

# Ein Framework für MANET Routing Protokolle

Jürgen Nagler, Frank Kargl, Stefan Schlott, Michael Weber

Abteilung Medieninformatik  
Universität Ulm  
89069 Ulm  
juergen.nagler@informatik.uni-ulm.de  
frank.kargl@informatik.uni-ulm.de  
stefan.schlott@informatik.uni-ulm.de  
weber@informatik.uni-ulm.de

**Abstract:** Es existieren zahlreiche Protokolle, die das Routing in „mobile ad-hoc networks“ (MANETs) übernehmen. Die meisten dieser Protokolle wurden für Simulatoren implementiert, um die spezifischen Eigenschaften unter bestimmten Umständen untersuchen und mit anderen Protokollen vergleichen zu können. Daneben existiert eine Reihe von „echten“ Implementierungen, die sich vor allem in ihrer Verfügbarkeit unterscheiden. Einige beschränken sich auf eine Lauffähigkeit unter bestimmten Betriebssystemen (z.B. BSD, Linux, Windows), andere sind nur mit eingeschränkten Fähigkeiten (z.B. nur für festgelegte Bewegungsmuster) verfügbar. In diesem Paper stellen wir deshalb ein gemeinsames Framework für MANET Routing Protokolle vor. Zuerst werden wir auf bestehende und verwandte Arbeiten eingehen. Dann beschreiben wir die Funktionen, die viele der MANET Routing Protokolle gemein haben und deshalb im Framework für alle zur Verfügung gestellt werden. Als nächstes wird die Architektur des Frameworks aufgezeigt. Danach werden die Änderungen an einigen der bekanntesten Routing Protokolle dargestellt, die nötig sind, um innerhalb des Frameworks funktionieren zu können. Anschließend stellen wir die Implementierung unseres so genannten MANET Gateway Daemons (manetgd) für Linux vor. Wir schließen das Paper mit Kommentaren zu Tests, die möglich sind, sobald eine ausreichende Anzahl von Routing Protokollen unter Zuhilfenahme unseres Frameworks implementiert worden ist.

## 1 Einführung

Wir werden an dieser Stelle keine Einführung über MANETs aufführen, da wir davon ausgehen, dass der Leser mit den Grundlagen bereits vertraut ist. Wir konzentrieren uns viel mehr auf die Frage: warum benötigen wir ein Routing Framework für MANETs?

Es lassen sich eine Reihe von Argumenten dafür finden. Wie wir später zeigen werden greifen die aktuellen MANET Routing Protokolle immer wieder auf ähnliche Grundfunktionen zurück. Werden diese jeweils in den einzelnen Protokollen implementiert, muss dafür auch jeweils die kostbare Zeit für die Entwicklung beansprucht werden. Dazu kommt, dass sich somit auch mehr Fehler im Code einschleichen können. So würde beispielsweise die Implementierung eines einzigen Flooding Algorithmus' im Framework, auf den

die Protokolle zugreifen könnten, vermutlich besser auf Fehler untersucht sein, als all die verschiedenen Versionen, die jeweils für nur ein Protokoll entwickelt wurden.

Wie bereits im RFC2501 [CM99] beschrieben und auch in vielen verschiedenen Threads auf der MANET Mailingliste [MWM01] diskutiert wurde, wird es aller Voraussicht nach in der Zukunft nicht ein bestimmtes Routing Protokoll geben, das die Bedürfnisse aller verschiedenen MANET Szenarien befriedigen kann. So werden Lösungen benötigt, die für große oder aber auch relativ kleine Netze genügen, oder für Netze, deren Knoten dicht beieinander liegen oder nur sehr dünn verteilt sind. Somit wird es eine ganze Reihe von Protokollen für die verschiedenen Einsatzmöglichkeiten geben. Dennoch wird es Situationen geben, in denen man solche Netzwerke untereinander verbinden will. Wenn die Routing Protokolle als Module in einem gemeinsamen Framework zum Einsatz kommen, besteht die Möglichkeit der Zusammenarbeit. So können z.B. Topologie-Informationen von allen Protokollen genutzt oder bestimmte Ereignisse an alle gemeldet werden.

Wie wir noch im Abschnitt über die Test sehen werden, bietet unser Framework eine effiziente Möglichkeit, die verschiedenen Protokolle in einer realen Testumgebung miteinander zu vergleichen. Auf einer Maschine können die Routing Protokolle als Module gleichzeitig und doch unabhängig voneinander laufen, so dass auch die jeweils erhaltenen Topologie-Informationen und Routing-Einträge pro Protokoll betrachtet werden können. Da die Anzahl der Maschinen samt der Bewegungsmuster für alle Protokolle identisch sind, erhält man hervorragend vergleichbare Ergebnisse im Gegensatz zu Ergebnissen aus verschiedenen Test, die in Reihe ausgeführt wurden.

