Implementierung und Evaluierung des Adaptive Demand-Driven Multicast Routing Protokolls

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

von: cand. inform. Christian Koch

Betreuer: Prof. Dr. Martina Zitterbart
Dipl.-Inform. Oliver Stanze

Tag der Anmeldung: 1. Oktober 2003
Tag der Abgabe: 31. Dezember 2003

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

Karlsruhe, den 9. Dezember 2003

So eine Arbeit wird eigentlich nie fertig,
man muss sie für fertig erklären,
wenn man nach Zeit und Umständen
das Möglichste getan hat.

Goethe, „Italienische Reise“, 16. März 1787

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Verwendung von drahtlosen Endgeräten, die z.B. über WLAN mit dem Internet verbunden werden, hat in den letzten Jahren immer mehr zugenommen. Da davon auszugehen ist, dass diese Entwicklung anhalten wird, lohnt es sich, einen genaueren Blick auf diese Techniken und die Möglichkeiten zu werfen, die sich bei der Verwendung eines drahtlosen Mediums ergeben.

Die Broadcast-Eigenschaft dieses Mediums, die dazu führt, dass ein Sender zwangsläufig alle Geräte in Sendereichweite erreicht, bietet sich geradezu an, um Multicast-Szenarien zu realisieren und so Daten mit relativ wenig Aufwand an mehrere Empfänger auszuliefern. Verwendet man ein mobiles Ad-hoc-Netzwerk (auch „MANET“) kann dafür jedoch nicht auf eine vorhandene Infrastruktur zurück gegriffen werden. Dies ist ein wesentlicher Unterschied zu herkömmlichen Mobilfunktechniken die bei Handys verwendet werden.

Der Verzicht auf eine bestehende Infrastruktur ist eine Eigenschaft, die z. B. bei militärischen Einsatzszenarien von großer Bedeutung ist, da so eine vollkommen autonome Kommunikation ermöglicht wird. Aber auch im zivilen Bereich kann dies nützlich sein, beispielsweise in einem Tagungsraum ohne die benötigte technische Ausstattung.

Durch die Anforderung auch ohne Infrastruktur Geräte außerhalb der Reichweite des Senders erreichen zu wollen ergeben sich besondere Anforderungen an ein Routing Protokoll. Es müssen - im Allgemeinen ohne Zusatzinformationen wie etwa GPS-Daten - bestimmte Geräte ausgewählt werden, die die Daten weiterleiten. Die Mobilität der Geräte macht es notwendig, diese Entscheidung immer wieder neu zu treffen.

1.1 Zielsetzung der Arbeit

Ziel dieser Studienarbeit ist es, das als Internet-Draft vorliegende „Adaptive Demand-Driven Multicast Routing“-Protokoll (ADMR) [Jet01b] für mobile Ad-hoc-Netze im Netzwerksimulator GloMoSim zu implementieren und zu evaluieren. Zur Evaluation wurden diverse Szenarien mit unterschiedlicher Mobilität,

Senderaten, Multicast-Gruppengrößen usw. simuliert. Dabei wurde ADMR dem „On-Demand Multicast Routing Protocol“ (ODMRP, [LSG00]) gegenübergestellt. Aus diesem Vergleich sowie den Messungen in anderen Arbeiten (z. B. [Jet01a], [LSHGB]) soll so eine Einschätzung der Performance von ADMR abgeleitet werden können.

1.2 Gliederung der Arbeit

Zunächst erfolgt im zweiten Kapitel eine Beschreibung der grundlegenden Begrifflichkeiten. Die Funktionsweise des Simulators GloMoSim sowie die Arbeitsweise von ADMR werden erläutert. Außerdem wird ODMRP vorgestellt.

Das dritte Kapitel widmet sich der Implementierung von ADMR in GloMoSim. An GloMoSim waren einige Änderungen notwendig, auf die dort eingegangen wird. Außerdem wird beschrieben, wie die Implementierung von ADMR aufgebaut und wie der Ablauf des Programms ist. Es wird darauf eingegangen, welche Besonderheiten ADMR aufweist und welche Statistikdaten von dieser Implementierung erzeugt werden.

Die Simulationen diverser Szenarien und deren Ergebnisse werden im vierten Kapitel präsentiert. Die Messwerte werden vorgestellt und interpretiert.

Als Abschluss werden im fünften Kapitel die Ergebnisse dieser Arbeit zusammengefasst. Es folgt ein Ausblick auf mögliche Verbesserungen.

2 Grundlagen

In diesem Teil werden zunächst die für das Verständnis der vorliegenden Arbeit wichtigen Begrifflichkeiten, Voraussetzungen sowie die grundlegenden Funktionsweisen der beteiligten Protokolle erläutert. Da es sich bei ADMR um ein Multicast-Protokoll für mobile Ad-hoc-Netze handelt, werden zunächst die Begriffe Multicast und MANET erklärt, gefolgt von einer kurzen Beschreibung von GloMoSim, dem verwendeten Netzwerksimulator. Darauf folgt eine Beschreibung des Multicast Routing Protokolls ADMR.

Da ADMR in Simulationen mit ODMRP, einem weiteren Multicast Routing Protokoll, verglichen wird, endet dieses Kapitel mit einer Funktionsbeschreibung von ODMRP.

2.1 Multicast

Grundsätzlich wird zwischen vier verschiedenen Kommunikationsformen unterschieden: Unicast, Multicast, Anycast und Broadcast. Im Fall von Unicast werden die Daten von einem Sender an genau einen Empfänger geschickt. Anycast-Daten werden an einen beliebigen Empfänger einer Gruppe von möglichen Empfängern ausgeliefert, und Broadcast-Daten werden an alle erreichbaren Geräte geschickt.

Bei Multicast dagegen werden die Daten von einem Sender an mehrere Empfänger geschickt. Eine typische Anwendung von Multicast ist z.B. das Senden eines Videobildes in einer Videokonferenz. Das von einer Kamera aufgenommene Bild soll nicht nur an einen, sondern an mehrere Empfänger geschickt werden. Dies ließe sich zwar auch mit Unicast-Verbindungen zu jedem einzelnen Empfänger realisieren, würde die Netzwerkbelastung jedoch deutlich erhöhen. Bei Multicast-Routing reicht es, die Daten für nah beieinander liegende Empfänger erst kurz vor Erreichen der Empfänger zu duplizieren, während dies bei Unicastverkehr bereits beim Sender geschehen müsste.

Beim Versenden von Multicast-Daten werden die Pakete nicht direkt an die Empfänger adressiert, sondern an eine so genannte Multicast-Gruppe ver-

schickt. Empfänger können dieser Gruppe nun beitreten und erhalten daraufhin die an diese Gruppe verschickten Daten. Die Daten vom Sender an die gerade interessierten (der Gruppe angehörenden) Empfänger zu verteilen ist die Aufgabe des Routing Protokolls. Im Internet kommen dafür Protokolle wie DVMRP [WaPa88], MOSPF [Moy94] oder PIM [EsFa98] zum Einsatz. Da ein MANET aber beispielsweise durch die Broadcast-Eigenschaft des verwendeten Mediums andere Voraussetzungen als ein leitungsgebundenes Netz hat, ist deren Einsatz dort nicht möglich. Ein auf die Gegebenheiten in einem MANET abgestimmtes Multicast Routing Protokoll ist z. B. ADMR. Da mehrere Sender an dieselbe Gruppe schicken können ist es möglich, dass ein Empfänger nicht alle an eine Gruppe verschickten Daten erhalten möchte. Er könnte sich beispielsweise nur für die Daten von einem bestimmten Sender interessieren. In diesem Fall spricht man von einer senderspezifischen Multicast-Gruppe.

2.2 Mobile Ad-hoc-Netze

Ein mobiles Ad-hoc-Netz ist ein Zusammenschluss von mobilen Geräten wie Handys, PDAs oder Notebooks zu einem selbst-organisierenden Netz, für das keine zusätzliche Infrastruktur in Form von Access-Points o. ä. benötigt wird. Jedes der beteiligten Geräte kann dabei als Sender und/oder Empfänger von Daten fungieren.

Besonders wichtig ist jedoch, dass jedes Gerät auch die Funktion eines Routers übernimmt und Daten weiterleitet. Dadurch können auch Geräte miteinander kommunizieren, deren Entfernung für eine direkte Funkverbindung zu groß ist.

Die Entscheidung an welche Geräte die Daten weitergeleitet werden obliegt dabei dem Routing Protokoll. Auf Grund der Mobilität in einem MANET muss diese Entscheidung immer wieder neu getroffen werden. Bleibt bei einer Änderung noch ein Teil des vorherigen Routingstatus erhalten, kann es durch die Weiterleitung an mehrere (Zwischen-)Knoten durchaus Redundanz geben. Die Daten werden u. U. über mehrere Pfade an die Empfänger verteilt. Um die Belastung des Netzes zu minimieren sollte diese Redundanz nicht zu hoch sein, jedoch groß genug um eine zuverlässige Weiterleitung der Daten zu gewährleisten. Ein Fluten des gesamten Netzes mit allen Daten würde - abgesehen von den zwangsläufig auftretenden Kollisionen - zwar alle Geräte erreichen, jedoch zu einer unverhältnismäßigen Belastung durch letztlich redundante Pakete führen.

2.3 Der Simulator GloMoSim

GloMoSim steht für „Global Mobile Information System Simulator“ und ist ein von der Universität von Kalifornien in Los Angeles entwickelter ereignisbasierter Netzwerksimulator. Für den universitären Einsatz ist er kostenlos verfügbar [gms].

GloMoSim benutzt ein Schichtenmodell, das sich grob an den OSI-Schichten orientiert, die Schichten fünf bis sieben jedoch als Anwendungsschicht zusammenfasst. Die einzelnen Schichten in GloMoSim sind (vgl. [BTATBG]):

- Bitübertragung, physikalische Schicht (Signalausbreitung)
- Sicherungsschicht (MAC-Schicht)
- Vermittlungsschicht (IP und Routing)
- Transportschicht
- Anwendungsschicht

Für jeder der Schichten stehen verschiedene Modelle zur Auswahl. So läßt sich in der Anwendungsschicht z.B. FTP, HTTP, telnet o. ä. simulieren. Durch diesen modularen Aufbau lassen sich einzelne Komponenten recht einfach ersetzen. Für diese Arbeit konnte somit ADMR als ein neues Routing Protokoll implementiert und eingesetzt werden.

Da GloMoSim bei der Ausführung möglichst stark parallelisiert ablaufen soll, wurde es in PARSEC (PARallel Simulation Enivronment for Complex systems) [Par] geschrieben. Diese Sprache orientiert sich stark an C und weist nur einige Erweiterungen auf. Das Programmieren neuer Protokolle erfolgt daher ausschließlich in C und es ist nur an wenigen Stellen auf PARSEC-Bestandteile zurückzugreifen. Lediglich für das Verändern des Kernbestandteile von GloMoSim sind detailliertere PARSEC-Kenntnisse erforderlich.

Die Benutzung von GloMoSim erfolgt durch die Einstellung der gewünschten Parameter in einer Konfigurationsdatei, die GloMoSim als Argument übergeben wird. So läßt sich beispielsweise die Dauer der Simulation, die Anzahl der Knoten, aber auch das zu verwendende Routing Protokoll auswählen. Andere Einstellungen, wie etwa der Datenverkehr der Anwendungsschicht, werden teilweise in gesonderten Dateien vorgenommen.

Zur Auswertung der Ergebnisse ist GloMoSim in der Lage am Ende der Simulation für jede Schicht Statistiken auszugeben.

2.4 Protokollbeschreibung ADMR

Die aktuelle Spezifikation des „Adaptive Demand-Driven Multicast Routing"-Protokolls (ADMR) liegt als IETF-Internet-Draft [Jet01b] vor. Eine Beschreibung der Funktionsweise durch die Entwickler gibt es auch in [Jet01a]. Es ist ein Multicast-Routingprotokoll für mobile Ad-hoc-Netze.

Die Autoren bezeichnen es in dem Draft, d.h. in der Protokollbeschreibung, als ein Protokoll für den Einsatz in relativ kleinen Netzen, in denen ein Paket zwischen Sender und Empfänger nicht mehr als maximal fünf bis zehn andere Knoten passieren muss. Laut [Jet01a] ist ein Vorteil von ADMR im Vergleich zu anderen derartigen Routingprotokollen der sehr geringe Anteil von Overhead ohne speziellen Sendewunsch eines Knotens. Es ist im Wesentlichen „demand-driven", versucht also möglichst wenige noch nicht oder nicht mehr gebrauchte Informationen vorzuhalten und nur solange Netzwerkaktivität zu verursachen, wie wirklich Daten zu verschicken sind.

ADMR wurde als völlig eigenständiges Multicast Routing Protokoll entwickelt, dessen Funktionsweise nicht auf einem vorhandenen Unicast-Routingprotokoll aufbaut. Dies verbessert zwar die Modularität und auch die Portierbarkeit, da kein spezielles Unicast-Routingprotokoll vorausgesetzt werden muss, kann dafür allerdings zu Nachteilen hinsichtlich der Effizienz führen. Unter Umständen wäre es sinnvoll, Informationen zwischen Multicast- und Unicast-Routingprotokoll auszutauschen und so die Funktionsweise von Beiden zu verbessern. Die Entwickler wollen laut [Jet01a] diesbezügliche Untersuchungen durchführen.

Hat ein Sender S Multicast-Daten zu versenden, kann er im Allgemeinen nicht alle Empfänger direkt erreichen, da zumindest einige außerhalb seiner Funkreichweite liegen. Für eine erfolgreiche Kommunikation zwischen Sender und Empfängern sind somit andere Knoten notwendig, die die Daten weiterleiten und so die Strecke zwischen Sender und Empfängern überbrücken. Diese Knoten zu bestimmen und bei Bedarf anzupassen ist Aufgabe des Routing Protokolls.

Bei ADMR besteht eine Verbindung zwischen einem Multicast-Sender S und Empfängern E aus Knoten W, die die Daten weiterleiten. Dieses Konzept veranschaulicht Abbildung 1 für einen Sender und vier Empfänger. Die Knoten W bilden dabei eine Gruppe, die im Folgenden Weiterleitungsgruppe genannt wird. Bei mehreren Empfängern kann so ein Baum entstehen. Ein Knoten kann dabei sowohl Empfänger, als auch Weiterleiter sein. Die Empfänger sind also nicht notwendigerweise die Blätter des Baums, sondern können ihrerseits die Daten an andere Empfänger weiterleiten. Bei dem mit „E/W“ markierten Knoten ist dies der Fall.

Die Entwickler von ADMR legen Wert darauf, dass es sich bei den Knoten W lediglich um eine Gruppe von Knoten und nicht um einen Baum handelt [Jet01b], es also keine expliziten Eltern/Kind-Beziehungen zwischen den Knoten gibt. Dies ist auch insofern richtig, als dass es durchaus möglich ist, dass beispielsweise die zwei Weiterleitungsknoten W_1 und W_2 der Abbildung 1 ihre Positionen im Baum vertauschen, ohne das dies Aktionen des Routing Protokolls erfordern würde (vorausgesetzt die Entfernungen zwischen den Knoten erlauben auch während des Vertauschens eine Weiterleitung aller Daten). Problematisch wird diese Auffassung (wie die Autoren selbst einräumen), wenn die Verbindung zwischen Sender und mindestens einem Empfänger abbricht. Dies erfordert eine Reparatur der Verbindung, bei der durchaus von einem Baum und nicht lediglich von einer Gruppe von Knoten ausgegangen wird. Auf diesen Vorgang wird später genauer eingegangen.

Die Verwaltung der Weiterleitungsgruppe und somit die Aktivitäten von ADMR lassen sich grob in vier Szenarien unterteilen:

Sender-initiierter Gruppenbeitritt: Ein Sender beginnt an eine Multicast-Gruppe zu senden. Es muss überprüft werden, ob im Netzwerk Empfänger vorhanden

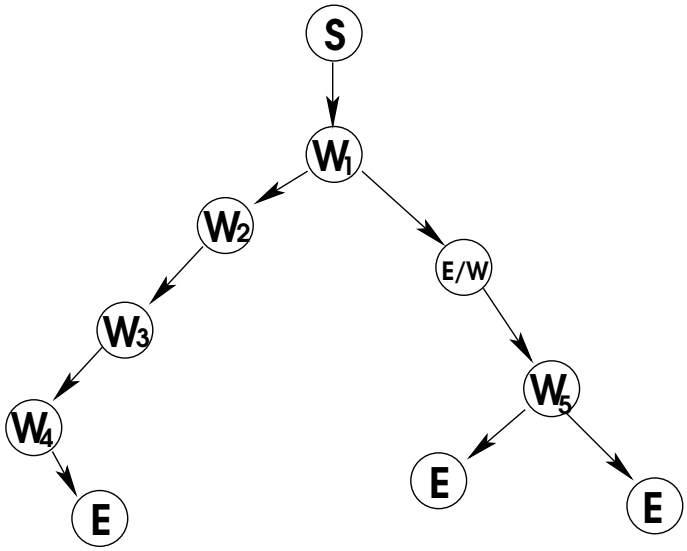

Abbildung 1: Multicast Paketweiterleitung

sind, die dieser Multicast-Gruppe angehören. In dem Fall muss eine Verbindung zu den Empfängern aufgebaut, also die Weiterleitungsgruppe bestimmt werden.

Empfänger-initiierter Gruppenbeitritt: Ein Knoten wird Empfänger einer Multicast-Gruppe. Es muss überprüft werden, ob ein passender Sender im Netzwerk vorhanden ist und u. U. muss eine Verbindung zu diesem aufgebaut werden.

Reparatur von Verbindungsabbrüchen: Die Verbindung zwischen Sender und Empfänger bricht ab. Dies geschieht im Wesentlichen durch die Bewegung der einzelnen Knoten, wodurch die Entfernungen so groß werden können, dass eine Datenübertragung nicht mehr möglich ist. In diesem Fall muss die Verbindung repariert werden.

Abbau des Routingstatus: Ein Knoten wird zur Weiterleitung von Daten nicht mehr benötigt und verlässt die Weiterleitungsgruppe oder ein Sender sendet keine weiteren Daten mehr und alle beteiligten Knoten verlassen die Weiterleitungsgruppe.

Der Ablauf dieser Szenarien wird im Folgenden erläutert. Zum besseren Verständnis wird zunächst jedoch der Aufbau des ADMR-eigenen Headers beschrieben.

2.4.1 ADMR-Header

ADMR fügt in allen Paketen zwischen dem IP-Header und dem Header der Transportschicht, wie in Abbildung 2 dargestellt, noch einen eigenen Header ein. Dieser wird benötigt, um die ADMR-spezifischen Informationen zu übertragen. Hier werden z. B. Kontrollpakete wie ein *Receiver Join*, eine *Multicast Solicitation* oder ein *Reconnect* gespeichert.

Abbildung 2: die Header eines Pakets bei Benutzung von ADMR

Problematisch ist, dass für die korrekte Funktionsweise von ADMR im IP-Header das „*Don't Fragment*"-Bit gesetzt werden muss. Die Pakete dürfen also nicht in kleinere Pakete aufgeteilt werden. Inwiefern dies zu Problemen im Zusammenhang mit den beschränkten Kapazitäten in mobilen Netzen führt, wurde in der vorliegenden Arbeit nicht untersucht.

Der ADMR-Header besteht zunächst aus einem 4 Byte großen Abschnitt, in dem die Gesamtgröße des ADMR-Headers sowie der Typ des darauf folgenden Headers kodiert sind. Danach folgen die von ADMR benutzten Optionen. Da deren Größe insgesamt ein Vielfaches von 32 Bit sein muss, gibt es auch Optionen, die lediglich zum Auffüllen dienen (Pad1 und PadN) und keine Informationen beinhalten. Jede Option beginnt mit einem 8 Bit großen Feld zur Identifikation der Option, gefolgt von weiteren 8 Bit mit deren Gesamtgröße (außer Pad1). Eine Übersicht über alle Optionen liefert Tabelle 1.

Nummer (Option Type)	**Bezeichnung**
0	Pad1
1	PadN
2	Source Information
3	Receiver Join
4	Multicast Solicitation
5	Repair Notification
6	Reconnect
7	Reconnect Reply
8	Multicast Group Address
9	Multicast Sender Address

Tabelle 1: ADMR-Optionen

2.4.2 Sender-initiierter Gruppenbeitritt

Wenn auf einem Knoten eine Applikation Multicast-Daten verschickt, ist es Aufgabe von ADMR zunächst herauszufinden, ob passende Empfänger vorhanden sind. Bis dieser Vorgang abgeschlossen ist werden von ADMR zunächst alle Pakete gepuffert, bis die Weiterleitungsgruppe aufgebaut ist, bzw. festgestellt wird, dass keine Empfänger vorhanden sind. Das erste Paket wird allerdings nicht gepuffert, sondern an alle Knoten verschickt, also im gesamten Netzwerk geflutet. Das Fluten über das gesamte Netzwerk wird im Folgenden in Übereinstimmung mit [Jet01a] und [Jet01b] als *Network-Flood* bezeichnet. Das Puffern der Pakete verhindert, dass zu viele Pakete als ineffizienter *Network-Flood* gesendet werden müssen. Sollte sich herausstellen, dass keine passenden Empfänger im Netz vorhanden sind, verschickt ADMR auch keine Pakete mehr, sondern puffert sie für eine gewisse Zeit im *Send Buffer*.

Erhält nun ein passender Empfänger das Paket, antwortet er dem Sender mit einem *Receiver Join*. Dies ist ein spezielles Paket, das dem Sender signalisiert, dass Empfänger für diese Multicast-Gruppe vorhanden sind. Diese Antwort ist direkt an die Adresse des Senders adressiert, also eigentlich eine Unicast-Nachricht, wird jedoch auch von ADMR und nicht von einem eventuell vorhandenen Unicast Routing Protokoll weitergeleitet. ADMR ist in der Lage dieses Paket weiterzuleiten, da jeder Knoten in seiner *Node Table* die MAC-Adresse des Knotens gespeichert hat, von dem das als *Network-Flood* gesendete Paket empfangen wurde. Das Paket kann so bis zum Sender weitergeleitet werden. Da der *Receiver Join* nur mit geringer zeitlicher Verzögerung verschickt wird ist das Risiko, dass sich die Verhältnisse im Netzwerk so stark geändert haben, dass das Weiterleiten bis zum Sender fehlschlägt, gering.

Das *Receiver Join*-Paket enthält die Sequenznummer des zuvor als *Network-Flood* verschickten Pakets, welches es beantwortet. Sollten mehrere Empfänger vorhanden sein, werden somit mehrere *Receiver Join*-Pakete mit identischen Sequenznummern verschickt. Da für den Sender nur der Erhalt von einem *Receiver Join* notwendig ist, um zu wissen, dass Empfänger vorhanden sind, hat jeder Knoten die Möglichkeit, nur eine begrenzte Anzahl von *Receiver Join*-Paketen (in Abbildung 3 sind dies zwei) mit identischer Sequenznummer weiterzuleiten. Dies ist vor allem bei einer großen Zahl von Empfängern sinnvoll, um nicht zu viele unnötige Pakete zu verschicken.

