs zu erzeugen, quantitativ abschätzen zu können, wurden aufeinanderfolgende, gültige CGAs unter Veränderung des Modifiers gebildet. Der Modifier ist Teil der CGA Parameter Datenstruktur (siehe Abbildung 2.5 und Kapitel 2.2.2), die als Eingabe in die CGA-Berechnungsfunktion dient. Die anderen Felder der Struktur wurden bei der CGA-Generierung nicht verändert. Tabelle 5.1 zeigt den Security Parameter *sec*, das Arithmetisches Mittel der Modifier-Schrittweite und die Standardabweichung. Die Modifier-Schrittweite bezeichnet dabei die Anzahl der Veränderungen des Modifiers, um von einer gültigen CGA zur folgenden zu gelangen.

Sec	**Arithmetisches Mittel der Modifier-Schrittweite**	**Standardabweichung**
0	1	0
1	66.113	256
2	2.591.220.608	50.901

Tabelle 5.1.: Komplexität der CGA Berechnung

Die Ergebnisse zeigen den erwarteten, exponentiellen Anstieg der Komplexität. Durch das Erhöhen des Security Parameters *sec* um 1 auf der Empfängerseite wird die Komplexität, zwei aufeinander folgende, gültige CGAs zu erzeugen um eine Größenordnung von 5 erhöht. Ein Knoten, der eine Attacke durch ungewöhnlich hohe Auslastung bemerkt, kann (temporär) verlangen, einen höheren Security Parameter *sec* zu benutzen. Dadurch können Angreifer zuvor generierte CGAs nicht weiter benutzen. Der Aufwand, CGAs zu generieren, ist wesentlich größer als diese zu überprüfen. Laut [20] ist der Aufwand, RSA-Signaturen zu generieren, $O(k^3)$, wobei k die Länge des Schlüsselmoduls ist. Das Verifizieren einer CGA hingegen kann durch das Berechnen zweier SHA-1 Hashwerte und die RSA-Verifikation mit dem Aufwand $O(k^2)$ durchgeführt werden.

Die Generierung von CGAs ist also wesentlich komplexer als deren Überprüfung - besonders wenn der Security Parameter *sec* auf einen hohen Wert gesetzt ist [6]. Daher ist es für Angreifer schwer, viele CGA-signierte Nachrichten für unterschiedliche HoAs zu senden, um dadurch viele States in den Routern auf der Unicast Source Route zwischen nDR und pDR zu etablieren und damit die Ressourcen (Multicast-Routingtabelle und damit Hauptspeicher) des Routers zu verschwenden.

6. Formale Verifikation des Tree Morphing Protokolls

In diesem Kapitel werden formale Untersuchungen zur Verifikation des Tree Morphings durchgeführt (siehe Abschnitt 3.1).

6.1. Konzept

Ein Tree Morphing-fähiger Router soll durch die Verifikation auf die Eigenschaften Deadlock-, Livelock-Freiheit und Liveness untersucht werden. Dazu werden Pakete in zufälliger Reihenfolge an diesen gesendet, um zu überprüfen, ob dieser auf alle Nachrichten-Kombinationen reagieren kann und weiterhin korrekt arbeitet.

Da es sich bei dem Tree Morphing Protokoll um ein asynchrones Kommunikationsprotokoll handelt, soll diese exemplarische Verifikation als Grundlage für weitere Verifikationen, z.B. der Loopfreiheit dienen. Ein Ausblick dazu findet sich am Ende des Kapitels.

Die Verifikation wird mit Hilfe des Tools Spin [37] durchgeführt. Dieses Tool erwartet sowohl die Prozess-Beschreibung als auch die Verifikationsbedingungen in der Sprache PROMELA (PROcess MEta-LAnguage). Für die Verifikation wurde Spin ausgewählt, da es sich sehr gut eignet, um den in Abschnitt 2.5.1 vorgestellten, endlichen Zustandsautomaten (FSM) zu modellieren.

6.2. Realisierung

6.2.1. Implementierung mit Hilfe von Promela

Um Pakete an einen Router senden zu können, müssen die Pakete selbst und der Kommunikationkanal definiert werden (siehe Listing 6.1).

Zunächst werden die Pakete definiert: *Join*, *Prune_nCoA*, *Prune_pCoA*, usw. Dann werden die Nachrichten für die Timersteuerung defniert: *startTimer*, *stopTimer*, usw. Darauf folgen die Kommunikationskanäle *msg* und *timerChan*. Die Pufferlänge des Kommunikationskanals *timerChan* ist Null (Zeile 4). Dabei handelt es sich um eine besondere Art der Kommunikation: die Rendezvous-Kommunikation. Der Kanal kann Nachrichten zwar weiterreichen, diese allerdings nicht puffern. Dadurch ist ein Schreibzugriff auf den Kanal

Listing 6.1: Initialisierung

```
1 mtype = {Join, Prune_nCoA, Prune_pCoA, StateUpdate_RPF_ok,
      StateUpdate_RPF_fail, DataPkt_RPF_ok, DataPkt_RPF_fail};
2 mtype = {startTimer, stopTimer, restartTimer, stop1Timer, timeoutTimer,
      wakeupTimer};
3 chan msg = [MSGBUFLEN] of {mtype};   /* RouterEvent */
4 chan timerChan = [0] of {mtype};     /* TimerEvent - Rendezvous
      Communication */
5 int timerCount = 0;
```

blockierend, wenn es keinen gleichzeitigen Lesezugriff eines anderen Prozesses gibt. Die Kommunikationsart ist dadurch synchron, was im Fall des Timer-Zugriffs erwünscht ist.

Listing 6.2 zeigt den Quelltext des Nachrichten-Generierungsprozesses. Zunächst wartet der Prozess auf ein *timeout*-Ereignis. Dieses tritt ein, wenn alle anderen Prozesse des Routers blockieren und nicht mehr weiterarbeiten können. Danach wird überprüft, ob der *msg*-Kanal leer ist. Dies muss an dieser Stelle immer der Fall sein, da der Router ansonsten (nach dem timeout) auf eine Nachricht nicht reagiert hat und somit nicht korrekt arbeitet. Ebenso darf die *timerCount*-Variable nur Werte zwischen 0 und 10 annehmen, doch dazu später mehr. Die Überprüfungen werden durch das Schlüsselwort *assert* definiert, welches bei der Verifikation ausgewertet wird und diese abbricht, sobald die Bedingung nicht erfüllt ist.

Ab Zeile 12 folgt der eigentliche, zufällige Nachrichtengenerator. Da die Auswahlmöglichkeiten der if-Anweisung keine Ausführungsbedingungen (sogenannte Guards) besitzen, kann jede der Alternativen ausgeführt werden. Dadurch wird entweder eine der möglichen Nachrichten in den *msg*-Kanal geschrieben oder der Timer über den *timerChan*-Kanal auf ein Timeout hingewiesen. Danach springt der Prozess zurück an den Anfang (Zeile 4) und wartet erneut darauf, dass kein weiterer Prozess mehr arbeitet.

Der Timer-Prozess (siehe Listing 6.3) besteht aus einer Endlosschleife, die durch das Schlüsselwort *endTimer*[1] in Zeile 3 als gültiger Endzustand definiert ist. Anstatt alle Timer einzeln zu starten und bei Ablauf zu beenden, wird analog zu der State Machine in Abbildung 2.14 (Kapitel 2.5.1) mit einem Care-of Address (CoA)-Zähler gearbeitet. Dieser hält die Anzahl der CoAs vor und damit auch die Anzahl der laufenden Timer. Dabei wird nicht zwischen den vorhergehenden CoAs (pCoAs) unterschieden, da es für die Verifikation unerheblich ist, welcher pCoA-Timer abläuft. Es ist lediglich wichtig das Ablaufen des letzten Timers zu identifizieren, da dies zu einer Zustandsänderung führt. Beim Wechsel aus dem Tree Morphing-Zustand werden alle Timer gestoppt und ein neuer wird gestartet, was dem gewünschten Verhalten entspricht. Über den Kanal *timerChan* erhält der Timer-Prozess die Anweisungen, Timer zu starten oder zu stoppen. Eine Besonderheit dabei ist, dass die Nachricht *stopTimer* alle, *stop1Timer* jedoch nur einen Timer beendet.

[1] In PROMELA definieren alle Marken, die mit dem Schlüsselwort *end* beginnen, gültige Endzustände.

Listing 6.2: Message Generator-Prozess

```
1  proctype MsgGenerator() {
2    mtype tmp;
3
4  progress_MsgGenerator:
5    do
6    :: timeout ->
7
8      assert (!msg?[tmp]);     /* State Machine hat einen State nicht
           verarbeitet */
9      assert (!(timerCount < 0));
10     assert (!(timerCount > 10));
11
12     if
13     :: msg!Join;
14         printf("Generating Join\n");
15     :: msg!Prune_nCoA;
16         printf("Generating Prune_nCoA\n");
17     :: msg!Prune_pCoA;
18         printf("Generating Prune_pCoA\n");
19     :: msg!StateUpdate_RPF_ok;
20         printf("Generating StateUpdate_RPF_ok\n");
21     :: msg!StateUpdate_RPF_fail;
22         printf("Generating StateUpdate_RPF_fail\n");
23     :: msg!DataPkt_RPF_ok;
24         printf("Generating DataPkt_PRF_ok\n");
25     :: msg!DataPkt_RPF_fail;
26         printf("Generating DataPkt_RPF_fail\n");
27     :: (timerCount > 0) -> timerChan!wakeupTimer;
28         printf("Waking up Timer\n");
29     fi;
30   od
31 }
```

Da es in PROMELA keine einfache Möglichkeit gibt, globale Zeiten zu definieren[2], werden die Timer durch den Message-Generator Prozess (Listing 6.2) beendet. Die Zeitspanne, bevor ein Timer abläuft, kann somit verschieden lang sein. Während dieser Umstand bei einer echten Implementierung absolut unerwünscht ist, kann er bei der Verifikation zur Vereinfachung hingenommen werden.

Beim Eintreffen der Nachricht *wakeupTimer* wird zunächst die *timerCount*-Variable dekrementiert und das *timeout*-Event dann ebenfalls über den *timerChan*-Kanal an den Router-Prozess weitergereicht.

Um einfach auf die Timer zugreifen zu können, wurden sogenannte inline-Funktionen definiert, von denen zwei in Listing 6.4 abgebildet sind. An der Stelle des Aufrufs wird die Funktion durch den definierten Quelltext ersetzt. Dadurch kann die eigentliche Timer-Implementierung leicht verändert werden, ohne den eigentlichen Router-Quelltext anpassen zu müssen.

[2]Die einzige Möglichkeit, globale Zeiten zu verwenden, ist, einen eigenen, globalen Timer-Tick einzuführen und alle Ereignisse über diese globale Zeit zu synchronisieren.

Listing 6.3: Timer-Prozess

```
1  proctype Timer()
2  {
3  endTimer:
4    do
5    :: timerChan?eval(startTimer);
6       timerCount++;
7    :: timerChan?eval(stopTimer);
8       timerCount = 0;
9    :: timerChan?eval(stop1Timer);
10      if
11      :: (timerCount > 0) ->
12        timerCount--;
13      fi;
14   :: timerChan?eval(wakeupTimer);
15      timerCount--;
16      timerChan!timeoutTimer;
17   od;
18 }
```

Listing 6.4: Timer-Zugriffsprozesse (Ausschnitt)

```
1  inline Timer_Start()
2  {
3    timerChan!startTimer
4  }
5
6  inline Timer_Stop()
7  {
8    timerChan!stopTimer
9  }
```

Ein Ausschnitt des eigentlichen Router-Prozesses ist in Listing 6.5 abgebildet. Dieser besteht im Wesentlichen aus den Marken *NI* (No Info), *J* (Join), *PP* (Prune-Pending) und *TM* (Tree Morphing), welche gleichzeitig den aktuellen Zustand der State Machine beschreiben. Darauf folgt jeweils ein *progress*-Label (Zeile 5). Analog zu den *end*-Labeln bezeichnet jede Marke, die mit *progress* beginnt, einen signifikanten, gewünschten Fortschritt im Prozess. Danach wird ein Folgezustand entsprechend der Nachricht im *msg*- bzw. *timerChan*-Kanal angesprungen und eventuell zusätzlich Timer gestartet bzw. angehalten. Dabei wird zusätzlich überprüft, ob die Zahl der Timer kleiner 10 ist (Zeile 35), um der in Abschnitt 3.1 beschriebenen State Space Explosion entgegenzuwirken. Durch diese Einschränkung können pro (HoA, G)-Zustand nur 10 pCoAs bestehen, was allerdings ausreichend ist, um die korrekte Funktionsweise des Protokolls nachzuweisen, da die Anzahl der CoAs bei der Verifikation nur für die Anzahl der Timer benötigt wird. Es würde für die Verifikation sogar ausreichen die korrekte Funktionsweise des Protokolls für maximal zwei Timer nachzuweisen, da alle weiteren Timer aufgrund der Implementierung durch eine Variable analog dem ersten Timer arbeiten.

Listing 6.5: Router-Prozess (Ausschnitt Tree Morphing)

```
1  proctype Router()
2  {
3
4  TM:
5  progress_TM:
6    printf("TM\n");
7  end_TM:
8    do
9    :: msg?Join ->
10     Timer_Restart();
11     goto TM
12   :: msg?Prune_nCoA ->
13     Timer_Stop();
14     Timer_Start();
15     goto PP
16   :: msg?Prune_pCoA ->
17     if
18     :: (timerCount > 1) ->
19       Timer_StopOne();
20       goto TM
21     :: else ->
22       Timer_Restart();
23       goto J
24     fi;
25   :: msg?StateUpdate_RPF_ok ->
26     if /* to prevent state space explosion, only 10 timers are allowed */
27     :: (timerCount < 10) ->
28       Timer_Start();
29       goto TM;
30     :: else ->
31       goto TM;
32     fi;
33   :: msg?StateUpdate_RPF_fail ->
34     if /* to prevent state space explosion, only 10 timers are allowed */
35     :: (timerCount < 10) ->
36       Timer_Start();
37       goto TM;
38     :: else ->
39       goto TM;
40     fi;
41   :: msg?DataPkt_RPF_ok ->
42     Timer_Restart();
43     goto J
44   :: msg?DataPkt_RPF_fail ->
45     Timer_Restart();
46     goto TM
47   :: timerChan?timeoutTimer ->
48     if
49     :: (timerCount > 0) ->
50       goto TM
51     :: else ->
52       goto NI
53     fi;
54     goto NI
55   od;
```

Schließlich müssen die Prozesse noch zu Beginn der Verifikation initialisiert werden. Dies geschieht im *init*-Prozess (siehe Listing 6.6), welcher von Spin standardmäßig ausgeführt wird.

Listing 6.6: Initialisierungs-Prozess

```
1 init {
2   run Timer();
3   run Router();
4   run MsgGenerator();
5 }
```

6.2.2. Sanity Check

Um die Plausibilität des in PROMELA geschriebenen Router-Modells nachzuweisen, wurden zwei Tests durchgeführt. Zum einen wurde überprüft, ob der Timer-Prozess ordnungsgemäß arbeitet. Dazu wurde ein einfacher Test-Prozess (siehe Listing 6.7) geschrieben, der anstelle des MsgGenerator-Prozesses in den Quelltext eingefügt und verifiziert wurde. Dadurch konnte nachgewiesen werden, dass der Timer-Prozess sowie die oben beschriebenen inline-Funktionen zum Aufruf der Timer Deadlock- und Livelock-frei sind.

Listing 6.7: Timer-Test Prozess

```
1 proctype TimerTest()
2 {
3  progress:
4   do
5   :: Timer_Start(); Timer_Restart(); Timer_Stop();
6   od
7 }
```

Zum anderen wurde der Prozessablauf mit Hilfe von XSpin schrittweise verfolgt und die Ausgaben des Router-Prozesses überprüft. Auch dieser Sanity Check war erfolgreich. Damit kann davon ausgegangen werden, dass das Modell gemäß der Spezifikation arbeitet und eventuelle Verifikationsfehler nicht auf die „Implementierung“ des Modells zurückzuführen sind.

6.2.3. Spin

Zur Verifikation des Modells mit dem Tool Spin kann entweder die Kommandozeilenversion *spin* oder die graphische Version *Xspin* (siehe Abbildung 6.1) benutzt werden. Letztere ist dabei nicht nur ein Wrapper für die Kommandozeilentools (siehe Abbildung 6.2), sondern untersützt auch die geführte Simulation und gibt ein Sequenzdiagramm der Prozesse aus, was sehr hilfreich beim Auffinden von fehlerhaften Quelltext-Stellen ist.

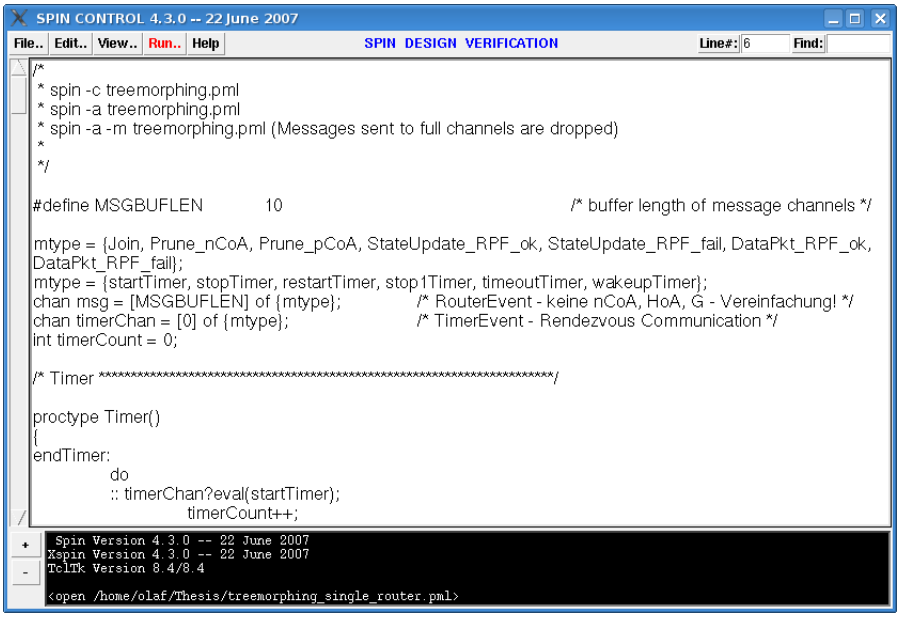


Abbildung 6.1.: Xspin-Screenshot: Main Window

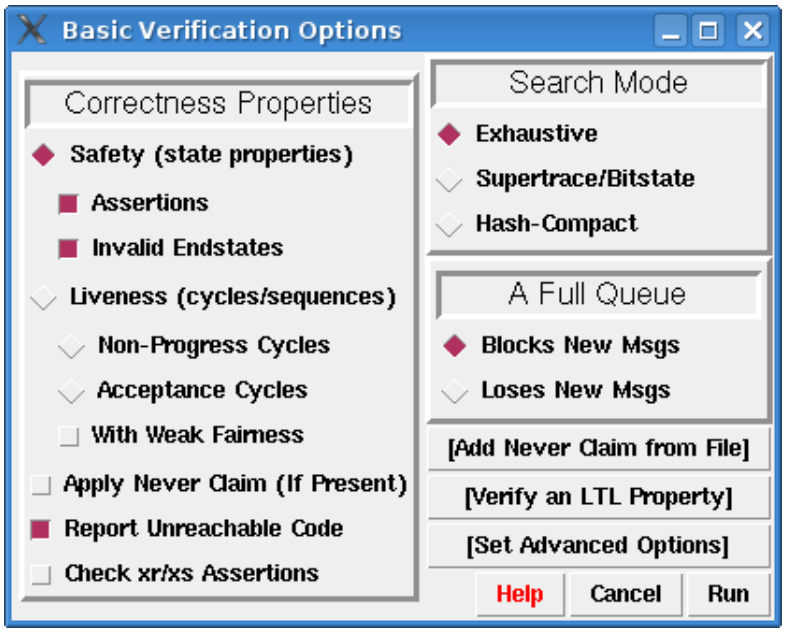


Abbildung 6.2.: Xspin-Screenshot: Verifikationseinstellungen

Der Verifikationsablauf soll nun anhand des Kommandozeilentools *spin* gezeigt werden.

1. Zunächst wird aus dem Verifikationsquelltext durch den Aufruf von spin −a treemorphing .pml ANSI C-Code erstellt, der die Verifikationsroutinen enthält.
2. Der resultierende ANSI C-Code (gespeichert in der Datei *pan.c*) kann nun durch den Aufruf eines C-Compilers[3] überstetzt werden: cc −o pan pan.c. Durch die Angabe weiterer Parameter kann nicht nur definiert werden, auf welche Eigenschaften hin das Modell verifiziert werden soll, sondern auch weitere Suchoptimierungen aktiviert werden. Um auf Safety-Eigenschaften (also Assertions und ungültige Endzustände) zu prüfen, müssen keine weiteren Parameter angegeben werden. Durch die Nutzung der Parameter cc −o pan −DNP pan.c wird das Modell hingegen mit einem Liveness-Verifier ausgestattet, welcher das Modell auf Non-Progress Cycles, also Prozesse die nicht fortschreiten, untersucht.
3. Schließlich kann die Verifikation durch den Aufruf des resultierenden, ausführbaren Programms *pan* durchgeführt werden. Bei der Überprüfung auf Liveness muss zusätzlich der Parameter ./pan −l angegeben werden.

6.3. Ergebnisse

Listing 6.8 zeigt die Ausgabe des Verifier-Aufrufs, bei dem Safety-Checks durchgeführt werden. Dabei wird auf Assertion-Verletzungen (Zeile 6), sowie auf ungültige Endzustände (solche, die nicht explizit durch *end*-Marken definiert wurden) (Zeile 8), geprüft. Da nicht auf Acceptance-Cycles geprüft werden soll, wurde der Kompilationsparameter *-DSAFETY* von *spin* automatisch gesetzt, um eine effizientere Überprüfung durchführen zu können (Zeile 7). In Zeile 10 wird ausgegeben, dass dabei keine Fehler aufgetreten sind, die Verifikation also erfolgreich durchgeführt wurde. Ab Zeile 11 folgen Informationen u.a. über die Anzahl der Zustände und Transitionen.

Durch diese Analyse wurde nachgewiesen, dass der Router-Prozess alle Eingaben verarbeitet hat und es keinen Zustand gibt, in dem dieser nicht auf Eingaben reagiert. Dies wird durch die Assertion im MsgGenerator-Prozess überprüft, bei der der Nachrichtenkanal leer sein muss. Damit wurde die Deadlock-Freiheit dieses Prozesses nachgewiesen.

Listing 6.9 zeigt die Verifier-Ausgabe entsprechend für die Liveness-Checks. Es wird dabei insbesondere auf Non-Progress Cycles (Zeile 7) geprüft. Obwohl kein Never-Claim definiert wurde, zeigt *spin* an, dass keine Fehler bei der Überprüfung aufgetreten sind (Zeile 5). Da die Events an den Router nichtdeterministisch gesendet werden, ist eine Fairness-Überprüfung der Abarbeitung nicht nötig (Zeile 6). Da hierbei ebenfalls keine Fehler auftraten (Zeile 10), wurde auch diese Verifikation erfolgreich durchgeführt.

Dabei wurde durch die Angabe von sogenannten *progress*-Marken nachgewiesen, dass alle Prozesse einen signifikanten Fortschritt durchführen. Würden diese Marken bei einem gesamten Verifikationslauf nicht betreten werden, würde der Prozess nicht korrekt arbeiten, da durch die nicht-deterministische Kombination aller möglicher Nachrichten jeder

[3]Hier wird der C-Compiler der GNU Compiler Collection benutzt.

Listing 6.8: Verifier-Ausgabe: Safety-Checks

```
1  (Spin Version 4.3.0 -- 22 June 2007)
2    + Partial Order Reduction
3
4  Full statespace search for:
5    never claim          - (not selected)
6    assertion violations  +
7    cycle checks         - (disabled by -DSAFETY)
8    invalid end states    +
9
10 State-vector 44 byte, depth reached 163, errors: 0
11       549 states, stored
12       103 states, matched
13       652 transitions (= stored+matched)
14         0 atomic steps
15 hash conflicts: 0 (resolved)
```

Zustand betreten wird. Da dies nicht der Fall ist, befindet sich der Prozess nicht in einem Livelock, verarbeitet die Nachrichten auch weiterhin und kann den aktuellen Zustand verlassen. Dies konnte bei der Safety-Verifikation nicht erkannt werden, da die Prozesse dabei lediglich auf ungültige Endzustände (Deadlocks) geprüft werden.

Listing 6.9: Verifier-Ausgabe: Liveness-Checks

```
1  (Spin Version 4.3.0 -- 22 June 2007)
2    + Partial Order Reduction
3
4  Full statespace search for:
5    never claim          +
6    assertion violations  + (if within scope of claim)
7    non-progress cycles   + (fairness disabled)
8    invalid end states    - (disabled by never claim)
9
10 State-vector 48 byte, depth reached 313, errors: 0
11      1764 states, stored (2547 visited)
12      1943 states, matched
13      4490 transitions (= visited+matched)
14         0 atomic steps
15 hash conflicts: 5 (resolved)
```

6.4. Ausblick

Da es sich beim Tree Morphing um ein asynchrones Protokoll handelt, ist die Verifikation eines einzelnen Routers exemplarisch. Dennoch kann die durchgeführte Verifikation als Basis für weitere Untersuchungen dienen, bei denen ein ganzes Netzwerk von Routern auf protokollspezifische Eigenschaften hin untersucht werden kann. Beispielsweise könnte dabei die Loopfreiheit des Protokolls nachgewiesen werden, was bedeutet, dass ein Paket nicht mehrfach über dasselbe Downstream-Interface eines Routers gesendet wird. Dazu könnten die Router-/Downstream-Interface Informationen im Paket gespeichert werden und durch einen *never*-Claim auf doppelte Einträge hin untersucht werden. Ebenso können dabei Paketverluste und -überholungen nachgewiesen werden. Um die Korrektheit des Protokolls zu überprüfen, können außerdem gültige Wege für ein Paket vorgegeben und diese dann verifiziert werden.

Da diese Untersuchung allerdings eine in PROMELA aufwändige Verifikations-„Implementierung“ eines ganzen Routernetzwerkes voraussetzt, soll sie im Rahmen dieser Arbeit nicht weiter untersucht werden. Die Loopfreiheit und weitere Eigenschaften werden stattdessen in den nächsten Kapiteln durch eine Implementierung und den Test dieser experimentell nachgewiesen.

7. Protokollimplementierung

Dieses Kapitel stellt die Protokollimplementierung des Tree Morphing Protokolls in OMNeT++ dar.