Ein Nebenbemerkung: wir haben uns dafür entschieden, unser Framework für IPv6 auszuliegen. Deshalb war es nötig, die implementierten Protokolle leicht anzupassen, um vor allem den Unterschied bzgl. der Längen der IP Adressen zu berücksichtigen. Andererseits war es uns dadurch möglich, von den Neuerungen in IPv6 (wie z.B. der erweiterbare Header-Mechanismus, die Flexibilität bei Source Routen, ...) zu profitieren. Da aber die betroffenen Änderungen der MANET Protokolle nicht den Hauptteil dieser Arbeit darstellen, werden wir nur teilweise darauf eingehen. Für eine spätere Version ist die Unterstützung von IPv4 vorgesehen.

## **2 Bisherige und verwandte Arbeiten**

Sicherlich ist die Idee, ein gemeinsames Routing Framework für verschiedene Routing Protokolle einzusetzen, nicht neu. Schon sehr bald gab es Implementierungen der klassischen Routing Protokolle innerhalb eines Frameworks. Ein sehr verbreitetes Beispiel dafür ist der gated von ISC [NHT02], ein weiteres des GNU Zebra Projekt [III02]. Diese Projekte haben bereits gezeigt, dass viele der erwähnten Vorteile wirklich genutzt werden können. Unser Ansatz soll zudem über den üblichen Funktionsumfang hinausgehen und ebenfalls allgemein benötigte Funktionen bieten. So kann beispielsweise jetzt schon über das Framework auf diverse Warteschlangen-, Listen- und Tabellen-Management-Funktionen zurückgegriffen werden.

Ein Nachteil unseres Ansatzes ist die enge Bindung zwischen dem zentralen Framework

und den Routing Protokoll Modulen innerhalb eines Prozesses. Eine andere Möglichkeit, wie sie z.B. ZEBRA nutzt, wäre, alle Komponenten in separaten Prozessen laufen und über UNIX domain sockets kommunizieren zu lassen. Dies würde zu einer besseren Wartbarkeit und Stabilität führen. Wir versuchen, dies zu kompensieren, indem wir für die Routing Protokoll Module dynamisch ladbare Bibliotheken verwenden.

Innerhalb der MANET Forschungsgruppen existiert bereits ein ähnlicher Vorschlag zur Bildung eines allgemeinen Routing Frameworks. Der NS2 Simulator [TNS02, WME02] und der OPNET Modelierer [OM02] werden aktuell von den meisten MANET Arbeitsgruppen zur Entwicklung und Evaluation ihrer Algorithmen eingesetzt. Beide bieten eine Art Integrationsumgebung, in der die verschiedenen Protokolle implementiert und getestet werden können. Im Gegensatz zu unserem Entwurf stellen diese Frameworks eine eigene API zur Verfügung oder basieren auf speziellen Programmiersprachen (z.B. otcl). Aus diesem Grund kann üblicherweise diese spezielle Implementierung für das Simulationssystem nicht für die reale Welt eingesetzt werden und eine weitere Implementierung muss entwickelt werden (TORA [PC97, PC01] zählt hierbei zu den Ausnahmen, da ein Großteil des für NS2 und für Linux entwickelten Codes identisch ist).

Da viele der Protokolle mit speziellen Mechanismen arbeiten (beispielsweise Source Routen, Verarbeitung von weiterzuleitenden Paketen, etc.), nutzen sie oft die proprietären Möglichkeiten einzelner Betriebssysteme und erschweren dadurch eine einfache Portierung der Software. Durch den Einsatz unseres Frameworks wird zwar die Problematik der Portierung auf andere Plattformen nicht umgangen, allerdings muss dies lediglich ein Mal getan werden und alle Routing Protokolle, die das Framework nutzen, können direkt davon profitieren.

Es gab bereits einen Versuch unter den MANET Arbeitsgruppen, ein allgemeines Framework mit integrierter Funktionalität für MANET Routing Protokolle zu etablieren. Das Internet MANET Encapsulation Protocol (IMEP) [CPPPQ98] wurde entwickelt, um den aufsetzenden Protokollen Nachrichtenaggregation, Adressauflösung auf Netzwerkebene, Statuserkennung von Verbindungen, zuverlässigen Broadcast, Multipoint Relaying und Authentisierung zu bieten. Leider wurde IMEP lediglich von seinen Entwicklern als Basis zur Implementierung des TORA Protokolls verwendet. Eine IMEP Implementierung ist als Teil der UMD TORA Implementierung [PC01] erhältlich. Es lassen sich einige Parallelen zwischen IMEP und unserem Framework ziehen. Momentan decken wir nicht alle der Aspekte, die IMEP bietet, ab (z.B. bieten wir keinen zuverlässigen Broadcast), allerdings wurde IMEP hauptsächlich als Protokoll entwickelt und somit sind auch viele praktische Aspekte, denen sich unser Framework annimmt (wie das Laden/Entladen von Modulen, Tabellen-Management, usw.), dort nicht berücksichtigt.