Jeder Knoten der ein *Receiver Join*-Paket weiterleitet gehört dadurch zur Weiterleitungsgruppe und muss einen entsprechenden Eintrag in seiner *Membership Table* vornehmen. Ein *Receiver Join*-Paket besitzt damit also zwei Funktionen:

- Es signalisiert dem Sender, dass Empfänger vorhanden sind.
- Es baut die Weiterleitungsgruppe auf.

Die nun folgenden Pakete des Senders sowie die im *Send Buffer* zwischengespeicherten Pakete werden nicht mehr als *Network-Flood* sondern als so genannter

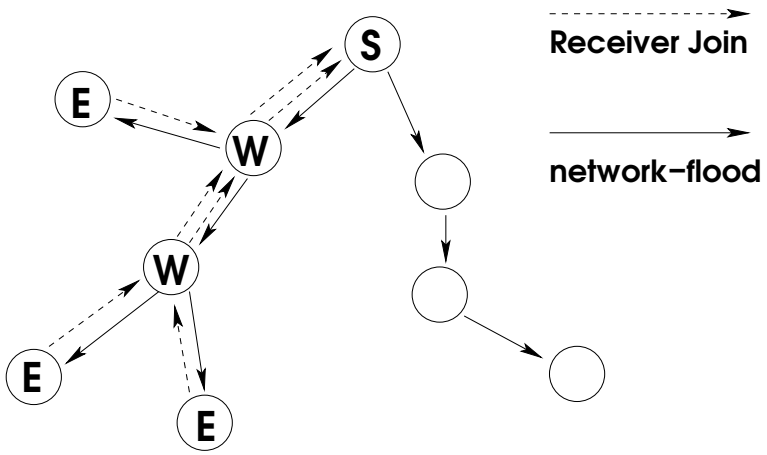

Abbildung 3: *Network-Flood* und *Receiver Joins*

Tree-Flood verschickt. Dies bedeutet, dass sie nicht mehr von allen Knoten sondern nur noch von Mitgliedern der Weiterleitungsgruppe weitergeleitet werden.

Es ist jedem Sender freigestellt periodisch einige seiner weiteren Multicast-Datenpakete als *Network-Flood* zu verschicken. Falls einige Empfänger auf Grund schlechter Übertragungsverhältnisse das erste *Network-Flood*-Paket nicht erhalten haben, erhalten sie auf diesem Wege die Möglichkeit dennoch von der Existenz des Senders zu erfahren.

Hat ein Empfänger ein *Receiver Join*-Paket gesendet, dann erwartet er demnächst ein Datenpaket vom Sender. Bleibt dies aus, z. B. auf Grund von ungünstigen Verhältnissen im Netzwerk, so sendet er das *Receiver Join*-Paket erneut. Sollte er daraufhin immer noch kein Paket empfangen, so verschickt er ein *Multicast Solicitation* Paket. Der Zweck und die Benutzung dieses Pakets wird im nächsten Abschnitt beschrieben.

2.4.3 Empfänger-initiierter Gruppenbeitritt

Tritt ein Knoten einer Multicast-Gruppe bei, so muss ADMR herausfinden, ob hierfür ein passender Sender vorhanden ist. Ähnlich wie beim Sender-initiierten Gruppenbeitritt geschieht dies durch ein Paket, welches als *Network-Flood* verschickt wird.

Der Empfänger verschickt hierzu eine *Multicast Solicitation*, in der die Adresse der betreffenden Gruppe und u. U. die Adresse des Senders gespeichert wird. Bei nicht-senderspezifischen Multicast-Gruppen leiten alle Knoten diese Nachricht weiter. Handelt es sich dagegen um eine *Multicast Solicitation* für eine senderspezifische Gruppe, so wird diese von Knoten, die bereits der passenden Weiterleitungsgruppe angehören, nicht weiter als *Network-Flood* verschickt. Das Paket erhält eine Unicast-Adresse und wird direkt zum Sender geschickt. Dadurch soll eine unnötige Belastung des Netzwerks vermieden werden. Da jedoch

alle anderen Knoten das Paket weiter als *Network-Flood* verschicken ist es fraglich, ob dieser Mechanismus tatsächlich nennenswert zu einer Entlastung des Netzes führt.

Ein Sender der nun eine *Multicast Solicitation* erhält, hat zwei Möglichkeiten auf diese zu antworten. Falls er ohnehin demnächst ein Paket als *Network-Flood* verschicken würde, kann er dieses vorziehen und so den Empfänger von seiner Existenz unterrichten. Ist dies nicht der Fall, so antwortet er mit einem *Unicast Keep-Alive*. Ein *Unicast Keep-Alive* wird auf dem umgekehrten Weg der *Multicast Solicitation* als Unicast bis zum Empfänger weitergeleitet. Wie im vorigen Abschnitt (beim *Receiver Join*) beschrieben, ist dies auf Grund der in der *Node Table* gespeicherten Informationen möglich.

Erhält ein Empfänger eine der beiden möglichen Antworten auf seine *Multicast Solicitation*, so beantwortet er diese mit einem *Receiver Join*. Dieses Vorgehen ist analog zum Verhalten beim Sender-initiierten Gruppenbeitritt. Der Empfänger signalisiert dem Sender seine Existenz und gleichzeitig wird die Weiterleitungsgruppe aufgebaut. Den Sender von der Existenz des Empfängers zu unterrichten ist insofern wichtig, als es möglich wäre, dass dies der erste und einzige Empfänger für diese Gruppe ist und ein Sender nur Daten verschickt, wenn ihm die Existenz von mindestens einem Empfänger bekannt ist.

Wie beim Sender-initiierten Gruppenbeitritt existieren auch beim Empfänger-initiierten Gruppenbeitritt Mechanismen die verhindern sollen, dass kurzfristige Störungen im Netzwerk zum Fehlschlagen der Prozedur führen. So wird auch hier ganz analog zum vorherigen Fall ein *Receiver Join* erneut gesendet, falls kein Datenpaket erhalten wurde. Außerdem wiederholt ein Sender das Verschicken des *Unicast Keep-Alives*, falls er keinen *Receiver Join* erhält. Erhält er auch dann noch kein *Receiver Join* Paket, so verschickt er sein nächstes Datenpaket als *Network-Flood*, um so den Empfänger doch noch zu erreichen.

Das erneute Verschicken des *Receiver Join* Pakets kann u. U. im Zusammenhang mit dem Mechanismus, nur eine gewisse Anzahl von *Receiver Join*s weiterzuleiten, zu Problemen führen. So könnte es passieren, dass dieses erneut verschickte *Receiver Join* Paket von anderen Knoten gar nicht weitergeleitet wird, da diese bereits entsprechend viele *Receiver Join*s mit dieser Sequenznummer weitergeleitet haben. Es kann somit seinen Zweck, eine Verbindung zum Sender herzustellen, nicht erfüllen.

2.4.4 Reparatur von Verbindungsabbrüchen

Beim Versenden von Daten versucht das Routing Protokoll beim Sender eine Annahme darüber zu treffen, in welchen Abständen die Applikation neue Daten versendet und so mit weiteren Paketen zu rechnen ist. Diese Information wird im ADMR-Header von jedem verschickten Paket gespeichert, um so die Weiterleiter und Empfänger davon zu unterrichten.

Kommt es nun z. B. durch die Bewegungen der Knoten zu einem Linkabbruch, so kann jeder Knoten merken, dass ihn ein eigentlich erwartetes Paket nicht

erreicht hat. Im Beispiel in Abbildung 4 wird zunächst noch (a) ein Multicast-Paket erfolgreich an alle Empfänger verschickt. Danach (b) bricht die Verbindung zwischen den Weiterleitungsknoten A und B auf Grund einer zu großen Entfernung ab. Knoten B bemerkt dies, da er das auf den Verbindungsabbruch folgende Paket von A nicht erhält. Selbstverständlich merken dies auch der Knoten C sowie die Empfänger. Die Funktionsweise von ADMR verbessert sich jedoch, wenn diese Knoten, wie auch in [Jet01b] vorgeschlagen, ihre Reparaturen etwas später als B einleiten. Es sollte für das Anlaufen der Reparatur folglich nicht nur die erwartete Zeit zwischen den Paketen sondern auch die Entfernung zum Sender berücksichtigt werden.

Knoten B beginnt also mit seiner Reparatur und verschickt zunächst (c) eine *Repair Notification* als *Tree-Flood*. Auf Grund des Verbindungsabbruchs zwischen A und B erreicht dieses Paket nur C und die beiden Empfänger. Diese merken dadurch, dass sie nicht direkt an der Störung beteiligt sind und ein anderer Knoten die Reparatur vornimmt. Sie werden also zunächst keine eigene Reparatur vornehmen. Eine *Repair Notification* sorgt somit dafür, dass immer nur eine Reparatur gleichzeitig vorgenommen wird.

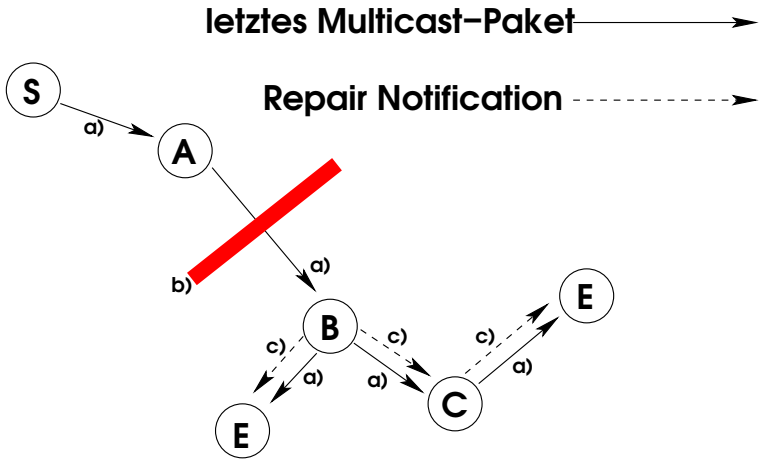

Abbildung 4: Linkabbruch

Die *Repair Notification* hat jedoch auch noch einen anderen Zweck und an dieser Stelle kann die von den ADMR-Entwicklern geforderte Vorstellung von einer Weiterleitungs*gruppe* und nicht von einem Baum nicht mehr aufrecht erhalten werden. So hat die *Repair Notification* explizit ein Feld „*Parent Address*“, in dem die Adresse des Elternknotens gespeichert wird. Dies ist der Knoten, von dem die vorherigen Multicast-Pakete empfangen wurden. Im obigen Beispiel wird Knoten B dieses Feld also auf die Adresse von Knoten A setzen.

Sollte Knoten B nun fälschlicherweise eine Reparatur eingeleitet haben bedeutet dies, dass auch Knoten A die von Knoten B verschickte *Repair Notification* erhält. Knoten A stellt nun fest, dass es der im Feld *Parent Address* gespeicherte Knoten ist und bemerkt so den „Fehlalarm“. Daraufhin verschickt Knoten A

selbst eine *Repair Notification*. So ist Knoten B der Meinung ein anderer Knoten würde sich um den (vermeintlichen) Fehler kümmern und verfolgt seine eigene Fehlerbehebung nicht weiter.

Erhält Knoten B nach einer gewissen Zeit keine andere *Repair Notification* setzt er seine Fehlerbehebung fort. Er leitet eine lokale Reparatur ein und versendet dazu ein *Reconnect*-Paket. Das *Reconnect*-Paket wird als *Network-Flood* verschickt, der jedoch nicht unbegrenzt häufig weitergeleitet werden darf. ADMR bedient sich dazu IP und setzt dort das TTL-Feld (time-to-live) z. B. auf 3. Dadurch wird die Reparatur auf einen Teil des Netzes beschränkt. Da ADMR aber ohnehin nur für relativ kleine Netze gedacht ist, kann sich schon eine lokale Reparatur auf große Teile des Netzes beziehen. Abbildung 5 zeigt den Ablauf einer lokalen Reparatur.

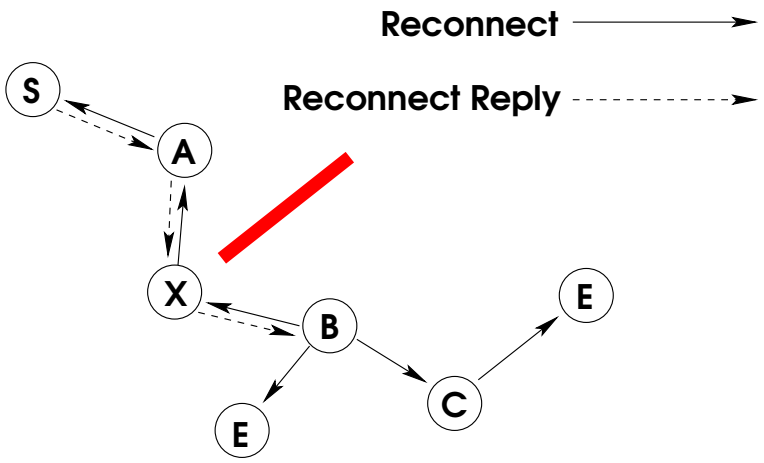

Abbildung 5: lokale Reparatur nach dem Linkabbruch zwischen Knoten A und B von Abbildung 4 - *Reconnect* und *Reconnect Reply*

Erreicht das *Reconnect* Paket einen Knoten, der noch mit dem Sender verbunden ist, so schickt dieser ihn als Unicast weiter bis zum Sender. Dies funktioniert wie schon in den vorigen Fällen an Hand der Informationen aus der *Node Table*. Im Beispiel in Abbildung 5 ist dies Knoten A. Der Knoten muss aber nicht wie dort eine direkte Verbindung zum Sender haben.

Der Sender antwortet auf das *Reconnect*-Paket mit einem *Reconnect Reply*-Paket. Dieses wiederum als Unicast verschickte Paket ist an den Knoten, der die Reparatur ausgelöst hat, adressiert. Die zum Weiterleiten benötigten Informationen in der *Node Table* wurden zuvor durch das *Reconnect* Paket bestimmt. Alle Knoten, die dieses Paket weiterleiten, gehören dadurch automatisch zur Weiterleitungsgruppe. Im Beispiel wurde die abgebrochene Verbindung zwischen A und B also durch Hinzunahme des Knotens X repariert.

Erhält ein Empfänger auch nach dem (vermuteten) Abschluss dieser lokalen Reparatur keine Daten, so vermutet er, dass sie fehlgeschlagen ist und unternimmt

eine globale Reparatur. Eine globale Reparatur ist identisch mit dem Empfängerinitiierten Gruppenbeitritt. Der Empfängerknoten verschickt eine *Multicast Solicitation*, erwartet z. B. ein *Unicast Keep-Alive* und antwortet darauf mit einem *Receiver Join*. Durch diesen Vorgang wird eine völlig neue Verbindung zum Sender gesucht, die bei einer lokalen Reparatur womöglich unerkannt geblieben wäre.

Zusätzlich können die Empfänger einen Sender über ein spezielles Flag in einem *Receiver Join*-Paket darüber informieren, dass sie sehr häufig eine globale Reparatur durchführen mussten und die Mobilität daher offensichtlich besonders hoch ist. Dann kann der Sender dazu übergehen, die Pakete für eine gewisse Zeit nicht mehr als *Tree-Flood* sondern als *Network-Flood* zu verschicken.

2.4.5 Abbau des Routingstatus

Auf Grund der relativ begrenzten Kapazitäten in einem MANET ist es wichtig, dass möglichst wenig überflüssige Pakete verschickt werden. Die Mitglieder der Weiterleitungsgruppe müssen also möglichst gut erkennen können, ob sie noch benötigt werden und weiterhin Pakete weiterleiten müssen. Es gibt zwei Fälle, in denen das Weiterleiten von Paketen nicht mehr notwendig ist:

- Ein Empfänger möchte keine weiteren Daten mehr empfangen bzw. ein Teil der Weiterleitungsgruppe ist zur erfolgreichen Vermittlung nicht mehr nötig.
- Ein Sender verschickt keine Daten mehr und somit wird die Weiterleitungsgruppe nicht mehr benötigt.

Das Verfahren für den Fall, dass ein Empfänger keine weiteren Pakete mehr empfangen möchte und dadurch, oder auf Grund der Bewegungen der Knoten, ein Teil der Weiterleitungsgruppe überflüssig geworden ist, baut auf der Broadcast-Eigenschaft des Mediums auf, durch die sich ein Signal in alle Richtungen ausbreitet. Betrachtet man beispielsweise Abbildung 5, so sieht man, dass ein von Knoten X an Knoten B weitergeleitetes Paket gleichzeitig auch von Knoten A empfangen wird. Knoten A erkennt dieses zwar als Duplikat, weiß somit aber, dass Knoten X die Pakete weiterleitet und offensichtlich weiterhin Pakete von Knoten A erhalten möchte. Aus diesem Grund verschickt auch jeder Empfänger ein von ihm empfangenes Paket nochmal, um den Weiterleitern zu signalisieren, dass die Pakete nach wie vor an diesen Empfänger weiterzuleiten sind. Diese Pakete werden im Folgenden als *Ack*-Pakete (acknowledgement) bezeichnet. Möchte ein Empfänger keine Daten mehr empfangen, so wird er erhaltene Pakete auch nicht mehr erneut verschicken. Um von der in einem MANET recht knappen Übertragungskapazität möglichst wenig für die *Ack*-Pakete zu beanspruchen, werden diese ohne die Nutzdaten verschickt.

Merkt nun ein Weiterleiter, dass eine gewisse Zahl aufeinanderfolgender Pakete nicht mehr von anderen Knoten erneut verschickt wurde, so geht er davon aus, dass er für die Weiterleitungsgruppe nicht mehr nötig ist. Er hört also auch

auf, Pakete weiterzuleiten. Dadurch wird die gesamte Weiterleitungsgruppe, bzw. der nicht mehr benötigte Teil, sukzessive abgebaut.

Der zweite Fall tritt ein, wenn die Applikation beim Sender plötzlich keine Daten mehr versendet. In diesem Fall wird die gesamte Weiterleitungsgruppe für diese Gruppe von diesem Sender nicht mehr benötigt.

Um bei einem kurzfristigen Ausbleiben von Daten nicht sofort die gesamte Weiterleitungsgruppe abzubauen, fängt ADMR beim Sender zunächst an, *Maintenance Keep-Alive*-Pakete zu verschicken. Diese werden wie herkömmliche Pakete als *Tree-Flood* an die Multicast-Gruppe verschickt, um die Weiterleitungsgruppe aufrecht zu erhalten.

Sie haben aber gleichzeitig den Zweck die anderen Knoten davon zu unterrichten, dass im Moment keine neuen Daten produziert werden. Daher ist in einem *Maintenance Keep-Alive* die Anzahl der noch zu verschickenden *Maintenance Keep-Alives* sowie der zeitliche Abstand zwischen diesen gespeichert. Jeder Knoten weiß damit, wann das letzte Paket zu erwarten ist. Erhält er bis dahin keine echten Datenpakete, so können die Informationen für diese Multicast-Gruppe verworfen werden.

Dieses Verhalten wirft jedoch zumindest mit dieser Spezifikation von ADMR ein Problem auf. Laut [Jet01b] soll die Zeit zwischen den *Maintenance Keep-Alive*-Paketen jeweils mit einem Faktor multipliziert werden. So wird erreicht, dass diese zu Beginn noch recht häufig und dann immer seltener gesendet werden. Problematisch ist nun, dass auch noch die größte Zeitspanne zwischen zwei Paketen, also zwischen dem vorletzten und letzten *Maintenance Keep-Alive*-Paket in dem Feld *Inter-Packet Time* des Pakets gespeichert werden muss. Dieses Feld hat allerdings nur eine Breite von 10 Bit und stellt Millisekunden dar. Die maximale Zeit zwischen zwei Paketen ist somit $(2^{10}) - 1\ ms = 1023\ ms$, also nur 1,023 Sekunden. Dies führt dazu, dass bei großen Abständen zwischen den Datenpaketen nur noch sehr wenige *Maintenance Keep-Alive*-Pakete verschickt werden können und so deren Zweck, die Weiterleitungsgruppe aufrecht zu erhalten, auch nur sehr kurzfristig erfüllt wird. Mindestens ein *Maintenance Keep-Alive* ist aber notwendig, um alle Knoten davon zu unterrichten, dass der Sender keine weiteren Daten mehr versendet.

Die *Inter-Packet Time* wird jedoch nicht nur bei *Maintenance Keep-Alive*-Paketen sondern auch in normalen Datenpaketen gespeichert, um so ausbleibende Pakete bei Linkabbrüchen zu erkennen. Daher wird diese Beschränkung problematisch, falls eine Applikation Datenpakete in Abständen von mehr als 1,023 Sekunden erzeugt. In diesem Fall läßt sich der Abstand zwischen den Paketen nicht mehr korrekt darstellen und die Weiterleitungsgruppe sowie die Empfänger wissen nicht, in welchen Abständen mit neuen Paketen zu rechnen ist. Dies kann dazu führen, dass zu früh Reparaturen eingeleitet werden. Die dabei verschickten Pakete führen zu einer unnötigen Belastung des Netzes.

2.5 ODMRP

Das in dieser Arbeit für Vergleiche mit ADMR herangezogene „On-Demand Multicast Routing Protocol“ (ODMRP) liegt als IETF-Internet-Draft vor [LSG00]. Die im Folgenden gegebene Beschreibung orientiert sich an [LSG00] sowie [Jet01a].

Ähnlich wie ADMR benutzt ODMRP eine Weiterleitungs*gruppe*, um die Daten vom Sender zu den Empfängern zu transportieren. Im Gegensatz zu ADMR wird diese jedoch nicht für jedes Tupel (Sender, Multicast-Gruppe) getrennt aufgebaut, sondern nur einmal für jede Multicast-Gruppe. Das kann bei mehreren Sendern dazu führen, dass die Pakete jedes einzelnen Senders zu oft, d. h. in die „falsche“ Richtung weitergeleitet werden, da Teile der Weiterleitungsgruppe nämlich nur zu einem anderen Sender und nicht zu weiteren Empfängern führen. Andererseits kann diese zusätzliche Redundanz auch zu besseren Ergebnissen führen, da unter ungünstigen Situationen womöglich mehr Pakete ihr Ziel erreichen.