7.1. Konzept

Zunächst sollen die konzeptionellen Grundlagen des verwendeten Simulators und des Modells aufgezeigt werden. Darauf folgt eine Übersicht, an welchen Stellen des Modells das Tree Morphing Protokoll eingefügt wird. Die modifizierte PIM-SSM State Table und die benötigten Pakete und Router werden anschließend vorgestellt. Zudem wird der prinzipielle Weg der Multicastpakete durch den Simulator vorgestellt.

7.1.1. OMNeT++

OMNeT++ ist ohne Erweiterungen „lediglich“ ein einfacher, diskreter Ereignissimulator ohne vorzugeben, welche speziellen Modelle ausgeführt werden können. Daher wird er – außer für Netzwerksimulationen – für die verschiedensten Simulationen, wie z.B. Geschäftsprozesse oder Hardwarearchitekturen, eingesetzt. Interessant wird er für die Netzwerksimulation erst durch die verschiedenen, verfügbaren Modelle (Frameworks). So gibt es nicht nur Modelle für TCP/IP Netzwerke, sondern auch solche für Peer-to-peer und Ad-hoc, Funk- und Sensor-Netzwerke [62].

Die für diese Arbeit grundlegenden Modelle sind das sogenannte INET- und das IPv6-SuiteWithINET-Framework. Das INET-Framework [61] enthält neben einem vollständigen IPv4- und TCP/UDP-Stack u.a. Unterstützung für die Protokolle 802.11, Ethernet und PPP. Das IPv6SuiteWithINET-Framework [58] basiert auf dem INET-Framework und erweitert es u.a. um die Protokolle IPv6 inklusive Mobile IPv6, Hierarchical Mobile IPv6 und ICMPv6. Dieses Framework soll genutzt werden, um das Tree Morphing Protokoll zu implementieren, da es bis auf MLD und PIM alle nötigen Netzwerkprotokolle unterstützt, welche jedoch nachträglich leicht eingefügt werden können.

7.1.2. Klassenmodell der IPv6Suite

Simulationen in OMNeT++ bestehen aus zwei Teilen: Einer Simulationsbeschreibung in der Sprache NED (NEtwork Description language) und der Funktionsimplementierung in C++. NED definiert dabei das Simulationsnetzwerk, die Zusammensetzung der einzelnen

Module zu größeren Einheiten, z.B. zu Routern, und die Verbindungen dieser Module untereinander. Die einfachsten Module (vom Typ SimpleModule) werden in C++ implementiert und kommunizieren mit anderen Modulen nur über sogenannte „Gates“. Diese werden in der NED-Definition mit den Gates anderer Module verbunden. Zusätzlich gibt es Module, die keine Implementierung besitzen, da sie lediglich andere Module kapseln, wie z.B. die Netzwerkschicht eines Routers.

Abbildung 7.1 zeigt ein einfaches Simulationsnetzwerk mit drei Routern, *ha*, *router1* und *router2*, zwei Access Points und einem Empfänger *rcv1*. Außerdem existiert ein *configurator6*-, ein *worldProcessor*- und ein *tmLogger*-Modul. Sehr anschaulich sind dabei die Verbindungen dieser Module untereinander, symbolisiert durch die Pfeile. Die Kreisausschnitte stellen die Funkradien der Access Points bzw. den Empfangsradius des *ms1* dar. Die nicht verbundenen Module stellen globale Funktionen zur Verfügung und können, da sie keine Gates besitzen, keine normalen Simulationsnachrichten empfangen.

Das Router-Modul *ha* besteht aus den in Abbildung 7.2 dargestellten Modulen. Die wichtigsten Module sind dabei das *networkLayer*-, das *interfaceTable*- und die *linkLayer*-Module.

Das *networkLayer*-Modul besteht wiederum aus den zwei in Abbildung 7.3 gezeigten Modulen. Das Relevante davon ist das *proc*-Modul, welches in Abbildung 7.4 dargestellt ist. Dieses enthält die wesentlichen Teile der IPv6-Paketverarbeitung:

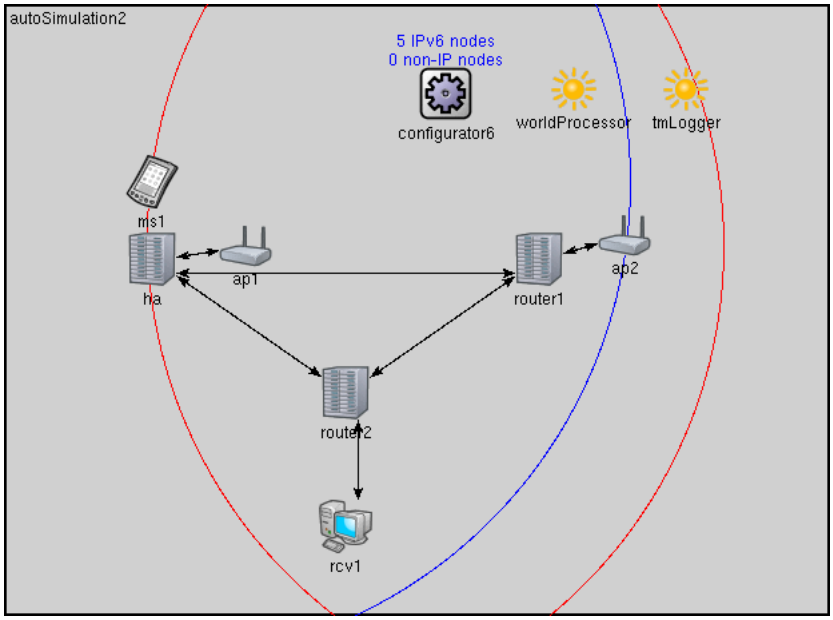


Abbildung 7.1.: Ein Simulationsnetzwerk in OMNeT++

- **preRouting** Pakete, die den Router erreichen, werden vom *linkLayer*- an das *networkLayer*-Modul weitergereicht. Dort passieren diese zunächst das *inputQueue*-Modul und treffen dann im *preRouting*-Modul ein. Dieses Modul verteilt die Pakete entsprechend ihres Inhalts entweder an das *ICMP*- oder das *forwarding*-Modul.
- **forwarding** Pakete werden im *forwarding*-Modul an benachbarte Module weitergeleitet. Ziele dafür sind das *multicast*-, das *tunneling*-, das *fragmentation*- und das

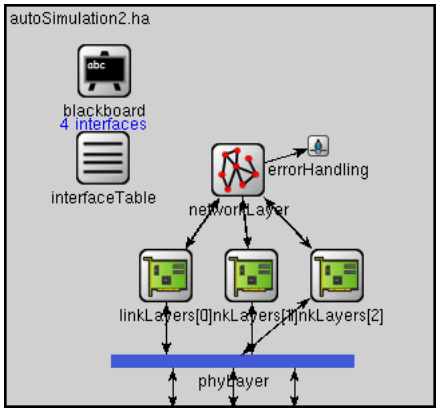


Abbildung 7.2.: Detailansicht des Routers „HA“

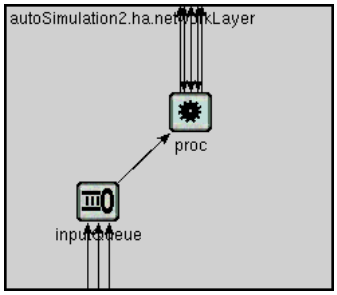


Abbildung 7.3.: Detailansicht des „networkLayer“-Moduls

addrResln-Modul. Ist die Zieladresse beispielsweise eine IPv6-Multicastadresse, wird das Paket ohne weitere Verarbeitung an das *multicast*-Modul weitergeleitet.

- **addrResln** Ist die Link-Layer Adresse zu einer bestimmten Ziel IP-Adresse nicht bekannt, wird das Address Resolution-Modul bemüht diese Adresse aufzulösen. Dazu besitzt dieses eine Verbindung zum *ICMP*-Modul, um die sogenannte Neighbour Discovery Prozedur durchzuführen. Die eigentlichen Datenpakete werden dabei in diesem Modul zwischengespeichert.
- **fragmentation** Das *fragmentation*-Modul dient als gemeinsamer Anlaufpunkt, um Pakete zu versenden. Diese werden dazu – sofern nötig – fragmentiert und diese Teile an das korrekte *output*-Modul weitergereicht. Das einzige Modul, das die Fragmentierung umgeht ist *ICMP*, dessen ausgehende Nachrichten gemäß [19] nicht fragmentiert werden müssen.
- **ICMP** Dieses Modul generiert und verarbeitet ICMP-Pakete.
- **send** Dieses Modul wird dazu genutzt, Pakete mit der Quell-IP Adresse des Hosts[1] zu senden. Pakete treffen von diversen anderen Modulen ein, z.B. den lokalen Transportschichten oder dem *tunneling*-Modul. Diese werden dann an das *forwarding*-Modul weitergereicht, um z.B. den korrekten „Next Hop“ zu finden.

[1] „Host“ bezeichnet in diesem Zusammenhang einen Knoten, der Pakete gemäß des IPv6-RFCs verarbeitet. Dazu zählen natürlich auch Router.

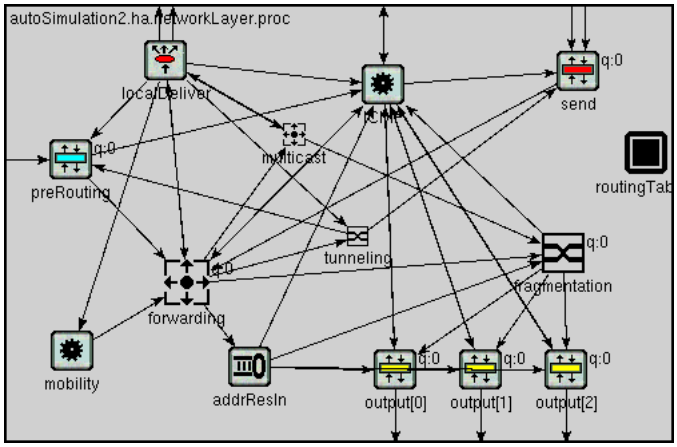


Abbildung 7.4.: Detailansicht des „proc“-Moduls

- **localDeliver** Pakete, die für den lokalen Host bestimmt sind, werden vom *localDeliver*-Modul verarbeitet. Anhand des „nextHeader“-Wertes wird bestimmt, an welches Modul das Paket weitergeleitet werden soll. Ziele sind u.a. die lokalen Transportschichten oder das *multicast*-Modul.
- **mobility** Dieses Modul ist dafür verantwortlich, die Mobile IPv6-Verwaltung durchzuführen. Es reagiert nicht nur auf Layer 2-Trigger[2], sondern sendet und empfängt auch die für Binding Updates nötigen Nachrichten und verwaltet den Binding Cache.
- **multicast** Das *multicast*-Modul ist schließlich für die MLD-, PIM- und Tree Morphing-Protokollbehandlung verantwortlich. Es verwaltet die in Abbildung 2.14 vorgestellte State Machine und verteilt Multicastpakete entsprechend der Weiterleitungszustände.

Die Module können mit Hilfe der graphischen Oberfläche von OMNeT++ weiter untersucht werden. Abbildung 7.5 zeigt beispielhaft die Gates des *multicast*-Moduls und deren Verbindungen.

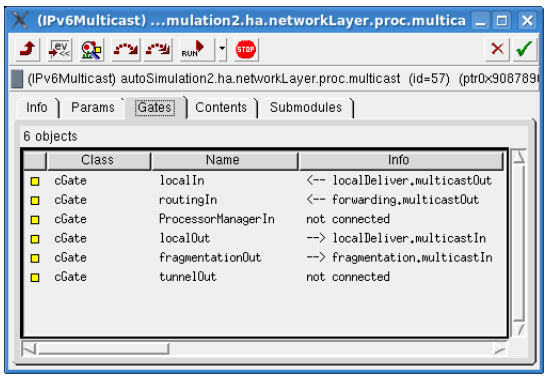


Abbildung 7.5.: Detailansicht der Gates des „multicast“-Moduls

[2]Layer 2-Trigger informieren den Netzwerkstack (Layer 3) über ein erfolgreiches Handover in ein neues Netz.

Mit Ausnahme des *ICMP*-Moduls, welches weiter unterteilt ist, sind alle zuvor vorgestellten Module sogenannte „SimpleModules“, deren Funktionen in jeweils einer C++-Klasse implementiert sind. Tabelle 7.1 listet die für diese Arbeit relevanten Module und deren Quelldateien innerhalb des IPv6SuiteWithINET-Verzeichnisses auf[3].

SimpleModule	**C++-Quelltextdatei**
preRouting	Network/IPv6/IPv6PreRouting.cc
forwarding	Network/IPv6/IPv6Forward.cc
addrResln	Network/ICMPv6/AddressResolution.cc
send	Network/IPv6/IPv6Send.cc
localDeliver	Network/IPv6/IPv6LocalDeliver.cc
mobility	Network/IPv6/IPv6Mobility.cc
multicast	Network/IPv6/IPv6Multicast.cc

Tabelle 7.1.: Ausgewählte Module des NetworkLayers und die Position ihrer Quelldateien

Die Module verarbeiten die eintreffenden Nachrichten entweder mit Hilfe einer Warteschlange (*Base/QueueBase.cc*), welche mehrere Nachrichten zwischenspeichern kann, oder durch die Methode *endService()*, die die Nachrichten direkt verarbeiten und eventuell weiterleiten muss, da sie keinen Nachrichtenpuffer bietet.

7.1.3. Eingliederung des Tree Morphing Protokolls in die IPv6Suite

Die zentrale Instanz der Multicast-Paketverarbeitung ist das Multicast-Modul, welches für die Behandlung der einfachen Multicast-, der Tree Elongation- und Binding-Acknowledgement Pakete verantwortlich ist. Außerdem übernimmt dieses Modul die Aufgabe der PIM- und MLD-Verarbeitung und hält die für die Paketweiterleitung nötigen State-Tabellen vor. Da das in der IPv6SuiteWithINET vorhandene Multicast-Modul weder MLD noch PIM unterstützt, müssen diese Protokolle neu implementiert werden. Aus Zeitgründen ist es nicht möglich, diese Protokolle in allen Einzelheiten zu implementieren[4]. Daher werden nur die wichtigsten Funktionen, wie das Senden von PIM- und MLD- Join- und Prune-Nachrichten und die Verwaltung der Multicast Forwarding State Table, vorgesehen. Weder die „Multicast Listener Query Message“, die zur Abfrage der lokalen Multicast-Empfänger genutzt wird, noch die PIM „Hello“-Nachrichten werden unterstützt. Außerdem wird nur der Bestandteil des PIM-Standards für Source Specific Multicast implementiert. Außer diesen Teilen müssen zusätzlich diverse Module des Netzwerk-Layers angepasst werden, um die Weiterleitung der Multicast- und Tree Morphing-Pakete an das Multicast-Modul durchzuführen. Einzelheiten dazu finden sich in Abschnitt 7.2.3.

[3]Da die Adressauflösung ICMP-Pakete nutzt, ist diese Datei als einzige im ICMPv6-Verzeichnis gespeichert.

[4]Da MLD und PIM-SSM nicht Teil der eigentlichen Untersuchung und in gängigen Betriebssystemen vorhanden sind, werden lediglich deren benötigte Grundfunktionen implementiert. Eine „echte“ Implementierung kann diese dann durch die vollständigen, vorhandenen Codeteile ersetzen.

7.1.4. Multicast Forwarding State Table

Multicast-fähige Router müssen Informationen über Weiterleitungszustände vorhalten, damit sie eingehende Pakete gezielt über dem Verteilbaum zugehörige Interfaces weiterleiten können. Ohne diese Informationen würde die Verteilung der Pakete der Broadcastübertragung entsprechen. [27] definiert die für SSM-Übertragungen benötigten Verteilinformationen. Wie in Kapitel 2.5 gesehen, benötigt das Tree Morphing Protokoll zusätzlich zu dem Tupel (S,G) Informationen über die Care-of Adresse des Senders[5]. Die Datenstrukturen aus [27] müssen daher angepasst werden.

Abbildung 7.6 zeigt das vereinfachte UML-Klassendiagramm der PIM Forwarding State Table. Die Klasse *TMStateTable* enthält dabei den C++ STL (Standard Template Library) Container *vector*, welcher eine einfache Sequenz von *TMStateTableEntry*-Klassen enthält. Jede dieser Klassen repräsentiert ein (HoA, G)-Tupel. Zusätzlich wird der *upstreamState* des Tupels gespeichert, welcher die Werte *NotJoined* oder *Joined* annehmen kann. Außerdem kann in jeder Klasse ein Verweis pro OMNeT++-Interface auf ein *TMInterfaceEntry* vorgehalten werden[6]. Dazu wird der STL Container *map* genutzt, welcher zu jedem einzigartigen Schlüssel einen Eintrag speichern kann. Der Schlüssel entspricht dabei der Nummerierung der Interfaces eines Routers. Die Klasse *TMInterfaceEntry* enthält wiederum einen *vector* mit Klassen des Typs *TMInterfaceEntryPerCoA*, da jedes Interface Informationen zu mehrere CoAs vorhalten kann. Diese Klassen speichern schließlich die *pimState*-Informationen des Interfaces, welche die Werte *NoInfo*, *Join* oder *PrunePending* annehmen kann.

Der State Table aus [27] wird also zusätzlich eine CoA-Zwischenschicht pro (HoA, G)-State hinzugefügt. Diese ist nötig, um die lokale Mitgliedschaft und den PIM-Status für jede CoA einzeln speichern zu können und um mehrere CoAs pro (HoA, G)-State zu unterstützen.

7.1.5. Zum Tree Morphing benötigte Pakete

Neben den regulären Multicast-Paketen, welche über UDP versendet werden, benötigt die Implementierung des Tree Morphing Protokolls einige weitere Pakete, welche in Tabelle 7.2 einschließlich ihrer Quell- und Ziel-IP Adressen aufgeführt sind. Die ersten drei genannten Pakete sind reguläre UDP Pakete, die sich durch die Quell- und Ziel-IP Adressen und die vorhandenen Header unterscheiden. Das reguläre Multicast-Paket wird von der CoA des Senders an die SSM-Adresse gesendet, wobei die Home Address Option die HoA des Senders übermittelt. Das nach einem Handover versandte Tree Elongation-Paket enthält ebenfalls die CoA des Senders als Quell-IP Adresse, allerdings ist das Ziel dabei der pDR. Dadurch werden die Tree Elongation-Pakete per Unicast übertragen. Zusätzlich zur Home Address Option werden gemäß Kapitel 4.2.2.3 die Router Alert Option, der Routing Header und die Binding Update Message in das Paket aufgenommen. Sobald das Tree Elongation Paket den pDR erreicht hat, wird der Routing Header entfernt und die dort enthaltene Multicast-Gruppenadresse als Ziel-IP Adresse in das Paket eingesetzt.

[5] Die Home Address wird dabei in S gepeichert.
[6] Die gespeicherte Information bezieht sich auf die Verteilrichtung des Interfaces (Downstream).

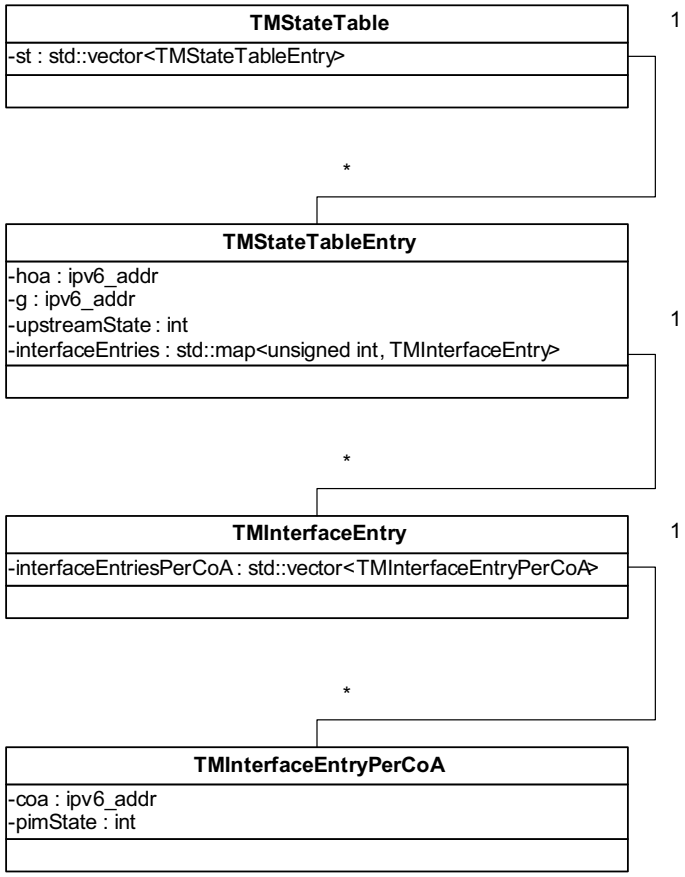


Abbildung 7.6.: Vereinfachtes UML-Klassendiagramm der PIM State Table

Alle anderen Header bleiben bei der weiteren Übertragung bestehen. Zusätzlich sendet der pDR eine Tree Elongation Binding Acknowledgement-Nachricht per Unicast an die CoA des MN zurück, um den Erhalt des Pakets zu quittieren.

Zusätzlich zu den eigentlichen Multicast- sind weitere Kontrollnachrichten nötig, um den Verteilbaum aufzubauen. Dazu kommt im Subnetz des Empfängers das MLD-Protokoll zum Einsatz. Der Empfänger meldet seinen Empfangswunsch bei dem für das Subnetz zuständigen Router über eine MLD-Join Nachricht an, welche von der Link-lokalen Adresse des Empfängers gemäß [100] an die Multicast-Adresse *ff02::16* gesendet wird. Alle Router, die das MLD-Protokoll unterstützen, müssen auf Anfragen an diese Adresse hören. Möchte ein Empfänger keine weiteren Multicastdaten empfangen, versendet er eine MLD-Prune Nachricht an dieselbe Multicast-Adresse.

Um den Verteilbaum zwischen den Routern aufzubauen, kommt das PIM-Protokoll zum Einsatz. Dazu versenden Router PIM-Join und PIM-Prune Nachrichten an den jeweils nächsten Router in Richtung der Multicastquelle. Als Quell- und Ziel-IP Adresse kommen dabei die Link-lokalen Adressen der beteiligten Router zum Einsatz, es findet also eine Unicast-Übertragung statt. Der letzte Router, an den die Multicastquelle direkt angeschlossen ist, sendet keine weiteren PIM-Nachrichten, sondern verteilt die Multicastpakete entsprechend der Anfragen.

Paketname	**Quell-IP Adresse**	**Ziel-IP Adresse**	**Anmerkungen**
Multicast (regulär)	CoA	Multicast-Gruppenadresse	HoA Option
Tree Elongation	CoA	pDR	HoA, RtAlert, RtHdr und BU Option
Multicast (nach Tree Elongation)	CoA	Multicast-Gruppenadresse	HoA, RtAlert und BU Option
Tree Elongation Binding Acknowledgement	pDR	CoA	
MLD-Join/Prune	Link-lokale Adresse des Senders	ff02::16 [100]	
PIM-Join/Prune	Link-lokale Adresse des Senders	Link-lokale Adresse des nächsten Routers	

Tabelle 7.2.: Beim Tree Morphing benutzte Multicastpakete

7.1.6. Prinzipieller Weg der Pakete durch den Netzwerklayer der IPv6Suite

Zur Durchführung der Implementierung ist es nötig zu wissen, welchen Weg die Pakete beim Durchlaufen des Netzwerk-Layers prinzipiell nehmen. Dadurch können die korrekten Module bei der Implementierung um die Weiterleitungsfunktionen erweitert werden. Im Folgenden werden die durchlaufenen Module innerhalb eines einzelnen Routers je Paketart aufgeschlüsselt:

- Multicast (regulär) / (nach Tree Elongation)
  - @MN: send → forwarding → multicast → fragmentation
  - @Router: preRouting → forwarding → multicast
  - @Receiver: preRouting → forwarding → localDeliver → UDP
- Tree Elongation
  - @MN: send → forwarding → multicast → fragmentation
  - @Router: preRouting → forwarding → fragmentation
  - @pDR: preRouting → forwarding → localDeliver → multicast
- Tree Elongation Binding Acknowledgement
  - @Router: preRouting → forwarding → fragmentation
  - @MN: preRouting → forwarding → localDeliver → multicast

- MLD-Join/Prune
  - @Receiver (Router): preRouting → forwarding → multicast
- PIM-Join/Prune
  - @Receiver (Router): preRouting → forwarding → localDeliver → multicast

Wie zu erkennen ist, müssen zum Multicast gehörende, eingehende Pakete zunächst stets das *preRouting*-Modul des Netzwerk-Layers passieren. Von dort werden diese immer an das *forwarding*-Modul weitergereicht[7]. Der weitere Weg des Pakets hängt von seinem Inhalt ab und wird in Abschnitt 7.2.3 im Detail erklärt.

7.2. Realisierung

Zur Durchführung der Implementierung standen zwei verschiedene Versionen der IPv6-SuiteWithINET [58] zur Verfügung: Die neuere, als unstabil und ungetestet veröffentlichte Version vom 09. August 2006 und die ältere, stabile vom 19. Januar 2006. Da bereits gute Erfahrungen mit der stabilen Version in der Projektgruppe an der HAW Hamburg gemacht wurden und um Abstürze des Simulators durch ungetesteten Code auszuschließen, wurde die stabile Version genutzt.

Ebenso standen zwei OMNeT++-Versionen zur Auswahl: Die neuere Version 3.4b2 vom 19. November 2006 und die ältere Version 3.3 vom 26. Oktober 2006. Da die Kompilierung der gewählten IPv6SuiteWithINET-Version gegen die neuere OMNeT++-Version fehlschlägt, wurde die ältere genutzt.

7.2.1. Paketimplementierung

Um Daten durch den Simulator zu transportieren, können entweder vorhandene Nachrichtentypen oder eigene Nachrichten verwendet werden. Zur Implementierung des Tree Morphing Protokolls wird zum einen ein einfacher UDP-Nachrichtentyp benötigt, der einige Informationen über den Verlauf des Pakets speichert, zum anderen ein Nachrichtentyp, der die für die MLD- und PIM-Protokolle benötigten Daten überträgt.

OMNeT++ bietet eine einfache Möglichkeit, eigene Pakete zu definieren. Dazu müssen die zu speichernden Felder lediglich in einer C-ähnlichen Notation in einer Datei mit der Dateiendung *.msg* definiert werden. Das Perl-Skript[8] *opp_msgc* überführt diese einfache Nachrichtendefinition in eine C++-Quelltextdatei. Dabei werden die Variablen als private Attribute implementiert. Zugriffsfunktionen, sogenannte Getter/Setter-Methoden, ermöglichen dabei den Zugriff auf diese Attribute. Das Skript unterstützt darüber hinaus auch

[7] Obwohl in Abbildung 7.4 eine Verbindung vom *preRouting*- zum *ICMP*-Modul existiert, wird diese in der eingesetzten IPv6SuiteWithINET dennoch nie genutzt.