### **3 Überblick der Funktionalität**

Indem wir die Anforderungen einiger der aktuellen Routing Protokolle genau untersucht haben, war es uns möglich, eine Liste von Eigenschaften zu identifizieren, die von den meisten Protokollen verwendet bzw. benötigt werden. Obwohl die genauen Einzelheiten pro Protokoll leicht unterschiedlich ausfielen, denken wir trotzdem, dass die meisten Pro-

tolle so angepasst werden können, dass unsere allgemeinen, im Framework eingebauten Mechanismen ausreichend sein sollten.

### **3.1 Adressautokonfiguration**

Bevor ein Knoten beginnen kann, Datenpakete mit anderen Knoten im Netzwerk auszutauschen, benötigt er eine gültige IP Adresse. Weder eine statische Konfiguration der Knoten, noch Maßnahmen, die zentrale Server (wie DHCP) benötigen würden, scheinen für MANETs geeignet. Ein Mechanismus wie die IPv6 Stateless Autoconfiguration [TN98] würde besser passen. Aber egal, welcher Mechanismus gewählt wird, dieser sollte Teil des Frameworks und nicht des aktuellen Routing Protokolls sein.

Bis jetzt unterstützt unser Framework die statische Adressekonfiguration wie auch die IPv6 Stateless Autoconfiguration unter Angabe eines Präfixes. Dieses Präfix wird im Netzwerk initial in einem bestimmten Knoten, dem sogenannten Bootstrap Knoten, konfiguriert. Dies kann z.B. das Internet Gateway des MANETs sein. Mehrere Bootstrap Knoten sind möglich, sofern sie alle die gleichen Präfixe konsistent verwalten. Eine offene Frage bleibt in Hinblick auf die Realisierung, wie doppelte Adressen innerhalb eines MANETs entdeckt werden können.

Für zukünftige Versionen planen wir den Einbau einer Art Election Algorithmus, der das anfängliche Aushandeln und Bestimmen eines gemeinsamen Präfixes übernehmen kann.

### **3.2 Nachbarschaftspflege**

Alle Routing Protokolle basieren darauf, in jedem Knoten herauszufinden, wer die direkten Nachbarn sind, die in nur einem Hop physikalisch erreicht werden können. Gewöhnlich lassen sich dafür vier Vorgehensweisen finden, dies zu tun:

- unter Zuhilfenahme der Informationen aus dem Link-Layer (z.B. IEEE802.11 im ad-hoc Modus)
- durch Senden und dadurch dem Empfang expliziter HELLO Pakete als Broad- oder Multicast
- indem Informationen verwertet werden, die das Routing Protokoll aus mitgelesenen Broad- und Multicast gewinnen kann
- durch Beobachtungen, welche Routen für die (Unicast-)Paketauslieferungen erfolgreich benutzt werden

Die Verwaltung einer Liste aller Nachbarn sollte in der Verantwortung des Frameworks liegen. Es ist Aufgabe des Frameworks herauszufinden, welche Nachbarn welches Routing Protokoll unterstützen. Einem laufenden Routing Modul meldet das Framework dann nur die Knoten als Nachbarn, die auch das gleiche Routing Protokoll verwenden.

Bis jetzt wurden in unserem Framework lediglich die zweite und dritte Möglichkeit implementiert, da wir uns weder den Details der Link-Layer Benachrichtigung angenommen, noch eine Überwachung des Routing Prozesses eingebaut haben. Die verwendeten HELLO Nachrichten erlauben es, dass die Routing Protokolle eigene Informationen mit anhängen können, sofern dies vom Protokoll benötigt wird. Weiterhin existiert ein Interface, über das es den Routing Protokoll Modulen möglich ist, die Nachbartabelle anhand der bestehenden Informationen über die Netzwerktopologie zu aktualisieren. Wie bereits erwähnt existiert zudem eine umfangreiche Unterstützung für Timer Operationen, einschließlich einer automatischen Verfallsregelung für die Einträge der Nachbartabelle.

### 3.3 Flooding/Broadcasting

Fast alle Protokolle benötigen einen Mechanismus, der dafür sorgt, dass Informationen durch das MANET verbreitet werden. Wir können dabei zwei zu unterscheidende Prozesse identifizieren: Broadcasting und Flooding. Da der Gebrauch der beiden Begriffe innerhalb der MANET Forschergruppen nicht einheitlich erfolgt, führen wir für unsere Verwendung folgende Definitionen ein:

**Definition Broadcast:** Der Prozess, bei dem ein Paket mit Informationen an alle direkten Nachbarn gesendet werden soll. Nachbarn, die dieses Paket erhalten, sollten dieses ihrerseits nicht an andere weiterleiten.

**Definition Flooding:** Das Senden von Paketen mit Informationen an alle (oder auch nur eine bestimmte Untermenge von) Knoten innerhalb des MANETs. Damit die Information auch zu Knoten gelangt, die nicht direkt erreicht werden können, müssen die direkten Nachbarn die Pakete unter Verwendung eines Flooding Algorithmus' weiterleiten.