Möchte bei ODMRP ein Sender Daten verschicken, so wird zunächst ein *Join Query* als *Network-Flood* verschickt. Wie bei ADMR merkt sich dabei jeder Knoten, von wem er dieses Paket erhalten hat. Erreicht dieses Paket einen Empfänger, so antwortet er darauf mit einem *Join Reply*. Dieser wird auf dem Weg, den vorher der *Join Query* genommen hat, an den Sender weitergeleitet. Jeder Knoten, der das Paket weiterleitet, setzt sein *forwarding group flag* und gehört damit zur Weiterleitungsgruppe. Verschickt ein Sender jetzt ein Datenpaket kann es durch diese Knoten bis zu den Empfängern weitergeleitet werden. Sucht man die Pendants der ODMRP-Kontrollpakete in ADMR, so wird die Aufgabe des *Join Query* von normalen Datenpaketen übernommen, die geflutet werden. Ein *Join Reply* entspricht bei ADMR einem *Receiver Join*, wird jedoch anders als bei ADMR an alle Sender einer Gruppe weitergeleitet.

Im Gegensatz zu ADMR kennt ODMRP keine expliziten Mechanismen zur Reparatur, falls dies beispielsweise auf Grund von Bewegungen der Knoten notwendig sein sollte. Stattdessen wird das *Join Query*-Paket vom Sender periodisch wiederholt und die Weiterleitungsgruppe so immer wieder neu aufgebaut. Durch dieses Verfahren können Ersatzrouten entstehen, die somit zu einer erfolgreicheren Auslieferung der Pakete, durch die stärkere Belastung des Netzes aber auch zu mehr Paketverlusten, führen können. Außerdem wird ein *Join Reply*-Paket nicht nur an einen, sondern an alle Sender geschickt, wordurch sich, wie weiter oben schon beschrieben, u. U. zusätzliche Redundanz ergibt. Es gibt bei ODMRP keine speziellen Verfahren um Knoten aus der Weiterleitungsgruppe ausscheiden zu lassen. Jeder Knoten verlässt diese automatisch nach einer gewissen Zeitspanne, falls er zwischenzeitlich durch einen neuen *Join Reply* nicht erneut zum Weiterleiter geworden ist.

ADMR benutzt an vielen Stellen des Protokolls die Vorstellung, dass die Weiterleitungsknoten einen Baum darstellen. Dies wird von den Autoren zwar verneint, ist aber beispielsweise bei den Reparaturmechanismen eindeutig gegeben. ODMRP benutzt dagegen tatsächlich eher eine Gruppe von Knoten, welche

bei ODMRP *Mesh*, also Gitter genannt wird. Bei der Weiterleitung der *Join Reply*-Pakete auf dem entgegengesetzten Weg des *Join Query*-Pakets entsteht zwar zwangsläufig ein Baum aus Knoten, betrachtet man aber wie in Abbildung 6 die Überlagerung der bei mehreren Sendern entstehenden Bäume, so ergibt sich tatsächlich ein Gitter.

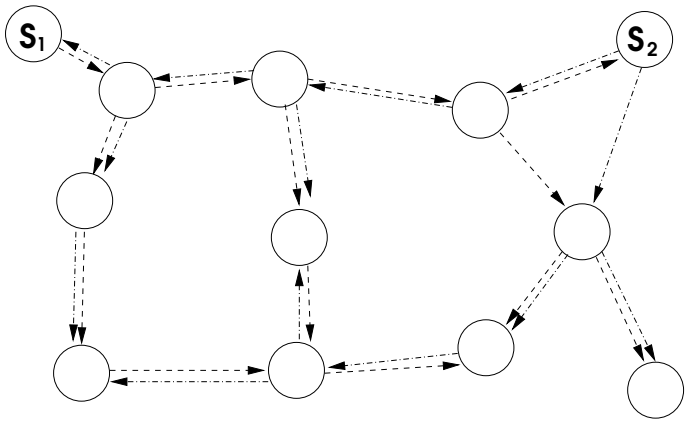

Abbildung 6: ODMRP-Mesh durch Überlagerung von zwei Bäumen

3 Implementierung

Dieser Abschnitt beschreibt die Implementierung von ADMR in GloMoSim. Zunächst wird auf das grundsätzliche Vorgehen und später auf Details der Umsetzung von ADMR eingegangen.

Es wird beschrieben welche Dateien für eine erfolgreiche Implementierung von ADMR in GloMoSim geändert werden mussten, welche ADMR-Funktionen wann von GloMoSim aufgerufen werden und welche Aufgaben diese übernehmen. Des weiteren wird erklärt, an welchen Stellen es Abweichungen zum ADMR-Draft [Jet01b] gibt und inwiefern sich diese auf die Funktionsweise auswirken. Außerdem wird darauf eingegangen, welche Datenstrukturen ADMR zum Speichern der benötigten Informationen benutzt und wie eingehende Pakete verarbeitet werden.

3.1 GloMoSim

Wie bereits angesprochen ist das Design von GloMoSim durch seinen modularen Aufbau für mögliche Erweiterungen recht gut geeignet. Außerdem ist mit der Datei `network/user_nwip.pc` bereits ein Grundgerüst zur Implementierung eines eigenen Routing Protokolls vorhanden.

GloMoSim verwaltet für jeden einzelnen Knoten eine Struktur `GlomoNode`, mit deren Hilfe jede Schicht Informationen speichern kann. So soll vermieden werden, dass ein Knoten - etwa durch eine fehlerhafte Programmierung - die Daten von anderen Knoten benutzen oder verändern kann. In `GlomoNode` existieren Zeiger, die auf die Strukturen der einzelnen Schichten verweisen. Ein Routing Protokoll kann so für jeden Knoten die benötigten Daten speichern. In diesem Fall erfolgt der Zugriff allerdings nicht direkt über einen Zeiger aus `GlomoNode`, sondern über einen Zeiger, der zunächst auf die Daten der IP-Schicht verweist. Erst diese enthalten dann den Zeiger auf die Strukturen des Routing Protokolls.

Ist für einen Knoten ein Aufruf des Routing Protokolls nötig, so wird diesem von GloMoSim die `GlomoNode`-Struktur dieses Knotens übergeben.

3.1.1 Funktionen

Für ein neues Routing Protokoll müssen vier Funktionen implementiert werden, die von GloMoSim aufgerufen werden. In `network/user_nwip.pc` sind dies `NetworkIpUserProtocolInit()`, `NetworkIpUserProtocolFinalize()`, `NetworkIpUserHandleProtocolEvent()` und `NetworkIpUserHandleProtocolPacket()`. Der Aufruf erfolgt von `network/nwip.pc` aus. Im Fall von ADMR rufen die genannten Funktionen lediglich ihre Pendants in `network/admr.pc` auf. Außerdem wird über `NetworkIpSetRouterFunction()` noch eine fünfte Funktion angegeben, die GloMoSim aufrufen soll, wenn eine Routingentscheidung getroffen werden muß.

Die Aufgaben dieser Funktionen werden im Folgenden kurz beschrieben.

RoutingAdmrProtocolInit(): Der Aufruf dieser Funktion erfolgt von `NetworkIpUserProtocolInit()`. Sie wird vor dem eigentlichen Start der Simulation gestartet und dient dazu die nötigen Initialisierungen vorzunehmen. Hier wird z. B. Speicher für die Datenstrukturen reserviert, die Gruppenmitgliedschaften aus einer Konfigurationsdatei eingelesen, die Variablen für die Statistiken initialisiert und die entsprechenden Timer gesetzt sowie `RoutingAdmrRouterFunction()` als Routingfunktion festgelegt.

RoutingAdmrProtocolFinalize(): Am Ende der Simulation erfolgt ihr Aufruf durch `NetworkIpUserProtocolFinalize()`. Einzige Aufgabe dieser Funktion ist es, die während der Simulation gesammelten statistischen Daten in die Statisitk-Datei (`glomo.stat`) auszugeben. Dabei handelt es sich beispielsweise um die Anzahl der gesendeten Pakete.

RoutingAdmrHandleProtocolEvent(): Hat ein Routing Protokoll eine Aufgabe nicht sofort sondern erst zu einem bestimmten Zeitpunkt zu erledigen, stellt GloMoSim hierfür Timer zur Verfügung. Dazu wird eine Nachricht erzeugt, die dem Protokoll dann zum gewünschten Zeitpunkt als „Event“ zugestellt wird. Dies wird z. B. dazu benutzt bestimmte Aktionen zu veranlassen, wenn ein Knoten einer Multicast-Gruppe beitritt. Der entsprechende Timer dafür wird in `RoutingAdmrProtocolInit()` gesetzt.

Die so auftretenden Nachrichten werden von `RoutingAdmrHandleProtocolEvent()` verarbeitet. Eine Aufstellung aller von ADMR verwendeten Nachrichtentypen ist in Tabelle 2 dargestellt.

RoutingAdmrHandleProtocolPacket(): Pakete, die als IP-Empfängeradresse entweder diesen Knoten oder die Broadcast-Adresse haben, werden (ohne IP-Header) an diese Funktion übergeben. Es kann sich hierbei sowohl um Daten- als auch um Kontrollpakete handeln. Unter Umständen müssen an diese Funktion übergebene Pakete auch von ADMR erneut verschickt werden. Da der IP-Header

Ereignis	Bedeutung
`MSG_NETWORK_JoinGroup`	Knoten tritt einer Gruppe bei
`MSG_NETWORK_LeaveGroup`	Knoten verlässt eine Gruppe
`MSG_ADMR_JoinTimer`	überprüfen ob Beitritt zur Gruppe erfolgreich war
`MSG_ADMR_StateSetupTimer`	beim Sender; Weiterleitungsgruppe müsste aufgebaut sein, gepufferte Pakete senden
`MSG_ADMR_NetworkFlood`	beim Sender; nächstes Paket als Network-Flood senden
`MSG_ADMR_CheckStateExpiration`	bei Sender bzw. Weiterleiter; überprüfen ob weiterhin Pakete verschickt/weitergeleitet werden müssen
`MSG_ADMR_UnicastKeepAlive-RetransmissionTimer`	Unicast Keep-Alive erneut bzw. nächstes Datenpaket als *Network-Flood* senden
`MSG_ADMR_MulticastKeepAlive-Timer`	falls keine Daten gesendet werden nächsten Multicast Keep-Alive schicken bzw. Sender endgültig inaktiv
`MSG_ADMR_CheckForwarding-ExpirationTimer`	beim Weiterleiter; prüfen ob noch Pakete empfangen werden
`MSG_ADMR_DisconnectionTimer`	auf Linkabbruch überprüfen
`MSG_ADMR_RepairNotification-DelayTimer`	Zeit seit dem Senden der Repair-Notification ist verstrichen
`MSG_ADMR_ReconnectTimer`	überprüfen ob Reconnect erfolgreich war
`MSG_ADMR_CheckHighMobility-Timer`	Zähler (`mobility counter`) für hohen Paketverlust zurücksetzen

Tabelle 2: ADMR-Ereignisse

bereits entfernt wurde muss er erneut angefügt werden. Dabei muss bei Bedarf die IP-Absenderadresse angepasst werden, da das Paket ja ursprünglich von einem anderen Knoten verschickt wurde.

RoutingAdmrRouterFunction(): Diese Funktion wird von GloMoSim aufgerufen wenn ein Paket eintrifft, dass nicht direkt an `RoutingAdmrHandleProtocol-Packet()` weitergeleitet wird. Auch Pakete aus der Anwendungsschicht treffen hier ein, so dass vor dem Versenden ein ADMR-Header hinzugefügt werden kann.

3.1.2 Anpassungen

Abgesehen von der Implementierung der genannten Funktionen waren für eine erfolgreiche Integration von ADMR in GloMoSim noch einige andere Änderungen notwendig. Eine Übersicht der veränderten Dateien liefert Tabelle 3.

Datei	**Zeilen** (in veränderter Datei), **Aktion**
`application/application.pc`	190-193 Überprüfung auf gültiges Routing-protokoll angepasst
`include/structmsg.h`	176-187 ADMR-Nachrichtentypen definiert
`main/Makefile`	60, 133-136 angepasst, um `network/admr.pc` und `network/admr.h` bei der Übersetzung zu berücksichtigen
`network/admr.h`	komplett neu
`network/admr.pc`	komplett neu
`network/nwip.pc`	2035-2038 Weitergabe der Adresse, von der das Paket empfangen wurde, an das Routing Protokoll hinzugefügt
`network/user_nwip.pc`	76, 85-89, 109-110, 138-139, 162-164; entfernt: 72, 78 (alt) Aufrufe an die entsprechenden Funktionen in `network/admr.pc` eingefügt

Tabelle 3: Angepasste Dateien im Vergleich zu GloMoSim 2.03

Zunächst musste `main/Makefile` angepasst werden, um die neu erstellten Dateien `admr.pc` und `admr.h` beim Kompilieren zu berücksichtigen. Da ADMR diverse neue Nachrichtentypen für Timer verwendet, musste die `enum`-Konstruktion in `include/structmsg.h` dementsprechend erweitert werden. In `application/application.pc` wird überprüft, ob in der Konfigurationsdatei ein gültiges Routing Protokoll angegeben wurde. Dies musste um ADMR erweitert werden.

Wie schon erwähnt werden bei einem neuen Routing Protokoll die Funktionen in `network/user_nwip.pc` aufgerufen. Da diese Aufrufe lediglich an die Funktionen in `network/admr.pc` durchgereicht werden sollten, musste `user_nwip.pc` entsprechend verändert werden.

Eine weitere Änderung, die für eine korrekte Funktion von ADMR nötig war, betrifft `network/nwip.pc`. ADMR muss wissen von welchem Knoten eine Nachricht empfangen wurde, um die entsprechenden Einträge in der *Node Table* vornehmen zu können. Nur so können Nachrichten wie ein *Receiver Join* und ein *Unicast Keep-Alive* erfolgreich geroutet werden (vgl. hierzu Abschnitte 2.4.2 und 2.4.3). Diese Information wird standardmäßig jedoch nicht an die Funktionen des Routing Protokolls übergeben.

GloMoSim stellt für einen solchen Fall eine Möglichkeit zur Verfügung, die

ohne eine Änderung der Schnittstellen der entsprechenden Funktionen auskommt und so relativ problemlos umzusetzen ist. Dafür hat jede von GloMoSim verschickte Nachricht das Feld `info`, für das über `GLOMO_MsgInfoAlloc()` Platz reserviert werden kann. Legt man in diesem Feld Daten ab, so werden diese zusammen mit der Nachricht an die weiteren Funktionen übergeben und können von diesen ausgelesen werden. Problematisch wird dieses Verfahren allerdings, wenn z. B. zwei verschiedene Schichten unabhängig voneinander das `info`-Feld der selben Nachricht benutzen wollen, da sich diese dann gegenseitig behindern.

Im Fall von ADMR gibt es dieses Problem jedoch nicht. Daher wurde die Funktion `NetworkIpReceivePacketFromMacLayer` so erweitert, dass sie die Adresse des Knotens, von dem das Paket empfangen wurde, im `info`-Feld ablegt.

Die eigentliche Implementierung von ADMR befindet sich in der Datei `network/admr.pc` sowie in der dazugehörigen Header-Datei `network/admr.h`.

3.1.3 Visualisierung

Um das Verhalten der einzelnen Knoten zu visualisieren wurde das Programm „The Network Animator“ [nam] verwendet. Im Folgenden wird dieses Programm kurz mit „nam“ bezeichnet. Es wurde ursprünglich für den Simulator ns-2 entwickelt.

Am Institut für Telematik an der Universität Karlsruhe (TH) wurde GloMoSim so erweitert, dass die benötigten Tracefiles für nam erzeugt werden können. Dieses Verhalten lässt sich über die Parameter `NAM-TRACE YES|NO` und `NAM-TRACE-FILENAME <dateiname>` in der GloMoSim-Konfigurationsdatei beeinflussen. Beim Aufruf von nam wird das somit erzeugte Tracefile als Parameter übergeben.

Da nam in der Lage ist Knoten in verschiedenen Farben anzuzeigen, wird dies von ADMR genutzt. Sender werden blau, Empfänger rot und Weiterleiter grün markiert. Zuverlässig funktioniert dieses Verfahren allerdings nur bei einfachen Szenarien. Da ein Knoten mehrere Funktionen übernehmen kann, also etwa sowohl Sender als auch Empfänger sein kann, ist bei komplexeren Szenarien eine eindeutige, fehlerfreie Zuordnung der Farben nicht mehr gegeben. Trotzdem läßt sich gerade in der Entwicklungsphase die Funktion des Routing Protokolls so leicht überprüfen. Außerdem visualisiert nam das Versenden von Paketen. Bei Broadcast-Paketen geschieht dies durch einen Kreis um den jeweiligen Sender, bei Unicast-Paketen durch einen kleinen Strich, der sich vom Sender zum Empfänger bewegt.

Ein Beispiel für eine Visualisierung durch nam zeigt Abbildung 7. In diesem Fall handelt es sich um einen Sender (blau, rechts), zwei Empfänger (rot, Mitte und links) sowie mehrere Weiterleiter (grün).

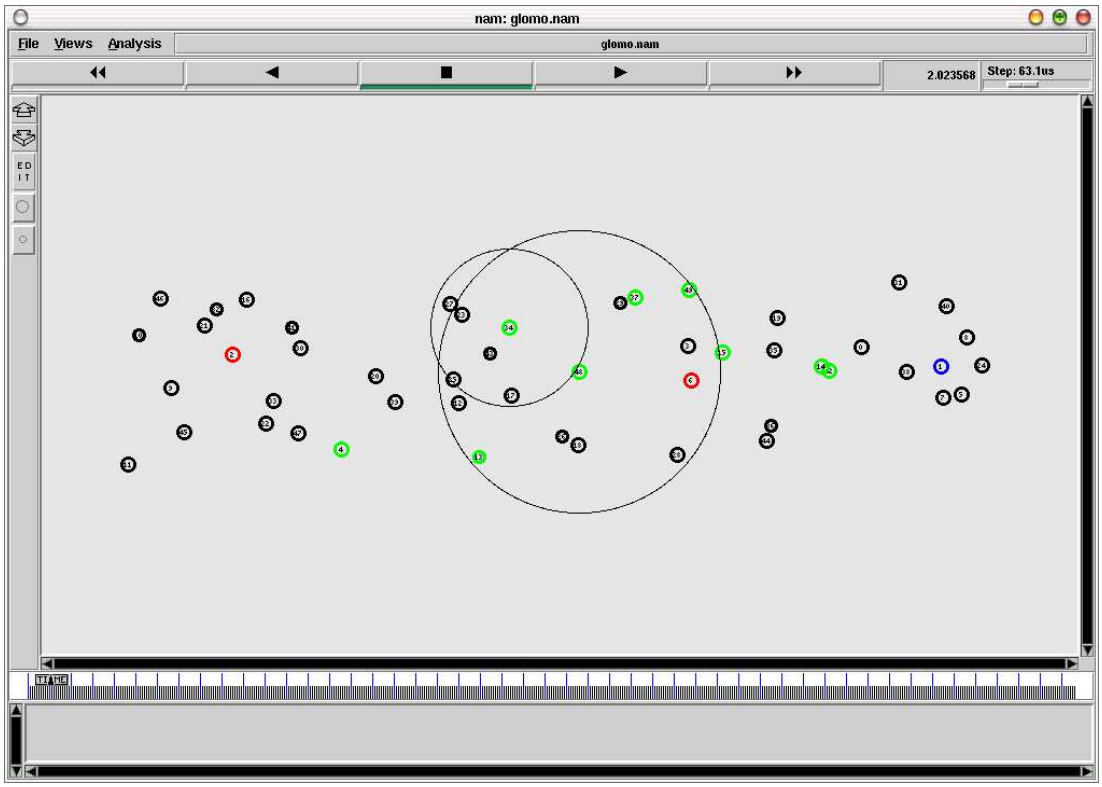

Abbildung 7: Visualisierung mit `nam`

3.2 ADMR als Routing Protokoll in GloMoSim

Für die Umsetzung von ADMR wurde der entsprechende Internet-Draft [Jet01b] verwendet. In diesem gibt es an einigen Stellen jedoch Unstimmigkeiten, so dass die darin gestellten Anforderungen nur bis auf einige Ausnahmen umgesetzt werden konnten. Andere Abweichungen sind durch die Funktionsweise von GloMoSim bedingt. Daher wird im Folgenden eine Übersicht über die Stellen, an denen die Implementierung vom Draft abweicht, gegeben. Zum besseren Verständnis der Funktionsweise von ADMR und den Auswirkungen der Abweichungen werden jedoch zunächst die von ADMR benutzten Datenstrukturen beschrieben.

Danach wird auf die Arbeitsweise der ADMR-Implementierung beim Senden von Daten, sowie beim Routing von Multicast-, Unicast- und *Network-Flood*-Paketen eingegangen. Darauf folgt eine Aufstellung einiger Besonderheiten von ADMR und der erzeugten Statistiken.

3.2.1 Datenstrukturen

Bei ADMR entscheidet jeder Knoten selbstständig, ob er ein empfangenes Paket weiterleitet oder nicht. Um diese Entscheidung fällen zu können, müssen vorher

diverse Informationen gesammelt werden, etwa ob für eine Multicast-Gruppe ein passender *Receiver Join* weitergeleitet wurde oder nicht.

Zur Speicherung dieser Informationen werden in jedem Knoten drei Tabellen, eine *Sender Table*, eine *Membership Table* und eine *Node Table* verwaltet. In [Jet01b] wird vorgeschlagen, welche Daten in den Tabellen zu speichern sind. Leider hat sich bei der Implementierung herausgestellt, dass diese Vorschläge teilweise nicht ausreichend sind und noch weitere Informationen zu speichern sind.

Sender Table: Die *Sender Table* enthält einen Eintrag für jede Gruppe, für die dieser Knoten ein aktiver Sender ist. Hier wird beispielsweise gespeichert, ob überhaupt Empfänger vorhanden sind oder wie groß der zeitliche Abstand zwischen den gesendeten Paketen ist. Abbildung 8 zeigt die hierfür verwendete Struktur.

```
typedef struct RoutingAdmrSenderTableEntry_Str {
    unsigned int    multicastGroup;
    unsigned int    interPacketTime;
    unsigned int    interKeepAliveMultiplicationFactor;
    unsigned int    interKeepAliveTime;
    unsigned int    keepAliveCount;
    int             mobilityCounter;
    int             packetLoss;
    BOOL            stateSetupFlag;
    BOOL            floodFlag;
    BOOL            floodModeFlag;
    BOOL            connectedFlag;
    struct RoutingAdmrSenderTableEntry_Str*  next;
    unsigned int unforwardedPackets;
    RoutingAdmrSendBufferEntry* sendBuffer;
    clocktype       lastPacket;
    RoutingAdmrJoinsListEntry* outstandingReceiverJoins;
    clocktype       lastFlood;
    clocktype       lastKeepAlive;
} RoutingAdmrSenderTableEntry;
```

Abbildung 8: *Sender Table*-Eintrag

Membership Table: Die *Membership Table* enthält einen Eintrag für jede Kombination aus Sender und Multicast-Gruppe, für die dieser Knoten entweder ein Empfänger ist oder der Weiterleitungsgruppe angehört. Die Struktur der Einträge zeigt Abbildung 9.