[8] Laut der OMNeT++-Dokumentation ist dies nur eine temporäre Lösung, bis das auf C++ basierende *nedtool* fertiggestellt ist.

Array-Definitionen, wobei diese automatisch ihre Größe ändern, sobald ein neues Element gespeichert[9] bzw. gelöscht wird. Die erzeugte Nachrichten-Klasse ist dabei von der Basisklasse *cMessage* abgeleitet, welche als kleinste Einheit zur Nachrichtenübertragung genutzt wird. Damit kann die neue Nachricht andere im Quelltext verwandte ersetzen, ohne die Übertragung selbst zu ändern, da alle Nachrichten von der selben Basisklasse abstammen.

Der Quelltext in Listing 7.1 definiert die Nachricht *McastPayloadDataCollector*, welche von der Multicast-Quelle gesendet wird. Dazu wird diese an den UDP-Stack übergeben und von dort weiter an den NetworkLayer weitergereicht. Die in der Nachricht enthaltenen Felder werden für die Simulationsauswertung benötigt. Details dazu finden sich in Kapitel 8.3.

Listing 7.1: Paketdefinition McastPayloadDataCollector

```
1 message McastPayloadDataCollector
2 {
3   fields:
4     unsigned int sequenceNr;
5     unsigned int routerInterfaceRecord[];
6     simtime_t addrResStartTime[];
7     simtime_t addrResEndTime[];
8 };
```

Listing 7.2 zeigt den in der Quelltextdatei *Network/IPv6/ocmld.msg* definierten Nachrichtentyp *OCMLD*, der die für die MLD- und PIM-Übertragung nötigen Informationen überträgt. Dies sind die SSM-Quelladresse S und die gewünschte Gruppenadresse G, beide vom Typ *ipv6_addr*. Da es sich dabei nicht um einen einfachen C-Typ handelt, muss dem Konvertierungsskript mitgeteilt werden, in welchem Header die Typdefinition gefunden werden kann (Zeile 2) und dass es sich dabei nicht um eine Klasse handelt (Zeile 5).

Listing 7.2: Paketdefinition OCMLD

```
1 cplusplus {{
2 #include "ipv6_addr.h"
3 }}
4
5 class noncobject ipv6_addr;
6
7 message OCMLD
8 {
9   fields:
10    ipv6_addr S;
11    ipv6_addr G;
12 }
```

[9] Die Implementierung erzeugt dazu ein neues, größeres Array, kopiert die Daten und löscht das alte Array.

7.2.2. Headerimplementierung

Neben den neuen Paketen müssen zusätzlich die Router Alert Option und der Routing Header Typ 7 implementiert werden, da diese nicht in der IPv6SuiteWithINET vorhanden sind.

Listing 7.3 zeigt die in der Datei *Network/IPv6/HopByHopOptMessage.h* implementierte Router Alert Option. Wie in Zeile 1 zu sehen ist, handelt es dabei um eine IPv6 Type Length Value (TLV)-Option. Zunächst wird der Typ *IPv6TLVOptionBase::ROUTER_ALERT_OPT*, danach die Länge 2 und schließlich der frei definierbare Wert gespeichert. Durch kleine Anpassungen (Hinzufügen einer Möglichkeit zu einem *switch*-Konstrukt) in *Network/IPv6/HdrExtProc.cc* wird diese Option im Zuge der bereits vorhandenen Hop-by-Hop Headerverarbeitung ausgewertet.

Listing 7.3: Definition der Router Alert Option

```
1  struct router_alert_opt: public ipv6_tlv_option
2  {
3    router_alert_opt(unsigned int value_in)
4      :ipv6_tlv_option((unsigned char)IPv6TLVOptionBase::ROUTER_ALERT_OPT,
5                       (unsigned int) 2),value(value_in){};
6    unsigned int value;
7  };
8
9  class TLVOptRouterAlert:public IPv6TLVOptionBase
10 {
11 public:
12
13   TLVOptRouterAlert(unsigned int value_in)
14     :IPv6TLVOptionBase(IPv6TLVOptionBase::ROUTER_ALERT_OPT, (unsigned int
       ) 2), value(value_in)
15     {}
16
17   virtual TLVOptRouterAlert* dup() const
18     {
19       return new TLVOptRouterAlert(value);
20     }
21
22   const unsigned int getValue(void) { return value; }
23
24   virtual bool processOption(cSimpleModule* mod, IPv6Datagram* dgram);
25 private:
26   unsigned int value;
27 };
```

Die Unterstützung des Routing Headers Typ 7 konnte ebenfalls einfach implementiert werden, da eine Verarbeitung der Routing Header inklusive Typ 1 bereits durch die Mobile IPv6-Unterstützung vorhanden war. Daher musste lediglich ein neuer Header definiert werden. Listing 7.4 zeigt einen Ausschnitt des Quelltextes, der in die Datei *Network/IPv6/Ipv6Headers.h* eingefügt wurde. Hier finden sich die in Kapitel 4.2.2.2 definierten Felder wieder. Im Gegensatz zu anderen Routing Headern kann dieser lediglich

eine IP-Adresse aufnehmen: die Gruppenadresse an die der pDR das Paket beim Tree Elongation weiterleiten soll.

Die Paketverarbeitung selbst konnte unverändert weiterverwendet werden, da die Routing-Header von derselben Basisklasse *ipv6_ext_rt_hdr* abgeleitet werden und somit leicht per Typecast in diese überführt werden können. Da der Stack zur Zeit nicht prüft um welchen Routing-Header Typ es sich handelt, kann der hier definierte standardkonform verarbeitet werden.

Listing 7.4: Definition des Routing Headers Typ 7

```
1  struct ipv6_ext_rt7_hdr: public ipv6_ext_rt_hdr
2  {
3    ipv6_ext_rt7_hdr(unsigned char next_header = 0,
4                     unsigned char hdr_ext_len = 0,
5                     unsigned char routing_type = IPv6_TYPE7_RT_HDR,
6                     unsigned char segments_left = 1, ipv6_addr* haddr = 0)
7      :ipv6_ext_rt_hdr(next_header, hdr_ext_len, routing_type,
           segments_left),
8       reserved(0), addr(haddr)
9      {}
10   unsigned int reserved;
11   ipv6_addr* addr;
12 };
```

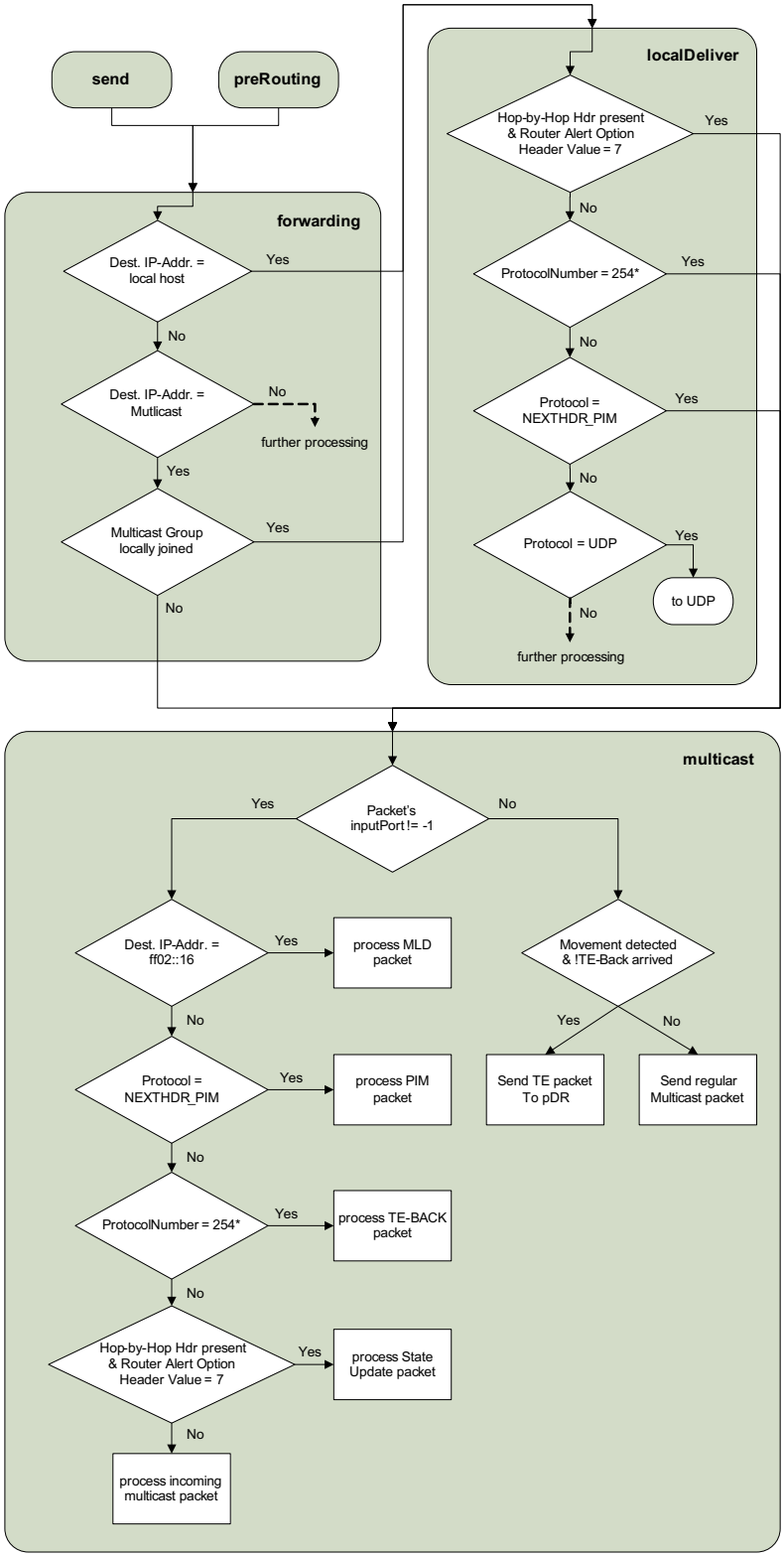
Die zwei weiteren für das Tree Morphing benötigten Header waren bereits in der IPv6-SuiteWithINET vorhanden: Die Home Address und die Binding Update Message Option.

7.2.3. Anpassung der Module des NetworkLayers

Sowohl eingehende, als auch ausgehende Pakete müssen den Netzwerk-Layer, wie in Abschnitt 7.1.6 beschrieben, passieren. Abbildung 7.7 zeigt das Flußdiagramm der Paketverarbeitung im NetworkLayer der IPv6SuiteWithINET.

Pakete, die vom lokalen Host gesendet werden, treffen im *send*-Modul im NetzwerkLayer ein. Von außen eintreffende Pakete gelangen über das *preRouting*-Modul in den Network-Layer. Dort werden sie an das *forwarding*-Modul weitergeleitet, welches die Zieladresse des Pakets zunächst mit einer der Adressen des lokalen Hosts vergleicht und das Paket bei einer Übereinstimmung an das *localDeliver*-Modul weiterreicht. Ist dies nicht der Fall, wird überprüft, ob die Zieladresse eine Multicastadresse ist. Schlägt diese Überprüfung fehl, folgt die weitere Paketverarbeitung, auf die hier nicht weiter eingegangen wird. Bei einer positiven Überprüfung wird das Paket an das *multicast*-Modul gesendet - es sei denn, der lokale Host ist Empfänger der Multicastübertragung, wobei es an das *localdeliver*-Modul übergeben wird.

Im *localDeliver*-Modul eintreffende Pakete werden zunächst auf eventuell vorhandene Header untersucht. Ist der Hop-by-Hop Header vorhanden, in diesem die Router Alert Option gespeichert und der Wert des *Value*-Feldes gleich 7, wird das Paket an das *multicast*-Modul



* Die gegenwärtige Implementierung verwendet die experimentelle Protokollnummer 254, um Tree Elongation Binding Acknowledgement Pakete zu erkennen. In einer echten Implementierung würde dafür eine ICMP-Nachricht eingesetzt werden. Da in der IPv6Suite jedoch keine Verbindung zwischen ICMP- und Multicast-Modul besteht, wird anhand der Protokollnummer unterschieden.

Abbildung 7.7.: Flussdiagramm der Paketverarbeitung im NetworkLayer

weitergereicht, da es sich um eine State Update Nachricht handelt. Ansonsten wird überprüft, ob die Protokollnummer des nächsten Headers 254 entspricht. Durch diese Überprüfung wird festgestellt, ob es sich um ein Tree Elongation Binding Acknowledgement Paket handelt. Eine reale Implementierung sollte dafür eine ICMP-Nachricht verwenden. Da der ICMP-Layer in der gewählten IPv6SuiteWithINET jedoch keine Verbindung zum *multicast*-Modul besitzt, wird das Tree Elongation Binding Acknowledgement Paket aktuell über die Protokollnummer erkannt und an das *multicast*-Modul weitergereicht. Diese Weiterleitung wird auch durchgeführt, wenn der nächste Header vom Typ *NEXTHDR_PIM* ist, wodurch PIM-Pakete, die laut [27] an die link-lokale Adresse des Routers gesendet werden, ebenfalls erkannt werden. Im Rahmen der regulären Paketverarbeitung wird ein (Multicast) UDP-Paket an das UDP-Modul weitergereicht. Ansonsten wird mit der weiteren Verarbeitung fortgefahren.

Das *multicast*-Modul unterscheidet eintreffende Pakete zunächst nach ihrem Ursprung. Pakete, die von anderen Modulen des selben Hosts eintreffen, werden anhand des fehlenden *inputPort*-Wertes (= -1) erkannt. Diese werden als reguläre Multicastpakete versandt, solange kein Handover stattgefunden hat oder nach einem Handover das erste Tree Elongation Binding Acknowledgement Paket eingetroffen ist. Unmittelbar nach einem Handover werden die Multicastpakete mit den in Abschnitt 4.2.2.3 beschriebenen Headern versehen und als Tree Elongation Pakete an den pDR gesendet.

Pakete mit einem *inputPort*-Wert größer als *-1* sind über ein gültiges Interface von einem anderen Host eingetroffen. Zunächst wird deren Ziel IP-Adresse auf eine Übereinstimmung mit dem Wert *ff02::16* überprüft, anhand dessen eine MLD-Anfrage erkannt werden kann. Daraufhin wird die State Table des Routers angepasst und eventuell ein PIM-Join gesendet. Eintreffende PIM-Pakete können, wie bereits im *localDeliver*-Modul gesehen, anhand des Wertes des nächsten Header-Felds identifiziert werden. Ist dort der Wert *NEXTHDR_PIM* hinterlegt, wird die State Table des Routers aktualisiert und eventuell weitere PIM-Pakete gesendet. Eintreffende Tree Elongation Binding Acknowledgement Pakete werden erneut durch die experimentelle Protokollnummer 254 zugeordnet. Schließlich werden State Update Pakete durch das Vorhandensein des Hop-by-Hop Option Headers und der dort hinterlegten Router Alert Option mit dem *Value*-Feld 7 unterschieden, was ebenfalls zu Änderungen an der State Tabelle führt und eventuell das Versenden von PIM-Nachrichten verursacht. Sollten alle dieser Überprüfungen fehlschlagen, handelt es sich um ein reguläres Multicastpaket, das nicht für den lokalen Rechner bestimmt ist. Daher wird es gemäß der in der State Tabelle gespeicherten Daten dupliziert und an weitere Router gesendet.

7.2.4. State Injection Algorithm

Unmittelbar nach einem Handover sendet der MN, wie in [84] beschrieben, State Update Nachrichten, um alle Knoten auf dem Multicast-Verteilbaum über die Adressänderung zu informieren. Nachdem das Tree Elongation Paket am pDR verarbeitet wurde, sendet dieser das Update weiter in den „alten“ Mutlicastverteilbaum. Jeder Router implementiert daraufhin den neuen (nCoA, HoA, G)-Verteilzustand. Erreicht das Paket den Router auf dem topologisch korrekten Interface, werden die „alten“ Zustände gelöscht, was als „State

Override“ bezeichnet wird. Ansonsten wird lediglich der neue Zustand hinzugefügt („State Injection“).

Listing 7.5 (aus der Datei *Network/IPv6/IPv6Multicast.cc*) zeigt den Quelltext des State Injection Algorithmus. In Zeile 2 wird zunächst eine Liste mit allen (HoA, G)-States, die den Werten des eingetroffenen Pakets entsprechen, erstellt. Diese Liste wird in Zeile 10 durchlaufen. Jeder der Einträge enthält eine weitere Liste mit CoAs, welche ebenfalls durchlaufen wird. Für jeden dieser CoA-Einträge wird überprüft, ob das Paket auf dem für die nCoA topologisch korrekten Interface eingetroffen ist (RPF-Check) (Zeile 12). Ist dies der Fall, wird das „State Override“-Ereignis zunächst aufgezeichnet (Zeile 13). Anschließend werden alle States des aktuell untersuchten Interfaces gelöscht (Zeilen 16-20) und der neue State eingefügt (Zeilen 21-24).

Schlägt der zuvor erwähnte RPF-Check fehl, wird das „State Injection“-Ereignis aufgezeichnet (Zeile 26) und zunächst nach einem vorhandenen State gesucht, um doppelte States in der Liste zu vermeiden (Zeilen 29-35), welche ansonsten auftreten könnten, da State Update-Nachrichten nach einem Handover mehrfach gesendet werden. Schließlich wird die Variable *doTreeOptimization* gesetzt (Zeile 44), um die Tree Optimization durch das Senden einer PIM-Join Nachricht in Richtung der neuen Multicastquelle (Zeile 50) durchzuführen. Durch das Setzen der Variable und das spätere Auslesen wird lediglich eine PIM-Join Nachricht initiiert.

Listing 7.5: Tree Morphing: State Injection Algorithm

```
1  vector<TMStateTableEntry *> v;
2  v = tmStateTable->getAllStates(hoa, g);
3  vector<TMStateTableEntry *>::iterator it;
4  map<unsigned int, TMInterfaceEntry>::iterator it2;
5  map<unsigned int, TMInterfaceEntry> interfaceList;
6  vector<TMInterfaceEntryPerCoA>::iterator it3;
7
8  TMInterfaceEntryPerCoA *tmInterfaceEntryPerCoA;
9
10 for (it = v.begin(); it != v.end(); it++) {
11   for (it2 = (*it)->interfaceEntries.begin(); it2 != (*it)->
       interfaceEntries.end(); it2++) {
12     if (rpfCheck(ncoa, datagram->inputPort())) {
13       tmStateTable->logStateOverride();
14       //replace all (., HoA, G) with (nCoa, HoA, G)
15       //instead of replacing, remove all states and insert new one
16 state_injection_again:
17       for (it3 = it2->second.interfaceEntriesPerCoA.begin(); it3 != it2->
           second.interfaceEntriesPerCoA.end(); it3++) {
18         it2->second.interfaceEntriesPerCoA.erase(it3);
19         goto state_injection_again;
20       }
21       tmInterfaceEntryPerCoA = new TMInterfaceEntryPerCoA();
22       tmInterfaceEntryPerCoA->coa = ncoa;
23       tmInterfaceEntryPerCoA->pimState = STATE_JOIN;
24       it2->second.interfaceEntriesPerCoA.push_back(*
           tmInterfaceEntryPerCoA);
25     } else {
26       tmStateTable->logStateInjection();
27
```

```
28        // only create new state, if no identical is available
29        bool ncoaStateFound = false;
30        for (it3 = it2->second.interfaceEntriesPerCoA.begin(); it3 != it2->
              second.interfaceEntriesPerCoA.end(); it3++) {
31          if (it3->coa == ncoa) {
32            it3->pimState = STATE_JOIN;
33            ncoaStateFound = true;
34          }
35        }
36        if (!ncoaStateFound) {
37          tmInterfaceEntryPerCoA = new TMInterfaceEntryPerCoA();
38          tmInterfaceEntryPerCoA->coa = ncoa;
39          tmInterfaceEntryPerCoA->pimState = STATE_JOIN;
40          it2->second.interfaceEntriesPerCoA.push_back(*
                tmInterfaceEntryPerCoA);
41        }
42
43        /* INIT TREE_OPTIMIZATION */
44        doTreeOptimization = true;
45      }
46    }
47  }
48
49  if (doTreeOptimization) {
50    sendPIMJoin(ncoa, hoa, g);
51  }
```

7.2.5. Extended Forwarding Algorithm

Der Quelltext des Extended Forwarding Algorithm gemäß [84] ist in Listing 7.6 abgebildet. Nachdem die Variablen und Iteratoren initialisiert und alle zu der HoA und G passenden States aus dem aktuell eingetroffenen Multicastpaket geholt wurden (Zeile 2), wird eine Schleife über diese *TMStateTableEntry*-Einträge ausgeführt. Darin befindet sich, wie in Abbildung 7.6 aufgeführt, eine Liste der Interfaces, über die ebenfalls iteriert wird (Zeile 9). Es wird analog zum State Injection Algorithm überprüft, ob das Paket auf dem für die nCoA topologisch korrekten Interface eingetroffen ist (Zeile 11). Ist dies der Fall, wird die Liste der pCoAs daraufhin untersucht, ob die zugehörige *pimState*-Variable den Wert *STATE_JOIN*, *STATE_PRUNEPENDING* oder *STATE_INCLUDE* enthält, woraufhin das Paket auf dem aktuellen Interface gesendet wird. Durch das Verwenden der *pktSentOnIf*-Variable (Zeilen 10, 19 und 26) wird sichergestellt, dass das Paket nur einmal pro Interface durch die Funktion *dupAndSendPacket()* gesendet wird. Ab Zeile 26 folgt die im State Injection Algorithm vorgestellte Implementierung, alle States außer des neuen (nCoA, HoA, G)-States zu entfernen. Allerdings muss dabei zusätzlich das „Prunen“ nicht weiter benötigter Zweige durchgeführt werden (Zeile 32). Zudem wird der Zeitpunkt gespeichert, an dem die State Tabelle für das gerade betrachtete Interface optimal ist.

Schlug der „erste“ RPF-Check über die nCoA fehl, wird die Abarbeitung des Programms in Zeile 49 fortgeführt. Dort wird für jede zum aktuellen Interface gehörende CoA, welche

ungleich der nCoA ist (Zeile 51), der RPF-Check durchgeführt (Zeile 52). Ist dieser erfolgreich, wird das Paket über das aktuelle Interface (einmal) versandt (Zeile 56). Schlägt der RPF-Check fehl, wird das Paket verworfen.

Dieser Extended Forwarding Algorithm wird auch dazu genutzt, Pakete weiterzuleiten obwohl die State Table optimal ist. Das führt zu minimal höherem Rechenaufwand, welcher sich in einer Simulation allerdings nicht auf das Timing auswirkt, da die Prozessierungszeit der Module nicht berücksichtigt wird.

Listing 7.6: Tree Morphing: Extended Forwarding Algorithm

```
1  vector<TMStateTableEntry *> v;
2  v = tmStateTable->getAllStates(hoa, g);
3  vector<TMStateTableEntry *>::iterator it;
4  map<unsigned int, TMInterfaceEntry>::iterator it2;
5  vector<TMInterfaceEntryPerCoA>::iterator it3;
6  bool pktSentOnIf;
7
8  for (it = v.begin(); it != v.end(); it++) {
9    for (it2 = (*it)->interfaceEntries.begin(); it2 != (*it)->
       interfaceEntries.end(); it2++) {
10     pktSentOnIf = false;
11     if (rpfCheck(ncoa, datagram->inputPort()) == true) {
12       mcastInOnCorrectIf = true;
13       mcastInOnCorrectIfForCoA = ncoa;
14
15       /* FORWARD_PACKET_ON_INTERFACE */
16       for (it3 = it2->second.interfaceEntriesPerCoA.begin(); it3 != it2->
           second.interfaceEntriesPerCoA.end(); it3++) {
17         if ((it3->pimState == STATE_JOIN) || (it3->pimState ==
             STATE_PRUNEPENDING)) {
18           // send out pkt only once per interface
19           if (!pktSentOnIf) {
20             pktSentOnIf = true;
21             dupAndSendPacket(datagram, it2->first);
22             datagram->setInputPort(inputPort);
23           }
24         }
25       }
26       if (pktSentOnIf) {
27         /* REMOVE (*, HoA, G) - states except (nCoA, HoA, G) */
28         bool ncoaStateFound = false;
29 IPv6Multicast_EFA_again:
30         for (it3 = it2->second.interfaceEntriesPerCoA.begin(); it3 != it2
             ->second.interfaceEntriesPerCoA.end(); it3++) {
31           if (it3->coa != ncoa) {
32             sendPIMPrune(it3->coa,hoa,g);
33             tmStateTable->logLocallyOptimal();
34             it2->second.interfaceEntriesPerCoA.erase(it3);
35             goto IPv6Multicast_EFA_again;
36           } else {
37             ncoaStateFound = true;
38           }
39         }
40         // if state has not been found, create new (nCoA, HoA, G)-state
41         if (!ncoaStateFound) {
```

```
42            TMInterfaceEntryPerCoA *tmInterfaceEntryPerCoA;
43            tmInterfaceEntryPerCoA = new TMInterfaceEntryPerCoA();
44            tmInterfaceEntryPerCoA->coa = ncoa;
45            tmInterfaceEntryPerCoA->pimState = STATE_JOIN;
46            it2->second.interfaceEntriesPerCoA.push_back(*
                  tmInterfaceEntryPerCoA);
47          }
48        }
49      } else {
50        for (it3 = it2->second.interfaceEntriesPerCoA.begin(); it3 != it2->
              second.interfaceEntriesPerCoA.end(); it3++) {
51          if (it3->coa != ncoa) { // only do this for pcoa states
52            if (rpfCheck(it3->coa, datagram->inputPort()) == true) {
53              // only send out pkt once per interface
54              if (!pktSentOnIf) {
55                pktSentOnIf = true;
56                dupAndSendPacket(datagram, it2->first);
57                datagram->setInputPort(inputPort);
58              }
59            }
60          }
61        }
62      }
63    }
64  }
```

8. Simulation des Tree Morphing Protokolls

Dieses Kapitel stellt die Simulation des Tree Morphing Protokolls in OMNeT++ dar. Die gewonnenen Ergebnisse werden analysiert und diskutiert.