Dabei benötigen alle (sinnvollen) Flooding Algorithmen einen Mechanismus, um zu verhindern, dass Pakete endlos im Netzwerk kursieren. Dies kann durch die Verwendung von TTL Zählern, Paket IDs, Merklisten oder Ähnlichem erfolgen. Des weiteren kann es sinnvoll sein, die Auslieferung mehrerer doppelt empfangener Pakete an den jeweiligen Prozess frühzeitig im Netzwerkstack zu unterbinden.

Flooding kann sehr einfach über einen „Broadcast und Weiterleiten“ Algorithmus realisiert werden. Allerdings würde sich dieser sehr ineffizient auf die Nutzung der Bandbreite auswirken, weshalb dies vor allem in drahtlosen Netzen vermieden werden sollte. Wir haben deshalb zusätzlich den Mechanismus der Multipoint Relays (MPR) [QVL00] in unser Framework integriert.

### 3.4 Zuverlässige Paketzustellung

Die Zustellung von Paketen der Routing Protokolle von einem zum nächsten Knoten hat in der Regel nach einer vorgegebenen Zuverlässigkeit zu erfolgen. Im einfachsten Fall (best-effort) werden vom Framework und dem darunterliegenden Zustellsystem keinerlei

Überprüfungen der Übertragung vorgenommen. Gehen in diesem Fall Pakete verloren, z.B. verursacht durch Überlastung oder Checksummenfehler, wird nichts unternommen und die Pakete fehlen einfach im Ablauf. Für einige Protokolle stellt dies kein Problem dar. Die meisten proaktiven Protokolle wiederholen die Übertragung von Protokoll Daten in regelmäßigen Abständen und erst eine gewisse Anzahl von hintereinander verloren gegangenen Paketen beeinflusst das Protokoll maßgeblich. In diesem Fall würde die Implementierung einer zuverlässigen Zustellung einen unnötigen Mehraufwand bedeuten. Andere Protokolle hingegen setzen auf einer garantierten Paketzustellung auf. Als Beispiel seien die Route Requests in den meisten reaktiven Protokollen genannt, die eine solche Garantie benötigen. Deshalb sollte das Routing Framework eine zuverlässige Zustellung anbieten. Dazu können die üblichen Mechanismen verwendet werden (Bestätigung und negative Bestätigung von Paketen).

Bisher kommt in unserem Framework lediglich ein einfacher Bestätigungsmechanismus zum Einsatz, der bei Bedarf erweitert werden kann.

### **3.5 Source Routing**

Einige der Protokolle gehen von Source Routen für die Paketzustellung aus. Diese Funktionalität kann, sofern das darunterliegende Betriebssysteme keine Unterstützung dafür bietet bzw. eine eigene protokollabhängige Semantik verwendet werden soll, als Teil des Frameworks realisiert werden. Das Framework greift durch folgende Schritte in den Source Routing Prozess ein:

1. Abfangen aller ausgehender Pakete, die mit einer Source Route versehen werden müssen (Erkennung z.B. anhand des Zielpräfixes und dem entsprechenden Protokoll)
2. Finden der entsprechenden Source Route zum angegebenen Ziel (in einem Kernel Modul)
3. Einfügen des Source Routing Headers in das Paket (unter Beachtung von MTU und Fragmentierung!)

Wird lediglich die Grundfunktionalität von Source Routen benötigt, besteht die Möglichkeit, den IPv6 Type 0 Routing Header [DH98] für die Source Routen zu verwenden. Dies hat zum Vorteil, dass die Zwischenknoten zwischen Sender und Empfänger die reguläre Weiterleitung des IPv6 Protokollstacks nutzen können, um das Paket richtig weiterzureichen. Da sich diese Funktion direkt im Kernel befindet, wird damit die Weiterleitung sehr schnell und effizient erledigt. Sofern ein Protokoll (z.B. DSR [JM96, JMHJ01]) zusätzliche Informationen benötigt, die im Source Routing Header übertragen werden sollen, aber vom Type 0 Routing Header nicht zur Verfügung gestellt werden, kann es sein eigenes Routing Header Format spezifizieren. In den Zwischenknoten müssen dann die Pakete mit Source Routen aus dem Weiterleitungsprozess des Systems herausgenommen werden und an ein eigenes Source Routing Modul übergeben werden. In diesem Modul wird dann der

nächste Hop bestimmt, der Source Routing Header evtl. angepasst und das Paket wieder dem System zur Weiterleitung übergeben.

Da wir bis zum aktuellen Zeitpunkt noch kein Protokoll implementiert haben, das auf Source Routen zurückgreift, kamen die definierten Schnittstellen im Framework bisher nicht zum Einsatz.

### **3.6 Hop-by-Hop Optionen**

Einige Protokolle sehen vor, dass Routing Daten im Huckepack-Verfahren mit dem normalen Datenverkehr transportiert werden können. AODV für IPv6 [PRD01] definiert beispielsweise eine sogenannte „Flooding Data Hop-by-Hop Option“. Die Verarbeitung von Huckepack-Daten soll ebenfalls vom Framework und nicht von den einzelnen Protokollen übernommen werden. Dazu zählen das Einfügen der Routing Daten in den Datenpaketen und das Versenden dieser.

