

Rheinische Friedrich-Wilhelms Universität Bonn  
Institut für Informatik

# **STAR**

\*

ein effizientes Routing Protokoll  
für ad hoc Netze

Seminar: Rechnernetze

Florian Röder  
[roederf@cs.uni-bonn.de](mailto:roederf@cs.uni-bonn.de)

# Inhalt

<u><a href="#">Inhalt</a></u>	2
<u><a href="#">1. Einleitung</a></u>	3
<u><a href="#">2. Charakterisierung von STAR</a></u>	3
<u><a href="#">2.1 Informationsaustausch zwischen Routern</a></u>	3
<u><a href="#">2.2 Aktualisierung der Pfade</a></u>	4
<u><a href="#">3. Beschreibung von STAR</a></u>	5
<u><a href="#">3.1 Konzept der genutzten Datenstruktur</a></u>	5
<u><a href="#">3.2 Gültigkeit von Updates überprüfen</a></u>	6
<u><a href="#">3.3 Austausch von Update-Informationen</a></u>	6
<u><a href="#">3.4 Auswirkung des Link-Layers</a></u>	10
<u><a href="#">3.5 Header Formate</a></u>	11
<u><a href="#">4. Leistung von STAR im Vergleich</a></u>	14
<u><a href="#">4.1 Vergleich mit table-driven Protokollen</a></u>	14
<u><a href="#">4.2 Vergleich mit on-demand Protokollen</a></u>	15
<u><a href="#">5. Stand der Implementation</a></u>	17
<u><a href="#">6. Zusammenfassung</a></u>	17
<u><a href="#">7. Literaturangaben</a></u>	19

# 1. Einleitung

Ad hoc Netze bestehen aus mobilen Routern und mobilen Endgeräten. Die Topologie solcher Netze ist naturgemäß sehr dynamisch, und die Anforderungen an ein Routing Protokoll sind sehr hoch. Eine Veränderung der Struktur wird nicht nur durch die Mobilität hervorgerufen, sondern auch durch Signalverluste, Interferenzen oder einfach durch Abschalten einzelner Geräte. Man hat also das Problem, dass nicht nur optimale Pfadinformationen schnell veraltet sind, sondern auch, dass bestehende Pfade nicht mehr genutzt werden können und neu gesucht werden müssen. Dies hat zur Folge, dass die relativ geringe Bandbreite, die wir heutzutage bei Funknetzen haben, durch den Informationsaustausch der Router zusätzlich belastet wird. Ein geringer Datenoverhead trägt auch dazu bei, die Energiereserven der Router zu schonen, da mobile Geräte ja meistens mit Batterie gespeist werden und eine möglichst lange Lebensdauer erwünscht wird.

STAR (Source Tree Addaptive Routing) ist ein Protokoll, das versucht mit einem Minimum an Informationsaustausch zwischen den Routern auszukommen und trotzdem effizientes Routing zu ermöglichen. Um dies zu erreichen, speichert jeder Router einen sogenannten Source Tree. Dabei handelt es sich um eine Baumstruktur, in der ein Weg vom Router zu allen Empfängern gespeichert ist. Zusätzlich werden die Source Trees der direkten Nachbarn und einige Informationen über die gesamte Struktur des Netzes gespeichert. Dadurch wird erreicht, dass oft geringe Updateinformationen ausreichen um den eigenen Source Tree korrekt zu aktualisieren. Dieser wird mit Hilfe der benachbarten Source Trees und der zusätzlichen Informationen über die Netztopologie berechnet. Nur in seltenen Fällen werden ganze Source Trees ausgetauscht.

## 2. Charakterisierung von STAR

Protokolle für ad hoc Netze lassen sich anhand verschiedener Merkmale charakterisieren. Zum einen kann man sie danach einteilen, auf welche Art die Router die Informationen austauschen, und zum anderen kann man danach gehen, von welcher Art die Informationen sind, die benutzt werden, um die Pfade zu den Empfängern zu berechnen. Beim erstgenannten Kriterium wird zwischen *table-driven* und *on-demand* unterschieden; beim zweitgenannten zwischen *link-state* und *distance-vector*. Weiterhin wird unterschieden nach welcher Strategie die benutzten Pfade aktualisiert werden. Auch hier gibt es zwei Ansätze: Entweder wird versucht, möglichst optimale Pfade zu benutzen, d.h. kürzeste Pfade oder Pfade mit minimalen Kosten, oder es wird versucht möglichst geringen Datenoverhead zu erzeugen, indem Pfade solange wie möglich benutzt werden.

### 2.1 Informationsaustausch zwischen Routern

Table-driven bedeutet, dass ein Router Pfadinformationen zu jedem Empfänger besitzt, die in einer Routingtabelle gespeichert werden. Diese wird aktualisiert, sobald es nötig ist, d.h. sobald z.B. ein neuer Router entdeckt wird. Die Information über die neue Verbindung wird an alle bekannten Router gesendet. Dieses Verhalten führt normalerweise bei ad hoc Netzen wegen der Dynamik zu einem hohen Datenoverhead. Deshalb sollte ein solches Protokoll zumindest optimale Pfade erstellen um effizientes Routing zu ermöglichen.

Meistens wird table-driven im Zusammenhang mit link-state eingesetzt. Routing Algorithmen nach dem link-state Verfahren arbeiten wie folgt: Jeder Router speichert die gesamte Netztopologie, d.h. Informationen über jede Verbindung zwischen zwei Routern. Wenn Updates verschickt werden, so muß nur die Information über eine Verbindung verbreitet werden, was im allgemeinen mit wenig Datenoverhead möglich ist. Da jeder Router die gesamte Netztopologie kennt, entscheidet jeder Router mittels Shortest Path Algorithmus selber, wo die Pakete hingeschickt werden. Ein bisschen im Gegenteil dazu verhält sich ein Algorithmus, der nach dem distance-vector Verfahren arbeitet. Dabei speichert jeder Router eine Liste, die alle bekannten Router enthält und die Entfernung (meist in Hops) zu dem jeweiligen Router. Um die Tabellen zu aktualisieren müssen die gesamten Tabellen von Zeit zu Zeit ausgetauscht werden, d.h. die Updates wachsen mit der Größe des Netzes. Ein weiterer Nachteil ist, dass alle Router sich aktiv an dem Verfahren beteiligen müssen, damit es effizient funktionieren kann. Bei sehr vielen Änderungen in der Netztopologie dauert es meist im Vergleich zu link-state Verfahren sehr lange, bis alle Tabellen aktualisiert sind, d.h. distance-vector Verfahren skalieren nicht so gut wie link-state Verfahren.

Beim On-demand Routing werden nur die nötigsten Informationen gespeichert, also die für die aktuellen Datenströme wichtig sind. Wird ein Empfänger adressiert, für den der Pfad unbekannt ist, wird mittels flooding eine Anfrage an alle Router gesendet um die fehlende Information zu erhalten. Mit on-demand wird meistens der distance-vector Ansatz in Verbindung gebracht. Dabei ist klar, dass keine optimalen Pfade benutzt werden können, da sich einmal erstellte Pfade nur ändern, wenn dies unumgänglich ist, z.B. weil ein Router nicht mehr erreichbar ist, der Pfad aber gerade benutzt werden soll.

STAR ist das erste table-driven Protokoll, das mit link-state Informationen arbeitet und, das in ad hoc Netzen effizienter arbeitet als on-demand Protokolle. Darüber hinaus muss man bei STAR allerdings nicht unbedingt den Ansatz verfolgen, dass die Pfade optimal sind, sondern kann Abweichungen zulassen, um den Informationsaustausch zwischen den Routern zu reduzieren.

## 2.2 Aktualisierung der Pfade

Wie bereits erwähnt, gibt es zwei Strategien für die Vorgehensweise um die Pfade der Router zu aktualisieren: Den *optimum routing approach* (ORA) und den *least overhead routing approach* (LORA).