8.1. Anforderungen

Die durchzuführenden Simulationen sollen zu möglichst aufschlußreichen, realitätsnahen und repräsentativen Ergebnissen führen. OMNeT++ bietet dazu die Möglichkeit, den kompilierten Simulator mit Topologiedaten zu parametrieren. Dadurch sollen einerseits in konstruierten Testnetzwerken der topologische Grundzustandsraum ausgetestet, andererseits reale Internettopologien zur Analyse realistischer Szenarien herangezogen werden. Die gewünschten Messgrößen wurden in Kapitel 3.2 eingeführt und werden hier noch einmal kurz aufgelistet:

- Delay Stretch
- Konvergenzzeit
- Paketduplikationen
- Paketüberholungen
- Paketverluste
- Routing-Loops

Die Simulation der Minimalnetze soll dabei die Schwächen des Protokolls aufzeigen, die Simulation auf realen Topologien Leistungsdaten des Alltagseinsatzes liefern. Dabei sollte stets beachtet werden, dass die Simulation bestimmte Eigenschaften eines echten Routers, wie z.B. das Prozessierungsdelay, die Speicherauslastung oder Verhalten unter Last, nicht abbilden kann.

8.2. Protokollanalyse

Die folgende Protokollanalyse soll anhand des durch die graphische Oberfläche visualisierten Simulationsverhaltens zeigen, dass das Tree Morphing Protokoll sich gemäß der Spezifikation verhält. Dies ist nötig, um Interpretationsfehler der Simulationergebnisse aufgrund einer fehlerhaften Implementierung auszuschließen. Dazu gehört die Überprüfung, ob ein Multicast-Verteilbaum korrekt erstellt und nach einem Handover erfolgreich „gemorphed“ wird. Tabelle 8.1 zeigt die Entwicklung der Router States für das in Abbildung 8.1 dargestellte Testnetzwerk.

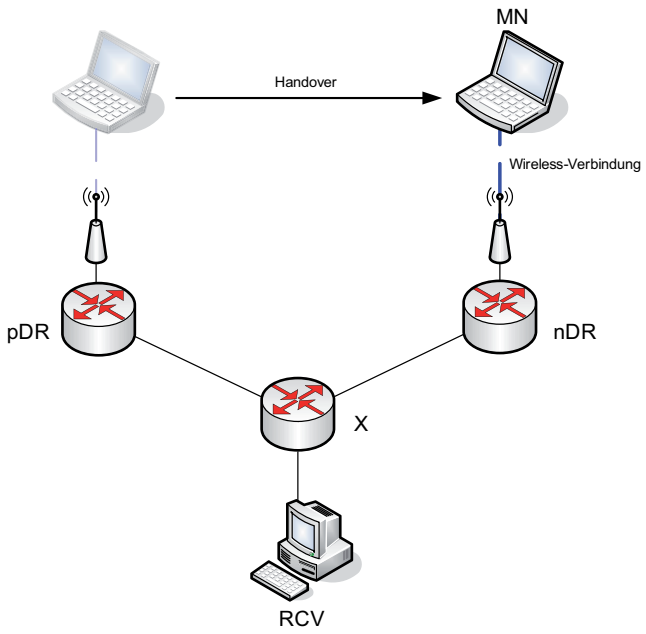


Abbildung 8.1.: Testnetz zur Protokollanalyse

Dieses Testnetz besteht aus drei Tree Morphing-Routern, pDR, nDR und X, welcher die Verbindung zu den Multicastempfängern (hier: ein Empfänger) herstellt. Der MN führt ein Handover vom Funknetz, welches an den pDR angeschlossen ist, zu dem am nDR anliegenden durch. Der als RCV bezeichnete Host ist der Empfänger der Multicastübertragung.

Das Netz wird jeweils zu den Zeitpunkten betrachtet, an denen ein Paket übertragen und verarbeitet wird, daher ist die Zeit in Tabelle 8.1 in Hops angegeben. Zum Zeitpunkt 0 existieren keine Weiterleitungszustände und der Empfänger (RCV) sendet den Empfangswunsch (MLD-Join) in das Netzwerk. Router X empfängt diesen zum Zeitpunkt 1 und etabliert einen Verteilzustand auf dem korrekten Interface. Dabei wird die pCoA des MN genutzt, die die aktuelle CoA vor dem Handover darstellt. Da der Upstream-Status des Routers X bisher nicht „joined“ ist, sendet dieser ein PIM-Join Paket an den pDR. Erreicht dieses den pDR (Zeit 2), etabliert auch dieser einen Weiterleitungzustand, allerdings nicht an den RCV, sondern standardkonform an den nächsten Hop, also X.

Zu diesem Zeitpunkt findet auch das Handover des MN statt. Nach erfolgter Layer 2- und Layer 3-Konfiguration, sendet dieser die Multicastdaten per Unicast als Tree Elongation-

Zeit (Hops)	Router-Zustände			Nächste Aktion
	pDR	X	nDR	
0	-	-	-	RCV sendet MLD-Join
1	-	pCoA → RCV	-	X sendet PIM-Join → pDR
2	pCoA → X	pCoA → RCV	-	MN Handover zu nDR, MN sendet TE-Paket → nDR
3	pCoA → X	pCoA → RCV	nCoA → X	TE-Paket wird weitergeleitet → X
4	pCoA → X	pCoA → RCV nCoA → pDR	nCoA → X	TE-Paket wird weitergeleitet → pDR
5	nCoA → X	pCoA → RCV nCoA → pDR	nCoA → X	pDR sendet TE-Back → MN, pDR sendet Mcast-Paket per regulärer Übertragung → X
6	nCoA → X	pCoA → RCV nCoA → RCV nCoA → pDR	nCoA → X	pDR sendet PIM-Join → nDR
7	nCoA → X	pCoA → RCV nCoA → RCV nCoA → pDR	nCoA → X	MN empfängt TE-Back und sendet Mcast-Pakete regulär → X
8	nCoA → X	nCoA → RCV nCoA → pDR	nCoA → X	X empfängt Mcast-Paket und sendet dieses → pDR und RCV
9	nCoA → X	nCoA → RCV nCoA → pDR	nCoA → X	pDR empfängt Mcast-Paket und sendet dieses → X
10	nCoA → X	nCoA → RCV nCoA → pDR	nCoA → X	X sendet PIM-Prune → pDR
11	-	nCoA → RCV nCoA → pDR	nCoA → X	pDR sendet PIM-Prune → X
12	-	nCoA → RCV	nCoA → X	

Tabelle 8.1.: Zuständsentwicklung der Router bei einem Handover für das in Abbildung 8.1 dargestellte Netzwerk

Pakete (TE) an den nDR. Durch die im TE-Paket enthaltene Router Alert Option und das State Update richtet der nDR einen Forwarding State zu X (Zeit 3), der Router X entsprechend einen zum pDR ein (Zeit 4). Dabei wird die nCoA des MN genutzt, da diese die nun aktuelle IP-Adresse darstellt. Erreicht das erste TE-Paket den pDR (Zeit 5), überschreibt dieser zunächst den pCoA-Zustand mit der nCoA, da dieses auf dem topologisch korrekten Interface eingetroffen ist und der vorherige Zustand ebenfalls auf diesem Interface vorhanden war. Anschließend sendet dieser das Multicastpaket (inklusive State Update) über den neuen Zustand an X. Zudem wird ein Tree Elongation Binding Acknowledgement Paket (TE-Back) per Unicast an den MN gesendet. Das am Router X zur Zeit 6 eintreffende Multicastpaket trifft nicht auf dem für die nCoA topologisch korrekten Interface ein. Daher wird der Verteilzweig lediglich hinzugefügt (nCoA → RCV) (State Injection) und das Paket aufgrund des Extended Forwarding Algorithm an den RCV weitergeleitet, da der RPF-Check für die zur HoA des Pakets gehörende pCoA erfolgreich ausfällt. Zusätzlich leitet der Router X die Tree Optimization Phase ein. Dazu sendet er ein PIM-Join in die „Unicast-Richtung“ der nCoA, um einen direkten Verteilzweig aufzubauen.

Zum Zeitpunkt 7 erreicht das TE-Paket den MN, welcher daraufhin keine weiteren TE-Pakete, sondern reguläre Multicastpakete sendet. Weiterhin erhält auch der Empfänger das Multicastpaket inklusive der State Update Nachricht (was in der Tabelle nicht weiter aufgeführt ist). Dieser findet eine Zuordnung auf Grund der HoA und aktualisiert seinen Binding Cache. Erreicht den Router X ein reguläres Multicastpaket (Zeit 8), entfernt dieser alle alten Verteilzustände pro Interface und etabliert lediglich den neuen. Daraufhin dupliziert dieser das Paket und sendet es an den Empfänger und den pDR, welcher dieses zum Zeitpunkt 9 auf dem etablierten State wieder zurück an X sendet. Dieser erkennt anhand des RPC-Checks, dass der Verteilzweig nicht länger benötigt wird (Zeit 10) und sendet ein PIM-Prune an den pDR. Da dieser keine weiteren Empfänger besitzt, sendet er ebenfalls ein PIM-Prune an den X (Zeit 11). Der Router X aktualisiert daraufhin seine Verteilzustände (Zeit 12) und die Multicastübertragung kann entlang eines neuen optimalen Verteilbaums stattfinden.

Dieser Ablauf beinhaltet alle Operationen des Tree Morphing Protokolls. Er wurde mit den durch Debug-Ausgaben der Router in der graphischen Simulationsumgebung von OMNeT++ erhaltenen Werten durch eine Schrittweise Simulation verglichen. Da keine Abweichungen auftraten, kann die Implementierung in den Grundfunktionen fehlerfrei angesehen werden.

8.3. Realisierung der Monitoring- und Analyse-Funktionen

Nachdem die Korrektheit der Implementierung angenommen werden kann, soll die Simulation zur Datenerhebung genutzt werden. Um Daten aus dem Simulator an andere Programme weiterzugeben, stellt OMNeT++ einige Klassen zur Verfügung. Zum einen besteht die Möglichkeit, einzelne Werte über die Funktion *recordScalar()* in die Datei *omnetpp.sca* zu schreiben. Die IPv6Suite speichert dort auch viele andere Werte z.B. Statistiken über Paketübertragungen. Daher muss diese Datei nach einem Simulationslauf nach den gewünschten Werten durchsucht werden. Weiterhin bietet Omnet mit der Funktion *recordWithTimestamp()* der Klasse *cOutVector* die Möglichkeit, Werte mit einem bestimmten Zeitstempel zu loggen. Diese Werte werden in der Datei *omnetpp.vec* gespeichert, welche mehrere Vectoren enthält und durch das Programm *splitvec* wieder in die einzelnen Vectoren getrennt werden kann.

Da Pakete grundsätzlich identifiziert werden müssen, wird jedes Multicastpaket vom MN mit einer fortlaufenden Sequenznummer und der Sende-Zeit versehen. Diese Daten werden zusätzlich an eine dafür entwickelte, globale Logging-Klasse *tmLogger* weitergegeben, welche eine Liste der gesendeten Pakete vorhält. Trifft ein Paket bei dem Empfänger[1] ein, so wird dieses Ereignis mit dem Empfangszeitpunkt an die *tmLogger*-Klasse gesendet. Anhand der Sequenznummer kann eine eindeutige Zuordnung der Pakete stattfinden.

Um auszuschließen, dass Paketverluste, die aus WLAN-Handovers auftreten, in die Auswertung einfließen, wird der Zeitpunkt der abgeschlossenen Layer 3-Konfiguration auf dem WLAN-Interface als Startzeitpunkt für sämtliche Auswertungen herangezogen.

Die Methoden zur Erhebung der gewünschten Daten sollen im Folgenden vorgestellt werden:

Konvergenzzeit Die Konvergenzzeit bezeichnet die Zeit, die die Routerinfrastruktur benötigt, um optimale Weiterleitungszustände in den Routern einzurichten. Dazu wird der Zeitpunkt, an dem der MN das erste Tree Elongation Paket sendet, als Startzeitpunkt in der globalen Klasse gespeichert. Die am Tree Morphing beteiligten Router informieren die globale *tmLogger*-Klasse, sobald eines der folgenden Ereignisse auftritt:

- **pDRReached** Erreicht den pDR das Tree Elongation-Paket, wird das erste dieser Ereignisse als Endzeitpunkt gespeichert.
- **State Injection** Findet in einem der Router eine State Injection statt, wird dies gespeichert. Jedes dieser Ereignisse wird ausgewertet, da dieses in vielen Routern stattfinden kann und jedes dieser die Konvergenzzeit beeinflusst.
- **StateJoined** Wird ein PIM-Join empfangen, signalisiert ein Router dieses. Ein solches Ereignis tritt nach einem Handover auf, wenn ein neuer Verteilzweig entsteht.

[1] Um die Analyse möglichst einfach durchführen zu können, wird lediglich ein Empfänger genutzt. Die hier vorgestellten Verfahren können allerdings auch auf mehrere Empfänger erweitert werden.

Im Gegensatz dazu wird das PIM-Prune nicht geloggt, da die minimale Konvergenzzeit gesucht wird, nach der Multicastpakete auf optimalen Wegen zu den Routern verteilt werden können.

- **TE-BackReceived** Die Multicastverteilung kann nur auf optimalen Wegen durchgeführt werden, wenn die Quelle keine Tree Elongation Pakete mehr sendet. Dies wird durch den Empfang des Tree Elongation Binding Acknowledgement Pakets festgestellt.

Von allen festgestellten Zeiten wird die größte als Endzeit genutzt. Beim Aufzeichnen aller Endzeitpunkte muss zusätzlich beachtet werden, dass die Paketauslieferung durch Adressauflösungen unerwünscht verzögert werden kann[2]. Um diesem Umstand entgegenzuwirken, enthält jedes Multicastpaket zusätzlich eine Liste, die die Verzögerungen durch Adressauflösungen enthält. Dazu wird die Zeit gemessen, die ein Paket im *addrResln*-Modul während der Adressauflösung verweilt. Soll ein Ereignis als Endzeitpunkt genutzt werden, wird die Liste durchsucht und die Summe aller Verzögerungszeiten von der aktuellen Simulationszeit substrahiert, um einen unverfälschten Wert zu erhalten.

Die Konvergenzzeit ergibt sich dann aus der Formel 8.1.

$$t_{conv} = t_{convergenceEnd} - t_{convergenceStart} \tag{8.1}$$

Delay Stretch Der Delay Stretch ist der Faktor, um den sich die Paketauslieferung während des Tree Morphings maximal erhöht. Dieser ist in Formel 8.2 angegeben.

$$\text{Delay Stretch} = \frac{\text{Maximum Delay during Tree Morphing}}{\text{Mean Delay after Tree Morphing}} \tag{8.2}$$

Das maximale Delay während des Tree Morphings wird durch die Klasse *tmLogger* bestimmt. Dazu wird die Liste der Pakete nach Simulationsende nach der maximalen Auslieferungszeit eines Pakets während der Baumreorganisation des Protokolls durchsucht. Dabei kann die Übertragungszeit durch Substraktion der *sendTime* von der *arrivalTime* errechnet werden. Jedoch muss auch hierbei die Zeit für die Adressauflösung substrahiert werden.

Weiterhin ist für die Berechnung die mittlere Auslieferungszeit nach dem Tree Morphing nötig, die durch Betrachten der Paketliste nach der Konvergenzendzeit gefunden werden kann. Dazu wird die Summe der Übertragungszeiten gebildet und durch die Anzahl der betrachteten Pakete geteilt.

Paketduplikationen Paketduplikationen können durch das Suchen von doppelten Sequenznummern im *pktLog* aufgedeckt werden. Diese werden mit der Sequenznummer, der Sende- und der Endzeit ausgegeben.

[2]Obwohl Layer 2-Adressauflösung für Multicastpakete nicht nötig ist, da für eine Multicastadresse kein einzelner Host zuständig ist, führt OMNeT++ diese durch. Dabei handelt es sich eindeutig um einen Fehler des Simulators.

Paketüberholungen Paketüberholungen können ebenfalls durch Durchsuchen der Paketliste gefunden werden. Dabei wird die Liste nach aufsteigender Sequenznummer sortiert[3]. Ist die Empfangszeit eines Pakets kleiner als die des Vorgängers, fand eine Paketüberholung statt. Die Anzahl der Paketüberholungen wird als absolute Zahl geloggt.

Paketverluste Paketverluste werden für jedes Paket durch das Fehlen einer Empfangszeit erkannt[4] und mit der Sequenznummer des Pakets ausgegeben.

Routing-Loops Um Routing-Loops zu erkennen, wird beim Senden eines Multicastpakets ein Hashwert aus der global eindeutigen Modul-ID des Routers und der Interface-ID, auf dem das Paket den Router verlässt, durch den Quelltext 8.1 gebildet. Dabei wird die Modul ID um 10 Bit nach links „geshiftet“ und die Interface-ID in den unteren 10 Bits gespeichert. Ein Router darf dabei bis zu 1023 Interfaces besitzen, was auch für sehr große Netze ausreichend ist.

Listing 8.1: Berechnung des RouterInterfaceID-Wertes

```
1 unsigned int routerInterfaceId = parentModule()->parentModule()->id() <<
      10;
2 routerInterfaceId |= (info->ifIndex() & 0x2ff );
```

Jeder Router prüft für ein zu sendendes Multicastpaket, ob diese ID bereits im Paket vorhanden ist. Sollte dies der Fall sein, wurde ein Routing-Loop erkannt und geloggt. Ansonsten wird die neue ID im Paket gespeichert. Da ein Paket einen Router mehrfach passieren kann, z.B. beim Tree Elongation und bei der anschließenden regulären Multicastverteilung, wird nicht nur die Router ID sondern auch die Interface ID gespeichert. Die Überprüfung auf Routing-Loops findet in jedem Router statt, da bei Paketverlust keine Informationen gesammelt werden könnten.

Zusätzlich zu den vorgenannten Monitoring-Funktionen kann für einen Empfänger ausgegeben werden, wie hoch die Übertragungszeit einzelner Pakete ist und welche das Ziel nicht erreicht haben. Diese werden in Abhängigkeit der Sequenznummer geloggt, wobei lediglich die Zeitdauer des Tree Morphings und die ersten Sekunden danach ausgegeben werden.

[3]Die Paketliste wird nicht nach der Empfangszeit sortiert, um Paketüberholungen von -verlusten unterscheiden zu können.

[4]Um Paketverluste durch ein abruptes Ende der Simulation auszuschließen, beendet der Sender die Übertragung von Paketen 5 Sekunden vor Simulationsende, wodurch alle Pakete ausgeliefert werden können.

8.3.1. Probleme der IPv6Suite

Die grundlegenden Simulationsmechanismen beruhen auf den Implementierungen der IPv6-Suite. Deshalb erben die Rechnungen Beschränkungen und inhaltliche Ausgestaltungsfehler dieser Implementierung. Die größte Einschränkung der IPv6Suite ist, dass der Ethernet-Layer Pakete verwirft, die in kürzeren Abständen als 15 ms eintreffen. Beim Versuch, die Senderate der Multicastübertragung im MN auf weniger als 15 ms einzustellen, traten reproduzierbar *EtherSignalJam*-Signal auf. Da für das genutzte mobile Scenario Access Points (AP) nötig sind und diese nur mit Ethernet-Schnittstellen arbeiten, konnten diese leider nicht ausgetauscht werden.

Dieses Phänomen trat auch bei den Verbindungen zwischen den Routern auf, die über das vordefinierte *intranetCable* verbunden waren. Dieses besitzt ein Link-Delay von 10ms. Ein Mailinglisten-Eintrag [45] führte das Problem auf die Ethernet-Spezifikation zurück, welche Link-Delays über 5.12 μs nicht erlaubt. Daher ist der Name *intranetCable* sehr irreführend und dieses sollte nicht für lokale Verbindungen genutzt werden. Stattdessen wurde ein eigenes Kabel mit einem Link-Delay von 2 μs definiert (siehe Quelltext 8.2).

Listing 8.2: Definition eines Ethernet-Kabels in NED

```
1 channel ethCable
2     delay 0.000002; //2us
3     datarate 100000000;
4 endchannel
```

Da in realen Topologien allerdings durchaus Link-Delays über 5.12 μs auftreten können, wurden alle Interfaces in den Simulationen durch den Typ *IPv6PPPInterface* ersetzt. Da dieser Punkt-zu-Punkt Verbindungen erstellt, sind die Link-Delays dabei frei wählbar.

Obwohl PPP-Verbindungen üblicherweise keine Adressauflösung benötigen, wird diese reproduzierbar durchgeführt. Dabei handelt es sich offensichtlich um einen Fehler in der Implementierung der IPv6Suite. Um diesen Verzögerungen entgegenzuwirken, wurde der im letzten Abschnitt beschriebene Logger eingeführt, der die Adressauflösungs-Delays für jedes Paket aufzeichnet.

Ein ebenfalls schwerwiegendes Problem trat bei einer Simulation auf, in dem sich der MN zu Beginn der Simulation nicht im Heimatnetz befindet. Der aktuelle Designated Router wurde dabei fälschlicherweise als HA angesehen. Bei einem Handover in das Netz des HA schlug die Handover-Erkennung fehl und es wurden daraufhin keine Tree Elongation Pakete gesendet. Die Entwickler der IPv6Suite haben ein Patch für dieses Problem unter [57] zur Verfügung gestellt.

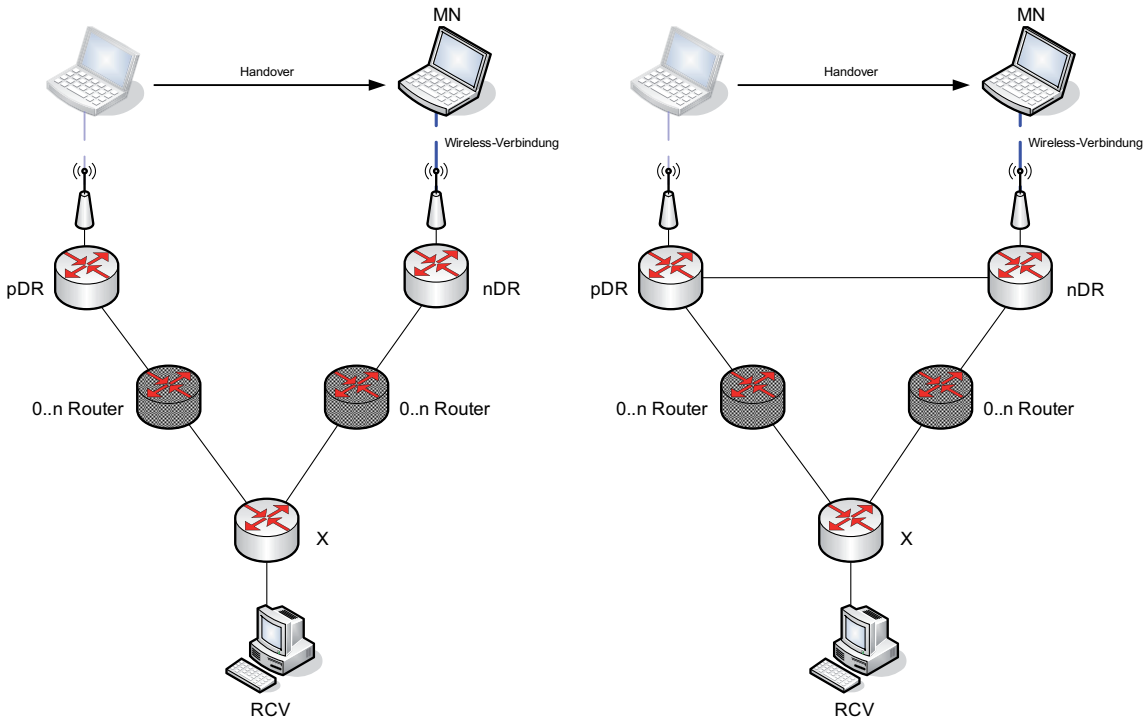
Eine kleine Merkwürdigkeit trat beim automatisierten Erstellen von Simulationsumgebungen auf (siehe nächsten Abschnitt). Die aus den Topologien generierte XML-Datei muss die Definitionen der Interfaces eines Nodes zwingend in aufsteigender Reihenfolge enthalten, da diese sonst nicht beachtet und nicht erstellt werden.

Beim Testen großer Topologien ist weiterhin zu beachten, dass das Multicast Hop-Limit per Voreinstellung auf 30 gesetzt ist. Dies kann und muss in der Datei *Transport/UDP/-UDPProccessing.cc* verändert werden, da ansonsten Paketverluste auftreten.

8.4. Simulationsanalytik

8.4.1. Simulationstopologien / Testnetze

Die Simulation soll Ergebnisse für verschiedene Netzwerktopologien liefern. Dazu werden die in Abbildung 8.2(a) und 8.2(b) dargestellten Testnetze erstellt, um zunächst den Topologieraum strukturell zu überdecken. Diese können skriptgesteuert durchgegeführt werden. Netz 1 besteht aus den Routern pDR, nDR und X, der der Intersection Point zwischen dem vorherigen und nächsten Verteilbaum ist. Der MN führt ein Handover vom Access Point (AP) des pDR zum AP des nDR durch. Zusätzlich können 0 bis n Router in die Verbindung zwischen pDR und X bzw. nDR und X eingefügt werden. Diese werden dann sequentiell miteinander verbunden, was bei 2 zusätzlichen Routern zwischen pDR und X zu den Verbindungen pDR-autoRouterA1-autoRouterA2-X führt. Die auf der Strecke pDR-X eingefügten Router tragen dabei das Prefix autoRouterA, die zwischen nDR-X liegenden autoRouterB.



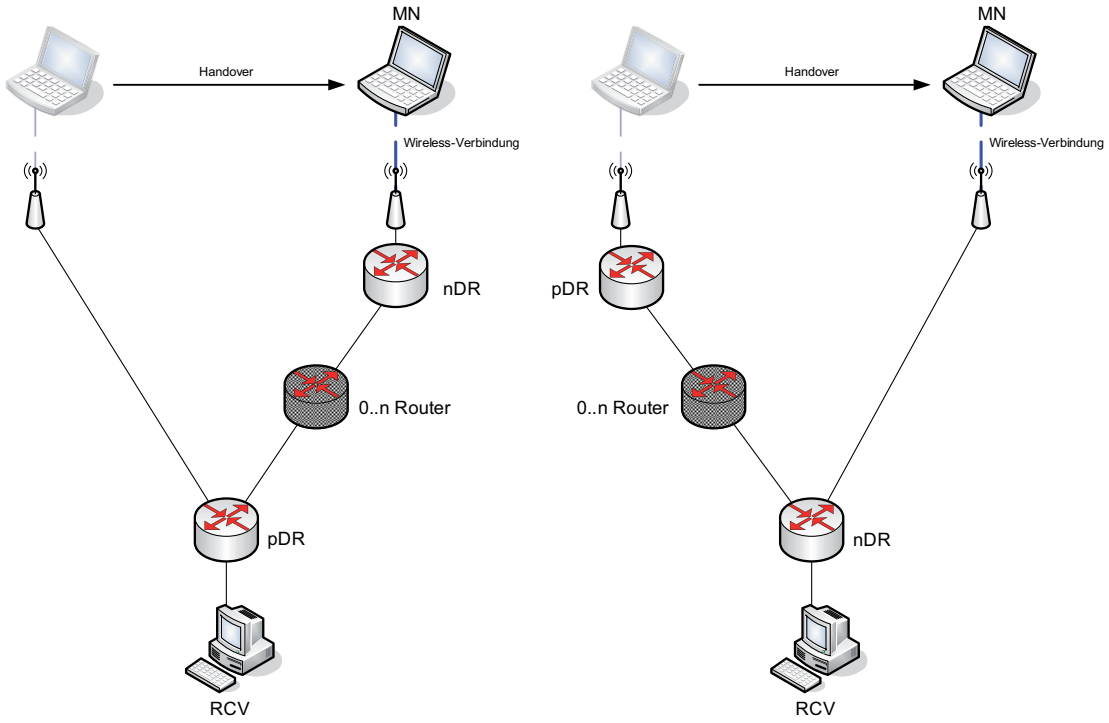
(a) Testnetz 1 (b) Testnetz 2

Abbildung 8.2.: Testnetze

Zusätzlich existieren zwei Spezialfälle: Abbildung 8.3(a) zeigt den Fall, dass die Gesamtentfernung zwischen pDR und X 0 ist, der pDR also die Stelle von X übernimmt. Entsprechendes gilt für den nDR und ist in Abbildung 8.3(b) dargestellt.