Unser Framework bietet einen Mechanismus, um Hop-by-Hop Options in bestimmte normale Pakete, spezifiziert über die Zieladresse, einzufügen. Ein zusätzlicher Timeout gibt an, wie lange anstehende Routing Daten auf ein zu verschickendes Datenpaket warten sollen, bevor sie als eigenständiges Paket verschickt werden sollen.

### **3.7 Schnittstelle zum System**

Ein MANET Routing Protokoll muss auf eine Reihe von Funktionen des Betriebssystems (BS) zurückgreifen. Da sich die APIs der verschiedenen BS gewöhnlich unterscheiden, ist mit viel Arbeit bei der Portierung eines Routing Protokolls zu rechnen. Falls die Routing Protokolle direkt auf die APIs zugreifen, muss eine Portierung für jedes Protokoll und jeden Port einzeln vorgenommen werden. Unsere Idee ist es, allen Routing Protokollen über das Framework eine generische API anzubieten. Eine Portierung des Frameworks mit all seinen Protokoll Modulen für ein BS wird wesentlich effizienter.

Die meisten Routing Protokolle greifen auf die System Routing Tabelle zu, um dort Routen hinzuzufügen, zu löschen oder zu ändern. Oft existiert dafür keine einheitliche API und in seltenen Fällen kann es sogar nötig sein, ein eigenes Kernel Modul zu schreiben, das diese Aufgabe übernimmt und erst dadurch eine Schnittstelle zum Programmierer bietet. Für BSe, in denen selbst dies nicht möglich ist, kann als Workaround auch das entsprechende Kommandozeilenprogramm mit den entsprechenden Parametern über die system() Funktion gerufen werden.

Die Socket Schnittstelle ist eine wohlbekannt Abstraktion der Netzwerkfunktionalität. Dabei sind die Basisfunktionen wie Senden und Empfangen von Daten fast identisch für die meisten Unix-basierten BS und zumindest ähnlich für Microsoft Windows. Allerdings sind spezielle Funktionen, die von MANET Routing Protokollen häufig benötigt werden wie Senden und Empfangen von Broad- und Multicasts, meist BS-abhängig. Deshalb enthält unser Framework eine generelle Abstraktion von Senden und Empfangen von Daten, die auch Broad- und Multicast beinhaltet.

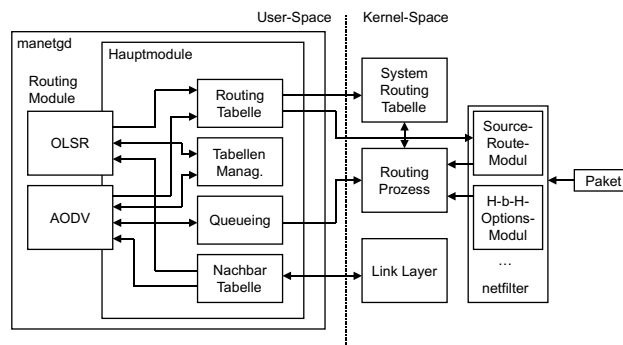


Abbildung 1: Architektur des Frameworks

### 3.8 Allgemeine Funktionen

Schließlich lässt sich ein großer Bereich von allgemein verwendeter Funktionalität finden, aus dem viele Routing Protokolle gemeinsam schöpfen. Die Verarbeitung von Warteschlangen und Tabellen (z.B. Routing Tabelle oder eine Darstellung der Topologie) ist allen zu eigen. Ein weiteres Gebiet ist die Verwaltung von Timern (z.B. ereignis- oder zeitgesteuert) und von auszulösenden Ereignissen und der Abarbeitung von ereignisgesteuerten Aufgaben.

Um die Entwicklung der Protokolle für unser Framework voranzutreiben, bietet es eine Unterstützung aller aufgeführten Punkte. Dabei ist unsere Lösung auf Basis der weit verbreiteten glib Bibliothek entstanden, die als Teil des Gimp Toolkits (GTK+) [GTK02] erstellt wurde. Programmierer, die bereits mit dem GTK+ vertraut sind, sollten somit keine Probleme haben, Protokoll Module für das manetgd Framework zu implementieren.

## 4 Aufbau des Frameworks

Abbildung 1 zeigt einen Überblick über unser MANET Routing Framework. Die zwei Routing Module (OLSR und AODV) links interagieren mit den verschiedenen Teilen des Hauptmoduls. Man sieht, dass jegliche Interaktion mit dem BS Kernel über das Hauptmodul erledigt wird und die Routing Module von BS-spezifischen Aspekten komplett getrennt gehalten werden. Der Großteil des Datenmanagements wird innerhalb des Hauptmoduls gehalten.