Bei ORA ist die Strategie, die Routingtabellen möglichst schnell auf den neuesten Stand zu bringen, damit optimale Pfade benutzt werden können. Diese Vorgehensweise ist natürlich in Verbindung mit on-demand Protokollen nicht sehr lohnenswert, da sonst das gesamte Netzwerk andauernd mit Updateinformationen überflutet wird. Im Moment wird ORA nur in Verbindung mit table-driven Protokollen in ad hoc Netzen eingesetzt. Oft handelt es sich dabei um Abwandlungen von bestehenden Protokollen für verdrahtete Netze. Um den Datenoverhead zu minimieren, kann ein Netz in Gruppen eingeteilt werden, oder es werden Updates nur in gewissen Zeitabständen verschickt.

Bei LORA wird ein Pfad solange wie möglich benutzt, um einen minimalen Kontrollfluss zu erhalten. Diese Strategie wird meist in Verbindung mit on-demand Protokollen verwendet, so dass Router nur Anfragen verschicken müssen, wenn sie die Adresse eines Empfängers nicht kennen. In Verbindung mit table-driven Protokollen wurde der LORA Ansatz noch nicht benutzt. Das liegt daran, dass anfängliche Protokolle meist mit Ausschnitten der gesamten Netztopologie oder mit Entfernungstabellen gearbeitet haben, und die Router nicht erkennen konnten, wenn sich z.B. Kreise gebildet hatten. Um dem vorzubeugen, mussten regelmäßige Updates verschickt werden, um die Routingtabellen zu

überprüfen. Diese Updates widersprechen aber schon der Idee von LORA, so dass ORA ein geeigneter Ansatz erschien.

STAR ist das erste table-driven Protokoll, das nach dem LORA Prinzip implementiert werden kann. Denn die eingesetzten Source Trees erlauben es den Routern Schleifenbildung zu erkennen und zu verhindern. Deshalb müssen keine regelmäßigen Updates verschickt werden, um die benutzten Pfade zu validieren. Natürlich kann STAR auch nach dem ORA Prinzip eingesetzt werden.

### 3. Beschreibung von STAR

Formal lassen sich ad hoc Netze als Digraph  $G=(V,E)$  beschreiben. Jeder Knoten steht für einen Router bzw. Sender/ Empfänger. Eine Kante zeigt an, dass zwischen zwei Sendern eine Verbindung besteht. Zur Vereinfachung kann man annehmen, dass alle Nachrichten einmal und in endlicher Zeit verarbeitet werden. Außerdem sollen die Router korrekt arbeiten und es sollen bei der Übertragung keine Fehler auftreten. Ein Source Tree ist ein Spannbaum, der die Kanten enthält, die benutzt werden, um die einzelnen Knoten zu erreichen. Dieser Source Tree wird an alle direkten Nachbarn verschickt. Diese wiederum bilden aus den ihnen schon bekannten Verbindungen und denen, die sie gesendet bekommen, einen Graphen, der die Topologie des Netzes widerspiegelt (im folgendem *Topology Graph* genannt). Mit Hilfe dieses Graphen wird der eigene Source Tree errechnet, indem ein Shortest Path Algorithmus ausgeführt wird (z.B. Dijkstra). Da jeder Router die Source Trees seiner Nachbarn kennt, braucht z.B. Router A keine Meldung von Router B, wenn bei Router B die Verbindung nach C ausfällt. Denn sobald Router B eine neue Verbindung hinzunimmt, wird Router A dies anhand des Source Trees von B erkennen (eine Ausnahme stellt natürlich der Fall dar, wenn Router A nun keinen Weg mehr zu C hat). Updates bestehen aus einer oder mehreren LSUs (link-state update unit). Eine LSU enthält Statusinformationen über eine Verbindung, die bei einer Verbindung von  $u$  nach  $v$  nur von  $u$  gesendet werden kann. Jede LSU besitzt eine Sequenz Nummer, anhand der erkannt werden kann, welche Nachrichten Gültigkeit besitzen. Ein Router löscht eine Verbindung aus dem Topology Graph, sobald sie in keinem Source Tree der Nachbarn vorkommt und nicht mit dem Router adjazent ist. Die Router müssen auch keine LSUs als regelmäßige Updates schicken, da die Informationen gültig bleiben, solange die Verbindung besteht. Nach welcher Regel Updates verschickt werden, hängt davon ab, welche Strategie bei der Implementierung verfolgt wird: ORA oder LORA.

#### 3.1 Konzept der genutzten Datenstruktur

Im folgenden wird eine formale Beschreibung der Daten gegeben, die jeder Router abspeichern muss, und die verschickt werden. Jeder Router  $i$  muss folgende Daten bereithalten:

- $TG[i]$  = Topology Graph
- $ST[i]$  = Source Tree
- $N[i]$  = Menge aller benachbarten Router
- $ST[i,x]$  = Source Tree von Router  $x$ , wobei  $x \in N[i]$
- $TG[i,x]$  = Topology Graph von Router  $x$ , wobei  $x \in N[i]$
- Routingtabelle
- $SN[i]$  = Zähler für die Sequenznummer

Dabei sind folgende Details zu beachten: Ein Eintrag für eine Kante in  $TG[i]$  besteht aus einem Tupel  $(u,v,c,sn,del)$ . Die Kosten der Kante  $(u,v)$  werden mit  $c$  bezeichnet (als Kosten könnte man z.B. die Anzahl der hops über diese Kante nehmen oder die Verzögerung der Verbindung) und  $sn$  ist die Sequenznummer der LSU. Die Variable  $del$  wird auf *true* gesetzt, wenn die Kante  $(u,v)$  aus  $TG[i]$  entfernt werden soll. Ein Eintrag für einen Knoten in  $TG[i]$  besteht auch aus einem Tupel  $v = (d,pred,suc,suc\_old)$ . Dabei steht  $d$  für die Entfernung von Router  $i$  zu Router  $v$ ,  $pred$  ist der Router der auf dem Weg von  $i$  nach  $v$  vor Router  $v$  kommt,  $suc$  ist der nächste Router nach  $i$  auf dem Weg von  $i$  nach  $v$ , und  $suc\_old$  speichert den alten Wert von  $suc$ , falls  $suc$  überschrieben wird. Die gleichen Daten müssen für die Knoten und Kanten in  $ST[i]$ ,  $ST[i,x]$  und  $TG[i,x]$  ebenfalls gespeichert werden. Die Routingtabelle besteht aus den Kanten des Source Trees  $ST[i]$ . Dabei wird für jede benutzte Verbindung  $(u,v)$  auf dem Weg von  $i$  nach  $w$  folgendes gespeichert: die Adresse des Empfängers  $w$ , die Kosten der Verbindung zum Empfänger  $w$  und die Adresse von  $v$  (next hop). Bekommt Router  $i$  ein Update von Router  $x$ , so wird die neue Verbindung zuerst in  $TG[i,x]$  eingetragen. Nachdem Router  $i$  die Nachricht komplett verarbeitet hat, wird  $ST[i,x]$  aktualisiert. Deshalb entsprechen sich danach  $ST[i,x]$  und  $TG[i,x]$  im Allgemeinen, wodurch Speicherplatz eingespart werden kann. Verfährt man nach LORA muß zusätzlich noch eine Kopie  $ST^*[i]$  von  $ST[i]$  angelegt werden, wenn  $ST[i]$  verändert wird.