Node Table: Die in Abbildung 10 dargestellte *Node Table* enthält einen Eintrag für jedes Tupel (Sender, Multicast-Gruppe) zu dem dieser Knoten ein Paket erhalten oder weitergeleitet hat. Hier werden unter anderem die Sequenznummern der Pakete gespeichert, um so Duplikate erkennen zu können. Außerdem wird jeweils die MAC-Adresse des Knotens gespeichert, von dem das letzte zu diesem Tupel passende Paket erhalten wurde. Dies ermöglicht später beispielsweise die

```
typedef struct RoutingAdmrMembershipTableEntry_Str {
    unsigned int   multicastGroup;
    unsigned int   sender;
    BOOL  isReceiver;
    BOOL  isConnected;
    BOOL  isForwarder;
    int   interPacketTime;
    int   keepAlivePacketCount;
    int   interKeepAliveMultiplicationFactor;
    struct RoutingAdmrMembershipTableEntry_Str* next;
    int   mobilityCounter;
    NODE_ADDR   previousHopAddress;
    unsigned int unforwardedPackets;
    BOOL  highMobility;
    clocktype   lastPacket;
    clocktype   disconnectionTimerTimeout;
    BOOL        gotRepairNotification;
} RoutingAdmrMembershipTableEntry;
```

Abbildung 9: *Membership Table*-Eintrag

```
typedef struct RoutingAdmrNodeTableEntry_Str {
    unsigned int        sender;
    unsigned int        receiver;
    unsigned int        sequenceNumber;
    BOOL                bitmapPreviousSequenceNumbers[10];
    unsigned int        previousHopAddress;
    struct RoutingAdmrNodeTableEntry_Str* next;
    unsigned int        hopCount;
    unsigned int        joinsCounter;
} RoutingAdmrNodeTableEntry;
```

Abbildung 10: *Node Table*-Eintrag

Weiterleitung eines *Receiver Joins* bis zum Sender und somit den Aufbau der Weiterleitungsgruppe.

Zusätzlich zu diesen Tabellen wird bei einem Sender noch ein *Send Buffer* benötigt. Dies ist ein Puffer für Pakete, die noch nicht gesendet werden können, da die Weiterleitungsgruppe noch nicht aufgebaut ist.

3.2.2 Abweichungen

Eine Änderung, die jedoch auf die Funktionsweise von ADMR keine Auswirkungen hat, betrifft die *Source Information Option*. An deren Ende steht laut [Jet01b] die *Previous Hop MAC Address*, also die Adresse des Knotens, von dem das Paket empfangen wurde. Das Feld davor (*Address Length*) gibt die Länge dieser Adresse an. Da GloMoSim jedoch ohnehin eine feste Länge für die Adressen verwendet, wurde das Feld *Address Length* gestrichen und für die *Previous Hop MAC Address* eine feste Länge von 32 Bit verwendet.

Außerdem darf laut [Jet01b] das Feld *Previous Hop MAC Address* in Pa-

keten die ein Knoten erzeugt, also nicht nur weiterleitet, nicht vorhanden sein. Dies bedeutet jedoch, dass ein Knoten der so ein Paket weiterleitet dieses Feld nachträglich einfügen müsste, was die Länge des Headers verändern würde. Dafür müsste somit zunächst der IP-Header entfernt, der ADMR-Header verändert und dann wieder der IP-Header hinzugefügt werden. Um dieses aufwändige Verfahren zu vermeiden, wird in dieser Implementierung das Feld direkt beim Sender eingefügt und auf die Adresse des Senders gesetzt.

Die Arbeitsweise von ADMR wird von dieser Änderung nicht beeinflusst. Die *Previous Hop MAC Address* dient ausschließlich dazu zu erkennen, ob die von einem Knoten K_1 versendeten/weitergeleiteten Pakete auch von anderen Knoten, z. B. K_2, weitergeleitet werden. Ist dies der Fall, so setzt K_2 die *Previous Hop MAC Address* auf die Adresse von K_1, K_1 empfängt die Nachricht auch und weiß so, dass K_2 die Daten weiterleitet. Wird ein Paket jedoch nicht weitergeleitet sondern erst erzeugt, so ist dieser Mechanismus nicht nötig. Auf Grund dessen bleibt die genannte Änderung ohne Konsequenzen.

In [Jet01b] wird im Abschnitt 7.1.1 beschrieben, in welchen Fällen ein Sender ein Paket nicht direkt versenden darf, sondern es im *Send Buffer* zwischenspeichern muss. Dies soll laut [Jet01b] der Fall sein, wenn (nach Kenntnis des Senders) die Weiterleitungsgruppe noch nicht aufgebaut ist, keine Empfänger vorhanden sind und das nächste Paket nicht als *Network-Flood* verschickt werden soll[1]. Falsch wird dieses Verhalten allerdings in dem Fall, wenn das *State Setup Flag* noch gesetzt ist (Weiterleitungsgruppe aufgebaut), aber keine Empfänger mehr vorhanden zu sein scheinen. Auch in diesem Fall sollte ein Paket, was nicht für den periodischen *Network-Flood* bestimmt ist, im *Send Buffer* abgelegt werden. Eine Alternative könnte sein, dass der Sender in dem Moment, in dem er feststellt, dass keine Empfänger mehr vorhanden sind auch das *State Setup Flag* löscht. [Jet01b] macht jedoch keine genauen Angaben dazu, was ein Sender in diesem Fall für Aktionen ausführen muss.

Diese Implementierung löscht das *State Setup Flag* nicht, wenn festgestellt wird, dass keine Sender mehr vorhanden sind. Daher werden Pakete auch im *Send Buffer* gespeichert, wenn das *State Setup Flag* noch gesetzt ist, aber keine Empfänger mehr vorhanden sind.

Ein Problem, welches durch GloMoSim bedingt ist, betrifft die Prüfsumme des IP-Headers. ADMR muss an einigen Stellen den IP-Header entfernen, um z.B. den ADMR-Header einzufügen. Fügt ADMR nachher den IP-Header wieder an, so müsste die Prüfsumme neu berechnet werden. Das wäre auch bei Änderungen am ADMR-Header der Fall. Da GloMoSim die Prüfsumme aber nicht benutzt und auch keine Funktion bereitstellt, um sie zu berechnen, unterbleibt dies. Sollte

[1] „If the flood flag, the state setup flag, and the connected flag are all cleared, then the packet is placed in the Send Buffer“

GloMoSim in zukünftigen Versionen die Prüfsumme doch benutzen, so müsste ADMR dahingehend angepasst werden.

Eine weitere, aber ebenfalls unproblematische, Abweichung der Implementierung von [Jet01b] gibt es bei den *Multicast Keep-Alive*s. Diese sollen eigentlich keinen Platz für Nutzdaten (also auf den ADMR-Header folgende Daten) enthalten. GloMoSim unterscheidet explizit zwischen Nutzdaten und Headern. Der Platz für Nutzdaten wird mit `GLOMO_MsgPacketAlloc()` reserviert, sollen Header hinzugefügt werden geschieht dies über `GLOMO_MsgAddHeader()`. Diese ADMR-Implementierung benutzt daher `GLOMO_MsgAddHeader()` zum Hinzufügen des ADMR-Headers. Soll ein *Multicast Keep-Alive* erzeugt werden, muss beim Aufruf von `GLOMO_MsgPacketAlloc()` also eigentlich eine `packetSize`[2] von Null angegeben werden. Dies führt bei GloMoSim im Laufe der Simulation jedoch zu einem Fehler, so dass die Ausführung des Programms abgebrochen wird. Daher benutzt diese ADMR-Implementierung für die `packetSize` eine Größe von einem Byte.

3.2.3 Senden von Daten

ADMR erhält die zu versendenden Pakete aus der Anwendungsschicht über `RoutingAdmrRouterFunction()`. Dass es sich tatsächlich um ein Paket direkt aus der Anwendungsschicht handelt erkennt ADMR daran, dass noch kein ADMR-Header vorhanden ist und die IP-Quelladresse mit der IP-Adresse dieses Knotens identisch ist. Daraufhin wird `RoutingAdmrSendData()` aufgerufen. Den Ablauf von `RoutingAdmrRouterFunction()` veranschaulicht Abbildung 11, einen Überblick über die Funktionsweise von `RoutingAdmrSendData()` liefert Abbildung 12.

`RoutingAdmrSendData()` überprüft zunächst, ob für die verwendete Multicast-Adresse schon ein Eintrag in der *Sender Table* vorhanden ist. Ist dies nicht der Fall, so wird ein neuer Eintrag erzeugt. Außerdem werden alle benötigten Timer zum ersten Mal gestartet: Einer für das periodische Fluten (`MSG_ADMR_NetworkFlood`), einer nach dessen Ablauf die Weiterleitungsgruppe aufgebaut sein sollte (`MSG_ADMR_StateSetupTimer`) und ein Timer zum Überprüfen ob der Sender noch aktiv ist bzw. zum Senden von *Multicast Keep-Alives* (`MSG_ADMR_MulticastKeepAliveTimer`).

Danach wird die Entscheidung darüber getroffen, ob das Paket als *Network-Flood* bzw. als *Tree-Flood* gesendet oder im *Send Buffer* gespeichert wird.

In jedem Fall wird zunächst der ADMR-Header zum Paket hinzugefügt. Als erstes wird der feste Teil des ADMR-Headers erzeugt und ins Paket eingefügt. Dies geschieht über `RoutingAdmrAddAdmrHeader()`. Danach wird eine *Source Information Option* erzeugt in der alle Werte entsprechend gesetzt werden. Beispielsweise muss der *Hop Count* auf Null gesetzt werden. Soll das Paket als *Network-Flood* versendet werden, wird zusätzlich eine *Multicast Group Address*

[2] = „`payLoadSize`: size of the payLoad to be allocated“ (vgl. `main/message.pc`)

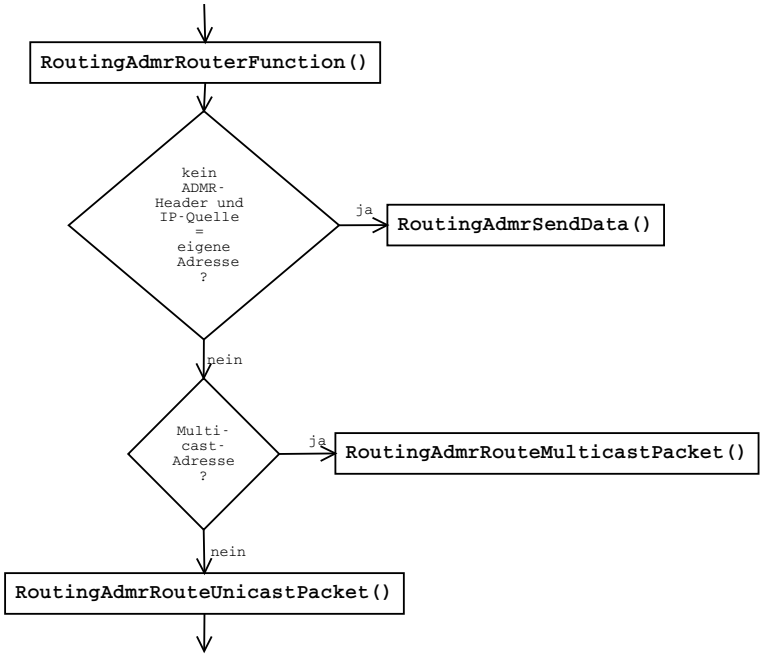

Abbildung 11: Ablauf von RoutingAdmrRouterFunction()

Option erzeugt. In ihr wird gespeichert an welche Gruppe das Paket adressiert ist, da dies bei einem *Network-Flood* nicht mehr über die IP-Zieladresse geschehen kann.

Sind die Optionen erzeugt und entsprechend initialisiert, werden sie mittels `RoutingAdmrInsertAdmrOption()` in das Paket eingefügt. Wie in Abschnitt 2.4.1 beschrieben, muss die Größe des ADMR-Headers ein Vielfaches von 4 Byte betragen. `RoutingAdmrAdjustAdmrHeaderSize()` passt die Größe des Headers durch Auffüllen mit Pad1- und PadN-Optionen entsprechend an.

Ist das Paket nicht für den *Send Buffer* bestimmt, wird es am Ende durch `NetworkIpSendPacketToMacLayerWithDelay()` mit einer Verzögerung zwischen Null und `ADMR_BROADCAST_JITTER` Millisekunden verschickt.

3.2.4 Routing von Multicast-Paketen

Abbildung 13 zeigt allgemein, an welche Funktionen eingehende Pakete zur Bearbeitung weitergegeben werden. Im Fall von Multicast-Paketen wird über `RoutingAdmrRouterFunction()` die Funktion `RoutingAdmrRouteMulticastPacket()` aufgerufen. Diese führt nacheinander die folgenden Schritte aus:

1. Enthält das Paket eine *Repair Notification Option* wird `RoutingAdmrProcessRepairNotification()` aufgerufen und die Bearbeitung abgebrochen.

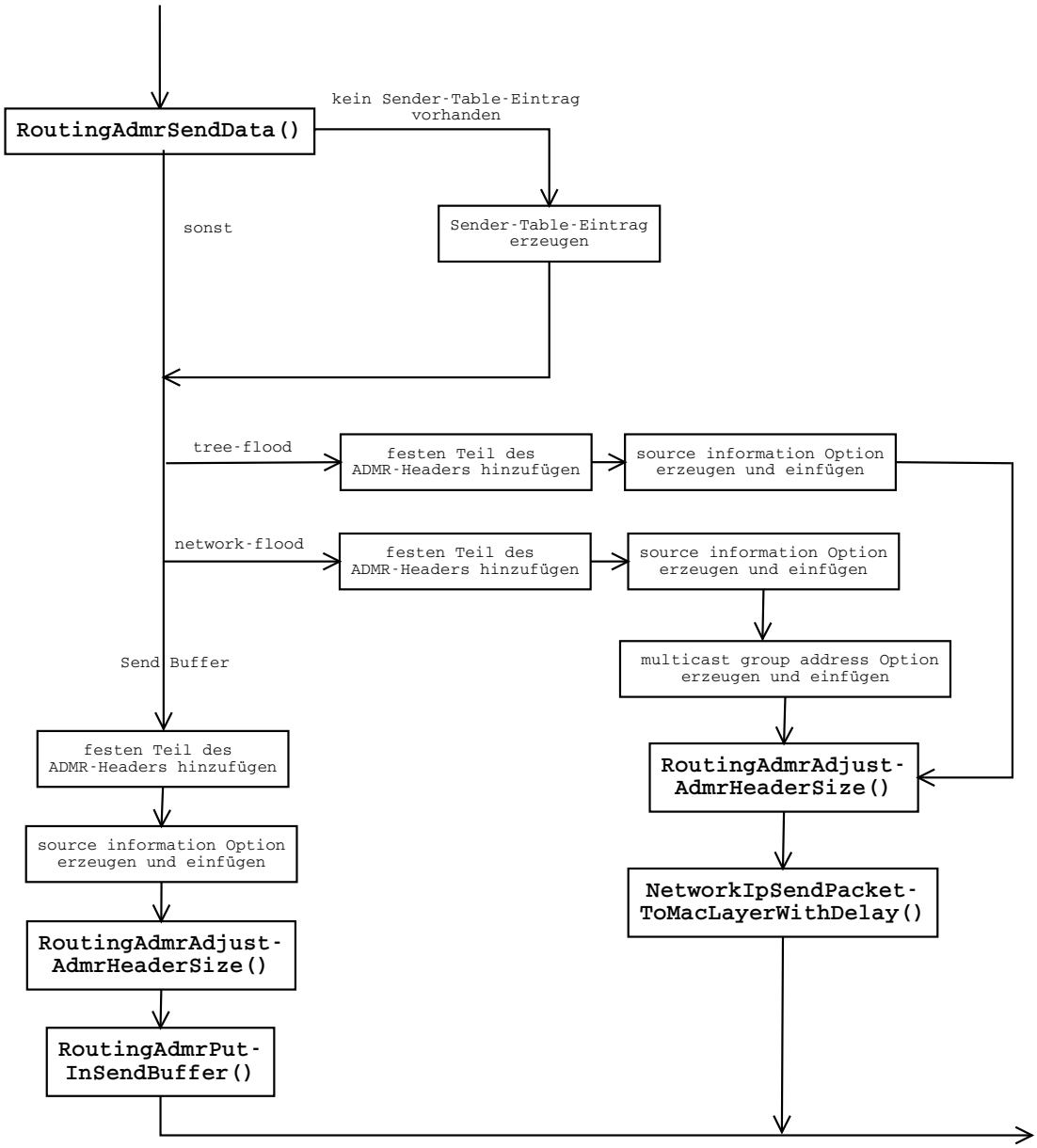

Abbildung 12: Ablauf von RoutingAdmrSendData()

2. Die *Source Information Option* wird ausgelesen. Ist das Feld `PreviousHopMacAdress`[3] mit der eigenen Adresse identisch wird dadurch erkannt, dass die eigenen Pakete noch weitergeleitet werden. Die Zähler für die Anzahl der nicht weitergeleiteten Pakete werden zurückgesetzt. Das Paket wird verworfen und die Bearbeitung an dieser Stelle abgebrochen.

3. Falls es sich um ein *Ack*-Paket von einem Empfänger handelt wird es verworfen und die Bearbeitung hier beendet. Die bei einem *Ack*-Paket notwendigen Aktionen wurden bereits in Punkt 2 ausgeführt. Dies muss vor der Duplikatserkennung geschehen, da das *Ack*-Paket dieselbe Sequenznummer wie das Datenpaket hat. Sollte ein *Ack*-Paket über einen anderen Weg vor dem Datenpaket ankommen würde das Datenpaket ansonsten verworfen werden.

4. Falls für das Tupel (Sender, Multicast-Gruppe) noch kein Eintrag in der *Node Table* vorhanden ist, so wird dieser mit den entsprechenden Angaben erzeugt. Andernfalls wird zunächst überprüft, ob es sich bei dem empfangenen Paket um ein Duplikat handelt, dieser Knoten dasselbe Paket also schon einmal empfangen hat. Da es in der Weiterleitungsgruppe durchaus redundante Wege geben kann ist dies leicht möglich. Sollte es ein Duplikat sein, so wird das Paket verworfen. Wenn es sich nicht um ein Duplikat handelt wird der entsprechende Eintrag in der *Node Table* aktualisiert.

5. Sollte es sich um einen *Maintenance Keep-Alive* handeln wird die Funktion `RoutingAdmrHandleMulticastKeepAlive()`[4] aufgerufen und die Bearbeitung beendet.

6. Ist die Bearbeitung an dieser Stelle angekommen muss es sich um ein Datenpaket handeln.

7. Es wird überprüft, ob dieser Knoten für das Tupel (Sender, Multicast-Gruppe) zu dem das Paket gehört, ein Mitglied der Weiterleitungsgruppe oder ein Empfänger ist. Ist dies nicht der Fall wird das Paket verworfen und die Bearbeitung beendet.

8. Die Einträge in der *Membership Table* (z.B. `previousHopAddress` oder `interPacketTime`) werden aktualisiert.

9. Falls der Knoten ein Empfänger des Pakets ist, wird es kopiert und die Kopie an die Anwendungsschicht übergeben. Das Kopieren des Pakets ist notwendig, da es u. U. noch weiter versendet werden muss.

[3] Adresse, von der der versendende Knoten das Paket empfangen hat

[4] [Jet01b] benutzt die Begriffe *Multicast* und *Maintenance Keep-Alive* synonym. Daher kommt es teilweise zu unterschiedlichen Bezeichnungen.

10. Ist der Knoten ein Empfänger, aber kein Weiterleiter, muss ein *Ack*-Paket geschickt werden. Dafür werden außer dem ADMR-Header und dem IP-Header die Nutzdaten vom Paket entfernt und das Paket wird erneut verschickt.

11. Ist der Knoten ein Mitglied der entsprechenden Weiterleitungsgruppe wird das Paket weitergeleitet.

Damit ist die Verarbeitung von Multicast-Paketen beendet. Für *Repair Notification*s und *Maintenance Keep-Alive*s werden die entsprechenden Funktionen aufgerufen, die im Folgenden betrachtet werden.

RoutingAdmrProcessRepairNotification(): Muss ein Knoten eine lokale Reparatur einleiten, verschickt er zunächst eine *Repair Notification*, um die anderen Knoten davon zu unterrichten. Das Prinzip ist in Abschnitt 2.4.4 genau beschrieben.

Erhält ein Knoten eine solche *Repair Notification* wird sie von `RoutingAdmrProcessRepairNotification()` verarbeitet. Zunächst wird bei Bedarf ein *Node Table*-Eintrag erzeugt bzw. ein bestehender angepasst. Duplikate werden verworfen.

Handelt es sich um eine fälschlicherweise verschickte *Repair Notification*[5], wird diese verworfen und eine eigene verschickt.

Der Zeitpunkt, an dem der Timer zur Auslösung einer eigenen Reparatur abläuft, wird nach hinten verschoben. Falls es sich bei diesem Knoten um einen Weiterleiter handelt wird die *Repair Notification* weitergeleitet.

RoutingAdmrHandleMulticastKeepAlive(): Wenn ein Sender keine weiteren Daten mehr sendet, beginnt ADMR damit *Maintenance Keep-alive*s zu verschicken. `RoutingAdmrHandleMulticastKeepAlive()`, welche diese *Maintenance Keep-Alive*s verarbeitet, muss drei Aufgaben ausführen:

- Das Ablaufen des Timers zum Erkennen von Linkabbrüchen muss verzögert werden, so dass dieser erst später ausläuft.
- Der Timer zum Erkennen des letzten Pakets vom Sender muss gesetzt werden.
- Das Paket muss weitergeleitet werden.

3.2.5 Routing von Unicast-Paketen

Eingehende Unicast-Pakete können entweder an diesen Knoten adressiert oder für einen anderen Knoten bestimmt sein. Dementsprechend wird entweder `RoutingAdmrHandleProtocolPacketForThisNode()` oder `RoutingAdmrRouteUnicastPacket()` aufgerufen.