Der BS Kernel bleibt weitestgehend unverändert. Die einzigen MANET-spezifischen Teile sind die Module für Paketauswahl und -modifikation. Wie wir später noch zeigen werden, bietet z.B. Linux mit seinem netfilter Framework eine einfache Möglichkeit, eigene Module für diese Aufgaben in den Kernel mit aufzunehmen.

## 5 Anpassung der Protokolle

Zwei Protokolle (OLSR [CJLMMQV01] und AODV [PR99, PBD01]) haben wir bisher für unser Framework implementiert, mit der Arbeit für das dritte Protokoll (DSR [JMHO1]) wurde bereits begonnen. Wir mussten eine Reihe von Änderungen an den ursprünglichen Protokollen vornehmen. Zuerst wurden die Protokolle für IPv6 angepasst. Für OLSR und AODV konnten wir auf die existierenden Ergänzungen bzgl. IPv6 [CJLMMQV01, PRD01] zurückgreifen. Für DSR sind die nötigen Änderungen zwar nicht bekannt, aber bereits geplant und in Arbeit [JMHO1].

Als nächster Schritt wurde versucht, die vom Framework bereitgestellten Funktionen möglichst effizient in den Protokollen einzusetzen. So wurde z.B. die Nachbarschaftspflege aus allen Protokollen entfernt und stattdessen durch entsprechende Aufrufe zu unserem Nachbarmodul ersetzt. An den Stellen, an denen die Protokolle zusätzliche Daten in den Paketen an die Nachbarn versenden oder empfangen mussten, wurden diese an die HELLO Pakete angefügt.

Dort, wo die Protokolle bisher Broadcasts einsetzten, um alle direkten Nachbarn zu erreichen oder das Netz zu fluten, mussten diese in entsprechende Multicasts umgesetzt werden. Wir entschieden uns dabei für die „ALL\_NODES\_LINK\_LOCAL“ (ff02::1) Multicastadresse [HD98], die wir entgegen der Semantik auch für zu flutende Pakete in Verbindung mit entsprechenden Hop-Limits einsetzten. Sobald allerdings die Multicastadresse „ALL\_IPv6\_MANET\_NODES“ [PRD00] definiert ist, werden wir diese zum Flooding einsetzen.

### **5.1 Änderungen für OLSR**

Die aktuelle Version des OLSR Drafts [CJLMMQV01] sieht außer den größeren Adressfeldern keine weiteren Änderungen in Bezug auf IPv6 vor, das Protokoll ist somit fast identisch zur IPv4 Version. Da der Draft auch keine Äußerungen bzgl. des zu verwendenden Broadcasts, um alle direkten Nachbarn zu erreichen, enthält, steht es dem Programmierer frei, einen der angebotenen Mechanismen zu verwenden.

### **5.2 Änderungen für AODV**

Zusätzlich zur AODV Protokoll Spezifikation existiert ein Draft [PRD01], der AODV in Zusammenhang mit IPv6 beschreibt. Die größte Änderung in diesem Draft ist die Möglichkeit, Route Requests und Replies als IPv6 Hop-by-Hop Optionen an normale Datenpakete anfügen zu können. Zwar bietet unser Framework eine Schnittstelle, Hop-by-Hop Optionen verwenden zu können, unsere aktuelle Implementierung von AODV nutzt jedoch lediglich normale RREQ/RREP Pakete. Für eine Realisierung dieser Erweiterung wäre zudem ein Eingriff im Protokollstack in Form eines weiteren Moduls nötig, um die Zieladresse von aus- und eingehenden Paketen von Unicast in Multicast und umgekehrt umzuschreiben.

### **5.3 Änderungen für DSR**

Wie bereits oben erwähnt ist die Implementierung von DSR für manetgd noch nicht abgeschlossen. Als erstes müssen alle Adressfelder auf 128 Bit erweitert werden. Eine noch diskutierte Frage stellt sich bzgl. der Verwendung des IPv6 Type 0 Routing Headers für die DSR Source Routen. Für die Grundfunktionalität von DSR scheint dieser ausreichend zu sein. Der große Vorteil dabei wäre, in allen beteiligten Zwischenknoten wie auch dem Zielknoten die Verarbeitung des Paketes dem vorhandenen IPv6 Protokollstack zu überlassen. Falls wir unseren eigenen Routing Header definieren und verwenden würden, müsste in allen Knoten die Weiterleitung durch den Kernel über ein eigenes (netfilter) Kernelmodul geregelt werden. Dies könnte eventuell zu einer verringerten Leistung führen. Andererseits müssen wir zusätzliche Informationen im Routing Header speichern, sofern wir erweiterte Funktionalität wie das Heilen von Routen oder die Verwendung von MANET-externen Routen nutzen wollen, das somit erst möglich wäre.