### 3.2 Gültigkeit von Updates überprüfen

Da in einem Netzwerk verschiedene Verbindungen auch verschiedene Bandbreite haben können, und da die meisten Verbindungen unterschiedliche Auslastung haben, kommt es bei der Versendung von Nachrichten von einem Router zum anderen zu Verzögerungen unvorhersehbarer Länge. Dadurch können sich Updateinformationen überschneiden und eine eintreffende LSU könnte überholte Informationen tragen. Deshalb benötigt man einen Mechanismus, der einem Router bei einer eintreffenden LSU sagt, ob die darin enthaltenen Informationen gültig sind oder nicht. Diesen Mechanismus kann man auch nutzen, um zu verhindern, dass Nachrichten immer im Kreis verschickt werden. So wird es zumindest bei STAR gemacht. STAR benutzt dazu Sequenznummern. Dabei handelt es sich um einen Zähler, der vom Sender einer LSU hochgezählt werden darf. Zur Vereinfachung reicht es aus, wenn jeder Router nur einen Zähler bereithält, wodurch sich aufeinanderfolgende Sequenznummern für den gleichen Datenstrom, um mehr als eins unterscheiden können. Eine LSU wird für gültig erklärt, wenn die darin eingetragene Sequenznummer größer als diejenige ist, die bei der letzten Nachricht über die gleiche Verbindung abgespeichert wurde. Ein Sonderfall ist natürlich dann gegeben, wenn für die Verbindung noch kein Eintrag im Topology Graph existiert. In STAR brauchen LSUs nicht durch regelmäßige Updates aktualisiert zu werden, wie dies in vergleichbaren Protokollen, die Sequenznummern benutzen, der Fall ist. Ein Router löscht eine LSU nur aus dem Topology Graph, wenn es sich bei der Verbindung um eine scheinbar abgebrochene Verbindung handelt oder wenn die Verbindung in keinem Source Tree mehr vorkommt (also weder in  $ST[i]$  noch in  $ST[i,x]$ ).

### 3.3 Austausch von Update-Informationen

In dem folgendem Abschnitt wird davon ausgegangen, dass ein am Netz neu angemeldeter Router in endlicher Zeit entdeckt wird. Gleiches gilt für den Abbruch einer Verbindung. Alle Update – Nachrichten sind Broadcast-Nachrichten an alle Nachbarn des jeweiligen Routers, der die Nachricht verschicken möchte. Wann bei STAR Updates verschickt werden ist davon abhängig, welcher Ansatz verfolgt wird: ORA oder LORA. Bei ORA reicht es aus, wenn ein Update verschickt wird, sobald sich der eigene Source Tree ändert. Ein Update enthält dann den ganzen Source Tree des jeweiligen Routers. Dadurch

wird erreicht, dass alle benachbarten Router ihren eigenen Source Tree eventuell aktualisieren können, um optimale Pfade zu benutzen. Wird der LORA Ansatz verfolgt, gestaltet sich das ganze etwas aufwendiger: Es werden Updates verschickt

- wenn ein Empfänger unerreichbar wird,
- wenn ein neuer Empfänger entdeckt wird,
- wenn es den Anschein hat, dass sich Schleifen gebildet haben
- oder wenn die Kosten für einen Pfad einen vorgegebenen Schwellwert überschreiten.

Die einzelnen Fälle werden dadurch entdeckt, dass der eigene Source Tree mit denen verglichen wird, die ein Router gesendet bekommt. Je nachdem welcher Fall eintritt, wird entweder der ganze Source Tree als Update geschickt (z.B. ein neuer Router), oder aber es werden nur Teilinformationen gesendet (z.B. bei einem unerreichbaren Router). Um die oben genannten Fälle zu unterscheiden, muss ein Router nach folgenden drei Regeln handeln:

1.) Router  $i$  sendet ein Update, wenn er einen neuen Empfänger entdeckt, oder einer seiner benachbarten Router einen Neuen meldet.

Entdeckt Router  $i$  einen neuen Empfänger, der gleichzeitig sein Nachbar ist, so wird die entsprechende LSU im Source Tree  $ST[i]$  eingetragen und ein Update verschickt. Für einen neu gebooteten Router bedeutet dies, dass er über seine Verbindungen LSUs verschickt, da er selbst ein neuer "Nachbar" ist. Am besten ist ein Support auf der Link-Layer Ebene geeignet, um neue Nachbarn schnell bekannt zu machen (siehe Kapitel 4).

2.) Router  $i$  sendet ein Update, wenn die Kosten zu mindestens einem Empfänger einen festgesetzten Schwellwert  $d$  überschreiten.

Nimmt man an, dass  $d = \text{unendlich}$ , so werden Updates verschickt, sobald eine Verbindung die Kosten unendlich hat, d.h. die Verbindung besteht nicht mehr. Wird ein Empfänger (oder sogar ganze Teile des Netzes, z.B. wenn der unerreichbare Knoten auf dem Pfad zu anderen Empfängern liegt) unerreichbar, so muss im entsprechenden Source Tree nur für die ausgefallenen Verbindungen ein Update generiert werden. Die darunter liegenden Knoten sind automatisch nicht mehr erreichbar, d.h. es muss kein Update für alle unerreichbaren Knoten verschickt werden.

3.) Router  $i$  sendet ein Update, wenn

- a) ein Pfad direkt eine auf eine Schleife schließen lässt,
- b) ein neu gewählter Nachfolger von Router  $i$  auf dem Pfad zu einem Empfänger eine größere Adresse besitzt als Router  $i$ , oder
- c) die Entfernung von einem neuen Nachfolger  $n$  auf dem Pfad zu einem Empfänger  $j$  länger ist, als die Entfernung von dem vorherigem Nachfolger zum gleichen Empfänger (Es sei denn  $j$  ist ein direkter Nachbar von  $i$  und die Verbindung  $(i,j)$  war ausgefallen).

Alle drei Regeln (a,b, und c) benötigt STAR, um Schleifenbildung zu verhindern. Bekommt Router  $i$  ein Update von einem Nachbarn  $x$ , so wird zuerst der entsprechende Source Tree  $ST[i,x]$  aktualisiert. Dann wird untersucht, für welche Empfänger Nachbar  $x$  den Router  $i$  benutzt (anhand von  $ST[i,x]$ ), und ob Router  $i$  für diese Empfänger eventuell  $x$  benutzt. Ist dies der Fall, so hat sich eine Schleife gebildet (Regel a). Also muss einer der beiden Router seinen Source Tree ändern.

Die Regel b) beruht auf folgender Beobachtung: Jeder Router hat eine eindeutige Adresse und in einer Schleife gibt es einen Router mit einer kleinsten Adresse. Wird immer ein Update verschickt, sobald ein neuer Nachfolger gewählt wird, der eine größere Adresse als der Router selber hat, so ist es nicht möglich, dass in einer Schleife kein Router eine Meldung verschickt. Nach Regel a) kann dann die Schleife gefunden und aufgelöst werden.

Regel c wird benötigt, wenn die Kosten einer bidirektionalen Verbindung in beide Richtungen unterschiedlich sind. Die Notwendigkeit dieser Regel wird im folgendem an einem Beispiel gezeigt. Angenommen wir haben ein Netzwerk mit sechs Knoten wie in Abbildung 1a. Alle Verbindungen seien bidirektional und die Kanten (a,b) und (b,c) haben die Kosten fünf, alle anderen haben die Kosten eins. Die Knoten sind von a bis f markiert und so mit einer eindeutigen Adresse (lexikografische Sortierung) ausgezeichnet. In Abbildung 1b bis 1d sind die Source Trees der markierten Knoten dargestellt. Angenommen die Regel c würde in STAR nicht benutzt, so kann man folgendes Szenario erzeugen, in dem es zur Schleifenbildung kommt. Nehmen wir an die Verbindung (c,d) fällt aus. So wird der Source Tree von Router c wie in Abbildung 1e aussehen. Da der neu gewählte Nachfolger b auf dem