[5] eigene Adresse = parentAddress

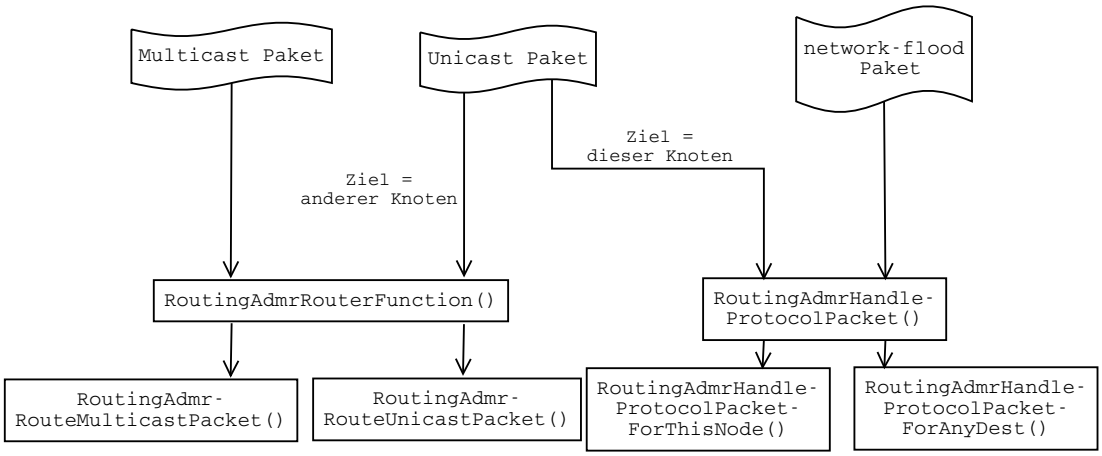

Abbildung 13: Verarbeitung eines eingehenden Pakets

Als Unicast werden ausschließlich Kontrollpakete verschickt. ADMR hat zunächst die Aufgabe zu erkennen, um welche Art von Kontrollnachricht es sich handelt und dann die betreffenden Aktionen auszuführen. Die auszuführenden Aktionen sind detailliert [Jet01b] zu entnehmen.

Mögliche Unicastnachrichten sind: Ein *Unicast Keep-Alive*, ein *Receiver Join*, eine als Unicast verschickte *Multicast-Solicitation* (vgl. Abschnitt 2.4.3), ein als Unicast verschickter *Reconnect* (vgl. Abschnitt 2.4.4) oder ein *Reconnect Reply*. Sie können alle entweder für diesen Knoten bestimmt sein oder müssen noch weitergeleitet werden. Daher können alle sowohl von `RoutingAdmrHandleProtocolPacketForThisNode()` als auch von `RoutingAdmrRouteUnicastPacket()` verarbeitet werden.

3.2.6 Routing von *Network-Flood*-Paketen

Pakete die an die IP-Broadcast-Adresse 255.255.255.255 adressiert sind werden über `RoutingAdmrHandleProtocolPacket()` an `RoutingAdmrHandleProtocolPacketForAnyDest()` übergeben (vgl. Abbildung 13). Das Design von GloMoSim geht dabei davon aus, dass solche Pakete ja *auch* für diesen Knoten bestimmt sind. Daher werden sie nicht an `RoutingAdmrRouterFunction()` übergeben.

Bei *Network-Flood*-Paketen kann es sich sowohl um Kontroll- als auch um Datenpakete handeln. Ihre Bearbeitung erfolgt in den folgenden Schritten:

1. Handelt es sich um eine *Multicast Solicitation* wird das Paket an `RoutingAdmrProcessMulticastSolicitationForAnyDest()` gegeben und die Bearbeitung beendet.

2. Sollte das Paket ein *Reconnect* sein wird es an `RoutingAdmrProcessReconnectForAnyDest()` gegeben und die Bearbeitung abgebrochen.

3. Ist die Bearbeitung an dieser Stelle angekommen muss es sich um ein Datenpaket handeln.

4. Bei Bedarf wird ein *Node Table*-Eintrag erzeugt. Sollte das Paket ein Duplikat sein wird es verworfen und die weitere Bearbeitung abgebrochen.

5. Ist der Knoten ein Empfänger für die betreffende Gruppe wird das Paket ohne ADMR-Header an die Anwendungsschicht weitergegeben.

6. Die Einträge in der *Membership Table* (z.B. `previousHopAddress` oder `interPacketTime`) werden aktualisiert.

7. Da es sich um ein *Network-Flood*-Paket handelt wird es versendet.

3.2.7 Besonderheiten

ADMR weist an einigen Stellen Besonderheiten auf, die bei der Benutzung, zumindest aber vor der Fehlersuche, berücksichtigt werden sollten. Daher werden sie an dieser Stelle kurz erwähnt. Sie sind teilweise allein auf die Spezifikation von ADMR, teilweise aber auch auf diese spezielle Implementierung von ADMR zurück zu führen.

Wie in Abschnitt 2.4.5 erklärt, sendet ein Sender beim Ausbleiben von Datenpaketen *Maintenance Keep-Alive*s. Die Anzahl der noch zu sendenden *Maintenance Keep-Alive*s wird dabei im Feld *Keep-Alive Count* der *Source Information Option* gespeichert, der zeitliche Abstand zwischen ihnen im Feld *Inter-Packet Time*. Da nun der Abstand zwischen den *Maintenance Keep-Alive*s immer erhöht werden soll (in dieser Implementierung immer um den Faktor zwei) und auch der größte Abstand zwischen zwei *Maintenance Keep-Alive*s, dem vorletzten und dem letzten, noch in dem Feld *Inter-Packet Time* gespeichert werden muss, können bei großen Abständen zwischen den Datenpaketen nur sehr wenige *Maintenance Keep-Alive*s gesendet werden. Der maximale Wert für die *Inter-Packet Time* beträgt 1023 Millisekunden. Sollte im Extremfall nur einer verschickt werden können und dieser auf Grund von ungünstigen Verhältnissen im Netzwerk verloren gehen, kommt es zu dem Problem, dass die Weiterleiter und Empfänger nicht mehr merken, dass der Sender aufgehört hat zu senden und mit eigentlich unnötigen Reparaturen beginnen werden.

Um den Wert für das Feld *Keep-Alive Count* zu berechnen benutzt diese ADMR-Implementierung die folgenden Ungleichungen. Sei die aktuelle *Inter-Packet Time* eines Senders t_i, der Faktor mit dem der Abstand zwischen den *Maintenance Keep-Alive*s erhöht wird m und die Anzahl der zu verschickenden *Maintenance Keep-Alive*s c, dann gilt:

$$m^c \cdot t_i \quad \leq 1023$$

$$\Leftrightarrow \quad m^c \quad \leq \frac{1023}{t_i}$$

$$\Leftrightarrow \quad c \quad \leq \frac{log(\frac{1023}{t_i})}{log(m)}$$

Um diese Ungleichung benutzen zu können muss ein Sender eine gute Annahme über die *Inter-Packet Time* t_i machen. Sie sollte sich nicht nur aus dem Abstand zwischen den beiden letzten Datenpaketen ergeben, sondern die „Vorgeschichte“ berücksichtigen. Bei unregelmäßig sendenden Quellen empfiehlt es sich, eine Art Durchschnitt über die letzten Pakete zu bilden. Diese Implementierung erreicht das gewünschte Verhalten durch die folgende, an der Berechnung der TCP-Round-Trip-Time orientierten, Formel (l = Abstand zwischen dem letzten und dem vorletzten Paket):

$$t_{i_{neu}} = (0,8 \cdot t_{i_{alt}}) + (0,2 \cdot l)$$

Da zu Beginn kein Wert für $t_{i_{alt}}$ vorhanden ist, wird hierfür `DEFAULT_INTERPACKETTIME` aus `admr.h` benutzt. Für die Simulationen im folgenden Abschnitt wurde hierfür ein Wert von 1000 Millisekunden gewählt. Wird dieser Wert zu klein gewählt, kommt es zu Beginn, wenn erst wenige Pakete verschickt wurden, zu sehr vielen Reparaturen, da auf Grund der geringen *Inter-Packet Time* zu früh mit weiteren Paketen gerechnet wird.

Ein weiterer Punkt, der beachtet werden sollte, ist die Tatsache, dass ADMR die ersten Pakete in den *Send Buffer* legt, bis die Weiterleitungsgruppe vermutlich aufgebaut ist. Zusammen mit der in [Jet01b] vorgeschlagenen und auch hier verwendeten Duplikatserkennung kann dies zu Paketverlusten führen. [Jet01b] schreibt als Vorschlag für die Duplikatserkennung in Punkt 5.3:

> „*A bitmap representing a number of previous sequence numbers of packets from this sender. The sequence number and bitmap are used to detect and discard duplicate packets during a flood: if the bit corresponding to some sequence number in this bitmap is set, the packet is assumed to be a duplicate; all sequence numbers prior to that corresponding to the first bit in the bitmap are also assumed to be duplicates (or are of no further interest and are discarded).*“

Diese Implementierung benutzt eine Bitmap mit zehn Einträgen. Das bedeutet also, dass für die letzten zehn Sequenznummern gespeichert wird, ob die dazugehörigen Pakete empfangen wurden oder nicht. Alle Pakete mit kleineren Sequenznummern werden als Duplikat behandelt. Werden nun aus dem *Send Buffer* mehr als zehn Pakete verschickt, deren Reihenfolge sich auf dem Weg vom Sender zum Empfänger ja durchaus ändern kann, könnte ein Knoten beispielsweise erst Paket 15 und direkt danach Paket 4 erhalten. Paket 4 würde in diesem Fall unnötigerweise verworfen werden.

Minimieren ließe sich dieses Problem selbstverständlich durch eine größere Bitmap, wenngleich das Problem damit im Grundsatz nicht gelöst wäre. Bei entsprechend vielen Paketen könnte es noch immer auftreten. In der Praxis tritt das Problem allerdings ohnehin nur in dem beschriebenen Fall beim Senden aus dem *Send Buffer* und einer besonders hohen Senderate, wenn viele Pakete im *Send Buffer* liegen, auf. Zu bedeutenden Einschränkungen in der Funktionsweise von ADMR kommt es somit nicht.

3.2.8 Statistiken

GloMoSim ist in der Lage am Ende eines Simulationslaufs Statistiken in eine Datei (`glomo.stat`) zu schreiben. Dies macht sich ADMR zu Nutze und hinterlegt seine eigenen Statistiken ebenfalls dort. Für jeden einzelnen Knoten werden die folgenden Werte erfasst:

- Anzahl der gesendeten[6] Kontrollpakete (ohne *Ack*-Pakete)
- Gesamtgröße der gesendeten Kontrollpakete (ohne *Ack*-Pakete) in Bytes
- Anzahl der weitergeleiteten Kontrollpakete (ohne *Ack*-Pakete)
- Gesamtgröße der weitergeleiteten Kontrollpakete (ohne *Ack*-Pakete) in Bytes
- Anzahl unterschiedlicher gesendeter Kontrollpakettypen
- Anzahl der gesendeten *Ack*-Pakete
- Gesamtgröße der gesendeten *Ack*-Pakete in Bytes
- Anzahl der gesendeten Datenpakete
- Gesamtgröße der gesendeten Datenpakete in Bytes
- Anzahl der empfangenen[7] Datenpakete
- Gesamtgröße der empfangenen Datenpakete in Bytes
- Anzahl der weitergeleiteten Datenpakete
- Gesamtgröße der weitergeleiteten Datenpakete in Bytes
- Anzahl lokaler Reparaturen (vgl. Abschnitt 2.4.4)

[6] gesendet bedeutet in diesem Zusammenhang immer *„von diesem Knoten erzeugt“*, empfangene und dann wieder versendete Pakete fallen unter „weitergeleitet“

[7] empfangene Pakete bezieht sich in diesem Zusammenhang auf die Pakete, die von Empfängern an die Transportschicht weitergegeben wurden. Pakete die von reinen Weiterleitern empfangen und dann wieder verschickt wurden zählen nicht dazu.

- Anzahl globaler Reparaturen (vgl. Abschnitt 2.4.4)

Zu jedem der obigen Punkte ist in der Statistikdatei vermerkt, dass es sich um einen von ADMR erzeugten Eintrag handelt. Eine Zeile der Datei kann dann beispielsweise das folgende Aussehen haben:

```
Node: 9, Layer: RoutingADMR, number of data-packets received: 3047
```

Mit Hilfe der so gewonnenen Informationen lassen sich komplexere Werte - wie z. B. der Prozentsatz der insgesamt empfangenen Pakete - berechnen. Auf Grund der einheitlichen Struktur der als reinem Text vorliegenden Statistikdatei kann dies auch in großem Umfang recht einfach geschehen.

Für die vorliegende Arbeit wurden hierfür PERL-Skripte verwendet. Da PERL reguläre Ausdrücke beherrscht, ließen sich so die benötigten Werte leicht extrahieren.

4 Simulationen

Um die Funktionsweise von ADMR bewerten zu können wurden mit GloMoSim verschiedene Szenarien simuliert. Dazu wurden die Messungen aus [Jet01a] und [LSHGB] nachgestellt. Um Vergleichswerte zu erhalten, wurden diese Messungen auch für das Routing Protokoll ODMRP durchgeführt.

Zunächst werden die verwendeten Metriken definiert. Für einfachere Vergleiche mit [Jet01a] oder [LSHGB] sind auch die jeweiligen englischen Begriffe angegeben. Da die ODMRP-Implementierung von GloMoSim einige Probleme aufweist, folgt ein Abschnitt über ODMRP, in dem auf die Probleme eingegangen wird, die sich durch den Einsatz dieser Implementierung ergeben. Im Anschluss daran erfolgt die Beschreibung und Auswertung der Simulationen.

4.1 Metriken

Definition 1: ***Paketankunftsrate*** (engl.: packet delivery ratio) *Verhältnis von der Anzahl der von den Empfängern tatsächlich empfangen Pakete zu der Anzahl der Pakete die hätten empfangen werden müssen.*

Die *Paketankunftsrate* lässt sich am besten an einem Beispiel verdeutlichen. Sendet ein Sender insgesamt 100 Pakete und existieren zwei Empfänger, von denen der eine 90 und der andere 60 Pakete empfängt, so ergibt sich eine *Paketankunftsrate* von 0,75. Die Empfänger hätten insgesamt 200 Pakete empfangen können, haben davon aber nur 150 erhalten, $((90 + 60)/(2 * 100) = 0,75)$.

Definition 2: ***Verschickte Pakete pro empfangenem Datenpaket*** (engl.: normalized packet overhead [Jet01a] *oder* total packets transmitted per data packet delivered [LSHGB]) *Die Anzahl aller versendeten (Kontroll- und Daten-)Pakete geteilt durch die Anzahl aller empfangenen Datenpakete. Bei den versendeten Paketen zählt jedes Versenden eines Pakets, bei Datenpaketen also nicht nur das Versenden beim Sender, sondern auch jedes Versenden durch Weiterleiter. Als empfangenen Pakete zählen diejenigen, die erfolgreich an die Applikationsschicht ausgeliefert wurden.*

Definition 3: ***Weiterleitungseffektivität*** (engl.: forwarding efficiency) *Die durchschnittliche Anzahl der Weiterleitungen eines Datenpakets. Sie berechnet sich also aus der Gesamtanzahl der Weiterleitungen dividiert durch die Anzahl der durch Sender verschickten Datenpakete.*

Definition 4: ***Verschickte Datenpakete pro empfangenem Datenpaket*** (engl.: Number of data packets transmitted per data packet delivered) *Ähnlich wie die* Weiterleitungseffektivität*, jedoch mit dem Unterschied, dass nicht nur die Weiterleitungen eines Datenpakets sondern auch das Versenden beim Sender gezählt wird. Außerdem wird nicht durch die Anzahl der versendeten, sondern durch die Anzahl der empfangenen Datenpakete geteilt.*

Definition 5: ***Kontrolloverhead*** (engl.: Number of control bytes transmitted per data byte delivered) *Die Anzahl der Kontrollbytes (versendet und weitergeleitet) geteilt durch die Anzahl der an die Applikationsschicht ausgelieferten Datenbytes. Zu den Kontrollbytes zählen dabei nicht nur die reinen Kontrollpakete, sondern auch die Header eines Datenpakets, da auch diese Kontrollinformationen enthalten.*

Für die Messung der Kontrollbytes wurden der IP-Header, der MAC-Header, usw. nicht mitgezählt. Das sind zwar auch Daten die bei reinen Kontrollpaketen zu einer zusätzlichen Netzwerkbelastung führen und auf Kontrollaktivitäten des Routing Protokolls zurückzuführen sind, deren Größe ist jedoch nicht vom Routing Protokoll beeinflussbar. Es wurden somit nur die Kontrollbytes, die das Routing Protokoll direkt verwendet, gezählt.

4.2 ODMRP

Für die Simulationen wurde eine Version von ODMRP benutzt, die zwar auf der in GloMoSim mitgelieferten aufbaut, jedoch vom Institut für Telematik an der Universität Karlsruhe (TH) an einigen Stellen ergänzt wurde. Das Intervall in dem *Join Query*s geschickt werden lässt sich in der GloMoSim-Konfigurationsdatei über den Parameter `ODMRP_JR_REFRESH-INTERVAL` einstellen. Nach welcher Zeit ein Knoten die Weiterleitungsgruppe verlässt gibt `ODMRP_FG_TIMEOUT-INTERVAL` an. Wie in [Jet01a] wurden hierfür Werte von drei bzw. neun Sekunden gewählt.

Obwohl die ODMRP-Implementierung von GloMoSim in [LSG00] (Abschnitt 6) als eine Art „Referenzimplementierung“ angegeben ist, gibt es teilweise deutliche Unterschiede zur Spezifikation in [LSG00]. Die Implementierung benutzt beispielsweise IP-Optionsfelder (`OdmrpIpOptionType`), die im Draft keine Erwähnung finden. Außerdem werden *Ack*-Pakete verschickt, die ebenfalls nicht im Draft beschrieben sind.

Die weiter vorne gegebene Beschreibung orientiert sich zwar an dem Draft, zur Simulation wurde allerdings trotzdem die erwähnte ODMRP-Implementierung benutzt - auch wenn es hier einige Abweichungen gibt. Aus diesem Grund gibt es

für die Statistiken über die Anzahl der Kontrollbytes im Folgenden immer zwei Werte: Einmal wurden die in [LSG00] angegebenen Pakete mit den dort aufgeführten Größen gezählt (ODMRP-draft). Der zweite Wert berechnet sich aus den in der Implementierung benutzten Paketgrößen und allen dort tatsächlich benutzten Paketen (ODMRP-Impl.). Zusätzlich zu den *Join Query*- und *Join Reply*-Paketen werden für *ODMRP-Impl.* also auch die *Ack*-Pakete und die IP-Optionen gezählt.

4.3 Szenarien nach [Jet01a]

ADMR wird in [Jet01a] vorgestellt und es werden die Ergebnisse von Simulationen zu zwei Szenarien präsentiert. Dabei wird ADMR jeweils ODMRP gegenüber gestellt. Um zu überprüfen, ob die für diese Arbeit angefertigte Implementierung korrekt ist, wurden diese Szenarien nachgestellt.

Bei den Szenarien handelt es sich einmal um eine Gruppe mit einem Sender und 15 Empfängern, sowie im anderen Fall um drei Gruppen mit je 10 Empfängern. Eine Übersicht über die Konfigurationen der verschiedenen Szenarien zeigt Tabelle 4. Alle Szenarien wurden jeweils mit zehn verschiedenen Initialisierungswerten für den Zufallszahlengenerator von GloMoSim simuliert. Aus den einzelnen Resultaten wurde dann der Durchschnitt gebildet. Der Initialisierungswert des Zufallszahlengenerators wird GloMoSim über den Parameter `SEED` in der Konfigurationsdatei übergeben. Es wurden hier die ganzzahligen Werte 1 bis 10 gewählt.

4.3.1 Konfiguration

GloMoSim wurde für diese Simulationen wie in [Jet01a] beschrieben konfiguriert. Das bedeutet, dass als MAC-Protokoll 802.11 mit einer Übertragungsrate von 2Mbps verwendet wurde. Die Reichweite der einzelnen Knoten betrug 250 Meter.

50 Knoten wurden über einen Zeitraum von 900 Sekunden simuliert. Diese wurden zufällig auf einem 1500m $\times$ 300m großen Gebiet platziert. Als Mobilitätsmodell wurde `RANDOM-WAYPOINT` verwendet. Das bedeutet, dass sich jeder Knoten zunächst für einen *Pausenzeit* (engl.: pause time) genannten Zeitraum nicht bewegt. Danach wählt er einen beliebigen Punkt in der Simulationsfläche und bewegt sich mit einer gleichmäßigen Geschwindigkeit dorthin. Hierfür wählt er eine Geschwindigkeit zwischen 0 m/s und 20 m/s. Für die *Pausenzeit* wurden die Werte 0, 30, 60, 120, 300, 600 und 900 Sekunden gewählt.

Die Sender beginnen sofort damit, Daten an die Multicast-Gruppe zu verschicken. Die einzelnen Empfänger treten der Gruppe jedoch erst zu einem Zeitpunkt zwischen 0 und 180 Sekunden nach dem Beginn der Simulation bei. Diese Verzögerung wurde erfasst und konnte somit beispielsweise bei der Berechnung der *Paketankunftsrate* berücksichtigt werden.

Als Programm der Anwendungsschicht wurde CBR (constant bit rate) ge-

Szenario	Konfiguration	
1	Gruppen:	1
	Sender:	1
	Empfänger:	15
2	Gruppen:	3
	Sender:	je 3
	Empfänger:	je 10
1 & 2	Knoten:	50
	Zeitraum:	900 Sekunden
	Gebiet:	1500m × 300m
	Mobilitätsmodell:	`RANDOM-WAYPOINT`
	Geschwindigkeit:	zufällig, 0 - 20 m/s
	Pausenzeit:	0, 30, 60, 120, 300, 600 und 900 Sekunden
	Traffic:	CBR, vier Pakete à 64 Bytes pro Sekunde
	MAC-Protokoll:	802.11
	Übertragungsrate:	2Mbps
	Reichweite:	250 Meter

Tabelle 4: Szenarien nach [Jet01a]

wählt. Jeder Sender verschickt 4 Pakete pro Sekunde, die jeweils eine Größe von 64 Bytes haben.

Die Konfiguration von ODMRP entspricht `ODMRP-baseline` aus [Jet01a].

4.3.2 Abweichungen von [Jet01a]

Bei der Simulation der Szenarien nach [Jet01a] stellte sich heraus, dass es signifikante Abweichungen zwischen den Resultaten der für diese Arbeit erstellten Implementierung und den Messungen aus [Jet01a] gibt. Beispielsweise ist die *Paketankunftsrate* der Messung von [Jet01a] deutlich höher als bei der vorliegenden Implementierung. Dies zeigen die Abbildungen 14 und 15. Auch bei der Anzahl der *verschickten Pakete pro empfangenem Datenpaket* (Abbildungen 17 und 18) sowie bei der *Weiterleitungseffektivität* (Abbildungen 20 und 21) gibt es deutliche Unterschiede.

Bei der Suche nach der Ursache für diese Abweichungen fiel auf, dass [Jet01a] in Abbildung 17 für die Anzahl der *verschickten Pakete pro empfangenem Datenpaket* bei hohen Pausenzeiten Werte von unter eins erreicht. Da aber nach der Spezifikation von [Jet01b] jedes empfangene Paket zum Versenden eines Pakets führt - es wird entweder ein *Ack*-Paket gesendet oder das Paket wird weitergeleitet - sind bei korrekter Messung hierfür nur Werte > 1 möglich.