Zur automatisierten Simulation wird das Skript *autoTest.sh* genutzt, welches zur Erzeugung der Netze zunächst ein weiteres Skript *generateNetwork.sh* aufruft. Diesem werden die Längen der pDR-X und nDR-X Distanzen als Parameter übergeben, wobei eine Länge

von 0 den Spezialfällen entspricht. Sind beide Strecken durch einen direkten Link verbunden, entsteht das bereits in Abbildung 8.1 vorgestellte Netz. Eine weitere Besonderheit entsteht, wenn beide Strecken 0 Router lang sein sollen. Dabei fallen alle drei Router zusammen. Dadurch führt der MN zwar ein Handover zwischen den Funknetzen durch, die Dauer des Tree Morphing ist aufgrund der weiterhin optimalen States allerdings 0ms.



(a) pDR-X Distanz = 0 (b) pDR-X Distanz = 0

Abbildung 8.3.: Testnetz 1: Sonderfälle

Das zweite Testnetz ist in Abbildung 8.2(b) dargestellt. Im Gegensatz zum Testnetz 1 besteht dabei eine direkte Verbindung zwischen pDR und nDR. Um alle Kombinationen abzudecken, müsste diese Strecke ebenfall mit einer variablen Routerzahl ausgestattet werden können. Da allerdings lediglich die (ganzzahligen) Verhältnisse der Verbindungen relevant sind, kann eine der Strecken auf 1 normiert werden. In diesem Testnetz gibt es die zuvor vorgestellten Spezialfälle nicht, da sich die Testnetze bei diesen Extremfällen gleichen. Daher ist auch die Bezeichnung der Anzahl zwischenliegender Router anders gewählt. Sind keine autoRouter vorhanden, wurde das Netz mit den Längen pDR-X = 0 und nDR-X = 0 erstellt. Dieses Netz soll die Überführung des Verteilbaums von der pDR-X zur nDR-X Verbindung zeigen.

Die zuvor erwähnten Netzgeneratoren finden sich in den Verzeichnissen *TreeMorphingNetwork/AutoTest/Net1* bzw. *TreeMorphingNetwork/AutoTest/Net2* unterhalb des IPv6-Suite-Verzeichnisses.

Während die Testnetze 1 und 2 Ergebnisse für bestimmte Netzwerkkonstellationen liefern, soll das Protokoll weiterhin auch in realen Internet-Topologien auf seine Leistungsfähigkeit getestet werden. Dazu wurde zum einen das AT&T Core-Netzwerk aus [32] als eng

vermaschtes Netzwerk mit 154 Routern eines einzelnen Providers gewählt. Zum anderen wurden zwei Ausschnitte unterschiedlicher Größe der im SCAN-Projekt [29, 55] erstellten Daten als Beispiel für Inter-Provider Netzwerke gewählt. Diese 154 und 1540 Router großen Ausschnitte entsprechen den in [84] genutzten und wurden von den Autoren mit Hilfe des Programms *nem* [50, 51] erstellt.

Da diese Netze noch keine Access Points, keinen MN und keinen Empfänger stellen, werden diese zufällig im Netz verteilt. Die Autoren in [98] liefern den Nachweis dafür, dass die zufällige, gleichverteilte Platzierung von Multicastempfängern der Realität entspricht. Für diese Verteilung ist das Perl-Skript *scan2ned.pl* im Ordner *TreeMorphingNetwork/-AutoTest/random* zuständig. Die weitere Bearbeitung erfolgt entsprechend den Vorgaben im folgenden Abschnitt. Lediglich die Datei *omnetpp.ini* muss bei jedem Simulationslauf modifiziert werden, da dort die Art der Interfaces angegeben wird und vor dem Randomisieren noch nicht feststeht, an welche Router die APs angeschlossen werden[5]. Die randomisierte Simulation auf realen Netzwerktopologien liefert als Ergebnisse den Delay Stretch, den Paketverlust und die Konvergenzzeit in Abhängigkeit der pDR-nDR Distanz. Diese wird vom nDR beim Empfang des TE-Back Pakets ermittelt und ausgegeben, da dieses Paket direkt vom pDR zum nDR gesendet wird. Die Anzahl der durchlaufenen Hops kann dabei anhand des RouterInterfaceID-Arrays festgestellt werden (vgl. Kapitel 8.3). Dadurch muss die OMNeT++-interne Dijkstra-Berechnung nicht durchgeführt werden.

[5]Die Routerinterfaces, an denen APs angeschlossen werden, müssen vom Typ „EtherModule“ sein, während alle anderen „IPv6PPPInterface“ sind (vgl. Kapitel 8.3.1).

8.4.2. Generierung der IP-Strukturen

Die Netzgeneratoren erstellen zunächst nur die NED-Datei, welche den Aufbau des Netzes beschreibt. Zusätzlich ist noch eine XML-Datei nötig, die den Ablauf der Simulation beschreibt, u.a. die Bewegung des MN. Quelltext 8.3 zeigt einen Ausschnitt dieser XML-Datei.

Listing 8.3: Ausschnitt der Datei autoSimulation.xml

```
<netconf>
  <local node="ms1" mobileIPv6Support="on" routeOptimisation="off"
      returnRoutability="off" signalingEnhance="None" mobileIPv6Role="
      MobileNode" hierarchicalMIPv6Support="off" optimisticDAD="on"
      ewuOutVectorHODelays="on" eagerHandover="on">
    <interface name="wlan0" HostDupAddrDetectTransmits="0"/>
  </local>

  <local node="pDR" routePackets="on" mobileIPv6Support="on">
  <interface name="0" AdvSendAdvertisements="on" MIPv6MinRtrAdvInterval="
      0.03" MIPv6MaxRtrAdvInterval="0.07" AdvHomeAgent="on">
    <inetAddr>2001:0:0aa0:2000:0:0:0:0/64</inetAddr>
    <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on" AdvRtrAddrFlag="on">2001
          :0:0aa0:2000:0:0:0:0/64</AdvPrefix>
    </AdvPrefixList>
  </interface>
  <interface name="1" AdvSendAdvertisements="on" MinRtrAdvInterval="1.0"
      MaxRtrAdvInterval="1.5">
    <inetAddr>2001:0:0ab8:1000:0:0:0:0/64</inetAddr>
    <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on" AdvRtrAddrFlag="on">2001
          :0:0ab8:1000:0:0:0:0/64</AdvPrefix>
    </AdvPrefixList>
  </interface>
</local>
...
  <misc>
    <ObjectMovement>
      <MovingNode NodeName="ms1" startTime="0">
        <move moveToX="500" moveToY="142" moveSpeed="15"/>
      </MovingNode>
    </ObjectMovement>
  </misc>

</netconf>
```

Darin kann man nicht nur die Definition der Bewegungsparameter, sondern auch die zugewiesenen IP-Adressen erkennen, welche automatisch vergeben wurden.

Dazu kam der in [25] beschriebene IP-Vergabealgorithmus (Recusive-Partitioner) zum Einsatz, der freundlicherweise als Vorabversion von seinem Entwickler Jonathon Duerig zur Verfügung gestellt wurde. Dieser nimmt als Eingabeparameter einen Graphen entgegen, wobei die Knoten den Routern und die Kanten den Links entsprechen. Das Programm

weist den Links Subnetze und den entsprechenden Router-Interfaces IP-Adressen aus diesem Subnetz zu. Dabei versucht es, topologisch angrenzenden Links Subnetze mit einer möglichst hohen Anzahl an gleichen Prefix-Bits zuzuordnen. Dadurch können die Routingtabellen aller Router besser aggregiert werden als bei einer zufälligen oder sequentiellen Subnetz-Zuweisung. Dadurch kann der für die Simulation benötigte Speicherplatz gering gehalten werden, da das Konfigurationsmodul in OMNeT++ ansonsten für jeden Host die Route berechnen und speichern muss.

Die vergebenen IP-Adressen sind dem IPv4-Adressbereich entnommen und damit für die durchzuführende Simulation erst einmal nicht nutzbar. Sie können jedoch durch ein Mapping in IPv6-Adressen umgewandelt werden. Dazu wird diese um 62 Bits nach Links geshiftet und erhält zusätzlich das Prefix *2001::* (siehe Quelltext 8.4).

Listing 8.4: Ausschnitt der Datei dijkstraOut2XML.pl

```
1  sub ipv4toipv6 {
2    my ($summe);
3    ($ip1, $ip2, $ip3, $ip4) = split(/\./, $_[0]);
4    my $ipv4 = ($ip1 << 24) | ($ip2 << 16) | ($ip3 << 8) | $ip4;
5    my $ifID = $ipv4 & 3;
6    $ipv4 >>= 2;
7    $ipv6_1 = ($ipv4 >> 16) & 0xffff;
8    $ipv6_2 = $ipv4 & 0xffff;
9
10   return sprintf("2001:0:%.4x:%.4x:%.4x:0:0:0",$ipv6_1,$ipv6_2,($ifID <<
         14));
11 }
```

Damit werden die ersten 30 Bits der IPv4-Adresse als Subnetz genutzt, die letzten 2 Bits als Interface Identifier. Da zwischen den Routern PPP-Links eingesetzt werden, sind diese 2 Bits vollkommen ausreichend. Dieses Mapping ist nötig, um dem MN die Möglichkeit zu bieten, an einem AP durch Autokonfiguration eine (fast) beliebige Adresse zu bilden. Daher können auch nicht die in [33] standardisierten „IPv4-Mapped IPv6 Addresses“ zum Einsatz kommen, da diese die IPv4-Adresse in der IPv6-Adresse ::ffff:X:Y/96 abbilden, wobei X und Y die IPv4-Adresse aufnehmen.

Aus den generierten IPv4-Adressdaten erzeugt daraufhin ein Routingtabellen-Generator, der ebenfalls von Jonathon Duerig stammt, eine aggregierte IPv4-Routingtabelle für jeden Host. Diese kann durch die oben angegebene *ipv4toipv6*-Routine in IPv6 umgewandelt werden. Zusätzlich muss jedoch die Subnetzmaske angepasst werden, was durch eine einfache Umwandlung der Subnetzmaske in Dezimalschreibweise und die Addition von 34 zu bewerkstelligen ist. Dadurch liegen die Subnetzmasken im Strukturbereich der gemappten IPv4-Adresse. Die entstandene Routingtabelle wird pro Host ebenfalls in die XML-Datei geschrieben.

Leider besteht das Problem, dass die erstellten Routingtabellen zwar von OMNeT++ eingelesen und die Werte entsprechend eingetragen werden, jedoch konnte zur Abgabe dieser Arbeit nicht ergründet werden, warum die Pakete nicht zugestellt werden. Daher wird als Workaround der in OMNeT++ vorhandenene Routingtabellen-Generator verwendet.

8.4.3. Formatumwandlungen

Die genutzten Programme sind untereinander leider nicht formatkompatibel, daher mussten folgende Konvertierungsskripte geschrieben werden:

- **generateNetwork.sh** Dieses Bash-Skript generiert das gewünschte Testnetz (Netz1 oder Netz2). Als Parameter erhält es die Anzahl der zusätzlich einzufügenden *autoRouter* und erstellt die Datei *autoSimulation.ned*.
- **ned2ipassign.pl** Die zuvor erstellte Datei wird von diesem Perl-Skript eingelesen und wandelt sie in das Eingabeformat des ipassign-Programms um.
- **ipassign2dijkstra.pl** Obwohl die Programme *ipassign* und der Routingtabellen-Generator *dijkstra* vom selben Autor stammen, sind sie nicht formatkompatibel und bedürfen einer Konvertierung.
- **dijkstraOut2XML.pl** Dieses Skript konvertiert die Ausgabe des Routingtabellen-Generators in das für OMNeT++ verständliche XML-Format.
- **dijkstraIn2nodeXML.pl** Da die Routingtabelle von OMNeT++ bisher nicht akzeptiert wird, wandelt dieses Skript die Ausgabe des IP-Vergabeprogramms in das XML-Format um.

Der Aufruf dieser Skripte ist in der Datei *autoTest* zusammengefasst, welche die Simulationen automatisch ablaufen lässt. Dazu Durchlaufen zwei Schleifen alle möglichen Kominationen für die Anzahl der zusätzlich zu generierenden *autoRouter*. Nach jeder Simulation werden die von OMNeT++ in die Datei *omnetpp.sca* ausgegebenen Resultate nach den gewünschten Ergebnissen durchsucht und diese in separaten Dateien mit dem Index der *autoRouter*-Anzahl gespeichert. Die Ausgabe „1 1 20.3“ in der Datei *convergenceTime.dat* gibt z.B. an, dass jeweils ein *autoRouter* in diesem Testlauf vorhanden war und dabei die Konvergenzzeit 20.3 ms betrug.

8.4.4. Simulationsverfahren

Um die Simulation vollautomatisch ablaufen lassen zu können, musste der Simulator ohne graphische Oberfläche übersetzt werden, da dieser ansonsten das manuelle Ausführen der Simulation erwartet. Für alle Simulationen wurden globale Parameter in der Datei *omnetpp.ini* vorgegeben, z.B. dass die Simulation nur bis zu der Simulationszeit 45 Sekunden durchgeführt wird. Durch die fixen Access Points und den immer gleichen Weg des MN, führt dieser das Handover stets zu derselben Zeit aus (bei 18.8 Sekunden).

Für die beiden Testnetzwerke sollen jeweils 900 Durchläufe (also alle Kombinationen aus 0-30 *autoRouterA* und 0-30 *autoRouterB*) durchgeführt werden. Die realen Topologien sollen so lange simuliert werden, bis die Ergebnisse bis die statistischen Schwankungen der Ergebnisse erkennbar insignifikant geworden sind. Dies ist bei dem ATT-Netzwerk bei 2500, bei dem Netzwerk mit 150 Knoten bei 3700 und bei dem mit 1540 Knoten bei 4000 Messungen der Fall gewesen.

8.5. Ergebnisse

Im Folgenden sollen die Ergebnisse der zuvor spezifizierten Tests dargestellt und bewertet werden. Zunächst ist festzustellen, dass bei keiner der Simulationen Paketüberholungen oder -duplikationen auftraten. Zudem wurden keine Routing-Loops erkannt. Alle Link-Delays zwischen den Routern und zwischen X und RCV wurden auf 10 ms eingestellt. Der Link-Delay zwischen Router und Access Point beträgt wie erwähnt 2 μs. Der MN sendet Multicastpakete mit einem Abstand von 15ms, die Sendefrequenz beträgt damit $\frac{1}{15ms} = 66,\bar{6}\frac{1}{s}$.

8.5.1. Testnetzwerk 1

Abbildung 8.4 zeigt den in Testnetzwerk 1 aufgetretenen Delay Stretch. In idealen Netzwerkwerken lässt sich diese Größe mit Hilfe der Formel 8.3 berechnen. Die in den Formeln im weiteren Verlauf des Kapitels genutzte Schreibweise (A-B) bezeichnet die Summe der einzelnen Link-Delays zwischen Host A und B in Millisekunden.

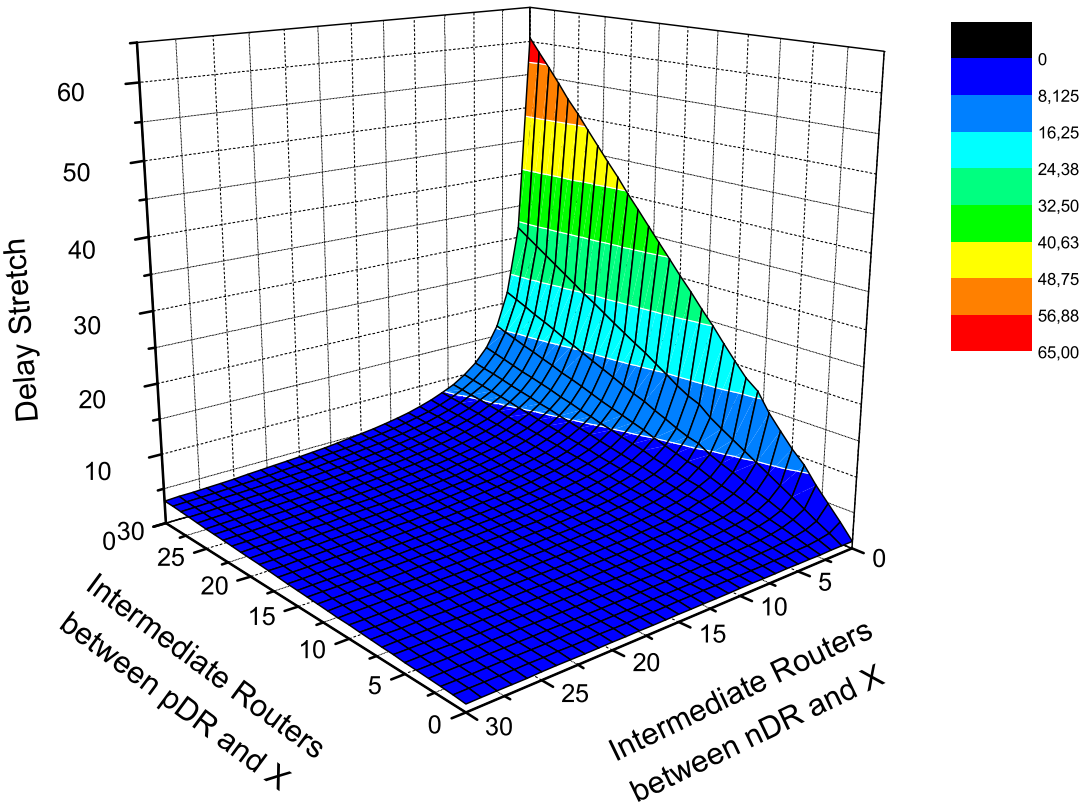


Abbildung 8.4.: Testnetz 1: Delay Stretch

$$\text{Delay Stretch} = \frac{\text{(MN-nDR)} + \text{(nDR-X)} + \text{(X-pDR)} + \text{(pDR-X)} + \text{(X-RCV)}}{\text{(MN-nDR)} + \text{(nDR-X)} + \text{(X-RCV)}} \tag{8.3}$$

Der Zähler der Formel beschreibt dabei den Weg des Pakets während der Tree Elongation Phase vom MN zum pDR und von dort zum RCV, der Nenner den direkten Weg nach dem Tree Morphing.

Im Graphen ist in der X- und Y-Achse die Anzahl der Hops zwischen pDR und X bzw. nDR und X aufgetragen. Die Z-Achse zeigt den zu dem Wertepaar gehörigen Delay Stretch. Zunächst einmal fallen die hohen Werte für kleine nDR-X und große pDR-X Distanzen auf, welche sich dadurch erklären lassen, dass die Distanz zwischen pDR und X zweifach durch den Hin- und Rückweg in den Zähler eingehen, die Distanz zwischen nDR und X jedoch nur einfach. Setzt man die Werte für nDR-X = 0 und pDR-X = 30 Hops in die Formel ein, ergibt sich als Ergebnis ca. 61.

Weiterhin ist ersichtlich auf, dass die Werte für pDR-X = 0 konstant 1 sind. Wird X durch den pDR ersetzt, erreicht das Tree Elongation Paket den pDR auf dem nach dem Handover optimalen Weg. Da keine zusätzliche Übertragungszeit benötigt wird, ist auch der Delay Stretch konstant 1. Für große nDR-X Werte steigt der Delay Stretch langsam an, da dieser Wert direkt in den Nenner der Formel eingeht.

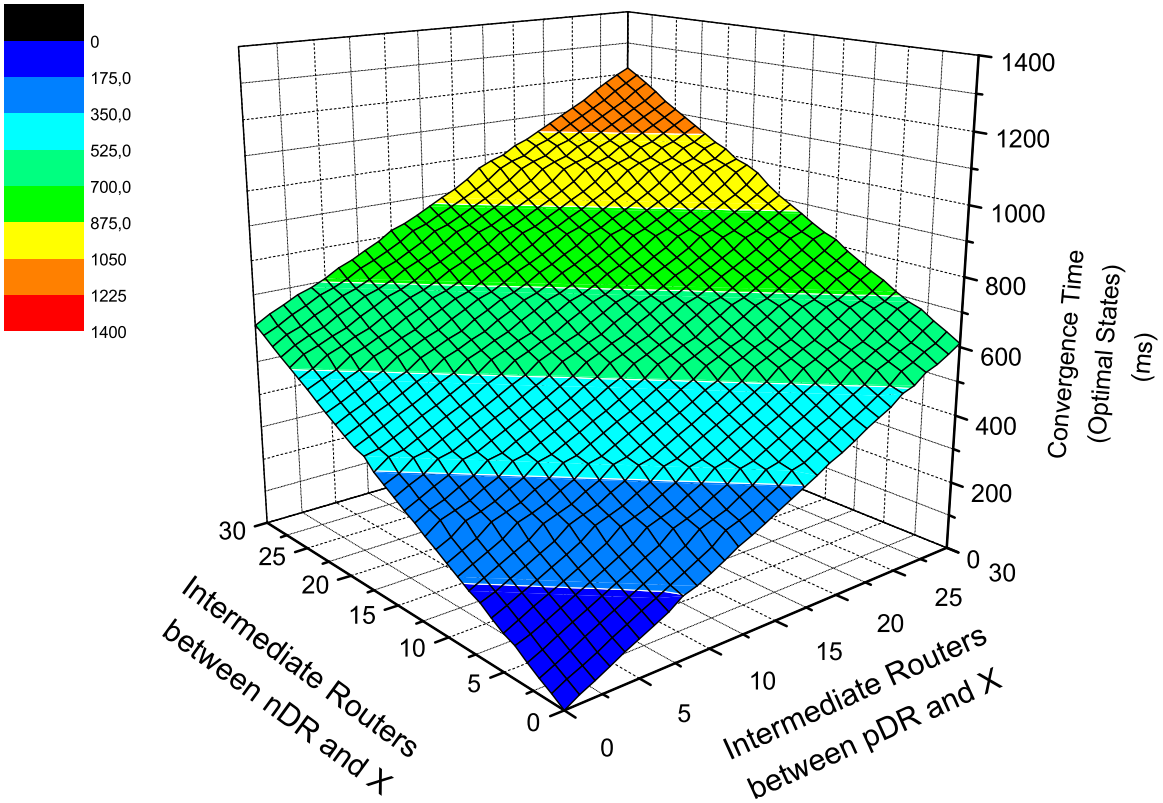


Abbildung 8.5.: Testnetz 1: Kovergenzzeit (optimale States)

$$t_{conv} = \underbrace{(\text{MN-nDR})}_{\text{Tree Elongation}} + \underbrace{(\text{nDR-X})}_{\text{Tree Elongation}} + \underbrace{(\text{X-pDR})}_{\text{Tree Elongation}} + \underbrace{(\text{pDR-X})}_{\text{State Update}} + \underbrace{(\text{X-nDR})}_{\text{TE-Back}} \tag{8.4}$$

Abbildung 8.5 zeigt die Simulationsergebnisse für die Konvergenzzeit des Protokolls bis zur Erlangung des neuen, optimalen Verteilbaums. Die zugehörige Vergleichsformel (8.4) begründet den ersichtlichen linearen Anstieg der Verlaufskurve in den beiden Distanzparametern (nDR-X) bzw. (pDR-X).

Die Konvergenzzeit ergibt sich aus den Zeiten, die das Paket vom MN zum pDR und von dort zurück zu X benötigt. Erreicht das State Update Paket den Router X, sind alle für optimale Paketverteilung nötigen Weiterleitungszustände etabliert. Dennoch erhöht sich die Konvergenzzeit, da diese Zustände nicht genutzt werden, so lange das Tree Elongation Binding Acknowledgement Paket nicht beim MN eingetroffen ist. Durch die Addition der Distanzen entsteht der linear ansteigende Graph.