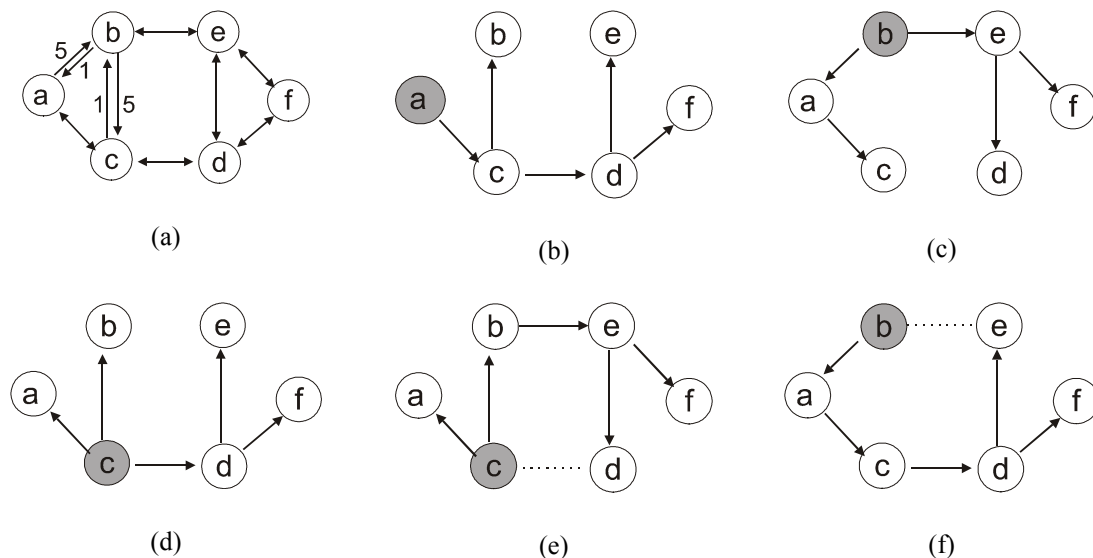


Abbildung 1

Pfad zu e eine kleinere Adresse hat als d, muss kein Update verschickt werden. Nehmen wir weiter an im direkten Anschluss fällt die Verbindung (b,e) aus, so wird der Source Tree von b geändert, wie in Abbildung 1f zusehen. Auch hier muss kein Update verschickt werden, da a eine kleinere Adresse hat als e. Jetzt hat sich aber eine Schleife gebildet. Denn soll z.B. von b nach d ein Paket gesendet werden, so wird es zuerst nach a geschickt, dann nach c und c schickt es wieder nach b.

Dieses Beispiel würde nicht funktionieren, wenn die Kosten der Kanten gleich wären. Denn so würde der Source Tree von b so aussehen, dass b den Knoten c als nächsten Knoten wählt, um d,e und f zu erreichen. Dann müsste b aber ein Update verschicken, da der Knoten c eine größere Adresse hat als b. Daß bei unterschiedlichen Kosten für eine Verbindung Regel b nicht ausreicht, liegt also daran, dass zwei benachbarte Router so nicht unbedingt die gleiche Verbindung wählen, um sich gegenseitig zu erreichen, und so auch die Schleifenbildung nicht mitbekommen, weil jeder Knoten einen mit kleinerer Adresse wählen kann und somit kein Update verschicken muss.

In Abbildung 2 ist der gleiche Fall wie oben dargestellt, diesmal aber mit Benutzung von Regel c. Sobald die Verbindung (c,d) ausfällt, wird ein Update verschickt, denn die Kosten für den Pfad nach f sind durch den neu gewählten Nachfolger, und den damit verbundenen Umweg, gestiegen. Dadurch aktualisiert a seinen Source Tree entsprechend. Fällt nun die Verbindung (b,e) aus, so weiß der Knoten b, dass d,e und f nicht mehr erreichbar sind, da er durch das vorherige Update weiß, dass auch (c,d) ausgefallen ist. Deshalb

verschickt b ein Update, in dem die Kosten für die Kante (b,e) auf unendlich gesetzt werden. Die Schleifenbildung wird hier im Beispiel dadurch verhindert. Hier sieht man auch, dass ein Update für die ausgefallene Verbindung ausreicht, damit alle Knoten wissen, daß der Subgraph bestehend aus d, e und f nicht mehr erreichbar ist.

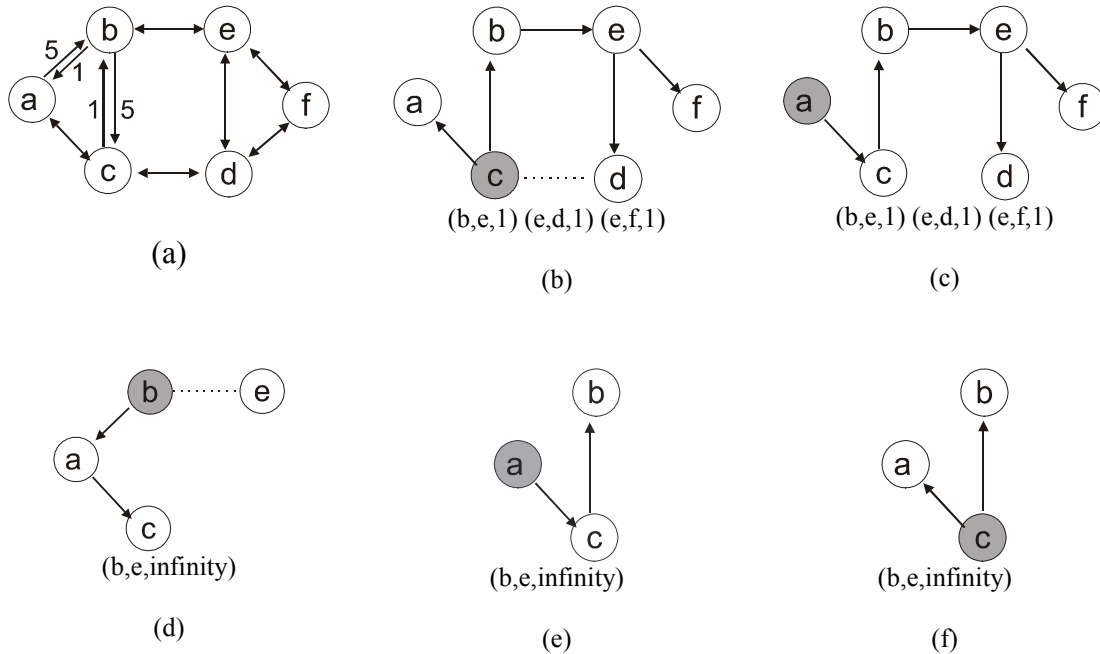


Abbildung 2

In Abbildung 3 ist schließlich noch der Ausnahmefall gezeigt, dass der Pfad zu mindestens einem Empfänger zwar länger geworden ist, der neue Nachfolger aber ein Nachbar dessen Knotens ist, zu dem die Verbindung ausgefallen ist. Anschaulich ist klar, dass dadurch keine Schleife gebildet werden kann.

Damit die oben aufgeführten Regeln auch funktionieren, wenn nur Informationen über eine einzelne Verbindung als Update gesendet werden (und nicht der gesamte Source Tree), muss jeder Router den Source Tree seiner Nachbarn kennen. Es gibt zwei unterschiedliche Arten von Updates, einmal der gesamte Source Tree oder aber nur einzelne LSUs: Sind in dem neuen Source Tree von Router i neue Nachbarn hinzugekommen, so sendet er den gesamten Source Tree als Update, damit seine Nachbarn möglichst schnell, von den Verbindungen, die Router i kennt, sich (hoffentlich) einen Vorteil schaffen können. Sind keine neuen Nachbarn hinzugekommen, so werden nur die benötigten LSUs verschickt, damit die benachbarten Router den neuen Source Tree errechnen können. Um sicherzustellen, dass in STAR nicht

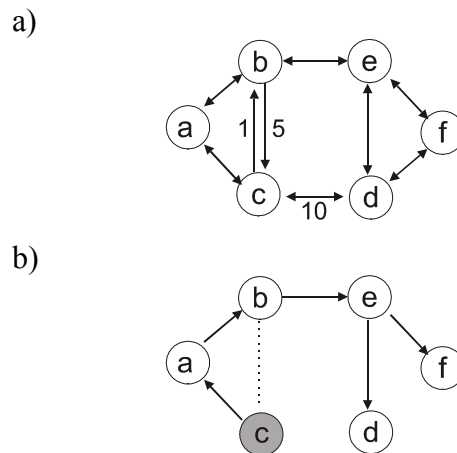


Abbildung 3

endlos Updates zwischen Routern hin und hergeschickt werden, wenn eine Schleife entdeckt wird, muss es eine Regel der Art „der Router mit der kleineren Adresse muss seinen Pfad ändern“ geben. Dadurch wäre eindeutig festgelegt, welcher Router die Schleife aufbrechen muß.