Auffällig ist auch, dass in [Jet01a] bei der Beschreibung von ADMR keine *Ack*-Pakete erwähnt werden. Es ist daher durchaus denkbar, dass zwischen der für

[Jet01a] benutzten Spezifikation von ADMR und der für diese Arbeit verwendeten [Jet01b] Unterschiede bestehen. Um eine Erklärung für die beobachteten Abweichungen zu erhalten wurde die Implementierung für diese Arbeit beim Versenden der *Ack*-Pakete verändert. Ziel war es, der in [Jet01a] benutzten Implementierung näher zu kommen. Statt die Nutzdaten für das *Ack*-Paket zu entfernen wurde das *Ack*-Paket inklusive der Nutzdaten verschickt. Das *Ack*-Paket ist somit identisch mit dem Datenpaket, was dazu führt, dass jeder Empfänger automatisch auch ein Weiterleiter ist. Wie Abbildung 16 zeigt, ergibt sich damit eine deutlich höhere *Paketankunftsrate*, die auch mit der Messung aus [Jet01a] übereinstimmt. Dadurch, das jeder Empfänger zwangsläufig auch Weiterleiter ist, müssen seltener Reparaturen eingeleitet werden, da schon durch das Verschicken der *Ack*-Pakete die Datenpakete ohnehin von mehr Knoten empfangen werden. Das erklärt die gemessenen höheren *Paketankunftsraten*.

Die durch diese Veränderung erzielte Übereinstimmung bei der *Paketankunftsrate* legt die Vermutung nahe, dass die so veränderte Implementierung mit der von [Jet01a] übereinstimmt. Allerdings lassen sich so die Werte unter eins bei den *verschickten Paketen pro empfangenem Datenpaket* noch nicht erklären. Daher wurden testweise die *Ack*-Pakete für die Erstellung der Statistiken nicht berücksichtigt. Das Ergebnis dieser Messungen zeigt Abbildung 19. Die Werte zeigen eine deutliche Übereinstimmung mit den Werten von [Jet01a] in Abbildung 17. Mit diesem Verfahren stimmen auch die Werte der *Weiterleitungseffektivität* (Abbildungen 20 und 22) überein. Das zeigt, wie in den folgenden Abschnitten noch näher beschrieben wird, dass die *Ack*-Pakete eine erheblichen Anteil an den verschickten Paketen haben.

Es wird daher im Folgenden davon ausgegangen, dass es bei der Behandlung der *Ack*-Pakete Unterschiede zwischen [Jet01b] und der in [Jet01a] verwendeten Spezifikation gibt und das es sich bei der für diese Arbeit erstellten Implementierung um eine korrekte Umsetzung von [Jet01b] handelt. Auf Grund der deutlichen Übereinstimmung zwischen [Jet01a] und der angepassten Version der für diese Arbeit erstellten Implementierung bei allen drei Messreihen erscheint dieser Ansatz als sehr wahrscheinlich. Ob die in [Jet01a] verwendete Implementierung tatsächlich nur die genannten Abweichungen enthält lässt sich nicht eindeutig klären.

4.3.3 Szenario 1

Die Resultate dieses Szenarios wurden bereits im vorangehenden Abschnitt vorgestellt. Hier soll jetzt eine genauere Analyse der Ergebnisse stattfinden. Es handelt sich um Simulationen, bei denen ein Sender alle 250 Millisekunden ein Paket an eine Multicast-Gruppe mit 15 Empfänger sendet. Die Knoten bewegten sich nach dem `RANDOM-WAYPOINT`-Modell mit einer Geschwindigkeit zwischen 0 und 20 m/s.

Abbildung 15 zeigt den Verlauf der *Paketankunftsrate* bei sinkender Mobili-

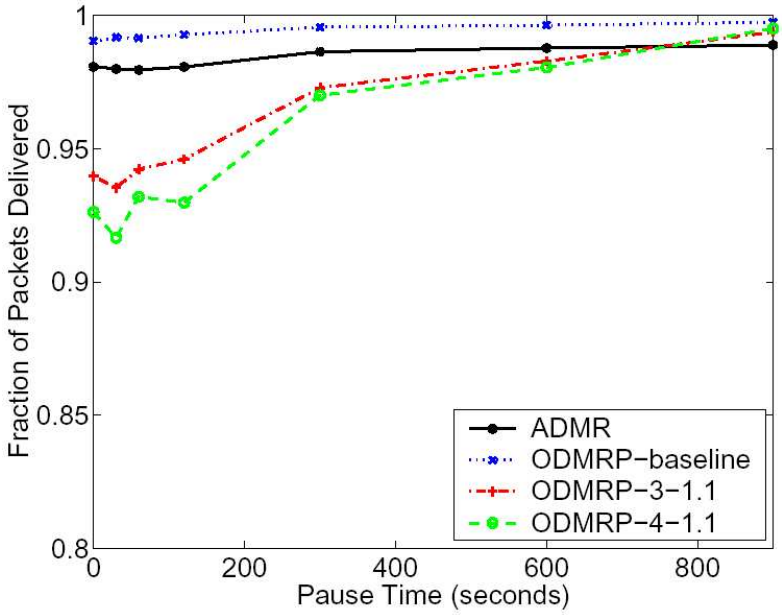

Abbildung 14: Paketankunftsrate, Szenario 1, aus [Jet01a]

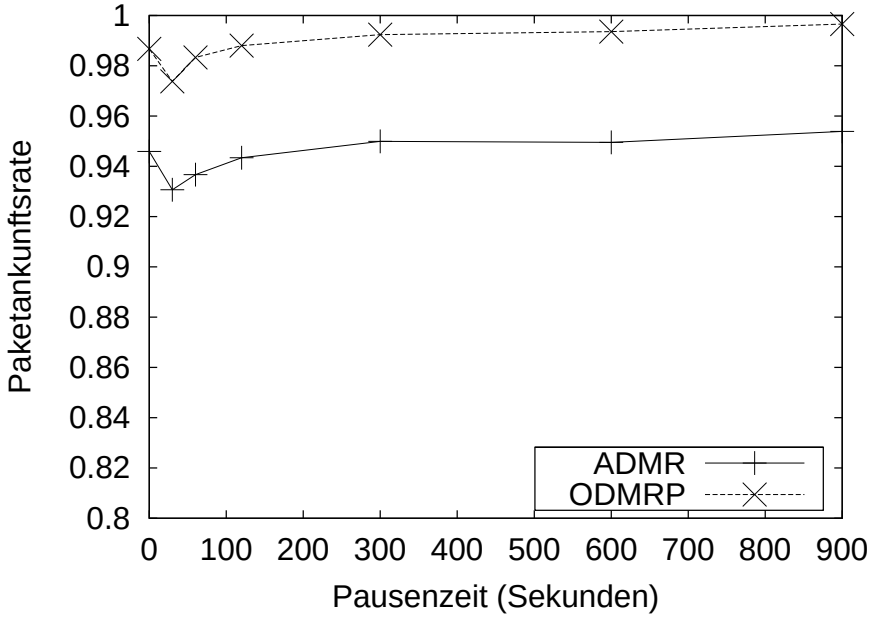

Abbildung 15: Paketankunftsrate, Szenario 1

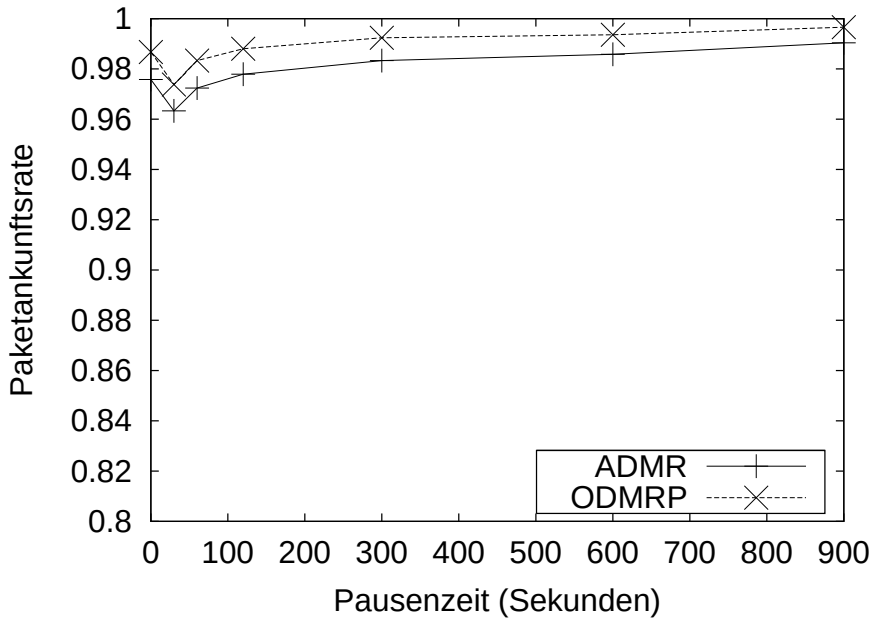

Abbildung 16: Paketankunftsrate, Szenario 1, Implementierung dieser Arbeit (mit fehlerbehafteten *Ack*-Paketen)

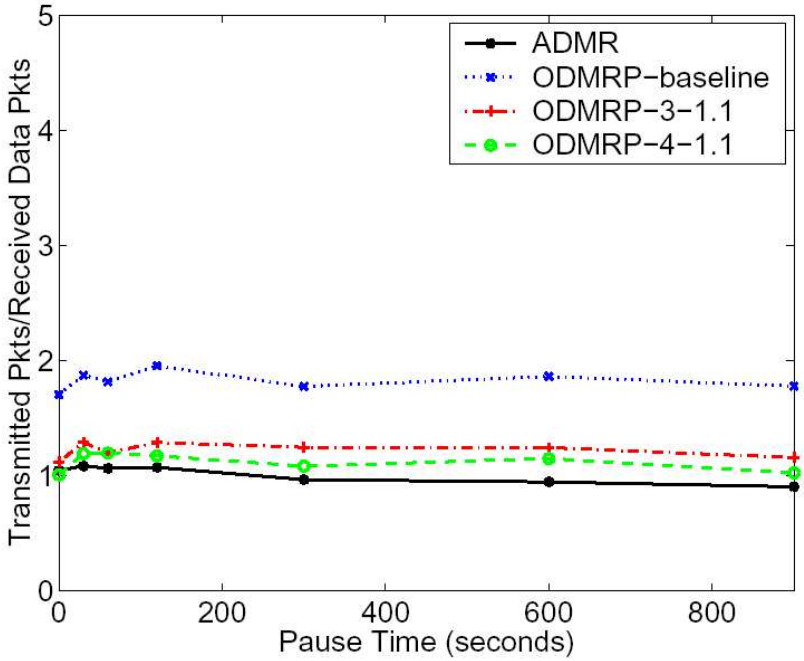

Abbildung 17: verschickte Pakete pro empfangenem Datenpaket, Szenario 1, aus [Jet01a]

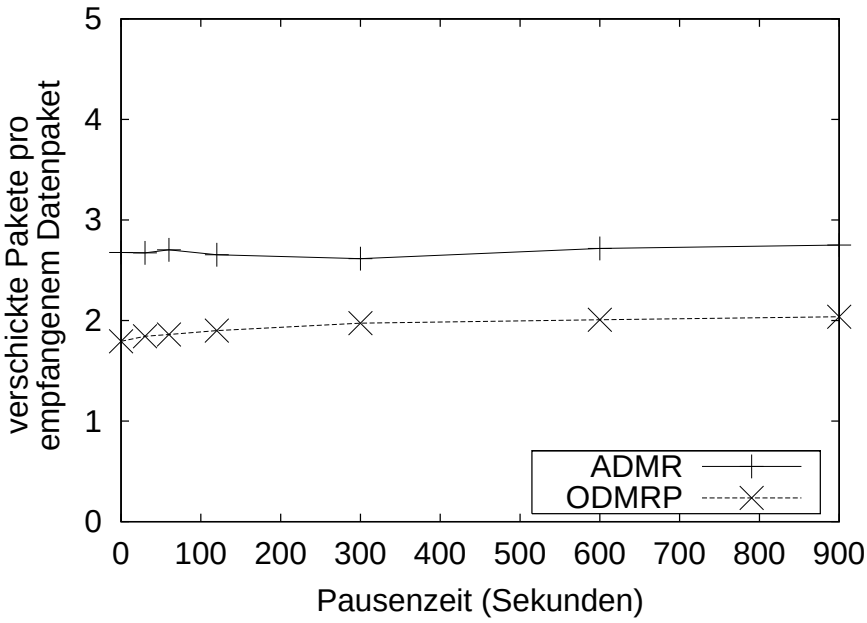

Abbildung 18: verschickte Pakete pro empfangenem Datenpaket, Szenario 1

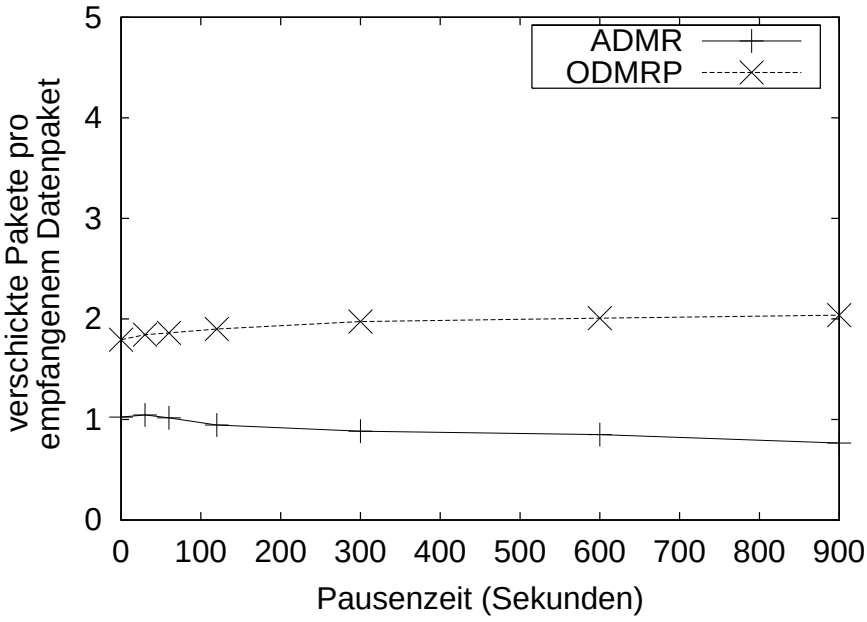

Abbildung 19: verschickte Pakete pro empfangenem Datenpaket, Szenario 1, Implementierung dieser Arbeit (mit fehlerbehafteten *Ack*-Paketen)

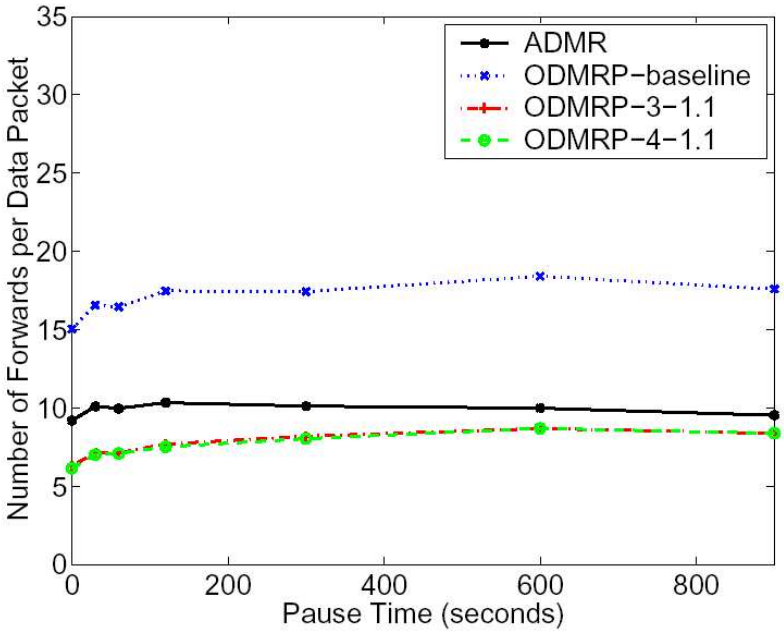

Abbildung 20: Weiterleitungseffektivität, Szenario 1, aus [Jet01a]

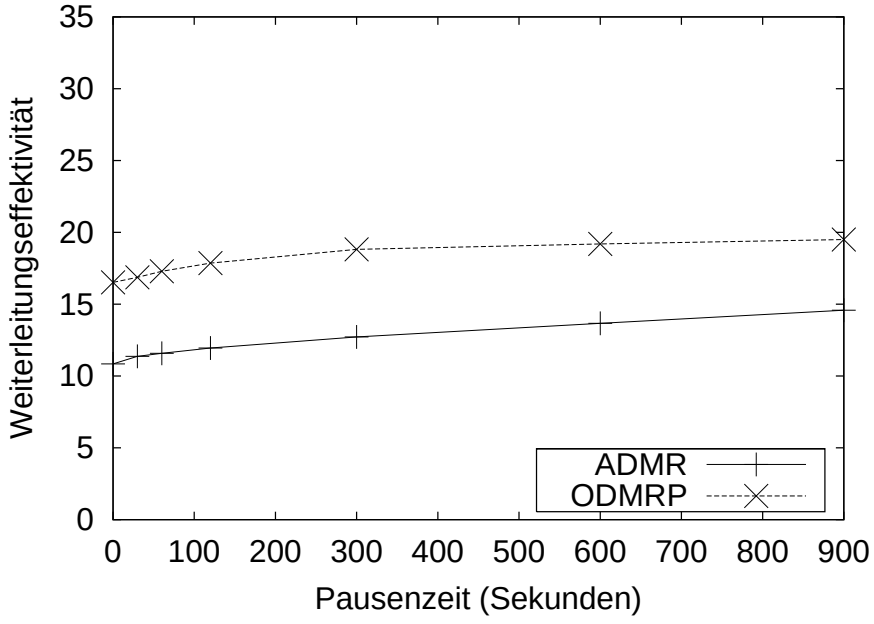

Abbildung 21: Weiterleitungseffektivität, Szenario 1

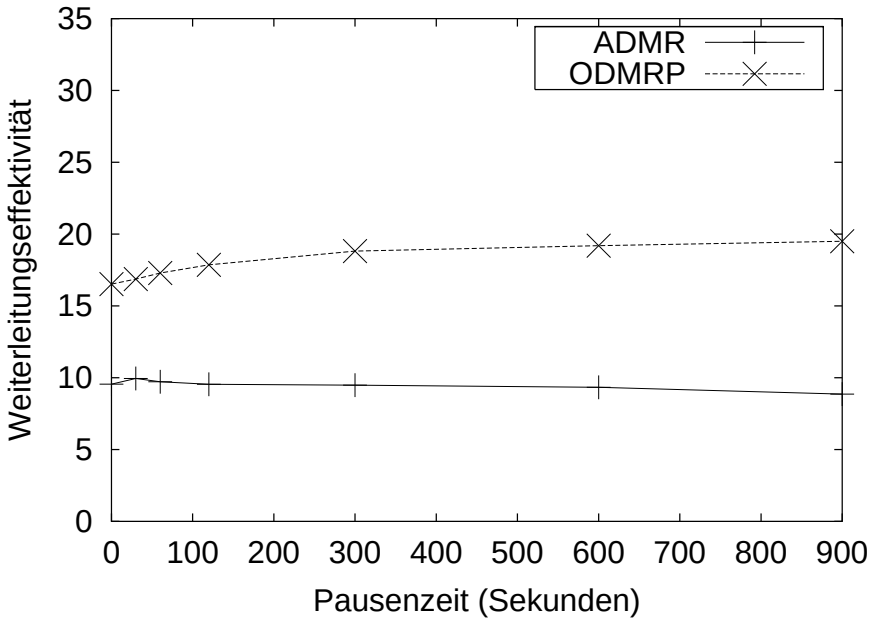

Abbildung 22: Weiterleitungseffektivität, Szenario 1, Implementierung dieser Arbeit (mit fehlerbehafteten *Ack*-Paketen)

tät. Zunächst fällt hierbei auf, dass ODMRP im gesamten Bereich bessere Werte als ADMR erzielt.

Da ein ADMR-Empfänger für jedes nicht-weitergeleitete empfangene Datenpaket ein *Ack*-Paket versendet, müssen insgesamt deutlich mehr Pakete als bei ODMRP verschickt werden, was zu deutlich mehr Kollisionen führt. Eine Messung zur Anzahl der Kollisionen aus einem Szenario von [LSHGB] bestätigt diese Annahme (vgl. Abbildung 30, Seite 58). Auch die *verschickten Pakete pro empfangenem Datenpaket* (Abbildung 18) zeigen, dass ADMR pro empfangenem Datenpaket mehr Pakete verschicken muss. Da Kollisionen zu Paketverlusten führen erreichen somit weniger Pakete ihre Empfänger, was die geringeren Werte von ADMR erklärt. Wie bereits im vorangehenden Abschnitt betrachtetet, liegen die Werte der *Paketankunftsrate* deutlich unter den Werten aus [Jet01a]. Dies kommt dadurch zustande, dass die *Ack*-Pakete nicht mehr zur Weiterleitung der Daten beitragen.

Der Einbruch der Werte bei 30 Sekunden ergibt sich durch ungewöhnlich starke Ausreißer bei den zehn verschiedenen Messwerten. Hier gibt es offensichtlich Schwächen im Zufallszahlengenerator von GloMoSim, da die Ausreißer sowohl bei ADMR als auch bei ODMRP auftreten und zudem beim gleichen Initialisierungswert. Berechnet man die *Paketankunftsrate* ohne diesen Ausreißer, so erhält man bei 30 Sekunden für ADMR einen Wert von 0,9455 und für ODMRP 0,9864. Damit ergeben sich deutlich glattere Kurvenverläufe.

Bei der in Abbildung 21 dargestellten *Weiterleitungseffektivität* erreicht ADMR deutlich geringere Werte als ODMRP. Da also die Datenpakete insgesamt weniger oft weitergeleitet werden, deutet dies auf den Aufbau einer kleineren Weiterleitungsgruppe als bei ODMRP hin. ADMR erzeugt weniger Redundanz, was sich allerdings negativ in der Anzahl der ausgelieferten Datenpakete niederschlägt. ADMR baut durch ein *Receiver Join*-Paket genau einen Pfad zwischen Sender und Empfänger auf. Im Gegensatz dazu bleibt bei ODMRP die alte Struktur bestehen, wenn durch ein neues *Join Query*/*Join Reply*-Paar schon ein neuer Pfad zwischen Sender und Empfänger aufgebaut wird. Trotz der besseren *Weiterleitungseffektivität* ist die Belastung des Netzes bei ADMR im Vergleich zu ODMRP allerdings nicht geringer. ADMR versendet die Datenpakete zwar nicht so häufig, dafür werden aber deutlich mehr Kontrollpakete verschickt. Abbildung 18 zu den *verschickten Paketen pro empfangenem Datenpaket* zeigt, dass ADMR pro empfangenem Datenpaket mehr Pakete als ODMRP verschicken muss.