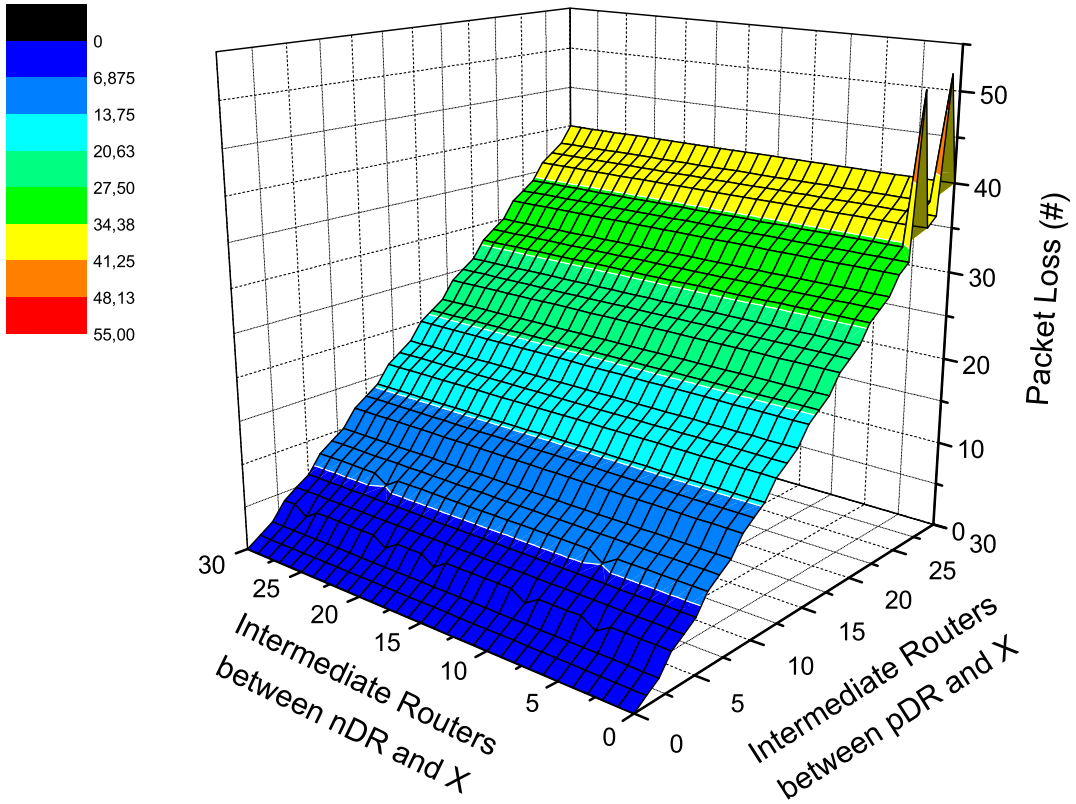


Abbildung 8.6.: Testnetz 1: Paketverlust

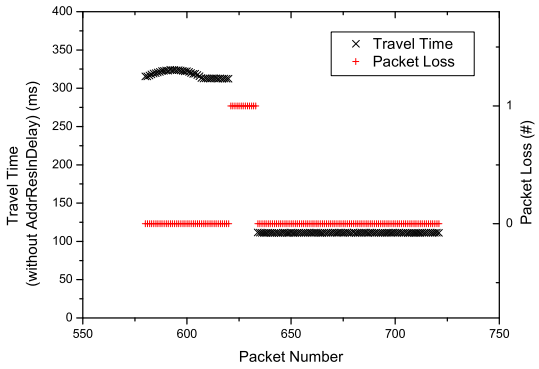
Die Paketverlustfunktion ist in Abbildung 8.6 dargestellt. Auffällig sind zunächst die zwei Messfehler bei nDR-X = 0 auf, welche zwar reproduzierbar, aber ohne erkennbaren Grund auftreten. Daher sollen sie nicht weiter betrachtet werden. Der Paketverlust ist offensichtlich nicht von der nDR-X Distanz abhängig, da sich dieser bei gleichbleibenden pDR-X Distanzen nicht ändert. Mit steigender pDR-X Distanz erhöht sich der Paketverlust linear.

$$\text{Packet Loss} = 2 * (\text{pDR-X}) * \text{Link Delay} * \text{Packet Frequency} \tag{8.5}$$

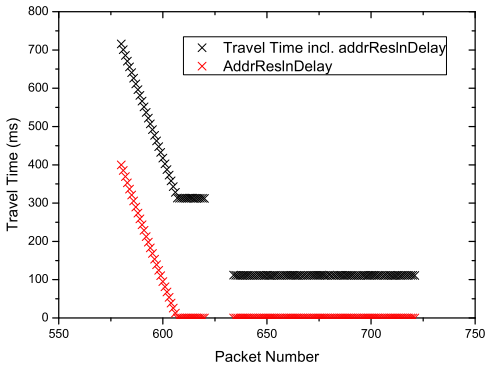
Zunächst soll erklärt werden, warum überhaupt Paketverluste auftreten: Erreichen den pDR Tree Elongation Pakete, sendet sie dieser als State Update Pakete in den alten Verteilbaum. Erreicht das erste dieser Pakete den Router X, sendet dieser eine PIM-Join Nachricht in Richtung des nDR. Dieser erhält, aufgrund der gleichen Übertragungswege, nicht nur das PIM-Join, sondern auch das TE-Back Paket und sendet letzteres an den MN weiter, welcher fortan Reguläre Multicastpakete sendet. Erreicht das erste regulär übertragene Multicastpaket den Router X, sendet dieser ein PIM-Prune Paket in Richtung pDR. Sobald dieses vom nächsten Router in Richtung pDR (oder dem pDR selbst) verarbeitet wurde, bestehen dort keine Weiterleitungszustände mehr und eintreffende Multicastpkete werden verworfen. Aufgrund des Link-Delays von 10ms und einer Multicastpaket-Senderate von 15 ms, befinden sich in Abhängigkeit der pDR-X Distanz einige Pakete auf diesem Verteilzweig, welche dann am Knotenpunkt X verworfen werden (vgl. Formel 8.5).

Die Abbildungen 8.7, 8.8 und 8.9 zeigen die individuellen Verlaufskurven der Übertragungszeit und des Paketverlusts für die Multicastpakete in Testnetz 1 für verschiedene Knotenanzahlen. Die pDR-X und nDR-X Distanzen sind dabei jeweils gleich und betragen 10, 20 bzw. 30 Hops. Die Ergebnisse wurden dabei vom Empfänger gemessen. Bei den jeweils mit (a) beschrifteten Plots wurde die zur Adressauflösung nötige Zeit jeweils substrahiert. In den mit (b) bezeichneten Plots ist diese explizit neben der gemessenen, gesamten Übertragungszeit (inklusive Adressauflösung) dargestellt. Die Wertebereiche der X-Achsen sind so gewählt, dass die Plots jeweils die übertragenen Pakete unmittelbar im Anschluss an das Handover visualisieren.

In allen drei Graphen ist zu sehen, dass die Übertragungszeit zunächst erhöht ist und danach Paketverluste auftreten. Ist die Tree Morphing-Phase abgeschlossen, werden die Pakete über den optimalen Weg ausgeliefert. Daher sinkt die Übertragungsdauer. Auffällig ist jedoch der zu Anfang der Kurve auftretende "Hügel", der durch sich in den Warteschlangen der Router einreihende und auf Adressauflösung wartende Pakete entsteht. Besonders gut ist dies in Abbildung 8.9(b) zu sehen. Dort sinkt die zwischen Paketnummern 580 und 650 gemessene Verzögerung durch Adressauflösungen nicht linear, sondern zunächst stärker und danach langsamer. Substrahiert man diese von der linear sinkenden Übertragungszeit, entsteht der „Hügel". Dieses Phänomen ist eindeutig auf die trotz PPP-Links durchgeführte Adressauflösung in OMNeT++ zurückzuführen und kann damit in realen Impementierungen sehr wahrscheinlich ausgeschlossen werden.

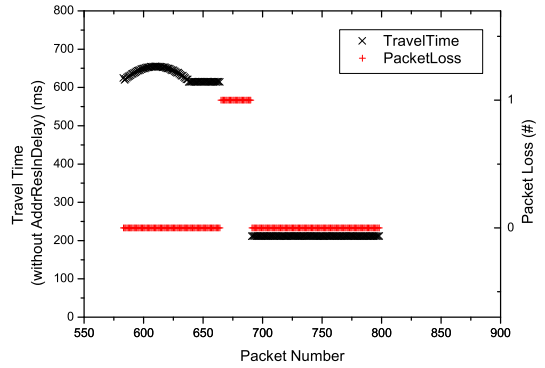


(a) Übertragungszeit und Paketverlust

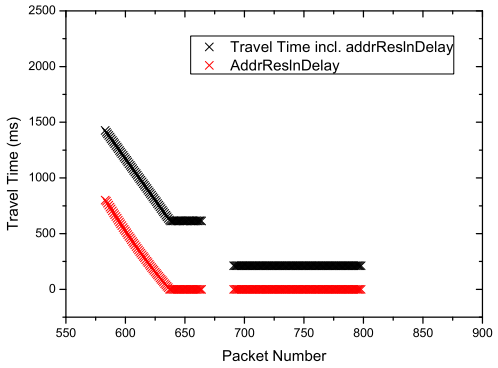


(b) Übertragungszeit und benötigte Zeit zur IP-Adressauflösung

Abbildung 8.7.: Simulationsergebnisse für einen Empfänger in Testnetz 1 bei (pDR-X) = (nDR-X) = 10

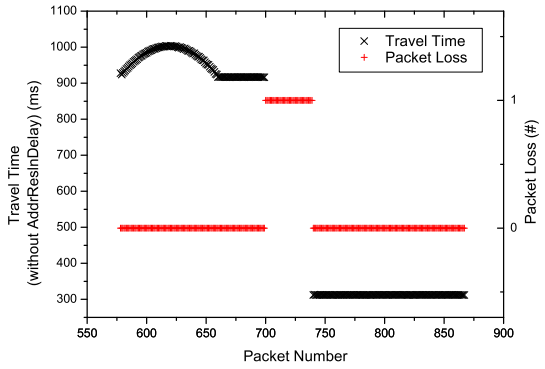


(a) Übertragungszeit und Paketverlust

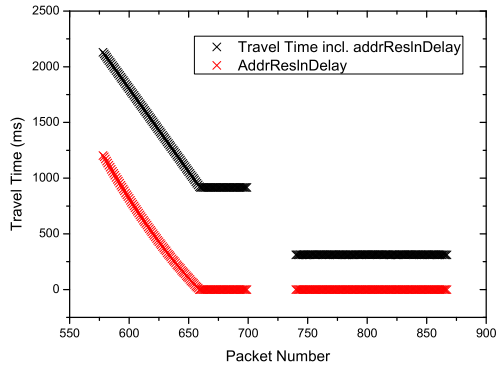


(b) Übertragungszeit und benötigte Zeit zur IP-Adressauflösung

Abbildung 8.8.: Simulationsergebnisse für einen Empfänger in Testnetz 1 bei (pDR-X) = (nDR-X) = 20



(a) Übertragungszeit und Paketverlust



(b) Übertragungszeit und benötigte Zeit zur IP-Adressauflösung

Abbildung 8.9.: Simulationsergebnisse für einen Empfänger in Testnetz 1 bei (pDR-X) = (nDR-X) = 30

8.5.2. Testnetzwerk 2

Der Delay Stretch-Graph für Testnetz 2 ist in Abbildung 8.10 dargestellt. Zunächst einmal sind die Messfehler auffällig, die sich als „Berge“ durch den Plot ziehen. Leider konnte nicht geklärt werden, was diese Fehler hervorruft.

Ist die pDR-X Strecke „kürzer“ als nDR-X, findet die Übertragung nach dem Handover auch weiterhin über den „alten“ Verteilbaum statt, der lediglich zum nDR verlängert wird. Dieses Verhalten ist vergleichbar mit dem von Testnetzwerk 1 für den Fall, dass die nDR-X Distanz gleich 0 ist (vgl. Abbildung 8.4). Der Delay Stretch ist dabei 1 (siehe Formel 8.6).

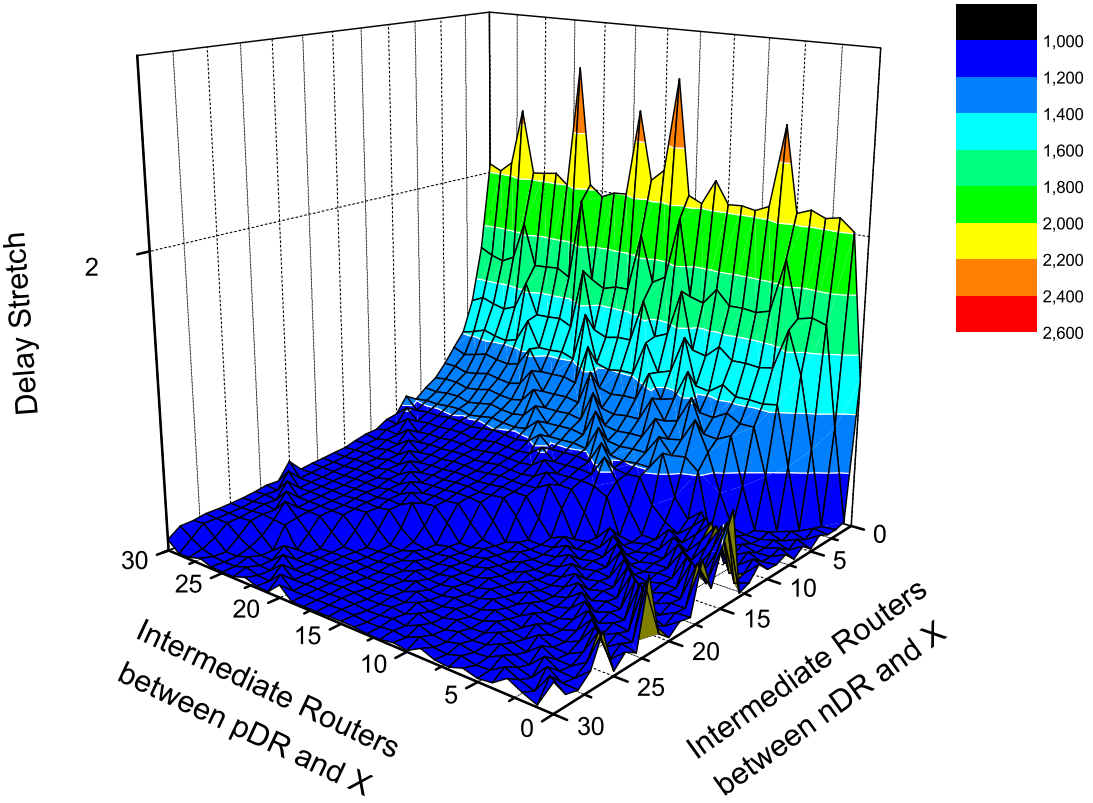


Abbildung 8.10.: Testnetz 2: Delay Stretch

$$\text{Delay Stretch} = \begin{cases} \frac{\text{(MN-nDR)+(nDR-pDR)+(pDR-X)+(X-RCV)}}{\text{(MN-nDR)+(nDR-pDR)+(pDR-X)+(X-RCV)}} = 1 & \text{für (pDR-X)} < \text{(nDR-X)} \\ \frac{\text{(MN-nDR)+(nDR-pDR)+(pDR-X)+(X-RCV)}}{\text{(MN-nDR)+(nDR-X)+(X-RCV)}} & \text{für (pDR-X)} = \text{(nDR-X)} \\ \frac{\text{(MN-nDR)+(nDR-pDR)+(pDR-nDR)+(nDR-X)+(X-RCV)}}{\text{(MN-nDR)+(nDR-X)+(X-RCV)}} & \text{für (pDR-X)} > \text{(nDR-X)} \end{cases} \tag{8.6}$$

Bei gleicher Distanz von pDR-X und nDR-X wird der Verteilbaum zunächst über die Strecke pDR-X aufgebaut. Nach einem Handover konvergiert dieser allerdings zu einer Übertragung über die Strecke nDR-X.

Ist die pDR-X Distanz größer als nDR-X, finden sämtliche Übertragungen über die Strecke nDR-X statt. Nach einem Handover ist die Tree Elogation Phase stets nach derselben Zeit abgeschlossen: (nDR-pDR) + (pDR-nDR), jedoch ändert sich der Delay Stretch durch den Link-Delay von nDR-X. Für den Extremwert (nDR-X) = 0 ergibt sich aus Formel 8.10 unter Berücksichtigung der aktuellen weiteren Linkdelays

$$\frac{2\mu s + 10ms + 10ms + 10ms + 10ms}{2\mu s + 10ms + 10ms} \approx 2 \tag{8.7}$$

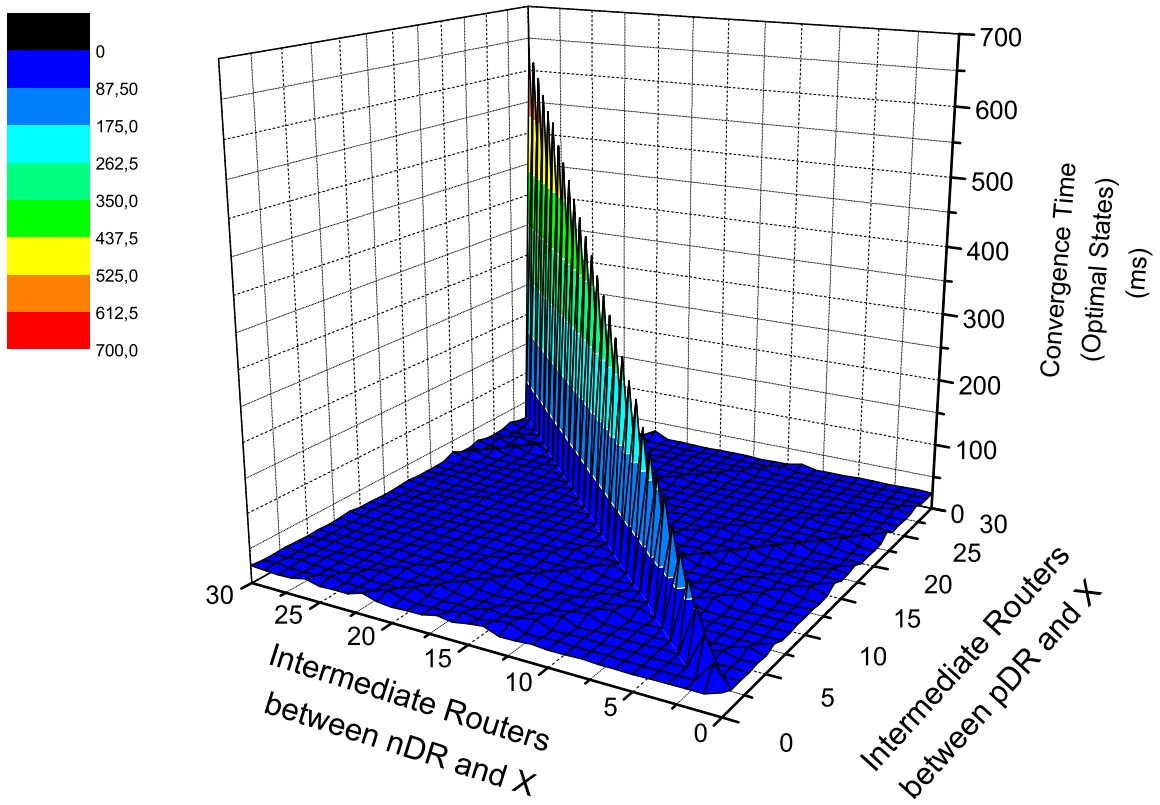


Abbildung 8.11.: Testnetz 2: Kovergenzzeit (optimale States)

$$t_{conv} = \begin{cases} (\text{MN-nDR}) + (\text{nDR-pDR}) & \text{für } (\text{pDR-X}) < (\text{nDR-X}) \\ (\text{MN-nDR}) + (\text{nDR-pDR}) + (\text{pDR-X}) + (\text{X-nDR}) & \text{für } (\text{pDR-X}) = (\text{nDR-X}) \\ (\text{MN-nDR}) + (\text{nDR-pDR}) + (\text{pDR-nDR}) & \text{für } (\text{pDR-X}) > (\text{nDR-X}) \end{cases} \tag{8.8}$$

Die Konvergenzzeit für Testnetz 2 ist in Abbildung 8.11 und als Konturplot in 8.12 dargestellt. Die korrespondierende Vergleichsformel ist 8.8. Dabei kann die Unterteilung der Betrachtungsbereiche analog zum Delay Stretch stattfinden. Ist die pDR-X Strecke kürzer als nDR-X, finden sämtliche Übertragungen über den pDR statt. Daher ist die Konvergenzzeit gleich der Tree Elongation-Zeit und konstant $2\mu s + 10ms = 10.002ms$. Ist das Verhältnis der Strecken umgekehrt, die nDR-X Strecke also kürzer als pDR-X findet die Übertragung zu X immer über den nDR statt. Daher muss zur vorher betrachteten Konvergenzzeit noch (pDR-nDR) addiert werden: $2\mu s + 10ms + 10ms = 20.002ms$. Bei gleich langen pDR-X und nDR-X Strecken findet das beschriebene „Baum-Umschwenken“ statt. Die Konvergenzzeit ergibt sich also aus der Übertragungsszeit vom MN zu X, welcher dann ein PIM-Join in Richtung nDR sendet. Somit ergibt sich für Distanzen von 30 eine Konvergenzzeit von: $2\mu s + 10ms + (30 * 10ms) + (30 * 10ms) = 610.002ms$

In diesem Plot treten erneut die „Wellen“ auf, welche ebenfalls als Messfehler angesehen werden müssen.

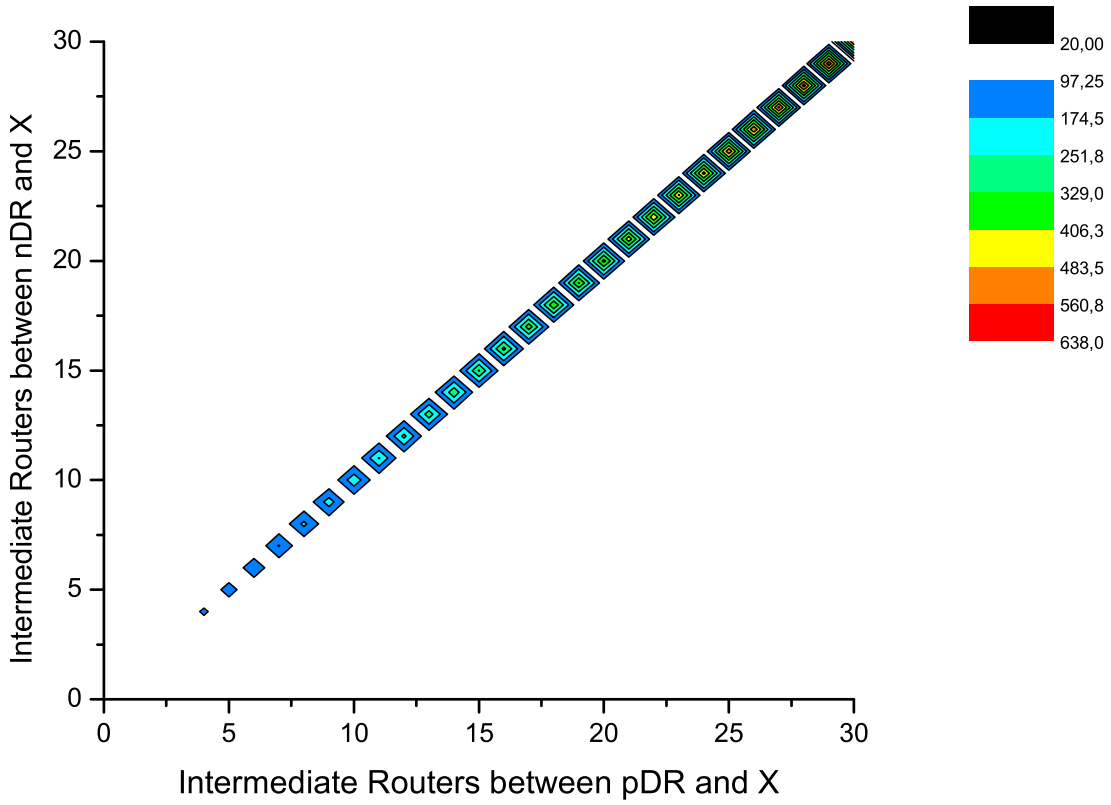


Abbildung 8.12.: Testnetz 2: Kovergenzzeit (optimale States) (Contour-Plot)

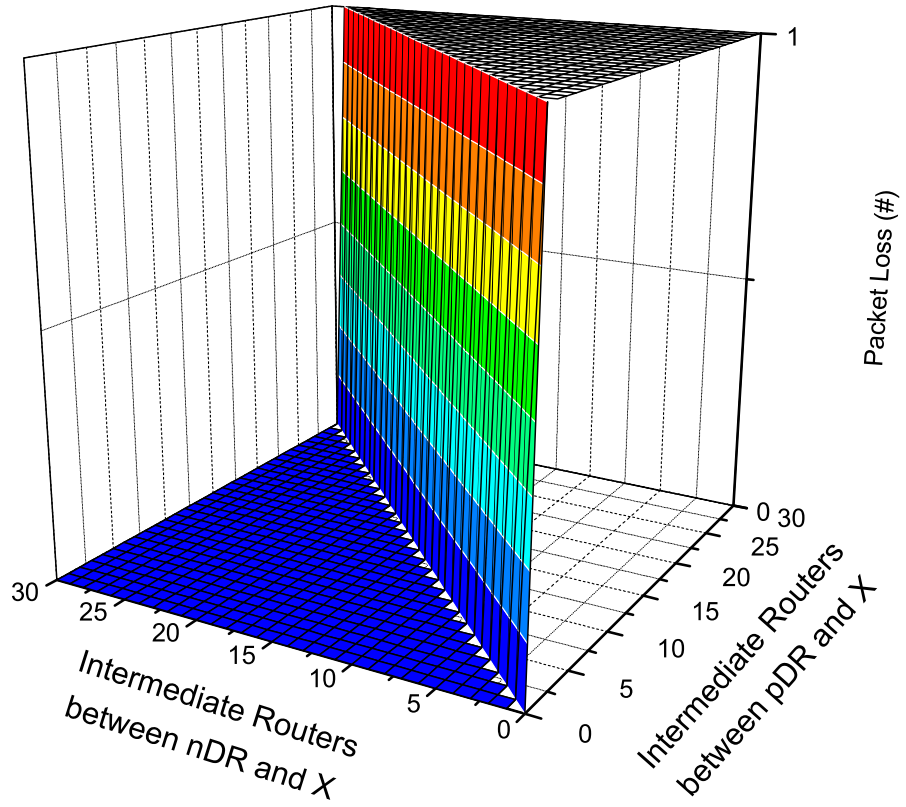


Abbildung 8.13.: Testnetz 2: Paketverlust

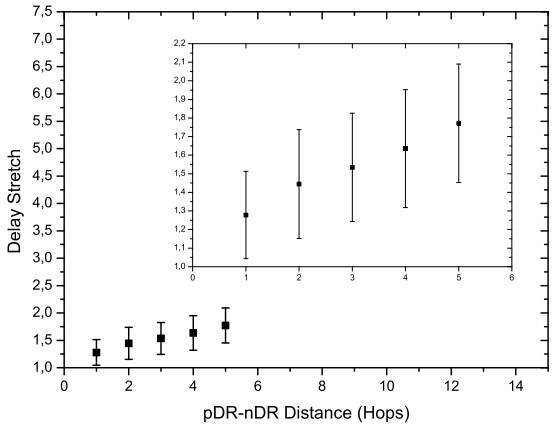
$$\text{Packet Loss} = \begin{cases} 0 & \text{für (pDR-X) <= (nDR-X)} \\ 1 & \text{für (pDR-X) > (nDR-X)} \end{cases} \tag{8.9}$$

Abbildung 8.13 zeigt den Paketverlust-Graphen für Testnetz 2. Dieser zeigt, dass sobald pDR-X größer als nDR-X ist, ein einzelnes Paket verloren wird (siehe Formel 8.9). Dies ist durch die Übertragungsstrecke zwischen nDR und pDR zu erklären, welche sich analog zu Testnetzwerk 1 für den Fall, dass nDR = X und pDR-X = 1 verhält. Sobald das TE-Back Paket vom pDR am nDR eintrifft, sendet dieser ein PIM-Prune zurück. Trifft es am pDR ein, werden alle weiteren Pakete verworfen. Da die Strecke zwischen pDR und nDR fest einen Hop beträgt, ändert sich dieses Verhalten bei Änderung der pDR-X und nDR-X Distanzen nicht.