## 6 Implementierung

Einige Bemerkungen zur Implementierung. Unser manetgd Prototyp läuft unter Linux 2.4 Systemen. Er wurde in reinem ANSI C geschrieben und greift auf die weit verbreitete glib Bibliothek [GTK02] zu. Da alle Komponenten streng modular aufgebaut sind, sollte es keine großen Probleme bei der Portierung auf z.B. BSD-basierte Systeme geben. Allerdings müssen Entsprechungen für die systemspezifischen Mechanismen gefunden werden, die bisher zum Einsatz kommen. So benötigen wir eine Möglichkeit, Pakete während des Weiterleitungsprozesses abzufangen und evtl. auch zu verändern. Unter Linux wird dies über das netfilter Framework realisiert.

Dieses Framework erlaubt es, in den Prozess des Paket Routings an ein paar bestimmten Stellen gezielt einzugreifen. Die Eintrittspunkte dafür sind `NF_IP6_PRE_ROUTING`, `NF_IP6_FORWARD`, `NF_IP6_POST_ROUTING` und `NF_IP6_LOCAL_OUT`. So kann beispielsweise ein reaktives Protokoll dazu veranlasst werden, einen Route Request abzuschicken, indem es ständig die Pakete an `NF_IP6_LOCAL_OUT` überprüft und bei fehlender oder ungültiger Route zum angegebenen Ziel entsprechend handelt. Das Paket wird an den User Space übergeben, dort in einer Warteschlange gespeichert und durch einen entsprechend gesetzten Callback wird die Routine im Routing Modul gerufen, die für das Versenden eines Route Requests zuständig ist. Sobald ein Route Reply eintrifft, kann das Paket wieder dem Routing Prozess zugeführt werden. Trifft kein Reply ein, sorgt ein Timeout dafür, dass wartende Pakete verworfen werden.

## 7 Tests

Mit den lauffähigen Implementierungen von OLSR und AODV konnten wir unser Framework bereits ersten Tests unterziehen. Die Funktionalität beider Protokolle wurde anhand einer relativ kleinen Anzahl von bis zu fünf Knoten überprüft.

Bisher wurden von uns nur Tests mit jeweils einem aktiven Protokoll durchgeführt. Durch diese Tests konnte einfach nachgewiesen werden, dass die Topology korrekt errechnet und Änderungen in einer angemessenen Zeitspanne bemerkt und darauf dann entsprechend

reagiert wurde.

Dies ist natürlich kein echter Vergleich der Protokolle. Deshalb entwickeln wir gerade einen Mechanismus, der es uns erlaubt, mehrere Protokolle parallel zu betreiben. Diese Protokollmodule und das Hauptmodul werden so konfiguriert, dass die Nutzung gemeinsamer Ressourcen wie z.B. der Topologie deaktiviert ist. Es soll sichergestellt werden, dass jedes Protokoll für sich arbeitet, um letzten Endes einen echten Vergleich anstreben zu können. Sofern das BS negativen Einfluss auf diese Separation haben sollte, z.B. durch die systemweite Routing Tabelle, müssen diese Teile durch eigene Kernelmodule ersetzt werden. Diese stellen dann eine strikte Trennung nach Protokollen sicher. Das Framework übernimmt das Versenden und Empfangen von Testpaketen, anhand derer der Datenfluss simuliert wird. Mit dieser Konfiguration und einer großen Anzahl von mobilen Knoten werden wir dann ausführliche Feldtests durchführen. Da alle Protokollmodule parallel auf den Knoten ablaufen werden, kommen überall die gleichen Bewegungsmuster und das gleiche Bewegungsmodell zum Tragen. Die durch die Testpakete erhaltenen Informationen sind somit unmittelbar vergleichbar.

## **8 Zukünftige Arbeiten**

Wie man deutlich sehen kann, ist die Arbeit an unserem Framework nicht wirklich abgeschlossen (und wird es wohl auch nie sein). Das nächste Ziel stellt die Implementierung weiterer Protokolle dar, um einen echten Vergleich aller relevanten MANET Routing Protokolle bieten zu können. In diesem Rahmen werden wir die Fähigkeiten unseres Frameworks bzgl. Tests ausbauen, so dass z.B. auch Bandbreitenbetrachtungen, usw. möglich werden.

Zudem planen wir, weitere Funktionalität im Framework zu integrieren. Momentan arbeiten wir an einem MANET Sicherheitsframework als Teil unseres manetgd. Dadurch beabsichtigen wir, Sicherheitsaspekte getrennt und unabhängig von den jeweiligen Protokollen zu behandeln. Die Entwickler von Routing Protokollen können dadurch ihre Arbeit unbeeinflusst von Überlegungen zur Sicherheit gestalten. Andere Aspekte wie z.B. Energiesparfunktionen, Sendeleistungsregelung, GPS Lokalisierung, usw. können ebenfalls als Teil des Frameworks in Frage kommen.