Der allgemeine Anstieg der *Weiterleitungseffektivität* bei sinkender Mobilität ist auf ein Phänomen des Mobilitätsmodells `RANDOM-WAYPOINT` zurück zu führen [BeWa02]. Bei höherer Mobilität führt `RANDOM-WAYPOINT` dazu, dass sich die Knoten nach einiger Zeit eher in der Mitte des Simulationsgebiets als am Rand aufhalten. Durch diese „Konzentration“ der Knoten in der Mitte des Gebiets müssen im Schnitt geringere Entfernungen überbrückt werden. Die Pakete müssen nicht mehr so häufig weitergeleitet werden und es ergeben sich bei hoher Mobilität geringere Werte für die *Weiterleitungseffektivität*.

4.3.4 Szenario 2

Für diese Messungen gab es drei Multicast-Gruppen mit jeweils drei Sendern und zehn Empfängern, an die wiederum alle 250 Millisekunden Pakete verschickt wurden.

Die Messung der *Paketankunftsrate* in Abbildung 23 zeigt, dass im Verhältnis weniger Pakete als im vorangegangenen Szenario ankommen. Es werden allerdings auch mehr Pakete verschickt, was zu einer größeren Belastung des Netzes und damit zu mehr Paketverlusten führt.

ADMR kommt mit hoher Mobilität offenbar schlecht zurecht. Wird die Entfernung zwischen zwei Knoten der Weiterleitungsgruppe zu groß, dauert es einen Moment bis ADMR dies bemerkt und eine Reparatur vornimmt. Bis die Reparatur eingeleitet wird ist aber schon mindestens ein Datenpaket verloren gegangen. ODMRP baut im Gegensatz dazu die Weiterleitungsgruppe mit jedem *Join Query* neu auf und kann so Paketverluste u. U. schon im Vorfeld verhindern. Außerdem bleibt bei ODMRP die alte Weiterleitungsgruppe noch für eine gewisse Zeit bestehen, was durch die so entstehende Redundanz auch zu einer zuverlässigeren Weiterleitung der Datenpakete beiträgt.

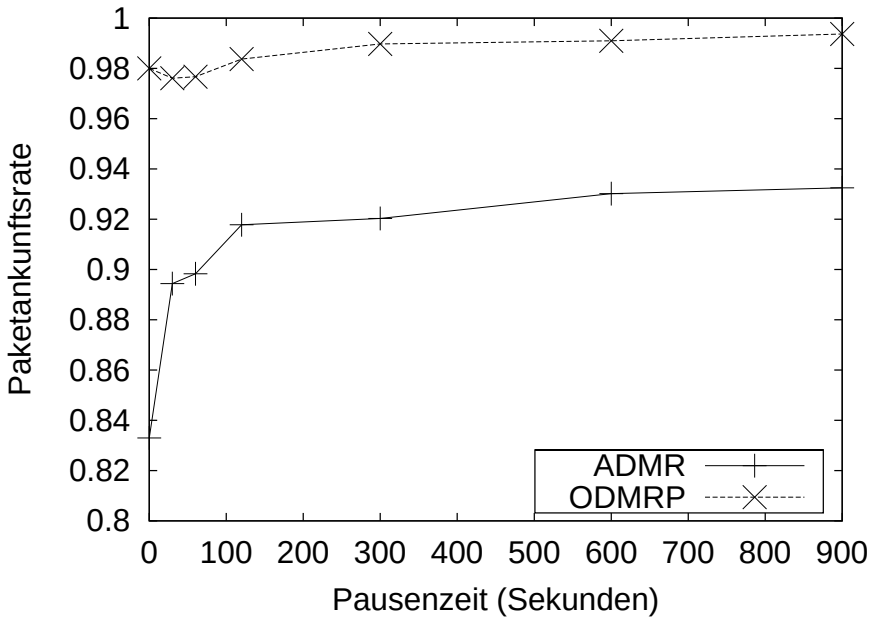

Abbildung 23: Paketankunftsrate, Szenario 2

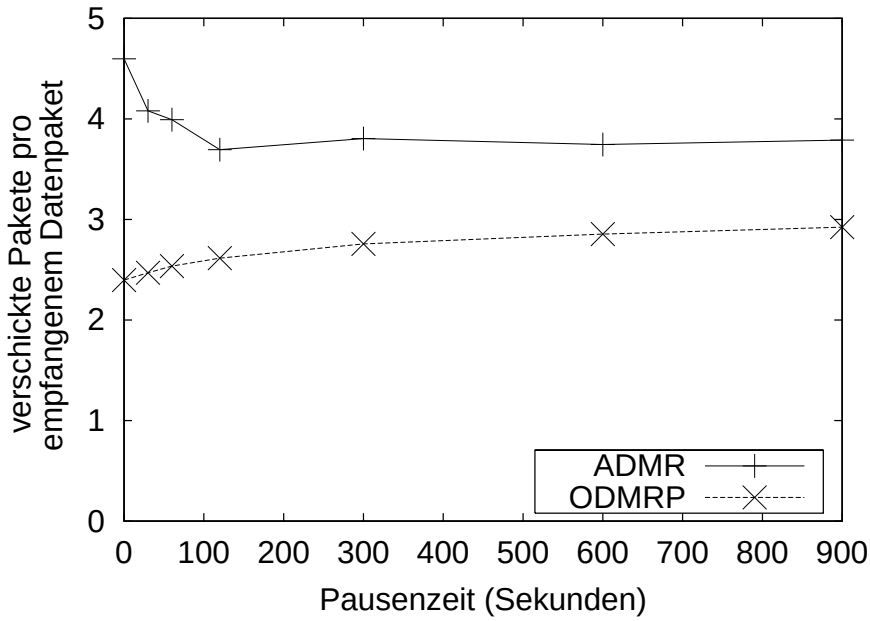

Abbildung 24: verschickte Pakete pro empfangenem Datenpaket, Szenario 2

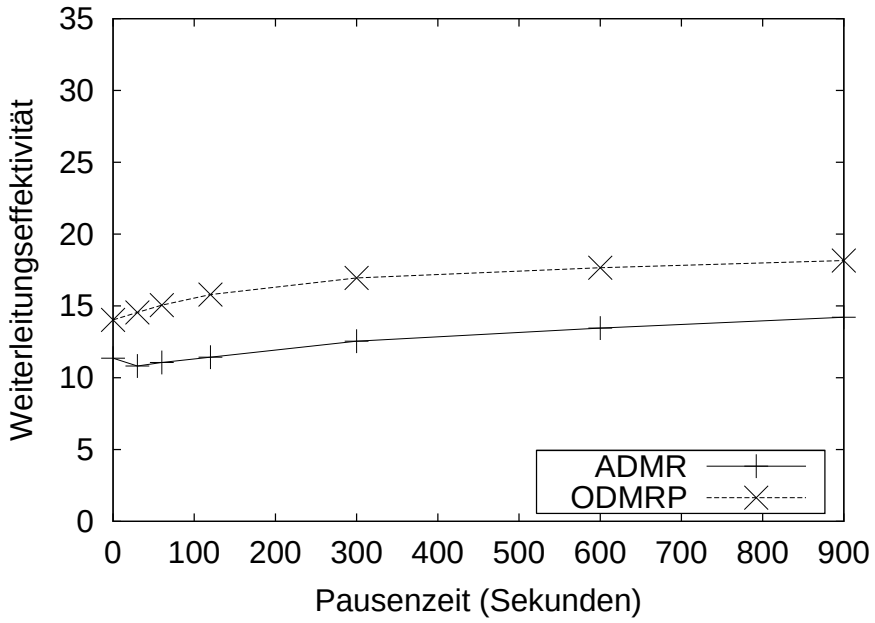

Abbildung 25: Weiterleitungseffektivität, Szenario 2

Bei den *verschickten Paketen pro empfangenem Datenpaket* in Abbildung 24 hat ADMR, wie schon bei der Konfiguration mit einer Multicast-Gruppe, deutlich schlechtere Werte als ODMRP. Die besonders hohen Werte bei geringer Pausenzeit ergeben sich zum einen direkt aus der schlechten *Paketankunftsrate*. Da für die *verschickten Pakete pro empfangenem Datenpaket* durch die Anzahl der erfolgreich empfangenen Pakete dividiert wird, ist dieser Wert direkt von der *Paketankunftsrate* abhängig. Werden wenig Pakete empfangen ergeben sich somit auch ohne sonstige Phänomene höhere Werte. Zum anderen führt eine hohe Mobilität zu mehr Reparaturen, also zu mehr Kontrollpaketen und damit zu mehr *verschickten Paketen pro empfangenem Datenpaket*.

Dass die Anzahl der *verschickten Pakete pro empfangenem Datenpaket* von ODMRP mit sinkender Mobilität ansteigt hängt mit den ebenfalls ansteigenden Werten der in Abbildung 25 dargestellten *Weiterleitungseffektivität* zusammen. Werden die Pakete häufiger weitergeleitet steigt auch die Anzahl der *verschickten Pakete pro empfangenem Datenpaket*. Der Anstieg der *Weiterleitungseffektivität* erklärt sich wie schon in Szenario 1 durch die Konzentration der Knoten in der Mitte des Simulationsgebiets [BeWa02].

4.4 Szenarien nach [LSHGB]

Um Vergleiche mit anderen Multicast Routing Protokollen ziehen zu können, wurden die in [LSHGB] simulierten Szenarien nachgestellt. Dort wurden ODMRP,

AMRIS, CAMP, AMRoute sowie ein einfaches Fluten des Netzes miteinander verglichen. Durch eine Simulation der dortigen Szenarien für ADMR ist somit eine Einordnung der Leistung von ADMR im Vergleich zu den anderen Routing Protokollen möglich. Für diese Arbeit wurden die Simulationen wie im vorhergehenden Abschnitt wieder für ADMR und ODMRP durchgeführt.

Die simulierten Szenarien setzen sich aus vier Typen zusammen. Variiert wurden: Die Geschwindigkeit der Knoten, die Anzahl der Sender, die Größe der Multicast-Gruppe und die Belastung des Netzes.

Last wird in [LSHGB] leider nicht exakt definiert. Für die vorliegende Arbeit wird daher die folgende Definition verwendet. Sie ist offenbar mit der in [LSHGB] benutzten identisch.

Definition 6: ***Last*** (engl.: network traffic load) *Die Anzahl der pro Zeiteinheit von allen aktiven Sender im Netz verschickten (= erzeugten) Datenpakete. Wird in dieser Arbeit in Paketen pro Sekunde gemessen.*

Wie bei den Simulationen zu [Jet01a] wurde auch hier immer der Durchschnitt von zehn Messungen mit jeweils unterschiedlichen Initialisierungswerten des Zufallszahlengenerators errechnet.

4.4.1 Konfiguration

Die Konfiguration für die Szenarien nach [LSHGB] ähnelt der aus Abschnitt 4.3. Wie auch Tabelle 5 zeigt, wurden wieder 50 Knoten verwendet, die jedoch über einen Zeitraum von 600 Sekunden auf einem Gebiet von 1000m × 1000m simuliert wurden. Die Reichweite der einzelnen Knoten betrug 250 Meter. Als Mobilitätsmodell wurde wieder `RANDOM-WAYPOINT` verwendet. Die Geschwindigkeit konnte jedoch nicht frei zwischen 0 m/s und 20 m/s gewählt werden sondern war für jedes Szenario fest vorgegeben. Die Größe der verschickten Pakete betrug 512 Bytes.

4.4.2 Szenario 3

Für dieses Szenario wurde die Geschwindigkeit, mit der sich die Knoten bewegen, variiert. Sie bewegten sich konstant, also mit einer *Pausenzeit* von 0 Sekunden. Die Geschwindigkeiten betrugen 0, 0.5, 1, 2, 5, 10 bzw. 20 Meter pro Sekunde. Es gab eine Multicast-Gruppe mit 20 Mitgliedern. Fünf dieser Mitglieder waren gleichzeitig Sender, die jeweils zwei Pakete pro Sekunde versendeten.

Abbildung 26 bestätigt die Vermutung aus Abschnitt 4.3, dass ADMR bei hoher Mobilität nur geringe *Paketankunftsraten* erzielt. Im Gegensatz zu ODMRP muss ADMR explizite Reparaturen vornehmen, die jedoch frühestens nach dem ersten Paketverlust eingeleitet werden können.

Außerdem macht sich schon hier bemerkbar, dass ADMR bei vielen Empfängern auf Grund der vielen *Ack*-Pakete und der damit verbundenen Kollisionen

Szenario	Konfiguration	
3	Sender:	5
	Empfänger:	20
	Geschwindigkeit:	0, 0.5, 1, 2, 5, 10 und 20 m/s
	Traffic (Last):	CBR, 10 Pakete à 512 Bytes pro Sekunde
4	Sender:	1, 2, 5, 10 und 20
	Empfänger:	20
	Geschwindigkeit:	1 m/s
	Traffic (Last):	CBR, 10 Pakete à 512 Bytes pro Sekunde
5	Sender:	5
	Empfänger:	5, 10, 20, 30 und 40
	Geschwindigkeit:	1 m/s
	Traffic (Last):	CBR, 10 Pakete à 512 Bytes pro Sekunde
6	Sender:	5
	Empfänger:	20
	Geschwindigkeit:	0 m/s (keine Mobilität)
	Traffic (Last):	CBR, 1, 2, 5, 10, 25 und 50 Pakete à 512 Bytes pro Sekunde
3, 4, 5 & 6	Knoten:	50
	Gruppen:	1
	Zeitraum:	600 Sekunden
	Gebiet:	1000m × 1000m
	Mobilitätsmodell:	`RANDOM-WAYPOINT`
	Pausenzeit:	0 Sekunden
	MAC-Protokoll:	802.11
	Übertragungsrate:	2Mbps
	Reichweite:	250 Meter

Tabelle 5: Szenarien nach [LSHGB]

Probleme hat. Dies wird später bei den Simulationen mit wechselnder Gruppengröße besonders deutlich. Bei einer Geschwindigkeit von 5 m/s ergibt die `RADIO-LAYER`-Statistik von GloMoSim für ADMR 1.143.616 Kollisionen, während bei ODMRP lediglich 406.063 Kollisionen auftreten, was lediglich 36% des Wertes von ADMR ist (vgl. Abbildung 30).

Die Werte aus Abbildung 27 zeigen das interessante Phänomen, dass ADMR pro empfangenem Datenpaket in etwa so viele Datenpakete verschickt wie ODMRP. Da bei ADMR aber weniger Datenpakete empfangen werden, werden insgesamt auch deutlich weniger Datenpakete verschickt als bei ODMRP. ADMR hat also offenbar eine kleinere Weiterleitungsgruppe. Diese wird nur bei einer expliziten Reparatur angepasst, während sie bei ODMRP durch jedes der periodisch

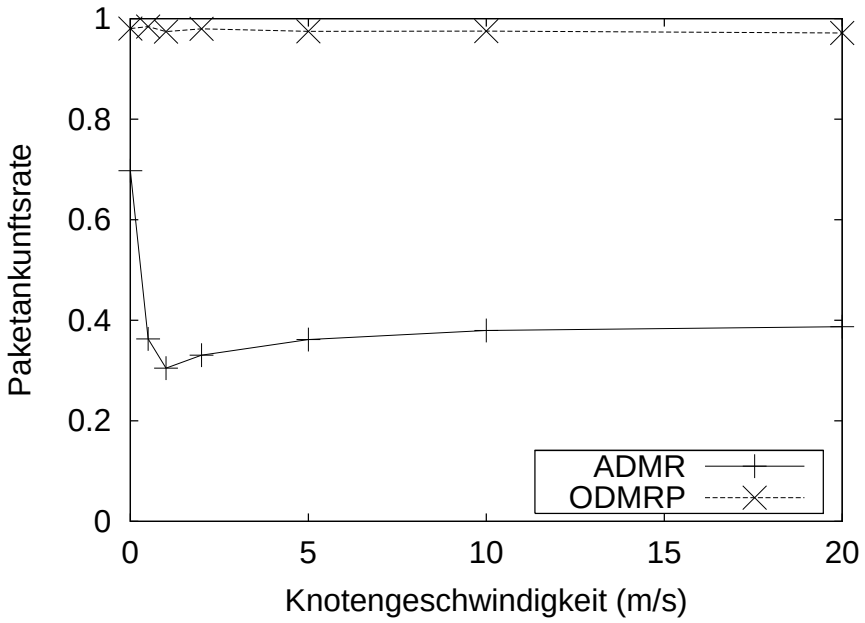

Abbildung 26: Paketankunftsrate, Szenario 3

verschickten *Join Query*-Pakete neu aufgebaut wird. Somit existieren häufiger als bei ADMR redundante Pfade, da bei ODMRP die alte Weiterleitungsgruppe noch für eine gewisse Zeit bestehen bleibt. Dies kann ein Indiz dafür sein, dass ADMR zu wenig Redundanz nutzt und daher schlechte *Paketankunftsraten* erzielt. Wie sich im Folgenden noch herausstellt, scheinen die *Ack*-Pakete und die dadurch bedingten Paketverluste jedoch ein entscheidenderer Faktor für die geringen *Paketankunftsraten* von ADMR zu sein.

Die Ausschläge der ADMR-Kurven in den Abbildungen 28 und 29 liegen zum Teil am Verlauf der *Paketankunftsrate*, da die Werte direkt mit der Anzahl der empfangenen Datenpakete zusammenhängen. Zum anderen tritt auch hier wieder der bei den Szenarien nach [Jet01a] beschriebene Effekt des Mobilitätsmodells `RANDOM-WAYPOINT` auf. Bei hoher Mobilität liegen die Knoten enger aneinander und es müssen seltener Reparaturen eingeleitet werden, da einige Knoten die Daten direkt vom Sender empfangen können und somit nicht auf andere Knoten angewiesen sind.

ODMRP zeigt nahezu geradlinige Verläufe, da keine Reparaturen vorgenommen werden und das Verhalten des Protokolls unabhängig von der Mobilität ist.

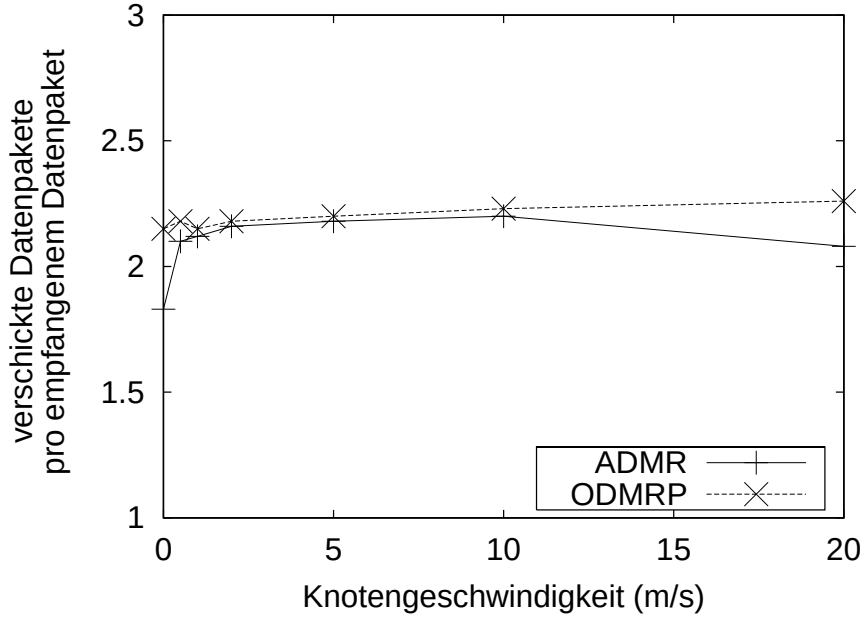

Abbildung 27: verschickte Datenpakete pro empfangenem Datenpaket, Szenario 3

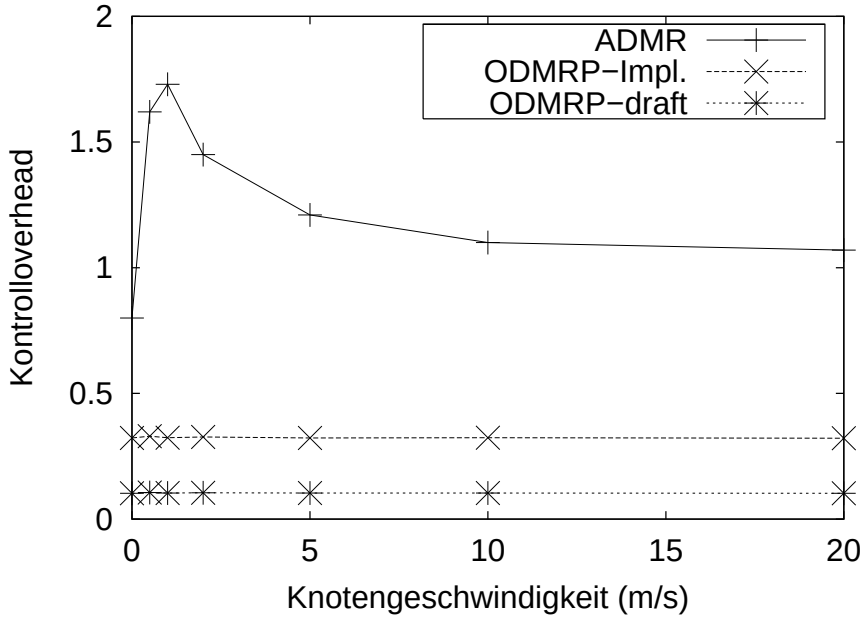

Abbildung 28: Kontrolloverhead, Szenario 3

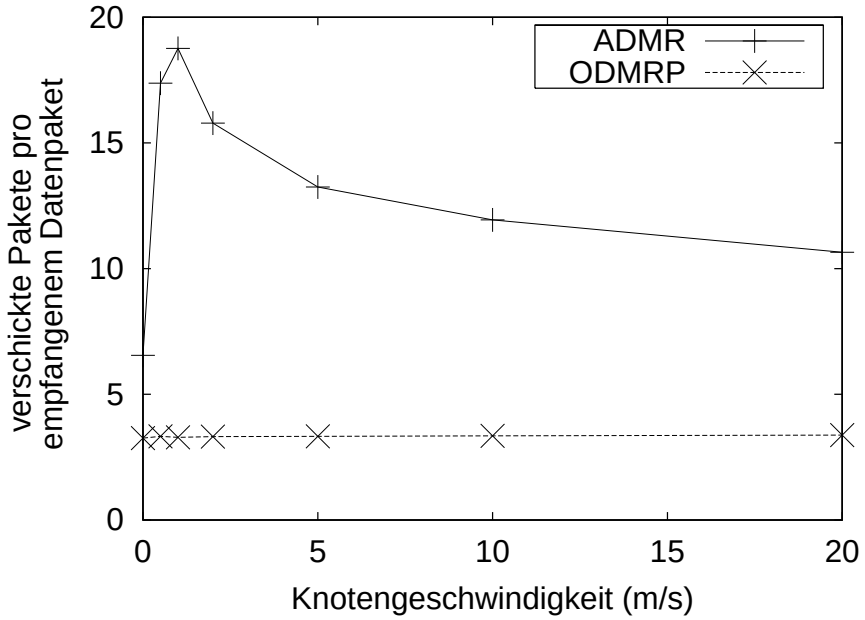

Abbildung 29: verschickte Pakete pro empfangenem Datenpaket, Szenario 3

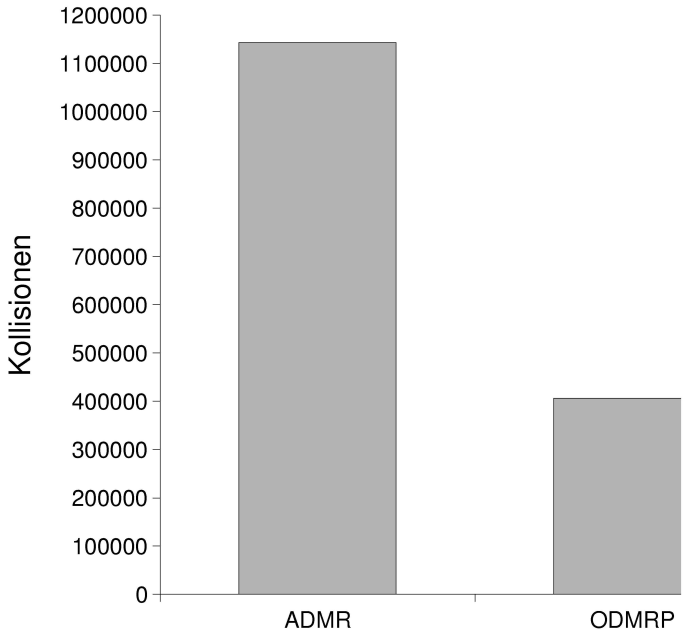

Abbildung 30: Kollisionen bei einer Knotengeschwindigkeit von 5 m/s, Szenario 3

4.4.3 Szenario 4

Bei diesen Simulationen wurde die Anzahl der Sender im Netzwerk variiert. Es gab 1, 2, 5, 10 bzw. 20 Sender. Diese erzeugten eine *Last* von 10 Paketen pro Sekunde. Die Größe der Multicast-Gruppe betrug immer 20 und die Sender waren gleichzeitig Mitglieder der Gruppe. Die Knotengeschwindigkeit betrug 1 m/s, was im Vergleich zu den anderen Szenarien relativ gering ist.