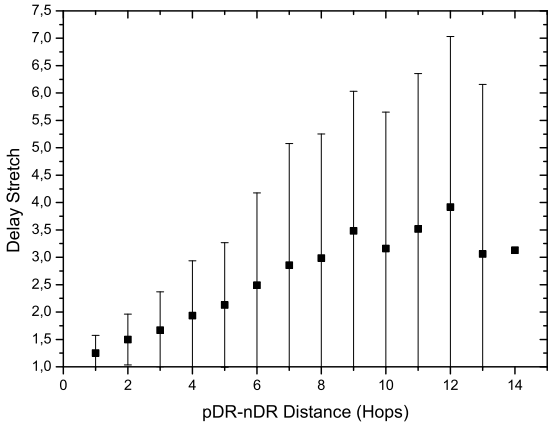
8.5.3. Reale Netzwerk-Topologien

Abbildung 8.14 zeigt die Delay Stretch Plots für das AT&T Core-Netzwerk (8.14(a)) und für Internet-Topologien mit 154 (8.14(b)) und 1540 (8.14(c)) Knoten. Auf der X-Achse ist dabei die Entfernungen des pDR zum nDR in Hops aufgetragen. Um die Plots miteinander vergleichen zu können, besitzen alle drei die gleiche Skalierung. Die Punkte symbolisieren die Mittelwerte der erhaltenen Ergebnisse, die Fehlerbalken die zugehörige Standardabweichung vom jeweiligen Mittelwert.

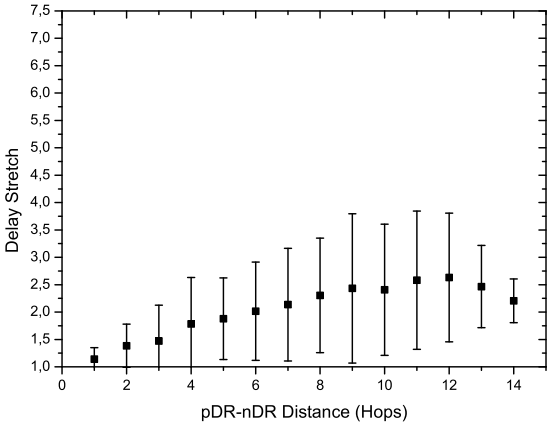
Da das AT&T-Netzwerk eng vermascht ist, können zwei beliebige Knoten über maximal 5 Hops miteinander kommunizieren. Daher bestehen die Plots in diesem und allen folgenden Plots lediglich aus Werten für 1-5 Hops. Um die Messwerte besser auswerten zu können, ist für dieses Netzwerk jeweils ein Detailplot vorhanden, welcher die Messwerte vergrößert darstellt.



(a) ATT Core Network



(b) Internet 154 Nodes



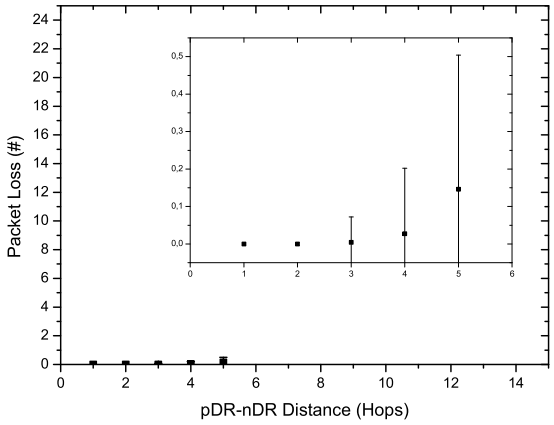
(c) Internet 1540 Nodes

Abbildung 8.14.: Delay Stretch für reale Netzwerktopologien

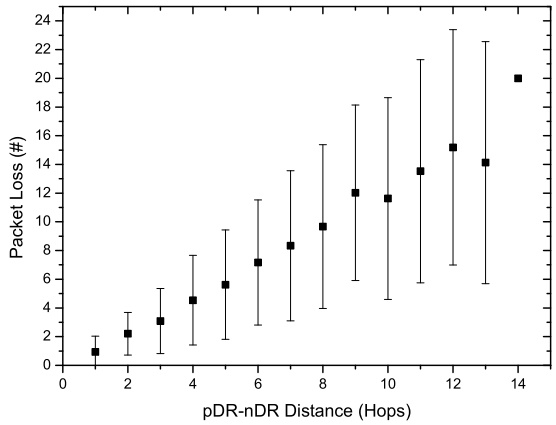
Wie zu erkennen ist, steigt der Mittelwert des Delay Stretch bei geringer Standardabweichung beim AT&T-Netzwerk lediglich auf 1,75 an, was aufgrund des kleinen, eng vermaschten Netzes den Erwartungen entspricht.

Die Mittelwerte für die Internet-Topologie mit 154 Knoten steigt bei hoher Standardabweichung auf etwa 4 an (Hop-Distanz 12), die Werte im Netzwerk mit 1540 Knoten dagegen lediglich auf 2,75. Dabei ist die Standardabweichung ebenfalls geringer. Dieser Effekt kann durch die Größe der Topologie erklärt werden. Bei 154 Knoten ist die Wahrscheinlichkeit eine für den Delay Stretch ungünstige pDR/nDR/RCV-Verteilung, bei der der pDR z.B. an einen Edge-Router angeschlossen wird, der nur ein Interface zum Netzwerk besitzt, deutlich größer als im Netz mit 1540 Knoten. Dieses bietet ausreichend Möglichkeiten auch Distanzen von 7 Hops zwischen pDR und nDR ohne die Auswahl von Edge-Routern zu simulieren.

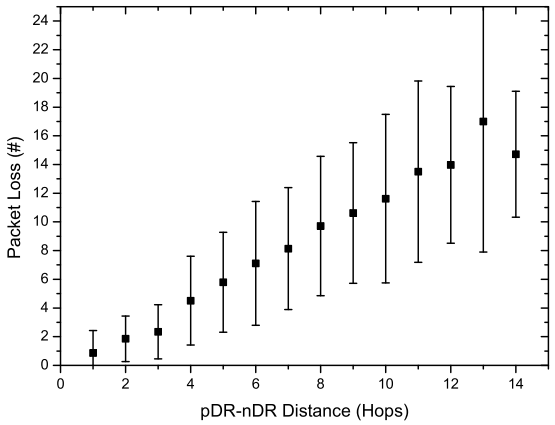
Da die Anzahl der Samples für Hop-Distanzen ab 12 zu gering ist, um aussagekräftige Werte zu liefern, können die an diesen Stellen sinkenden Delay Stretch-Werte als nicht signifikant angesehen werden.



(a) ATT Core Network



(b) Internet 154 Nodes



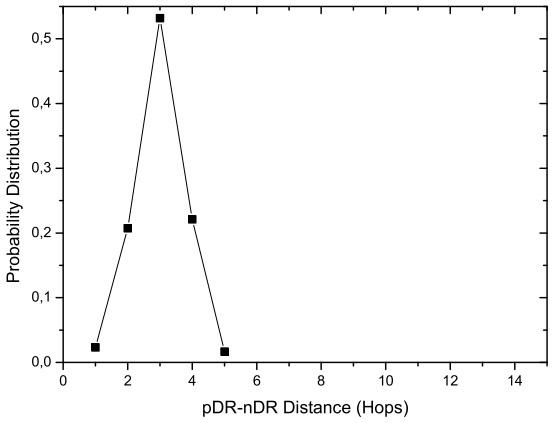
(c) Internet 1540 Nodes

Abbildung 8.15.: Paketverlust für reale Netzwerktopologien

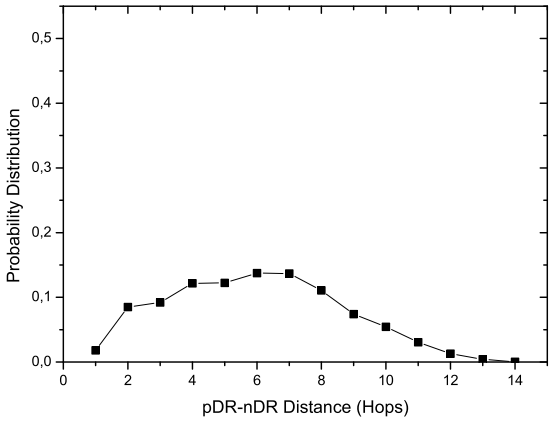
Dies gilt auch für den Paketverlust, der im AT&T-Netz erwartungsgemäß sehr gering ist. Im Mittel liegt dieser bei nur 0,15 für eine Hop-Distanz von 5, da die Link-Delays dieses Netzes im Mittel wesentlich geringer sind als die der anderen Netze. Durch die konstante Sendefrequenz des MN werden die Pakete wesentlich schneller ausgeliefert und weniger Pakete können sich auf dem „abgeschnittenen Ast“ befinden.

Die Ergebnisse der Simulation für die beiden Internet-Topologien sind ähnlich hoch. Dies kann durch den Plot der Wahrscheinlichkeitsverteilung der Samples begründet werden (siehe Abbildung 8.16). Die durchschnittliche Distanz zwischen pDR und nDR ist dabei ca. 6 für das Netz mit 154 Knoten und eindeutig 7 für das größere Netz. Durch diese vergleichbaren Ergebnisse entstehen auch mit derselben Wahrscheinlichkeit Netzwerkkonstellationen, die zu Paketverlusten führen.

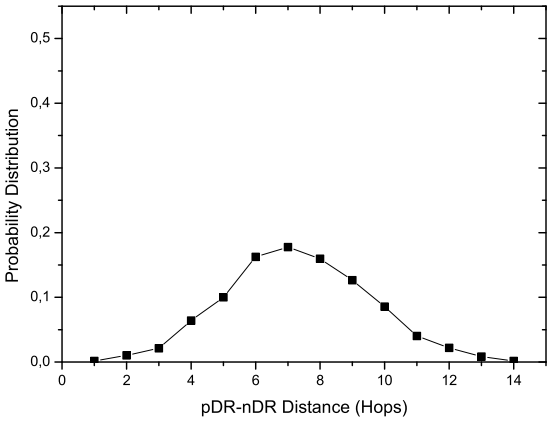
Die Standardabweichung für die Hop-Distanz 14 im Plot mit 154 Knoten fehlt, da lediglich ein Sample dafür aufgezeichnet wurde.



(a) ATT Core Network



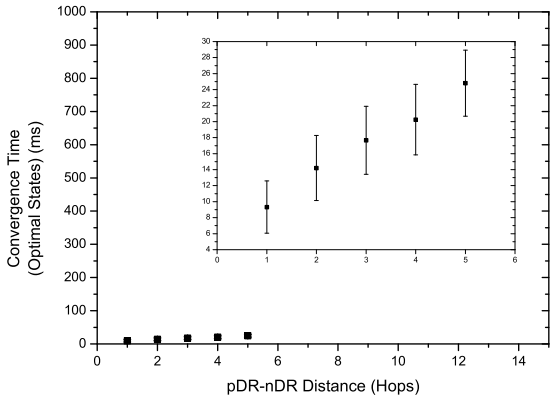
(b) Internet 154 Nodes



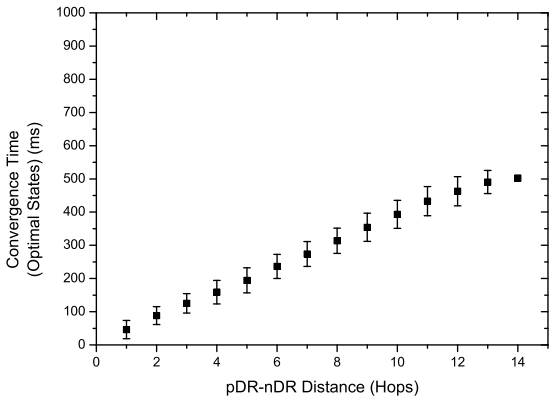
(c) Internet 1540 Nodes

Abbildung 8.16.: Wahrscheinlichkeitsverteilung der Samples für reale Netzwerktopologien

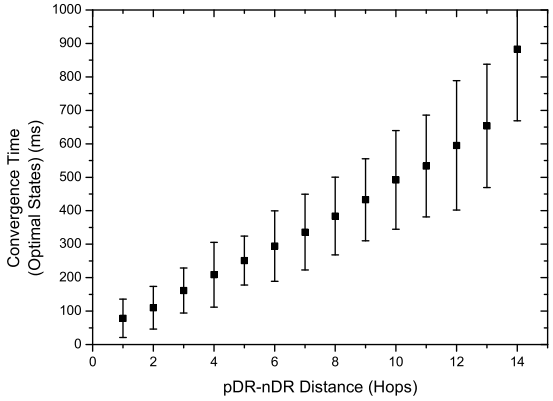
Abbildung 8.17 zeigt die Konvergenzzeit, die verstreicht, bis die Router optimale States besitzen. Neben den erwartet Geringen im AT&T-Netz ist festzustellen, dass die Konvergenzzeiten im Netz mit 154 Knoten überraschend geringe Standardabweichungen besitzen. Weiterhin sind die Mittelwerte durchgehend geringer als in der Topologie mit 1540 Knoten.



(a) ATT Core Network



(b) Internet 154 Nodes



(c) Internet 1540 Nodes

Abbildung 8.17.: Konvergenzzeit (optimale States) für reale Netzwerktopologien

8.6. Auswertung und Diskussion

Die Ergebnisse verdeutlichen die Stärken und Schwächen des Tree Morphing Protokolls in seiner gegenwärtigen Ausgestaltung. Insbesondere die Konvergenzzeit für das Netz mit 1540 Knoten ist von Bedeutung. Geht man von einer Handover-Schrittweite von 5 Hops aus, ist der Verteilbaum bereits nach durchschnittlich 250 ms konvergiert, was auch für schnelle Handover ausreichend ist. Dabei tritt jedoch ein durchschnittlicher Verlust von 6 Paketen auf. Dieser Paketverlust entsteht durch die in Testnetz 1 forcierte Konstellation, welche in realen Netzen weitaus seltener auftreten wird, da meist kein einzelner Router existiert, der als letzter Punkt vor der Verteilung des Pakets in die Multicastgruppe dient. Zudem treten diese Verluste nicht zwangsweise bei allen Empfängern auf, da bereits am pDR eine Duplikation der Pakete stattfinden kann. Durch die weitere topologische Auffächerung der Empfänger minimiert sich dieser Effekt weiterhin, so dass viele Empfänger überhaupt keine Verschlechterung der Übertragungsqualität bemerken.

Der im Netz mit 1540 Knoten auftretende Delay Stretch von durchschnittlich 1,9 bei einer Handover-Schrittweite von 5 fügt eine nicht zu unterschätzende zusätzliche Latenzen zu Beginn des Tree Morphings ein. Im Gegensatz zum dadurch gewonnenen Nutzen ist dies minimal. Einen komplett neuen Verteilbaum aufzubauen dauert dagegen wesentlich länger. Zudem ist der Verteilbaum bei dieser Messung nach 250 ms konvergiert, wodurch der zusätzliche Delay Stretch lediglich für eine kurze Dauer auftritt. Bei einer VoIP-Konferenz würde ein Handover somit im ungünstigsten Fall als kurzer Übertragungsaussetzer auffallen.

Größere Handover-Schrittweiten führen zu erhöhten Paketverlust, Konvergenzzeit und Delay Stretch-Werten. Wie in den Plots für die Internet-Topologie mit 1540 Knoten zu sehen ist, steigen alle Werte jedoch linear an. Daher kann auch für „weite“ Handover zuverlässig vorgesagt werden, in welchem Ausmaß sich die Übertragungsqualität durchschnittlich während des Tree Morphings verschlechtert.

Die dargestellten Konvergenzzeiten und vor allem die Paketverlustanzahlen wurden durch die in dieser Arbeit eingeführte Tree Elongation Binding Acknowledgement Nachricht verlängert, welche den Baumanschluß sicherstellt. Kürzere Konvergenzzeiten und geringere Paketverluste können erreicht werden, indem zunächst weiter optimiert wird. Ein Ansatz wäre, dass der MN nach einem Handover nur eine bestimmte Anzahl an Tree Elongation Paketen sendet, dann auf reguläre Übertragung umstellt und darauf vertraut, dass die Forwardings-States zum pDR dann bereits installiert wurden. Erhält der MN das Tree Elongation Acknowledgement Paket nach einer bestimmten Zeitspanne nicht, könnte er als Fallback wieder in den sicheren Modus zurückschalten und so lange Tree Elongation Pakete senden, bis der pDR erreicht ist.

9. Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Protokoll für den Tree Morphing-Algorithmus entworfen, um mobile Sender bei Source Specific Multicast-Übertragungen zu unterstützen. Dabei wurde inbesondere auf ein leichtgewichtiges Design und die kryptographische Absicherung der Updates geachtet. Das entwickelte Protokoll überträgt die nötigen Informationen einzig durch die zweckgebundene Kombination vorhandener IPv6-Header unter der Definition eines neuen Routing Header Typs. Sämtliche Updates sind durch den Einsatz von kryptographischen Paketadressen gegen Übernahme-Versuche des Verteilbaums geschützt. Zudem wurde die Robustheit gegenüber Netzwerkfehlern nachgewiesen.

Durch die theoretische Evaluierung und formale Verifikation des Protokolls wurde die fehlerfreie Arbeitsweise des Protokolls für einen Router nachgewiesen. In weiteren Arbeiten sollte diese Verifikation auf ein ganzes Routernetzwerk ausgedehnt werden.

Das Protokoll wurde nachfolgend auf der Netzwerksimulationsplattform OMNeT++ implementiert, um seine Leistungseigenschaften in praktischen Simulationen untersuchen zu können. Da Multicastverteilbäume vor allem von den topologischen Netzwerkeigenschaften geprägt werden, gingen insbesondere unterschiedliche Topologiedaten in die Simulationen ein. Die Untersuchungen stützten sich einerseits auf synthetische Geometrien, welche den prinzipiell möglichen Zustandsraum überdeckten, andererseits auf real gemessene Daten des gegenwärtigen Internets.

Da OMNeT++ die Ausgaben des Routingtabellen-Generators bisher nicht verarbeitet hat, konnten weitere Tests mit größeren Topologien nicht durchgeführt werden. Da beim Start der Simulation für jeden Knoten eine Routingtabelle mit Host-Routen für jeden Ziel-Host erstellt wird, erfordert dies einen hohen Bedarf an Speicher.

Es wurden alle relevanten Leistungsdaten des Protokolls untersucht. Dabei konnten zunächst die uneingeschränkte Funktionsfähigkeit sowie ein weitgehend überzeugendes Leistungsverhalten des Tree Morphing Protokolls aufgezeigt werden. Aber auch Protokollschwächen und Optimierungspotentiale konnten als Ergebnis dieser Arbeit identifziert werden. So stellten die Simulationen klar heraus, dass die Absicherung der Übertragung durch das Tree Elongation Binding Acknowledgement Paket die Konvergenzzeit bei bestimmten Netzkonstellationen erhöht, obwohl die in den Routern vorhandenen States bereits konvergiert sind. Dadurch kann der Baumanschluß jedoch sichergestellt werden, was der Robustheit des Protokolls zugute kommt. Weiterhin konnten die Simulationsergebnisse des Tree Morphing Protokolls in realen Internet-Topologien dessen Leistungsfähigkeit nachweisen.

Es besteht weiteres Optimierungspotential in der Phase des Tree Elongation. Bei Empfang der Tree Elongation Nachricht könnten Router, die Intersection Points darstellen - also

bereits Teil des vorherigen Verteilbaums waren, diese Pakete bicasten. Das Update erreicht die Empfänger dadurch wesentlich schneller, da der zusätzliche Weg über den pDR eingespart wird. Dies wirkt sich ebenfalls positiv auf die Paketverlustzahlen aus, da die Pakete auf dem „abgeschnittenen“ Zweig bereits vom Intersection Point in den Verteilbaum gesendet wurden. Allerdings entstehen durch diese Vorgehensweise Paketduplikationen. Diesen könnte der Router, der den Intersection Point darstellt, entgegenwirken, indem er eine Liste der Sequenznummern bereits weitergeleiteter Pakete vorhält und Duplikate verwirft, was allerdings mit erhöhtem Speicherbedarf und zusätzlichem Prozessierungsaufwand verbunden ist.

Weiterhin könnte das Tree Morphing Protokoll mit Beschleunigungs-Protokollen, wie z.B. FMIPv6, verwendet werden. Dabei entspricht bei einem Handover der pAR dem pDR und der nAR dem nDR. Dadurch kann der für das Weiterleiten der Pakete genutzte Tunnel zwischen pAR und nAR auch in umgekehrter Richtung genutzt werden, um die State Update Pakete abgesichert zum pDR zu übertragen. Dies könnte den zur Zeit auftretenden Paketverlusten entgegengewirken, da durch das Tunneln der Pakete kein „Tree Elongation Pfad“ vom nDR zum pDR exististiert.

Nachdem die korrekte Arbeitsweise des Protokolls im Simulator nachgewiesen wurde, soll die Implementierung in weiterführenden Arbeiten auf eine reale Routerplattform, z.B. XORP portiert werden. Dazu kann die in dieser Arbeit implementierte Forwarding State Table inklusive der Zugriffsfunktionen direkt übernommen werden, da sie nicht von OMNeT++ abhängt und ausschließlich auf Standardfunktionen der Sprache C++ zurückgreift. Obwohl sich die Methoden, Multicastpakete zu behandeln, in XORP gänzlich von denen in OMNeT++ unterscheiden, können die für das Tree Morphing wichtigen Quelltext-Stellen dennoch als Vorlage dienen, eine Implementierung durchzuführen.

A. Übersicht über vergebene Routing Header Typen

Um einen freien Typ des für das Tree Morphing benötigten Routing Headers bestimmen zu können, sind im Folgenden die in Internet-Drafts vorhandenen Typen dargestellt.

Routing Header Typ	**Titel**	**Quelle**
0	Internet Protocol, Version 6 (IPv6) Specification	[22]
1	Mobile IPv6 Neighborhood Routing for Fast Handoff	[106]
2	Mobility Support in IPv6	[43]
3 und 4	IPv6 Reverse Routing Header and its application to Mobile Networks	[97]
5	Optimized Routing in Nested NEMO Networks	[71]
6	Secure Nested Tunnels Optimization using Nested Path Information	[67]

Tabelle A.1.: Übersicht über vergebene Routing Header Typen

Aktuell ist damit der Routing Header Typ 7 für das Tree Morphing Protokoll verfügbar.

Literaturverzeichnis

[1] Adams, A. ; Nicholas, J. ; Siadak, W. : *Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised)*. RFC 3973 (Experimental). http://www.ietf.org/rfc/rfc3973.txt. Version: Jan. 2005 (Request for Comments)

[2] Albanna, Z. ; Almeroth, K. ; Meyer, D. ; Schipper, M. : *IANA Guidelines for IPv4 Multicast Address Assignments*. RFC 3171 (Best Current Practice). http://www.ietf.org/rfc/rfc3171.txt. Version: Aug. 2001 (Request for Comments)

[3] Arkko, J. ; Devarapalli, V. ; Dupont, F. : *Using IPsec to Protect Mobile IPv6 Signaling Between Mobile Nodes and Home Agents*. RFC 3776 (Proposed Standard). http://www.ietf.org/rfc/rfc3776.txt. Version: Jun. 2004 (Request for Comments). – Updated by RFC 4877

[4] Arkko, J. ; Kempf, J. ; Zill, B. ; Nikander, P. : *SEcure Neighbor Discovery (SEND)*. RFC 3971 (Proposed Standard). http://www.ietf.org/rfc/rfc3971.txt. Version: März 2005 (Request for Comments)

[5] Arkko, J. ; Vogt, C. ; Haddad, W. : *Enhanced Route Optimization for Mobile IPv6*. RFC 4866 (Proposed Standard). http://www.ietf.org/rfc/rfc4866.txt. Version: Mai 2007 (Request for Comments)

[6] Aura, T. : *Cryptographically Generated Addresses (CGA)*. RFC 3972 (Proposed Standard). http://www.ietf.org/rfc/rfc3972.txt. Version: März 2005 (Request for Comments). – Updated by RFCs 4581, 4982

[7] Bagnulo, M. ; Arkko, J. : *Cryptographically Generated Addresses (CGA) Extension Field Format*. RFC 4581 (Proposed Standard). http://www.ietf.org/rfc/rfc4581.txt. Version: Okt. 2006 (Request for Comments)

[8] Baker, F. ; Savola, P. : *Ingress Filtering for Multihomed Networks*. RFC 3704 (Best Current Practice). http://www.ietf.org/rfc/rfc3704.txt. Version: März 2004 (Request for Comments)

[9] Ballardie, A. : *Core Based Trees (CBT) Multicast Routing Architecture*. RFC 2201 (Experimental). http://www.ietf.org/rfc/rfc2201.txt. Version: Sept. 1997 (Request for Comments)

[10] Ballardie, A. : *Core Based Trees (CBT version 2) Multicast Routing – Protocol Specification –*. RFC 2189 (Experimental). http://www.ietf.org/rfc/rfc2189.txt. Version: Sept. 1997 (Request for Comments)

[11] Bates, T. ; Chandra, R. ; Katz, D. ; Rekhter, Y. : *Multiprotocol Extensions for BGP-4*. RFC 4760 (Draft Standard). http://www.ietf.org/rfc/rfc4760.txt. Version: Jan. 2007 (Request for Comments)

[12] Begg, L. ; Liu, W. ; Pawlikowski, K. ; Perera, S. ; Sirisena, H. : *Survey of simulators of Next Generation Networks for studying service availability and resilience*. http://www.cosc.canterbury.ac.nz/research/reports/TechReps/2006/tr_0605.pdf

[13] Bérard, B. ; Bidoit, M. ; Finkel, A. ; Laroussinie, F. ; Petit, A. ; Petrucci, L. ; Schnoebelen, P. : *Systems and Software Verification: Model-Checking Techniques and Tools*. New York, NY, USA : Springer-Verlag New York, Inc., 2001. – ISBN 3–540–41523–8

[14] Cain, B. ; Deering, S. ; Kouvelas, I. ; Fenner, B. ; Thyagarajan, A. : *Internet Group Management Protocol, Version 3*. RFC 3376 (Proposed Standard). http://www.ietf.org/rfc/rfc3376.txt. Version: Okt. 2002 (Request for Comments). – Updated by RFC 4604

[15] Castellucia, C. ; Montenegro, G. : Securing group management in IPv6 with cryptographically generated addresses. In: *Computers and Communication, 2003. (ISCC 2003). Proceedings. Eighth IEEE International Symposium on* IEEE, IEEE Computer Society Press, 588–593 vol.1