### 3.4 Auswirkung des Link-Layers

Inwieweit sich das verwendete Protokoll auf Link-Layer Ebene auf das Verhalten von STAR auswirkt, ist davon abhängig, welcher Ansatz bei STAR implementiert wird. Bei ORA spielt es keine Rolle, ob das zugrundeliegende Protokoll sicheres broadcasting unterstützt. Denn hier werden immer als Updateinformationen die gesamten Source Trees versendet. Kommt eine Nachricht nicht an, wird spätestens beim nächsten Update die Verlorene implizit mitgeschickt. Bei LORA sieht das etwas anders aus: Hier können sich durch verloren gegangene Updates Schleifen bilden. Denn angenommen, Router A sendet B eine LSU, die besagt, dass Router A B benutzen möchte, um nach C zu gelangen; aber die Nachricht kommt nicht an. Router B könnte aber Router A benutzen, um Router C zu erreichen, wodurch sich eine Schleife gebildet hat. Bisher sind wir in Kapitel 3 davon ausgegangen, dass alle Updates auch wirklich ankommen. Unterstützt das zugrundeliegende MAC Protokoll also kein sicheres Broadcasting, so reichen die unter 3.3 definierten Regeln nicht mehr aus, damit STAR Schleifenbildung verhindern kann. Mit zusätzlichen Regeln kann man dies aber wieder sicherstellen. Man verschiebt dabei sozusagen die Aufgabe des Link Layer Protokolls, eine sichere Übertragung zu leisten, auf STAR-Ebene. Dabei müssen alle Pakete die schon durchlaufene Route in ihrem Header speichern. Weiterhin sind vier zusätzliche Regeln nötig.

4.) Router  $i$  sendet ein Update als zuverlässige unicast Nachricht an den Nachbarn, der Router  $i$  dazu veranlasst hat, seinen Source Tree zu aktualisieren und ein Update zu verschicken.

Darunter fällt auch der Fall, dass Router  $i$  den Source Tree von Nachbar  $j$  aktualisiert hat, also  $ST[i,j]$ . Zusätzlich wird die Nachricht natürlich auch noch als broadcast Nachricht an alle anderen Nachbarn geschickt.

5.) Ein Router  $i$  sendet periodische Updates an einen Nachbarn, wenn dieser in seinem Source Tree keinen Pfad zu einem Empfänger aufweist, den Router  $i$  aber in seinem Source Tree hat.

Diese regelmäßigen Updates sollten ca. alle 60 sec. geschickt werden und zwar solange, bis der entsprechende Nachbar den fehlenden Eintrag in seinem Source Tree berichtet. Diese Regel ist einleuchtend, denn solch ein Fall deutet darauf hin, dass ein Update von Router  $i$  nicht angekommen sein könnte. Zusätzlich kann man noch vereinbaren, dass jeder Router alle 600 sec (oder in längeren Zeitabständen) Updates verschickt.

6.) Router  $i$  sendet ein Update, wenn er zu einem Empfänger keinen Pfad hat, zu diesem aber ein Paket verschicken soll.

Router  $i$  bekommt also ein Datenpaket, das an einen Empfänger  $j$  gesendet werden soll. Router  $i$  hat aber nicht den vollständigen Pfad zu dem Empfänger. Deshalb schickt er eine Nachricht an all seine Nachbarn, um zu melden, dass ihm der Pfad nach  $j$  fehlt. Darauf hin schickt ihm jeder benachbarte Router aufgrund von Regel 4 eine zuverlässige unicast Meldung zurück. Router  $i$  verschickt diese Meldung solange, bis er den entsprechenden Pfad mitgeteilt bekommt. Dabei sollten die Intervalle zu Beginn etwas kürzer sein, z.B. 600msec, 6sec, 60sec und danach immer 600 sec oder mehr.

7.) Router  $i$  sendet ein Update, wenn er aufgrund der durchlaufenden Route eines Paketes feststellt, dass eine Schleife vorhanden ist.

Bekommt ein Router  $i$  ein Paket, das für einen Empfänger  $j$  bestimmt ist, muss er prüfen, ob in dem Pfad, den das Paket schon durchlaufen hat, ein Router vorkommt, den  $i$  benutzt, um den Empfänger  $j$  zu erreichen. Dies kann man anhand des Source Trees sehr schnell erkennen. Ist dies der Fall, muss eine Schleife vorhanden sein, und das Paket wird zunächst verworfen. Die Schleife kann folgendermaßen aufgelöst werden: Der Router  $i$  schickt eine Route-Repair Nachricht. Diese enthält den Source Tree  $ST[i]$  und die Route des Pakets. Die Route-Repair Nachricht muss zuverlässig an den Knoten gesendet werden, der am Anfang des Route-Repair-Path steht. Dies entspricht dem letzten Knoten auf dem durchlaufenden Pfad, der zuerst in dem Source Tree  $ST[i]$  auf dem Weg nach  $j$  vorkommt. Dabei handelt es sich sozusagen um den Beginn der Schleife. Der nächste Router, an den  $i$  die Nachricht gesendet hat, entfernt sich selber vom Route-Repair-Path und überschreibt den Source Tree der Nachricht mit seinem eigenem. Danach schickt er die Meldung weiter, bis sie schließlich bei dem Kopf des Route-Repair-Path ankommt. Dieser aktualisiert seinen Source Tree dahingehend, dass die Schleife aufgelöst wird. Dabei ist noch zu beachten, dass eine Route-Repair Nachricht nur gesendet wird, wenn nicht schon innerhalb der letzten 30 Sekunden eine gesendet wurde. Dies ist deshalb sinnvoll, da den Router innerhalb sehr kurzer Zeit viele Pakete erreichen können, für die diese Schleife entdeckt wird (z.B. wenn die Pakete alle zum gleichen Empfänger sollen). Ansonsten würde das Netz mit Route-Repair Nachrichten überflutet werden.

Diese Regeln machen im gewissen Umfang den großen Vorteil von STAR, keine regelmäßigen Updates verschicken zu müssen, wieder zunichte. STAR versucht zwar, nur im Ausnahmefall periodische Abfragen zu machen, aber wie oft diese Ausnahmen auftreten und wie groß sie auf die Performance des Protokolls sind, muss sich sicherlich erst noch bei der Simulation zeigen. Eine weitere Alternative wäre, dass jeder Router auch unter LORA bei jedem Update den gesamten Source Tree sendet. Diese Lösung hat aber ebenfalls zur Folge, dass ein Teil der Vorteile, die STAR bietet, nämlich möglichst geringen Datenoverhead zu erzeugen, wieder reduziert wird.

In den bis jetzt speziell für ad hoc Netze implementierten Protokollen für die Link-Layer Ebene ist sicheres Broadcasting noch nicht umgesetzt worden. Dies liegt daran, dass durch die dynamische Struktur des Netzes hohe Anforderungen an ein solches Protokoll gestellt werden müssen. Bisher gibt es z.B. nur ein Standard von IEEE 802.11, der aber auch nur für den Einsatz im ISM (Industrial, Science, Medicine) Band vorgesehen ist.

### 3.5 Header Formate

In den vorangegangenen Kapiteln wurden verschiedene Arten von Updates beschrieben. Diese sollen hier noch mal genauer spezifiziert werden; insbesondere soll der Unterschied zwischen den verschiedenen Formten aufgezeigt werden. Abbildung 4 zeigt den groben Aufbau eines Update Pakets in STAR. Es besteht aus einem Header, einer Bestätigungsliste (*Acknowledgement List*) und einer LSU Liste. Alle drei Bereiche werden im folgendem genauer beschrieben.

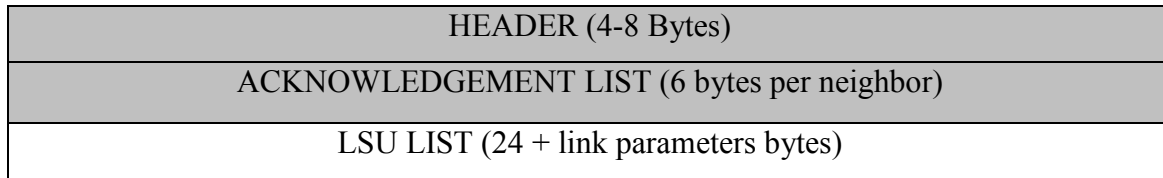


Abbildung 4

Der Header hat eine variable Länge von vier oder acht Bytes und spezifiziert eine von fünf möglichen Nachrichtenarten: GUM, PUM, TUM, RUM und SUM. Einen kurzen Überblick über die Bedeutung gibt Tabelle 1.