Um bei verschieden vielen Sendern eine konstante *Last* zu erreichen, müssen die Sender die Anzahl der pro Sekunde verschickten Pakete variieren. Ist nur ein Sender vorhanden muss dieser alle 0,1 Sekunden ein Paket verschicken. Gibt es dagegen 20 Sender muss jeder einzelne davon nur noch alle zwei Sekunden ein Paket versenden.

Dies bedeutet bei einem Sender, dass die Pakete innerhalb der Weiterleitungsgruppe schnell aufeinander folgen. Die Belastung dieses Teils des Netzes ist also relativ hoch. Möchte ADMR nun Reparaturen initiieren wird die Belastung der Knoten der Weiterleitungsgruppe weiter erhöht. Dies führt somit zu Kollisionen und Paketverlusten. Eingeleitete Reparaturen können dann u. U. fehlschlagen. Die Anfälligkeit von ADMR für dieses Problem zeigt Abbildung 31.

Dass die Werte bei mehr als zwei Sendern trotzdem wieder schlechter werden hat mehrere Gründe. ADMR baut für jeden Sender eine eigene Weiterleitungsgruppe auf. Damit steigt die Anzahl der Stellen, an denen ADMR Reparaturen vornehmen muss. Dass mehr Reparaturen vorgenommen werden (mehr Kontrollpakete verschickt werden) ist in Abbildung 32 dargestellt.

Außerdem tritt ein Effekt auf der sich dadurch ergibt, dass bei mehr Sendern die einzelnen Sender in größeren Abständen senden. Um eine Reparatur einzuleiten muss ADMR erkennen, dass ein Paket verloren gegangen ist. Ist nun aber der Abstand zwischen zwei Paketen sehr groß, dauert es entsprechend lange bis das nächste Paket erwartet wird. Frühestens nachdem das erwartetet Paket nicht eingetroffen ist, kann mit der Reparatur begonnen werden.

Die Konfiguration für einen Sender ähnelt den Einstellungen von Szenario 1. Allerdings wird dort eine deutlich höhere Paketankunftsrate erzielt. Dies hängt damit zusammen, dass in Szenario 1 das Gebiet in dem sich die Knoten bewegen deutlich kleiner ist (450.000 m^2 statt 1.000.000 m^2). Das kleinere Gebiet führt dazu, dass die Pakete weniger oft weitergeleitet werden müssen. Außerdem werden kleinere Paket verschickt, was zusammen mit der geringeren Anzahl von Weiterleitungen zu einer geringeren Netzbelastung führt. Hinzukommend sind in Szenario 1 nur 15 statt 20 Empfänger vorhanden. Somit müssen auch weniger *Ack*-Pakete verschickt werden, was die Netzbelastung weiter verringert und damit zu weniger Paketverlusten und einer höheren Paketankunftsrate beiträgt.

ODMRP erreicht bei der *Paketankunftsrate* bei wenigen Sendern wesentlich bessere Ergebnisse als ADMR. Steigt die Anzahl der Sender jedoch an, so werden mehr Kontrollpakete verschickt, da jeder Sender seine eigenen *Join Query*s ver-

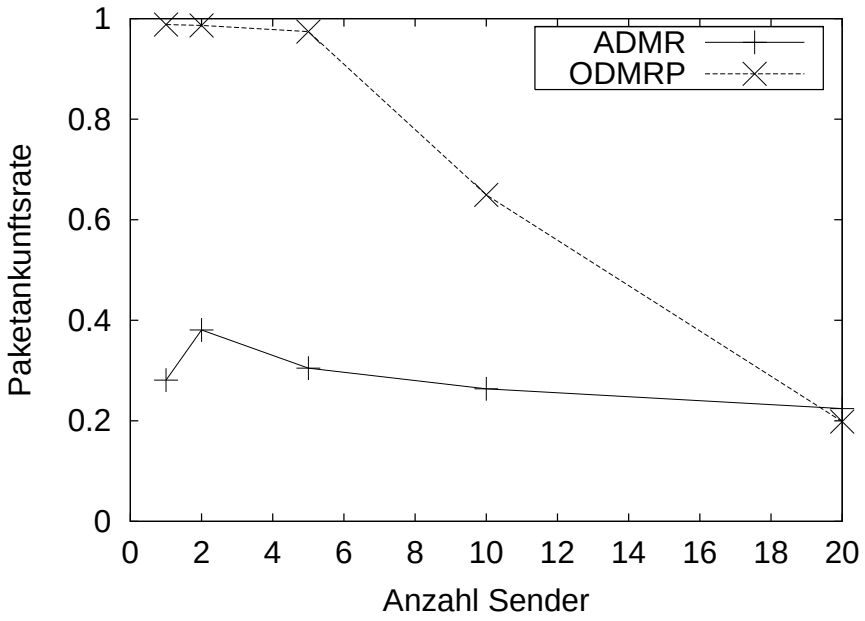

Abbildung 31: Paketankunftsrate, Szenario 4

schickt. Das führt durch die stärkere Belastung des Netzes zu einem Einbruch der *Paketankunftsrate*, die bei 20 Sendern sogar unter dem Wert von ADMR liegt. Auffällig ist hierbei, dass sich die Werte von ODMRP nicht mit den in [LSHGB] erzielten Ergebnissen decken. In [LSHGB] liegen die Werte von ODMRP durchweg über 90% und steigen bei mehr Sendern sogar noch an. Dies hängt möglicherweise mit unterschiedlichen Implementierungen von ODMRP zusammen. Auf die Probleme der verwendeten ODMRP-Implementierung wurde in einem früheren Abschnitt bereits eingegangen. Es ist aber nochmals darauf hinzuweisen, dass sich die für diese Arbeit verwendete Implementierung von ODMRP an der im ensprechenden Draft [LSG00] als Referenz angegebenen orientiert.

4.4.4 Szenario 5

Hierbei erzeugten fünf Sender eine *Last* von 10 Paketen pro Sekunde. Wie schon in den anderen Szenarien zu [LSHGB] gehörten sie auch der Multicast-Gruppe an. Deren Größe wurde variiert und betrug 5, 10, 20, 30 bzw. 40. Die Geschwindigkeit der Knoten betrug 1 m/s.

Wie in Abbildung 33 dargestellt zeigt ODMRP nur eine geringe Abhängigkeit von der Multicast-Gruppengröße. Bei einer größeren Gruppe fallen geringfügig mehr Kontrollpakete an und die Datenpakete müssen u. U. häufiger weitergeleitet werden. Die Belastung des Netzes steigt somit und führt zu mehr Paketverlusten.

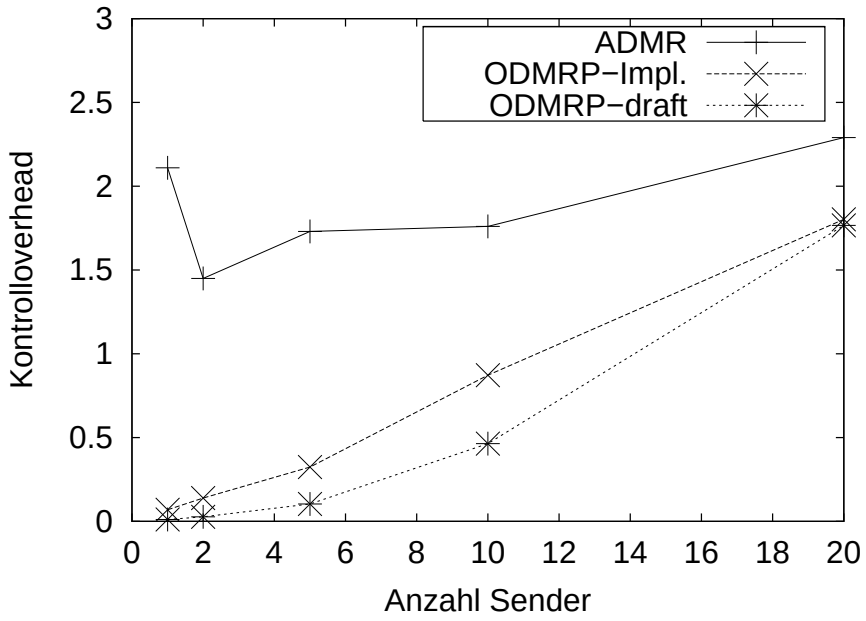

Abbildung 32: Kontrolloverhead, Szenario 4

ADMR hat hier ein gänzlich anderes Verhalten. Satt 95% bei fünf Empfängern werden bei 40 Empfängern nur noch 14% der Pakete empfangen. Der Grund hierfür liegt, wie schon durch die vorherigen Simulationen und die Werte aus Abschnitt 4.3 vermutet, im Wesentlichen in den *Ack*-Paketen der Empfänger. Da jeder Empfänger jedes Paket quittieren muss, steigt bei mehr Empfängern auch die Anzahl der versendeten *Ack*-Pakete an. Dadurch steigt die Anzahl der Kollisionen und der dadurch bedingten Paketverluste.

4.4.5 Szenario 6

Diese Messungen sollten zeigen, wie gut die Routing Protokolle mit einer hohen Netzbelastung zurecht kommen. Daher bewegten sich die Knoten überhaupt nicht, so dass es keine daraus resultierende Effekte gab. Es gab fünf Sender, die auch der Multicast-Gruppe angehörten. Insgesamt gab es 20 Multicast-Gruppenmitglieder. Die *Last* wurde über die Werte 1, 2, 5, 10, 25 und 50 Pakete pro Sekunde variiert.

Die erzielten *Paketankunftsraten* bei diesem Szenario sind in Abbildung 34 dargestellt. Beide Protokolle zeigen hier das erwartete Verhalten, dass bei steigender *Last* der Anteil der empfangenen Pakete sinkt. ADMR erreicht, wie auch schon durch die vorangegangenen Messungen zu erwarten, insgesamt schlechtere Werte als ODMRP. Wenn bei steigender Last von den Sendern mehr Pakete verschickt werden, müssen auch entsprechend mehr Pakete weitergeleitet werden.

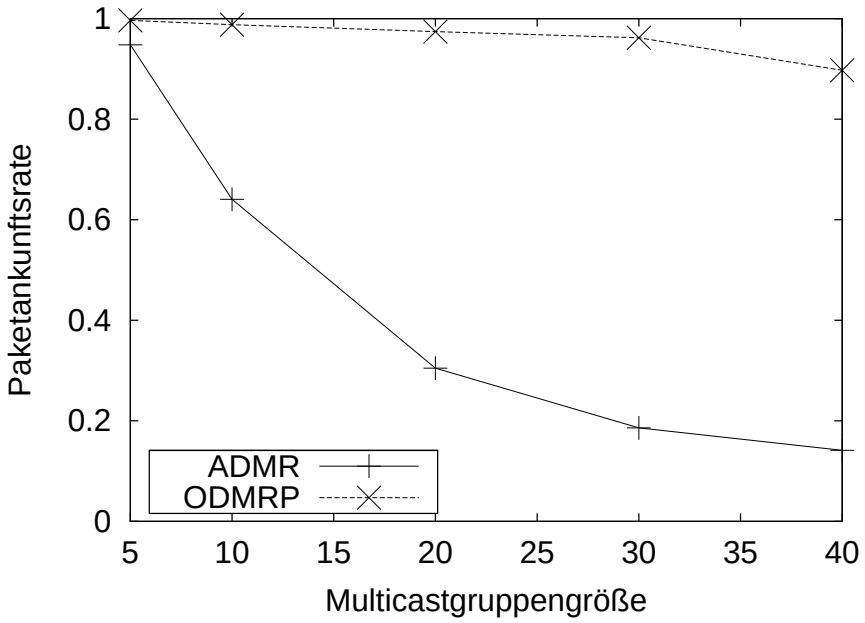

Abbildung 33: Paketankunftsrate, Szenario 5

Die Anzahl der von den Knoten insgesamt verschickten Pakete steigt also stark an. Dies führt zu mehr Kollisionen und dadurch zu Paketverlusten, die sich in einer schlechteren *Paketankunftsrate* niederschlagen. Bei ADMR kommt hinzu, dass bei einer hohen *Last* auch mehr *Ack*-Pakete verschickt werden müssen.

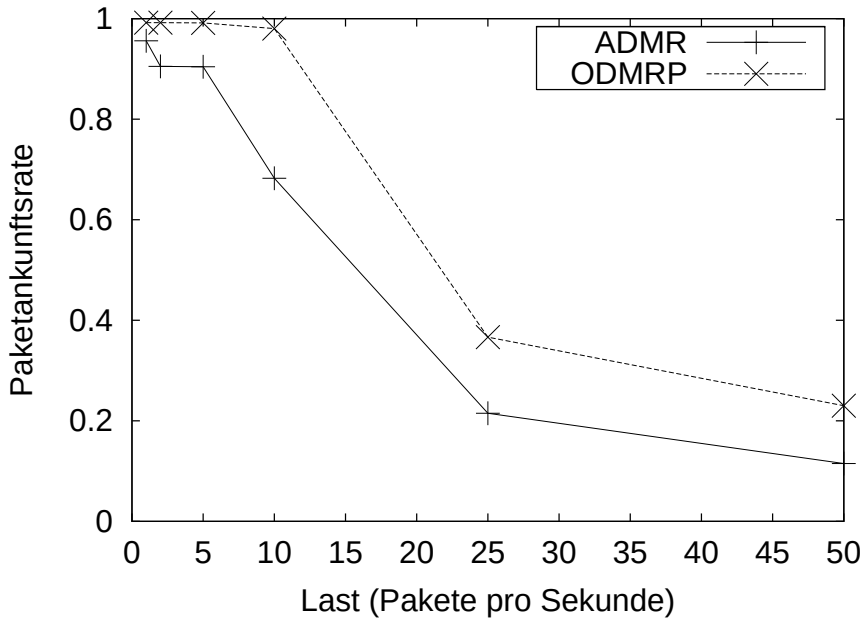

Abbildung 34: Paketankunftsrate, Szenario 6

5 Zusammenfassung und Ausblick

ADMR ist ein Multicast Routing Protokoll für mobile Ad-hoc-Netze. Wie im Namen von ADMR bereits angedeutet (Adaptive Demand-Driven Multicast Routing), wird der Routingstatus erst bei Bedarf aufgebaut und danach dynamisch an die Verhältnisse im Netz angepasst. Dies hat das Ziel, mit möglichst wenig Kontrolldaten trotzdem große Mengen an Datenpaketen auszuliefern.

ADMR bestimmt mit Hilfe von gefluteten Paketen und der Information darüber, von welchem Knoten diese Pakete empfangen wurden, eine Gruppe von Knoten, die die verschickten Daten eines Senders weiterleiten. So kann die Entfernung zwischen Sender und Empfängern überbrückt werden. Diese Gruppe wird nach Bedarf vergrößert oder verkleinert. Trifft ein erwartetes Paket nicht ein, wird sie durch eine Reparatur vergrößert; stellt ein Knoten fest, dass er nicht mehr gebraucht wird, wird sie verkleinert. Für eine korrekte Funktionsweise ist bei ADMR jeder Knoten darauf angewiesen zu erfahren, ob von ihm weitergeleitete Pakete weiterhin benötigt werden. Andere Knoten müssen ein Paket also entweder auch weiterleiten, so dass der ursprüngliche Knoten es wiederum empfängt, oder aber ein *Ack*-Paket senden.

Im Rahmen dieser Studienarbeit wurde ADMR in Netzwerksimulator GloMoSim implementiert, womit dann diverse Szenarien simuliert wurden. Dabei zeigte sich, dass ADMR in fast allen Bereichen schlechter als das vom Aufbau her eigentlich einfachere ODMRP abschneidet. Die Problembereiche von ADMR sind vor allem Multicast-Gruppen mit vielen Empfängern sowie hohe Mobilität. Der von den ADMR-Entwicklern erhoffte geringe Kontrolloverhead konnte in dieser Arbeit nicht bestätigt werden. Dies hängt wesentlich mit dem Versenden der *Ack*-Pakete zusammen. Im direkten Vergleich zwischen ODMRP und ADMR schneidet ODMRP deutlich besser ab. Lediglich bei sehr vielen Sendern (≥ 20)

könnte es einen leichten Vorteil für ADMR geben.

Die Probleme von ADMR sind die *Ack*-Pakete sowie das späte Einsetzen der Reparaturen, nachdem mindestens schon ein Paket verloren gegangen ist. Die Reparaturen ließen sich durch die Hinzunahme weiterer Informationen, wie etwa der Signalstärke, u. U. optimieren.

Das Problem der *Ack*-Pakete ist jedoch im grundsätzlichen Design von ADMR verankert und lässt sich nicht ohne tiefgreifende Änderungen beheben. Die gesamte Funktionsweise ist darauf angelegt, dass jeder Knoten erfährt, ob die von ihm versendeten Pakete noch von anderen Knoten benötigt werden.

Literatur

[BeWa02] C. Bettstetter, C. Wagner. „The spatial node distribution of the random waypoint mobility model“ in Proc. German Workshop on Mobile Ad Hoc Networks (WMAN), (Ulm, Germany), März 2002

[Bl03] R. Bless, Folien der Vorlesung „Next Generation Internet“ vom SS 2003 an der Universität Karlsruhe (TH). `http://www.tm.uka.de/lehre/SS03/vorlesungen/V_NGI.html`

[Bra97] S. Bradner, RFC 2119, „Key words for use in RFCs to Indicate Requirement Levels“. `http://www.ietf.org/rfc/rfc2119.txt`

[BTATBG] Lokesh Bajaj, Mineo Takai, Rajat Ahuja, Ken Tang, Rajive Bagrodia, Mario Gerla. „GloMoSim: A Scalable Network Simulation Environment“. `http://www.cs.ucla.edu/NRL/wireless/papers.htm` bzw. GloMoSim-Distribution

[EsFa98] D. Estrin, D. Farinacci et al. RFC 2362 „Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification“. `http://www.ietf.org/rfc/rfc2362.txt`

[gms] GloMoSim-Homepage. `http://pcl.cs.ucla.edu/projects/glomosim/`

[Jet01a] J. G. Jetcheva, D. B. Johnson. „Adaptive Demand-Driven Multicast Routing in Multi-Hop Wireless Ad Hoc Networks“. `http://www.monarch.cs.rice.edu/papers.html`

[Jet01b] J. G. Jetcheva. Internet-Draft, draft-jetcheva-manet-admr-00.txt, 13. Juli 2001. `http://www.monarch.cs.rice.edu/ietf.html`

[LSG00] S.-J. Lee, W. Su, M. Gerla. Internet-Draft, draft-ietf-manet-odmrp-02.txt, Januar 2000. `http://www.cs.ucla.edu/NRL/wireless/papers.html`

[LSHGB] S.-J. Lee, W. Su, J. Hsu, M. Gerla, R, Bagrodia. „A Performance Comparison Study of Ad Hoc Wireless Multicast Protocols“. `http://www.cs.ucla.edu/NRL/wireless/papers.html`

[Moy94] J. Moy. RFC 1584 „Multicast Extensions to OSPF“. `http://www.ietf.org/rfc/rfc1584.txt`

[nam] NAM-Homepage. `http://www.isi.edu/nsnam/nam/`

[Nil02] Thomas Nilsson. „A Tutorial On GloMoSim“. `http://www.cs.umu.se/kurser/TDBD16/HT02/lab2.html`

[Par] PARSEC-Homepage. http://pcl.cs.ucla.edu/projects/parsec

[WaPa88] D. Waitzman, C. Partridge. RFC 1075 „Distance Vector Multicast Routing Protocol“. http://www.ietf.org/rfc/rfc1075.txt

[Zit03] M. Zitterbart, Folien der Vorlesung „Telematik“ vom WS 2002/03 an der Universität Karlsruhe (TH). http://www.tm.uka.de/lehre/WS0203/vl/telematik.html

Index