Unsere Implementierungen der Routing Protokolle sind sicher nicht perfekt. Viele der in den Drafts vorgeschlagenen Optimierungen wurden von uns noch nicht umgesetzt. Um einen fairen Vergleich der Protokolle erreichen zu können, sollten die Protokollmodule die Möglichkeit haben, ihr Bestes zu geben und müssen deshalb nochmals überarbeitet werden. Wir hoffen, dass in Zukunft einige der Protokollentwickler auf unser Framework zurückgreifen werden, um ihre Ideen zu realisieren. Dadurch werden beide Seiten, die Entwickler wie auch die Benutzer, davon profitieren. Die Entwickler werden weniger Arbeit mit den systemspezifischen Aspekten haben und zudem Implementierungen besitzen, die sehr leicht auf andere BSe portiert werden können. Die Benutzer hingegen werden von einer großen Anzahl verfügbarer Protokolle für die unterschiedlichsten Einsätze profitieren.

Schließlich werden wohl alle Forschungsgruppen von der Möglichkeit profitieren, ver-

gleichbare Tests im „wirklichen“ Leben durchzuführen und damit die bisherigen Ergebnisse aus aktuellen Simulationen erweitern und vervollständigen zu können.

## Literatur

- [CM99] Corson, S.; Macker, J.: Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations, 1999. RFC 2501.
- [MWM01] Thread on manet wg mailinglist (manettd.nrl.navy.mil). Subjects: „manet and ip layering“ and „manet addressing, neighborcasting etc.“. March/May 2001.
- [NHT02] NextHop Technologies. Technical Specs for the GateD Product Suite. <http://www.nexthop.com/products/specs.shtml>. Januar 2002.
- [III02] IP Infusion Inc. GNU Zebra. <http://www.zebra.org/>. Januar 2002.
- [TNS02] The Network Simulator - ns2. <http://www.isi.edu/nsnam/ns/>. Januar 2002.
- [WME02] Wireless and Mobility Extensions to ns-2. <http://www.monarch.cs.cmu.edu/cmu-ns.html>. Januar 2002.
- [OM02] OPNET Modeler. <http://www.opnet.com/products/modeler/home.html>. Januar 2002.
- [PC01] Park, V.; Corson, S.: Temporally-Ordered Routing Algorithm (TORA) Version 1 Functional Specification, 2001. <http://www.ietf.org/internet-drafts/draft-ietf-manet-tora-spec-04.txt>
- [PC97] Park, V.; Corson, M.S.: A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. Proceedings of IEEE INFOCOM '97, April 1997.
- [PPPPQ98] Corson, M.; Papademetriou, S.; Papadopoulos, P.; Park, V.; Qayyum, A.: An Internet MANET Encapsulation Protocol (IMEP) Specification. Internet Draft (work in progress), 1998 (expired).
- [TN98] Thomson S.; Narten T.: IPv6 stateless address autoconfiguration, 1998. RFC 2462.
- [QVL00] Qayyum, A.; Viennot, L.; Laouiti, A.: Multipoint relaying: an efficient technique for flooding in mobile wireless networks. INRIA research report RR-3898, 2000. <http://www.inria.fr/rrrt/rr-3898.html>
- [DH98] Deering, S.; Hinden, S.: Internet Protocol, Version 6 (IPv6) Specification, 1998. RFC 2460.

- [JM96] Johnson, D.B.; Maltz, D.A.: Dynamic Source Routing in Ad Hoc Wireless Networks. Mobile Computing, T. Imielinski and H. Korth, eds. The Kluwer International Series in Engineering and Computer Science (vol. 35), Kluwer Academic Publishers, Norwood, Mass., 1996; S. 153-181.
- [JMHJ01] Johnson, D.; Maltz, D.; Hu, Y.-C.; Jetcheva, J.: The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR), 2001.  
<http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-06.txt>
- [PRD01] Perkins, C.; Royer, E.; Das, S.: Ad Hoc On Demand Distance Vector (AODV) Routing for IPv6, 2001.  
<http://www.cs.ucsb.edu/eroyer/txt/aodv6.txt>
- [GTK02] GTK+ - The GIMP Toolkit.  
<http://www.gtk.org/>. Januar 2002.
- [CJLMMQV01] Clausen, T.; Jacquet, P.; Laouiti, A.; Minet, P.; Muhlethaler, P.; Qayyum, A.; Viennot, L.: Optimized Link State Routing Protocol, 2001.  
<http://www.ietf.org/internet-drafts/draft-ietf-manet-olsr-05.txt>
- [PBD01] Perkins, C.; Belding-Royer, E.; Das, S.: Ad hoc On-Demand Distance Vector (AODV) Routing, 2001.  
<http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-09.txt>
- [PR99] Perkins, C.; Royer, E.: Ad-Hoc On-Demand Distance Vector Routing. Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications, February 1999.
- [HD98] Hinden, R.; Deering, S.: IPv6 Multicast Address Assignments, 1998. RFC 2375.
- [PRD00] Perkins, C.; Royer, E.; Das, S.: IP Broadcast in Ad hoc Mobile Networks, 2000.  
<http://www.ietf.org/internet-drafts/draft-perkins-manet-bcast-00.txt>
- [RT99] Royer, E.M.; Toh, C.-K.: A Review of Current Routing Protocols for Ad-Hoc Mobile Networks. IEEE Personal Communications 6(2), April 1999; S. 46-55.