Typ	Wert	Name	Art der Nachricht	Bedeutung
SUM	0x1	Start Update Message	Broadcast/ Unicast	Signalisiert, dass der Router neu gestartet wurde
GUM	0x2	Goodbye Update Message	Broadcast	Signalisiert, dass der Router für einige Zeit un erreichbar sein wird
PUM	0x3	Partial Update Message	Broadcast/ Unicast	Enthält eine oder mehrere LSUs
TUM	0x4	Total Update Message	Broadcast/ Unicast	Enthält den kompletten Source Tree
RUM	0x5	Reset Update Message	Broadcast	Signalisiert ein Reset und veranlaßt andere Router die Sequenznummer nicht mit vorherigen zu vergleichen

Tabelle 1

Die Header unterscheiden sich kaum (Abbildung 5). SUM hat eine Länge von 8 Bytes und die anderen haben eine Länge von 4 Bytes. SUM muß zusätzlich die Adresse des neuen Routers beinhalten, damit seine Nachbarn diese speichern können. Das Feld *version* muss auf eins gesetzt werden. *Num of ACKs* gibt an, wie viele Einträge in der Bestätigungsliste sind. Die Sequenznummer ist aus dem Wertebereich  $[0; 2^{15} - 1]$ . Das *most significant bit* wird gesetzt, wenn die Sequenznummer als Nummer des Eintrages in der Bestätigungsliste interpretiert werden soll.

Die Bestätigungsliste besteht aus Einträgen von Adressen der Nachbarn, von denen

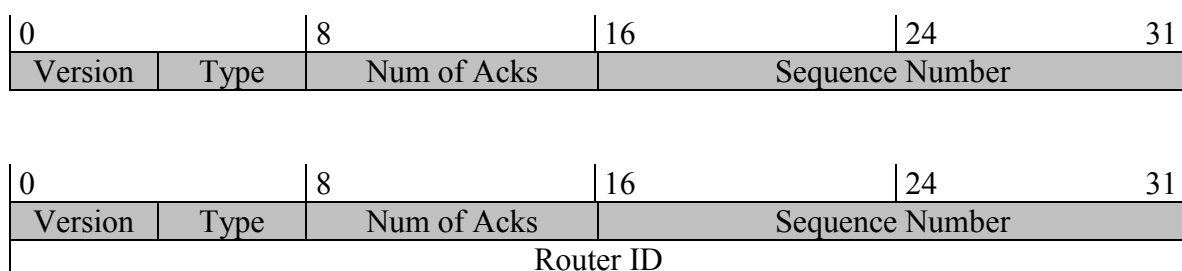


Abbildung 5

Pakete erhalten werden und, denen deshalb Bestätigungen geschickt werden müssen. In dem Feld *Expected Sequence Number* wird wie der Name schon sagt, die als nächstes zu erwartende Sequenznummer des jeweiligen Routers eingetragen. Das Bit *R* ist gesetzt, wenn der Router selber auf eine Bestätigung des Routers wartet, der im Adressfeld spezifiziert ist (Abbildung 6).

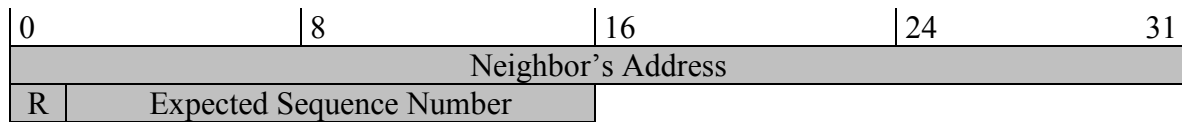


Abbildung 6

Die LSU Liste besteht wiederum aus einzelnen Einträgen für LSUs (Abbildung 7). Das Feld *Head Prefix* und *Tail Prefix* geben die Anzahl der Einsen an, die sich in der Netzwerkmaske zu den IP Adressen der Verbindung befinden. Das Bit *H* bzw. *T* gibt an, ob das Feld *ID of the Head of the link* bzw. *ID of the Tail of the link* vorhanden ist. In den beiden Feldern ist die eindeutige ID der Router gespeichert, zwischen denen die Verbindung besteht. Das Feld *Link Type* kann benutzt werden, um zu spezifizieren, ob über die Verbindung Broadcast-Nachrichten geschickt werden, es sich um eine Point-to-Point Verbindung handelt, oder ob es sich beispielsweise um eine Verbindung zu Endgeräten handelt. In dem *TOS Bit Vector* wird das entsprechende Bit gesetzt, wenn die Verbindung, den damit verbundenen Type of Service unterstützen soll. Dies können z.B. Forderungen sein, dass diese Verbindung eine hohe Auslastung haben sollte. Diese Forderungen, die sich damit angeben lassen, werden allerdings von keinem Router garantiert. Das Feld *Time Stamp* enthält die vergangene Zeit seit dem 1. Januar 1990 in Sekunden. Daran kann erkannt werden, ob die Informationen noch aktuell sind. In den nächsten beiden Feldern werden die IP Adressen der beiden Geräte gespeichert, zwischen denen die Verbindung besteht. Die Feldern, in denen die eindeutige ID gespeichert ist, wird benötigt, falls ein Router mehrere IP-Adressen besitzt.

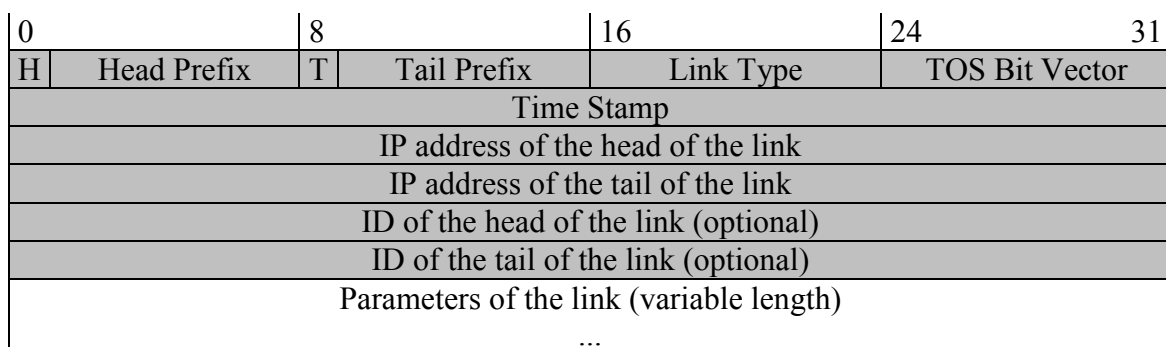


Abbildung 7

## 4. Leistung von STAR im Vergleich

Der Aufwand, den STAR betreibt entspricht bei Speicherbedarf, Zeit und Kommunikation im worst case dem von anderen effizienten Protokollen, z.B. ALP, ein table-driven Protokoll, das nach dem distance-vector Verfahren arbeitet. Doch wie sieht es mit dem durchschnittlichem Aufwand unter realen Bedingungen aus? Um darüber Aussagen machen zu können, muß man eine Simulation durchführen, die möglichst den Bedingungen entspricht, denen das Protokoll auch unter echtem Einsatz ausgesetzt ist. Die Ergebnisse solch einer Simulation beschreiben J.J. Garcia-Luna-Aceves und Marcelo Spohn in ihrem Paper „Source-Tree Routing in Wireless Networks“. Die Testbedingungen waren dabei wie folgt: Das Protokoll selber wurde mit dem C++ Protocol Toolkit (CPT) implementiert, in dem der Protokoll Stack so ziemlich den gleichen Code benutzt, wie er in echten Systemen eingesetzt wird. Als Netzwerkprotokoll wurde IP eingesetzt, als MAC Protokoll ein ähnliches wie IEEE 802.11 und als Physical Layer wurde direct sequence spread spectrum radio, das eine Bandbreite von 1 Mbit/s erlaubt, eingesetzt. STAR wurde so konfiguriert, dass eine Verbindung als ausgefallen gilt, wenn sie zehn Sekunden nicht mehr erreichbar ist.