[16] Chang, R.-S. ; Yen, Y.-S. : A Multicast Routing Protocol with Dynamic Tree Adjustment for Mobile IPv6. In: *Journ. Information Science and Engineering* 20 (2004), S. 1109–1124

[17] Cheshire, S. ; Aboba, B. ; Guttman, E. : *Dynamic Configuration of IPv4 Link-Local Addresses*. RFC 3927 (Proposed Standard). http://www.ietf.org/rfc/rfc3927.txt. Version: Mai 2005 (Request for Comments)

[18] Christ, O. ; Schmidt, T. C. ; Wählisch, M. : A Light-Weight Implementation Scheme of the Tree Morphing Protocol for Mobile Multicast Sources. In: Müller, P. (Hrsg.) ; Liggesmeyer, P. (Hrsg.) ; Maehle, E. (Hrsg.): *Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Application SEAA 2007*. Los Alamitos, CA, USA : IEEE Computer Society Press, August 2007. – ISBN 0–7695–2977–1, S. 149–156

[19] Conta, A. ; Deering, S. ; Gupta, M. : *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. RFC 4443 (Draft Standard). http://www.ietf.org/rfc/rfc4443.txt. Version: März 2006 (Request for Comments). – Updated by RFC 4884

[20] Cormen, T. H. ; Leiserson, C. E. ; Rivest, R. L. ; Stein, C. : *Introduction to Algorithms - Second Edition*. MIT Press, 2001. – ISBN 0–262–03293–7

[21] Crawford, M. : *Transmission of IPv6 Packets over Ethernet Networks*. RFC 2464 (Proposed Standard). http://www.ietf.org/rfc/rfc2464.txt. Version: Dez. 1998 (Request for Comments)

[22] Deering, S. ; Hinden, R. : *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460 (Draft Standard). http://www.ietf.org/rfc/rfc2460.txt. Version: Dez. 1998 (Request for Comments)

[23] Deering, S. : *Host extensions for IP multicasting*. RFC 1112 (Standard). http://www.ietf.org/rfc/rfc1112.txt. Version: Aug. 1989 (Request for Comments). – Updated by RFC 2236

[24] Devarapalli, V. ; Dupont, F. : *Mobile IPv6 Operation with IKEv2 and the Revised IPsec Architecture*. RFC 4877 (Proposed Standard). http://www.ietf.org/rfc/rfc4877.txt. Version: Apr. 2007 (Request for Comments)

[25] Duerig, J. ; Ricci, R. ; Byers, J. ; Lepreau, J. : *Automatic IP Address Assignment on Network Topologies*. http://www.cs.utah.edu/flux/papers/ipassign-ftn2006-02.pdf

[26] Eastlake 3rd, D. ; Jones, P. : *US Secure Hash Algorithm 1 (SHA1)*. RFC 3174 (Informational). http://www.ietf.org/rfc/rfc3174.txt. Version: Sept. 2001 (Request for Comments)

[27] Fenner, B. ; Handley, M. ; Holbrook, H. ; Kouvelas, I. : *Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)*. RFC 4601 (Proposed Standard). http://www.ietf.org/rfc/rfc4601.txt. Version: Aug. 2006 (Request for Comments)

[28] Fenner, B. ; Meyer, D. : *Multicast Source Discovery Protocol (MSDP)*. RFC 3618 (Experimental). http://www.ietf.org/rfc/rfc3618.txt. Version: Okt. 2003 (Request for Comments)

[29] Govindan, R. ; Tangmunarunkit, H. : Heuristics for Internet Map Discovery. In: *Proceedings IEEE INFOCOM 2000* Bd. 3. Piscataway, NJ, USA : IEEE Press, March 2000, S. 1371–1380

[30] Gupta, S. ; Shankar, V. ; Lalwani, S. : Reliable multicast MAC protocol for wireless LANs. In: *Communications, 2003. ICC '03. IEEE International Conference on* IEEE, IEEE Computer Society Press. – ISBN 0–7803–7802–4, 93–97 vol.1

[31] Haberman, B. ; Thaler, D. : *Unicast-Prefix-based IPv6 Multicast Addresses*. RFC 3306 (Proposed Standard). http://www.ietf.org/rfc/rfc3306.txt. Version: Aug. 2002 (Request for Comments). – Updated by RFCs 3956, 4489

[32] Heckmann, O. ; Piringer, M. ; Schmitt, J. ; Steinmetz, R. : On Realistic Network Topologies for Simulation. In: *MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*. New York, NY, USA : ACM Press, August 2003, S. 28–32

[33] Hinden, R. ; Deering, S. : *IP Version 6 Addressing Architecture*. RFC 4291 (Draft Standard). http://www.ietf.org/rfc/rfc4291.txt. Version: Febr. 2006 (Request for Comments)

[34] Holbrook, H. ; Cain, B. : *Source-Specific Multicast for IP*. RFC 4607 (Proposed Standard). http://www.ietf.org/rfc/rfc4607.txt. Version: Aug. 2006 (Request for Comments)

[35] Holbrook, H. ; Cain, B. ; Haberman, B. : *Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast*. RFC 4604 (Proposed Standard). http://www.ietf.org/rfc/rfc4604.txt. Version: Aug. 2006 (Request for Comments)

[36] Holzmann, G. J.: *Design and Validation of Computer Protocols*. Prentice Hall http://www.spinroot.com/spin/Doc/Book91.html. – ISBN 0–135–39925–4

[37] Holzmann, G. J.: *The Spin Model Checker: Primer and Reference Manual*. Redwood City, CA, USA : Addison Wesley Longman Publishing Co., Inc., 2004. – ISBN 0–321–22862–6

[38] Huitema, C. : *Routing in the internet - 2nd Edition*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 2000. – ISBN 0–130–22647–5

[39] IEEE: *802.16-2004 Part 16: Air Interface for Fixed Broadband Wireless Access Systems*. 3 Park Avenue, New York, NY 10016-5997, USA: IEEE Computer Society, October 2004

[40] IEEE: *802.3-2005 Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*. 3 Park Avenue, New York, NY 10016-5997, USA: IEEE Computer Society, Dezember 2005

[41] IEEE: *802.11-2007 Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. 3 Park Avenue, New York, NY 10016-5997, USA: IEEE Computer Society, June 2007

[42] International Telecommunications Union: *Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. ITU-T Recommentation X.690, July 2002

[43] Johnson, D. ; Perkins, C. ; Arkko, J. : *Mobility Support in IPv6*. RFC 3775 (Proposed Standard). http://www.ietf.org/rfc/rfc3775.txt. Version: Jun. 2004 (Request for Comments)

[44] Kim, S.-E. ; Jin, J.-S. ; Lee, S.-C. ; Lee, S.-H. : Multicast Transport on IEEE 802.16 Networks / 16ng Working Group. Version: July 2007. ftp://ftp.rfc-editor.org/in-notes/internet-drafts/draft-sekim-802-16-multicast-01.txt (01). – IETF Internet Draft – work in progress

[45] Köhnen, C. : *[omnetpp] Ethernet6 JAM*. http://www.omnetpp.org/listarchive/msg07002.php

[46] König, H. : *Protocol Engineering*. Wiesbaden, Germany : B.G. Teubner Verlag, 2003. – ISBN 3–519–00454–2

[47] Koodli, R. : *Fast Handovers for Mobile IPv6*. RFC 4068 (Experimental). http://www.ietf.org/rfc/rfc4068.txt. Version: Jul. 2005 (Request for Comments)

[48] Lai, J. ; Wu, E. ; Varga, A. ; Sekercioglu, Y. A. ; Egan, G. K.: *A Simulation Suite for Accurate Modeling of IPv6 Protocols*. http://ctieware.eng.monash.edu.au/twiki/pub/Simulation/Documentation/omnetpp2002.pdf

[49] Lee, H. ; Han, S. ; Hong, J. : Efficient Mechanism for Source Mobility in Source Specific Multicast. In: Kawahara, K. (Hrsg.) ; Chong, I. (Hrsg.): *Proceedings of ICOIN2006* Bd. 3961. Berlin, Heidelberg : Springer–Verlag, 2006 (LNCS). – (in press)

[50] Magoni, D. : *nem:* A Software for Network Topology Analysis and Modeling. In: *Proceedings of the 10th IEEE Symposium on Modeling, Analysis and Simulation of Computer & Telecomm. Systems (MASCOTS'02)*. Fort Worth, Texas, USA : IEEE Computer Society, October 2002, S. 364–371

[51] Magoni, D. ; Pansiot, J.-J. : Internet Topology Modeler Based on Map Sampling. In: *Proceedings of the 7th IEEE Symposium on Computers and Communications*. Taormina, Italy : IEEE Computer Society, July 2002, S. 1021–1027

[52] Free Software Foundation, Inc.: *GNU General Public License Version 2*. http://www.gnu.org/licenses/old-licenses/gpl-2.0.html

[53] Internet Assigned Numbers Authority: *Internet Multicast Addresses*. http://www.iana.org/assignments/ipv6-multicast-addresses

[54] Internet Assigned Numbers Authority: *Internet Protocol Version 6 Multicast Addresses*. http://www.iana.org/assignments/multicast-addresses

[55] SCAN Project: *Internet Maps. SCAN+Lucent map*. http://www.isi.edu/scan/mercator/maps.html, 2005

[56] The BRITE Project Team: *BRITE: Boston University Representative Internet Topology Generator*. http://www.cs.bu.edu/brite/

[57] The IPv6Suite Project Team: *Git-Repository: Fixed case when eager handover was on and we used first router as HA*. http://repo.or.cz/w/IPv6SuiteWithINET.git?a=commit;h=6a3faba174f6e3ce5e3e3d5c12d53b570e5ce3a9

[58] The IPv6Suite Project Team: *IPv6Suite Website*. http://ctieware.eng.monash.edu.au/twiki/bin/view/Simulation/IPv6Suite

[59] The ns-2 Project Team: *The Network Simulator - ns-2*. http://www.isi.edu/nsnam/ns/

[60] The OMNeT++ Project Team: *OMNeT++ Community Site*. http://www.omnetpp.org

[61] The OMNeT++ Project Team: *OMNeT++ Community Site - INET Framework Documentation and Tutorials*. http://www.omnetpp.org/staticpages/index.php?page=20041019113420757

[62] The OMNeT++ Project Team: *OMNeT++ Community Site - Simulation Models*. http://www.omnetpp.org/staticpages/index.php?page=20031008083109542

[63] The Trustees of Princeton University: *PlanetLab Website*. http://www.planet-lab.org

[64] The XORP Project Team: *XORP Open Source IP Router*. http://www.xorp.org

[65] McBride, M. ; Meylor, J. ; Meyer, D. : *Multicast Source Discovery Protocol (MSDP) Deployment Scenarios*. RFC 4611 (Best Current Practice). http://www.ietf.org/rfc/rfc4611.txt. Version: Aug. 2006 (Request for Comments)

[66] Moore, N. : *Optimistic Duplicate Address Detection (DAD) for IPv6*. RFC 4429 (Proposed Standard). http://www.ietf.org/rfc/rfc4429.txt. Version: Apr. 2006 (Request for Comments)

[67] Na, J. ; Cho, S. ; Kim, C. ; Lee, S. ; Kang, H. ; Koo, C. : Secure Nested Tunnels Optimization using Nested Path Information. Version: Mar 2004. http://tools.ietf.org/id/draft-varaporn-nemo-optimized-nested-nemo-01.txt (00). – IETF Internet Draft – work in progress

[68] Narten, T. ; Nordmark, E. ; Simpson, W. : *Neighbor Discovery for IP Version 6 (IPv6)*. RFC 2461 (Draft Standard). http://www.ietf.org/rfc/rfc2461.txt. Version: Dez. 1998 (Request for Comments). – Obsoleted by RFC 4861, updated by RFC 4311

[69] Nikander, P. ; Kempf, J. ; Nordmark, E. : *IPv6 Neighbor Discovery (ND) Trust Models and Threats*. RFC 3756 (Informational). http://www.ietf.org/rfc/rfc3756.txt. Version: Mai 2004 (Request for Comments)

[70] O'Neill, A. : Mobility Management and IP Multicast / IETF. 2002 (01). – Internet Draft – work in progress (expired)

[71] Pangboonyanon, V. ; Udugama, A. ; Goerg, C. ; Pittmann, F. ; Plasser, E. : Optimized Routing in Nested NEMO Networks. Version: Mar 2007. http://tools.ietf.org/id/draft-varaporn-nemo-optimized-nested-nemo-01.txt (01). – IETF Internet Draft – work in progress

[72] Park, S. D. ; Madanapalli, S. ; Rao, O. : Transmission of IPv6 Packets over 802.11/WLAN Networks. Version: July 2004. http://tools.ietf.org/html/draft-daniel-ipv6-over-wifi-01 (01). – IETF Internet Draft – work in progress

[73] Partridge, C. ; Jackson, A. : *IPv6 Router Alert Option*. RFC 2711 (Proposed Standard). http://www.ietf.org/rfc/rfc2711.txt. Version: Okt. 1999 (Request for Comments)

[74] Patel, A. ; Leung, K. ; Khalil, M. ; Akhtar, H. ; Chowdhury, K. : *Authentication Protocol for Mobile IPv6*. RFC 4285 (Informational). http://www.ietf.org/rfc/rfc4285.txt. Version: Jan. 2006 (Request for Comments)

[75] Patil, B. ; Xia, F. ; Sarikaya, B. ; Choi, J. ; Madanapalli, S. : IPv6 Over the IP Specific part of the Packet Convergence sublayer in 802.16 Networks. Version: March 2007. ftp://ftp.rfc-editor.org/in-notes/internet-drafts/draft-ietf-16ng-ipv6-over-ipv6cs-09.txt (09). – IETF Internet Draft – work in progress

[76] Perkins, C. : *IP Mobility Support for IPv4*. RFC 3344 (Proposed Standard). http://www.ietf.org/rfc/rfc3344.txt. Version: Aug. 2002 (Request for Comments). – Updated by RFC 4721

[77] Postel, J. : *Internet Protocol*. RFC 791 (Standard). http://www.ietf.org/rfc/rfc791.txt. Version: Sept. 1981 (Request for Comments). – Updated by RFC 1349

[78] Rekhter, Y. ; Li, T. ; Hares, S. : *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271 (Draft Standard). http://www.ietf.org/rfc/rfc4271.txt. Version: Jan. 2006 (Request for Comments)

[79] Romdhani, I. ; Bettahar, H. ; Bouabdallah, A. : Transparent Handover for Mobile Multicast Sources. In: *ICNICONSMCL '06: Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06)*. IEEE Computer Society. – ISBN 0–7695–2552–0, 145

[80] Romdhani, I. ; Kellil, M. ; Lach, H.-Y. ; Bouabdallah, A. ; Bettahar, H. : IP Mobile Multicast: Challenges and Solutions. In: *IEEE Comm. Surveys & Tutorials* 6 (2004), Nr. 1, S. 18–41

[81] Roscoe, A. W. ; Hoare, C. A. R. (Hrsg.) ; Bird, R. (Hrsg.): *The Theory and Practice of Concurrency*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 1997. – ISBN 0–136–74409–5

[82] Savola, P. ; Haberman, B. : *Embedding the Rendezvous Point (RP) Address in an IPv6 Multicast Address*. RFC 3956 (Proposed Standard). http://www.ietf.org/rfc/rfc3956.txt. Version: Nov. 2004 (Request for Comments)

[83] Schmidt, T. C. ; Wählisch, M. : Extending SSM to MIPv6 — Problems, Solutions and Improvements. In: *Computational Methods in Science and Technology* 11 (2005), November, Nr. 2, 147–152. http://tnc2005.terena.org/core/getfile.php?file_id=629. – Selected Papers from TERENA Networking Conference, Poznań, May 2005

[84] Schmidt, T. C. ; Wählisch, M. : Morphing Distribution Trees – On the Evolution of Multicast States under Mobility and an Adaptive Routing Scheme for Mobile SSM Sources. In: *Telecommunication Systems* 33 (2006), December, Nr. 1–3, 131–154. http://dx.doi.org/10.1007/s11235-006-9010-4

[85] Schmidt, T. C. ; Wählisch, M. : Multicast Mobility in MIPv6: Problem Statement and Brief Survey / MobOpts. Version: July 2007. ftp://ftp.rfc-editor.org/in-notes/internet-drafts/draft-irtf-mobopts-mmcastv6-ps-01.txt (01). – IRTF Internet Draft – work in progress

[86] Schmidt, T. C. ; Wählisch, M. ; Cycon, H. L. ; Palkow, M. : Scalable Mobile Multimedia Group Conferencing based on SIP initiated SSM. In: *Proc. of 4th European Conference on Universal Multiservice Networks – ECUMN'2007*. IEEE Computer Society Press, 200–209

[87] Soliman, H. ; Castelluccia, C. ; Malki, K. E. ; Bellier, L. : *Hierarchical Mobile IPv6 Mobility Management (HMIPv6)*. RFC 4140 (Experimental). http://www.ietf.org/rfc/rfc4140.txt. Version: Aug. 2005 (Request for Comments)

[88] Soliman, H. : *Mobile IPv6: Mobility in a Wireless Internet*. Redwood City, CA, USA : Addison Wesley Longman Publishing Co., Inc., 2004. – ISBN 0–201–78897–7

[89] Spät, W. : *Entwurf und Realisierung einer Verarbeitungskette für reale Internet Topologiedaten und deren Anwendung in der Strukturanalyse von Multicast Verteilbäumen, Bachelorarbeit Informatik, HAW Hamburg 2007*. http://users.informatik.haw-hamburg.de/~schmidt/thesis/waldemar_spaet.pdf

[90] Suh, Y.-J. ; Shin, H.-S. ; Kwon, D.-H. : An efficient multicast routing protocol in wireless mobile networks. In: *Wireless Networks* 7 (2001), Nr. 5, 443–453. http://dx.doi.org/10.1023/A:1016780525280. – ISSN 1022–0038

[91] Sun, M. ; Huang, L. ; Arora, A. ; Lai, T. : Reliable MAC layer multicast in IEEE 802.11 wireless networks. In: *Parallel Processing, 2002. Proceedings. International Conference on* IEEE, IEEE Computer Society Press. – ISBN 0–7695–1677–7, 527–536

[92] Tanenbaum, A. S. ; Woodhull, A. S.: *Operating Systems: Design and Implementation*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2006. – ISBN 0–131–42938–8

[93] Tang, K. ; Gerla, M. : MAC layer broadcast support in 802.11 wireless networks. In: *MILCOM 2000. 21st Century Military Communications Conference Proceedings* IEEE, IEEE Computer Society Press. – ISBN 0–7803–6521–6, 544–548 vol.1

[94] Thaler, D. : *Border Gateway Multicast Protocol (BGMP): Protocol Specification*. RFC 3913 (Informational). http://www.ietf.org/rfc/rfc3913.txt. Version: Sept. 2004 (Request for Comments)

[95] Thaler, D. : Unicast-Prefix-based IPv4 Multicast Addresses / MobOpts. Version: March 2007. ftp://ftp.rfc-editor.org/in-notes/internet-drafts/draft-ietf-mboned-ipv4-uni-based-mcast-03.txt (03). – IETF Internet Draft – work in progress

[96] Thomson, S. ; Narten, T. : *IPv6 Stateless Address Autoconfiguration*. RFC 2462 (Draft Standard). http://www.ietf.org/rfc/rfc2462.txt. Version: Dez. 1998 (Request for Comments). – Obsoleted by RFC 4862

[97] Thubert, P. ; Molteni, M. : IPv6 Reverse Routing Header and its application to Mobile Networks. Version: Feb 2007. http://tools.ietf.org/html/draft-thubert-nemo-reverse-routing-header-07 (07). – IETF Internet Draft – work in progress

[98] Van Mieghem, P. ; Hooghiemstra, G. ; Hofstad, R. van d.: On the Efficiency of Multicast. In: *IEEE/ACM Trans. Netw.* 9 (2001), Nr. 6, S. 719–732. http://dx.doi.org/http://dx.doi.org/10.1109/90.974526. – DOI http://dx.doi.org/10.1109/90.974526. – ISSN 1063–6692

[99] Varga, A. : *Parametrized Topologies for Simulation Programs*. http://www.omnetpp.org/download/docs/papers/cnds98-paramtop.pdf

[100] Vida, R. ; Costa, L. : *Multicast Listener Discovery Version 2 (MLDv2) for IPv6*. RFC 3810 (Proposed Standard). http://www.ietf.org/rfc/rfc3810.txt. Version: Jun. 2004 (Request for Comments). – Updated by RFC 4604

[101] Waitzman, D. ; Partridge, C. ; Deering, S. : *Distance Vector Multicast Routing Protocol*. RFC 1075 (Experimental). http://www.ietf.org/rfc/rfc1075.txt. Version: Nov. 1988 (Request for Comments)

[102] Wang, Y. ; Chen, W. : Supporting IP Multicast for Mobile Hosts. In: *Mobile Networks and Applications* 6 (2001), Nr. 1, 57-66. http://citeseer.ist.psu.edu/wang99supporting.html

[103] Wittmann, R. ; Zitterbart, M. : *Multicast Communication*. Morgan Kaufmann, 2000. – ISBN 1–558–60645–9

[104] Wu, J. ; Maguire, G. : *Agent Based Seamless IP Multicast Receiver Handover*. http://citeseer.ist.psu.edu/wu00agent.html. Version: 2000

[105] Xylomenos, G. ; Polyzos, G. C.: IP Multicast for Mobile Hosts. In: *IEEE Comm. Mag.* 35 (1997), Nr. 1, S. 54–58

[106] Yegin, A. E. ; Parthasarathy, M. ; Williams, C. : Mobile IPv6 Neighborhood Routing for Fast Handoff. Version: Nov 2000. http://tools.ietf.org/html/draft-yegin-mobileip-nrouting-01 (01). – IETF Internet Draft – work in progress

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

ACK ..... Acknowledgement
AH ..... Authentication Header
ANSI ..... American National Standards Institute
AP ..... Access Point
AR ..... Access Router
ARP ..... Address Resolution Protocol
AS ..... Autonomous System
ASM ..... Any Source Multicast
ASN.1 ..... Abstract Syntax Notation One
ATM ..... Asynchronous Transfer Mode
Back ..... Binding Update Acknowledgement
BC ..... Binding Cache
BFS ..... Breadth-First Search
BGMP ..... Border Gateway Multicast Protocol
BGP ..... Border Gateway Protocol
BS ..... Base Station
BSS ..... Basic Service Set
BT ..... Bidirectional Tunneling
BU ..... Binding Update
CA ..... Certificate Authority
CBT ..... Core Based Tree
CGA ..... Cryptographically Generated Address
CGHID ..... Cryptographically Generated Host Identifier
CID ..... Connection Identifier
CN ..... Correspondent Node
CoA ..... Care-of Address
CoA-ET ..... Care-of Address Expiry Timer
CoT ..... Care-of Test
CoTi ..... Care-of Test Init
CRL ..... Certificate Revocation List
CToken ..... Care-of Keygen Token
DAD ..... Duplicate Address Detection
DER ..... Distinguished Encoding Rules

DFS ..... Depth-First Search
DHCP ..... Dynamic Host Configuration Protocol
DM ..... Dense Mode
DNS ..... Domain Name Service
DoS ..... Denial of Service
DVMRP ..... Distance Vector Multicast Routing Protocol
ESP ..... Encapsulated Security Payload
ESSID ..... Extended Service Set Identifier
ET ..... Expiry Timer
FDDI ..... Fiber Distributed Data Interface
FSM ..... Finite State Machine
GNU ..... GNU is not Unix
GPL ..... GNU General Public License
GPLv2 ..... GNU General Public License Version 2
GSM ..... Global System for Mobile Communications
HA ..... Home Agent
HoA ..... Home Address
HoT ..... Home Test
HoTi ..... Home Test Init
HToken ..... Home Keygen Token
IANA ..... Internet Assigned Numbers Authority
ICMP ..... Internet Control Message Protocol
ICMPv6 ..... Internet Control Message Protocol Version 6
IGMP ..... Internet Group Management Protocol
IGMPv3 ..... Internet Group Management Protocol Version 3
IP ..... Internet Protocol
IPsec ..... Internet Protocol security
IPv4 ..... Internet Protocol Version 4
IPv6 ..... Internet Protocol Version 6
J ..... Join (State)
MA ..... Multicast Agent
MBGP ..... Border Gateway Protocol with multi-protocol extensions
MD5 ..... Message Digest Algorithm 5
MIPv6 ..... Mobile IPv6
MitM ..... Man-in-the-Middle
MLD ..... Multicast Listener Discovery
MLDv2 ..... Multicast Listener Discovery Version 2
MN ..... Mobile Node
MRIB ..... Multicast Routing Information Base
MRP ..... Mobility-aware RendezVous Point

MS ..... Mobile Source
MSA ..... Mobility Support Agent
MSDP ..... Multicast Source Discovery Protocol
nAR ..... next Access Router
nCoA ..... next Care of Address
NDP ..... Neighbor Discovery Protocol
nDR ..... next Designated Router
NED ..... NEtwork Description language (OMNeT++)
NI ..... No Info (State)
OMNeT++ ..... Objective Modular Network Testbed in C++
pAR ..... previous Access Router
pCoA ..... previous Care of Address
pDR ..... previous Designated Router
PIM ..... Protocol Independent Multicast
PIM-DM ..... Protocol Independent Multicast: Dense Mode
PIM-SM ..... Protocol Independent Multicast: Sparse Mode
PKI ..... Public Key Infrastructure
PP ..... Prune-Pending (State)
PPT ..... Prune-Pending Timer
PROMELA ..... PROcess MEta-LAnguage
REQ ..... Request
RFC ..... Request for Comments
RP ..... RendezVous Point
RPF ..... Reverse Path Forwarding
RRP ..... Return Routability Procedure
RS ..... Remote Subscription
RSA ..... Rivest, Shamir, Adleman
Sec ..... Security Parameter (siehe CGA)
SEND ..... Secure Neighbor Discovery
SHA-1 ..... Secure Hash Algorithm 1
SIP ..... Session Initiation Protocol
SM ..... Sparse Mode
SPT ..... Shortest Path Tree
SR ..... Source Registration
SS ..... Subscriber Station
SSM ..... Source Specific Multicast
TCP ..... Transmission Control Protocol
TE ..... Tree Elongation
TE-Back ..... Tree Elongation Binding Acknowledgement
TLV ..... Type Length Value

TM ..... Tree Morphing (State)
TO ..... Tree Optimization
UDP ..... User Datagram Protocol
UMTS ..... Universal Mobile Telecommunications System
VCoIP ..... Videoconferencing over Internet Protocol
VoIP ..... Voice over Internet Protocol
WiMAX ..... Worldwide Interoperability for Microwave Access
WLAN ..... Wireless Local Area Network
XML ..... Extensible Markup Language
XORP ..... Extensible Open Router Platform

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 19. Oktober 2007	
Ort, Datum	Unterschrift