Das Simulationsszenario war folgendermaßen aufgebaut: 20 Knoten wurden auf eine Ebene der Größe 5km x 7km zufällig mit einer Dichte von 1 Knoten pro km<sup>2</sup> verteilt. Um die Bewegung der Knoten zu simulieren, wurde das Random Waypoint Model eingesetzt. Die Arbeitsweise der Knoten in diesem Model lässt sich wie folgt beschreiben:

- Jeder Knoten bleibt für die Dauer von *pause\_time* an seiner Position
- Dann wird ein zufälliger Punkt in der Ebene bestimmt
- In diese Richtung bewegt sich der Knoten für fünf bis elf Sekunden mit einer Geschwindigkeit von 20 m/s
- Der Knoten verweilt wieder für die Dauer von *pause\_time* und der Vorgang beginnt von vorne

Dieses Verhalten wird für die ganze Dauer der Simulation angenommen.

### 4.1 Vergleich mit table-driven Protokollen

Als table-driven Protokolle wurden TOB (topology broadcast) und ALP zum Vergleich herangezogen, weil TOB einerseits streng den traditionellen link-state Ansatz umsetzt und ALP andererseits als eines der effizientesten table-driven Protokolle gilt. TOB arbeitet mit flooding von link-state Informationen innerhalb des gesamten Netzes oder nur in Teilbereichen. ALP ist ebenfalls ein link-state Protokoll. Beide Protokolle, ALP und TOB, arbeiten mit optimalen Pfaden, weshalb zum fairen Vergleich STAR mit ORA-Strategie eingesetzt wurde. Weiterhin wurde auf Link-Layer Ebene sicheres Broadcasting verwendet. Da dies bei STAR unter ORA keinen so großen Einfluß hat (wie im Vergleich zu unsicherem Broadcasting) spielt dies für die wohl eine vernachlässigbare Rolle. Es wurden mehrere Durchläufe gemacht mit unterschiedlichen Werten für *pause\_time*, wie man Tabelle 2 entnehmen kann. Die gesamte Simulation lief jeweils 900 Sekunden.

<i>pause_time</i>	Verbindungswechsel	generierte Pakete (Updates)		
		STAR	ALP	TOB
30	154	411	1765	5577
45	102	262	1304	3908
60	90	239	1144	2502
90	50	138	623	1811

Tabelle 2

Wie man sieht, steigt die Anzahl der Verbindungswechsel mit Verkürzung der Zeit, die sich die Knoten nicht bewegen. Dies ist klar, da die Verbindungen viel länger aufrechterhalten werden können, wenn die Knoten an einem festen Ort verweilen. Desto mehr Verbindungswechsel man hat, desto größer ist natürlich auch die Anzahl der verschickten Updates, da ständig Verbindungen nicht mehr verfügbar sind und dies den benachbarten Routern mitgeteilt wird. Bei STAR ist die Anzahl der generierten Pakete ca. um den Faktor drei größer als die Anzahl der Verbindungswechsel, sie steigt also linear an. Bei ALP ist die Anzahl der Pakete mindestens vier mal so groß und bei TOB um mindestens 10 mal so groß wie bei STAR. STAR generiert also sehr viel weniger Datenoverhead, insbesondere wenn das Netz eine hohe Dynamik aufweist.

Der Grund dafür liegt an der Art und Weise, wie und wann Updates verschickt werden. Bei ALP speichert jeder Router die gesamte Topologie des Netzes. Erhöhen sich die Kosten einer Verbindung, die ein Router benutzt, und wird sie deshalb entfernt, so wird dies den Nachbarn mitgeteilt, wenn die Verbindung den Status zwei hat. Andernfalls muß sie den Status eins haben und wird auf zwei gesetzt anstatt eine Nachricht zu verschicken. Bei STAR dagegen sind im Topology Graph nur Kanten gespeichert, die die direkten Nachbarn des Routers benutzen. Deshalb wird nur ein Update verschickt, wenn die ausgefallene Verbindung auch von einem Nachbarn benutzt wird. Deshalb ist bei hoher Anzahl an Verbindungswechsel der Datenoverhead bei STAR geringer als bei ALP.

#### 4.2 Vergleich mit on-demand Protokollen

Als on-demand Protokoll wurde DSR (Dynamic Source Routing) zum Vergleich gewählt, da es als eines der effizientesten Protokolle für ad hoc Netze gilt. STAR wurde nach dem LORA Prinzip implementiert und alle sieben Regeln, die unter 3.3 und 3.4 beschrieben sind kamen zum Einsatz, d.h. das MAC-Protokoll unterstützte kein kollisionsfreies Broadcasting. Beide Protokolle wurden so konfiguriert, dass sie bis zu 20 Pakete puffern können, wenn kein Pfad für sie vorhanden ist. Es wurde wieder ein Netz mit 20 Knoten generiert und die *pause\_time* wurde auf die Werte 0, 15, 30, 45, 60 Sek. gesetzt. Die gesamte Simulation lief wieder 900 Sekunden. Das Ziel sollte sein, herauszufinden, wie die Protokolle bei sehr hoher Dynamik reagieren, wie hoch unter diesen Bedingungen der *Goodput* zu den Empfängern ist. Dafür wurden drei verschiedene Szenarios betrachtet:

- 8 Datenströme mit CBR = 4 Pakete/s, d.h. insgesamt 32 Pakete/s (Pakete zu 64 Bytes), zu 8 eindeutigen Empfängern;
- 20 Datenströme mit CBR = 1,6 Pakete/s, d.h. insgesamt 32 Pakete/s (Pakete zu 64 Bytes), zu 8 eindeutigen Empfängern;
- 14 Datenströme mit CBR, davon 7 mit 4 Pakete/s zu **einem** Empfänger D, sowie 7 mit 4 Pakete/s von D zu 7 beliebigen Empfängern.

Die Datenströme wurden alle zwischen der 20. und 120. Sekunde gestartet, damit sich die Router in den ersten 20 Sekunden ihren Nachbarn bekannt machen. In Tabelle 3 sind die Ergebnisse zusammengefasst. Dabei ist zu sehen, dass bei STAR die Anzahl der Updatepakete linear zu der Anzahl der Verbindungswechsel ist (und das mit einem Faktor von  $\frac{1}{2}$  während es bei STAR unter ORA noch der Faktor 3 war), während die Anzahl der verschickten Updates bei DSR nicht nur von den Verbindungswechseln abhängig ist, sondern auch stark von der Anzahl der Datenströme im Netz. DSR kann sich bei der Anzahl der Updates nur einen Vorteil gegenüber STAR verschaffen, wenn wenig Ströme im Netz sind und die Dynamik nicht allzu groß ist. In den meisten Fällen verschickt aber STAR wesentlich

pause_time	Verbindungswechsel	Anzahl an Datenströmen	Update Pakete		überbrachte Datenpakete		generierte Pakete
			STAR	DSR	STAR	DSR	
0	1461	8	908	791	15110	14740	24100
		14	930	1460	15845	10975	25917
		20	916	3122	13689	6830	23718
15	605	8	615	460	19544	20831	24396
		14	636	702	23027	23210	25989
		20	686	1535	17254	10129	23649
30	424	8	559	350	20180	20492	24160
		14	551	464	23086	23228	25892
		20	580	763	19929	18341	23716
45	350	8	517	280	21685	22683	24100
		14	526	2352	23776	20481	25917
		20	507	1880	20749	19898	23731
60	322	8	522	482	22536	19102	24100
		14	507	1357	24473	23436	25917
		20	493	744	22218	21899	23775

Tabelle 3

weniger Updates als DSR. Auffällig ist auch, dass bei 20 Datenströmen und einer *pause\_time* von 0 DSR erheblich weniger Daten zu den Empfängern bringt, während STAR sich kaum beeinflussen lässt. Die hohe Differenz zwischen generierten und überbrachten Paketen liegt daran, dass das MAC-Protokoll ungesendete Pakete aus seiner Warteschlange verwirft, wenn die Verbindung ausfällt. Der Nachteil, den DSR bei dieser Simulation hat, liegt an seiner Arbeitsweise: jedes Paket enthält eine Route, die das Paket voraussichtlich nehmen kann, um sein Ziel zu erreichen. Diese Liste kann bei Bedarf von den Routern aktualisiert werden, wenn das Paket bei ihnen vorbeikommt. Wenn das Netz zwischenzeitlich partitioniert wird, weil mehrere Verbindungen ausfallen, so werden bei DSR die Pakete, die nicht mehr weiterkommen verworfen und müssen neu gesendet werden. Die Router versuchen in der Zeit mittels Flooding einen Weg für die nicht erreichbaren Router zu finden – natürlich vergebens, zumindest solange das Netz geteilt bleibt. Bei hoher Dynamik kann es aber vorkommen, dass kurzzeitig das Netz öfters in mehrere nicht verbundene Teile getrennt wird. Sind dann auch noch mehrere Datenströme (z.B. in der Simulation mit 20) unterwegs, werden natürlich auch sehr viele Datenpakete verworfen und das Netz ist wegen des einsetzenden Flooding ausgelastet, obwohl keine Daten verschickt werden. Bei STAR geschieht dies eben nicht. Wird das Netz einmal getrennt, so wird das Fehlen der Verbindungen maximal einmal durch das gesamte erreichbare Netz als Update gesendet. Deshalb sinkt die Auslastung bei STAR in diesem Fall ab, und sobald wieder eine Verbindung besteht, kann viel schneller reagiert werden, und somit müssen nicht so viele verlorene Pakete erneut gesendet werden. Um diesen Unterschied zwischen STAR und DSR hervorzuheben, wurde zusätzlich noch ein Szenario getestet, in dem nach 900 Sekunden ein Empfänger komplett ausfällt und die Simulation noch 900 Sekunden weiterläuft. Bei STAR erhöht sich die Anzahl der Updates um 15% (1823 statt 1583), während es bei DSR 55% (3043 statt 1963) sind (dabei werden die Zahlen mit einem Simulationslauf verglichen, der ohne Ausfall des Empfängers 1800 Sekunden lief).

Ob diese Simulationen nun wirklich zeigen, dass STAR eindeutig besser ist als DSR ist fraglich. Denn ein großer Kritikpunkt ist, dass das verwendete Szenario für DSR nicht gut gewählt ist, d.h. DSR kann seine Stärken nicht ausspielen. In den Testläufen beträgt die Anzahl der Empfänger 40% von der Anzahl der gesamten Knoten. Dadurch hat DSR eine sehr geringe Auswahlmöglichkeit, was die Route für die Pakete angeht, und der Vorteil, dass

die Router den vorgegebenen Weg verändern können und somit das Paket einen möglichst kurzen Weg findet, geht praktisch verloren. Bei den Simulationen war DSR in den Fällen STAR überlegen, wo die Anzahl der Datenströme nicht so hoch war. Interessant wäre mit Sicherheit ein Szenario, indem die Anzahl der Knoten etwas größer im Vergleich zu den Datenströmen gewählt würde. Ob also STAR oder DSR besser ist, hängt sehr von dem Zustand des Netzes ab. Was sich jedoch mit Sicherheit sagen lässt, ist, dass STAR DSR auf keinen Fall unterlegen ist.

## 5. Stand der Implementation

STAR läuft zwar noch nicht in kommerziellen Systemen, jedoch ist das Protokoll schon mehr oder weniger lauffähig. In dieser Version arbeitet es oberhalb von UDP und der Code dieser Version ist der MANET Gruppe zugänglich. Die Korrektheit von STAR ist bewiesen und soll noch veröffentlicht werden.

Ein großer Nachteil von STAR ist, dass es bei sehr großen Netzen auch sehr viel Speicher verbraucht. Man stelle sich mal als schlimmsten Fall einen vollständigen Graphen als Netz vor, so muss jeder Router den Source Tree eines jeden Knoten speichern. Damit wird auch der Rechenaufwand, den STAR betreibt, um zumindest optimale Pfade in dem ihm bekannten Teilnetz zu finden, erheblich größer. Diese beiden Punkte sind ein großer Nachteil, wenn man bedenkt, dass mobile Router ihren Strom aus einer Batterie beziehen, die möglichst lange halten soll; denn beides bringt wahrscheinlich einen erhöhten Stromverbrauch mit sich. Genau diesen Nachteil versucht eine Weiterentwicklung des STAR Protokolls auszumerzen. Dabei handelt es sich um SOAR (Source-Tree On-Demand Adaptive Routing). Dieses Protokoll funktioniert ähnlich wie STAR, nur speichert es nicht Informationen von allen möglichen Empfängern im Netz, sondern nur von denen, die gebraucht werden, um sogenannte *active destinations* zu erreichen. Dabei gilt ein Empfänger für einen Router als *active destination*, wenn der Router Datenpakete zu diesem Empfänger weiterleitet oder selber sendet. Die Vorgehensweise ähnelt der von STAR unter LORA mit dem Unterschied, dass SOAR ein on-demand Verfahren ist. Die Router geben auf Anfrage ihrer Nachbarn nur Informationen über Pfade, die zur Zeit in Benutzung sind. In der Veröffentlichung [4] wird die Korrektheit von SOAR bewiesen und es wird gezeigt, dass dieses Protokoll bei Simulationen mit hoher Mobilität genauso wie bei geringer Mobilität besser abschneidet als DSR.

## 6. Zusammenfassung

STAR ist mit Sicherheit nicht das Routing Protokoll schlechthin und ist noch verbesserungsbedürftig. Aber es ist jetzt schon den meisten anderen eingesetzten Protokollen überlegen und ist mindestens so effizient wie DSR, wenn die Netzstruktur dies zulässt. Den größten Vorteil kann STAR für sich verbuchen, wenn das Netz eine hohe Mobilität mitbringt. Denn dann kann STAR sogar das bewährte DSR Protokoll übertreffen. Dies liegt zum einen an dem geringem Datenoverhead, den STAR produziert, zum anderen an der Eigenschaft

schnell auf Verbindungsabbrüche und –wechsel zu reagieren, da es auch in kritischen Situationen die Bandbreite nicht mit Routinginformationen auslastet. Eine Schlüsselfunktion ist, dass STAR erlaubt von einem optimalen Pfad abzuweichen, obwohl es sich um ein table-driven Protokoll handelt. Das zeigt, dass auch Protokolle dieser Art für ad hoc Netze interessant sein können. Da STAR auch im Zusammenspiel mit Mechanismen wie Clustering benutzt werden kann, kommt es für viele Anwendungsmöglichkeiten im Bereich ad hoc Netze in Frage.

## 7. Literaturangaben

- [1] J.J. Garcia-Luna-Aceves und Marcelo Spohn, *Bandwidth-Efficient Link-State Routing in Wireless Networks*, 2001
- [2] J.J. Garcia-Luna-Aceves und Marcelo Spohn, *Source Tree Routing in Wireless Networks*, ICNP '99
- [3] J.J. Garcia-Luna-Aceves, Marcelo Spohn und David Beyer, *Source Tree Adaptive Routing (STAR) Protocol*, IETF MANET Working Group, Internet-Draft, Oktober 1999
- [4] Soumya Roy und J.J. Garcia-Luna-Aceves, *Using Minimal Source Trres for On-Demand Routing in Ad Hoc Networks*, 2001
- [5] David B. Johnson, David A. Maltz, Yih-Chun Hu und Jorjeta G. Jetcheva, *The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks*, IETF MANET Working Group, INTERNET-DRAFT, March 2001
- [6] Douglas E. Comer, *Internetworking with TCP/ IP*, 4<sup>th</sup> Edition, Prentice Hall, 2000