MiNEMA: Kommunikationsparadigmen

Christian Dreier, Michael Jakab, Elmar Stellnberger

{cdreier, mjakab, estellnb}@edu.uni-klu.ac.at

1 Gruppenkommunikation in Mobilen Ad-hoc-Netzwerken

1.1 Einleitung

Mensch und Tier arbeiten zur Erreichung ihrer Ziele häufig in Gruppen zusammen; man denke nur an ein Team von Softwareentwicklern oder Murmeltieren. Während ein Murmeltier stets Wache hält, grasen die anderen gemütlich. Ist Gefahr im Verzug, etwa durch einen Adler, der seine Kreise zieht, so warnt der Aufpasser seine Familienmitglieder durch ein schrilles Pfeifen. Auch der Zusammenschluß von Rechnern in Gruppen bietet zahlreiche Vorteile, wenn es um die Erfüllung von Anforderungen wie Verfügbarkeit, Ressourcennutzung, Leistung und Sicherheit oder einfach um die Lösung komplexer Probleme geht. Ein zentraler Server wäre weder von den Systemressourcen noch von der Netzwerkanbindung her in der Lage, diese Anforderungen zu erfüllen. In großen Netzwerken wie dem Internet versucht man diese Ziele durch die Spiegelung von Daten auf mehreren Servern sowie die Einrichtung von Proxies, welche Daten für den lokalen Gebrauch zwischenspeichern, zu erreichen. Ein Netz solcher Größe ist ohne eine hierarchische Struktur wie der eben beschriebenen kaum denkbar.

In mobilen Ad-hoc-Netzwerken hingegen ist der Aufbau einer fixen Infrastruktur zur Erfüllung oben genannter Aufgaben nicht möglich. Hier sind Mechanismen gefragt, welche den spontanen Zusammenschluß von lokal verfügbaren Rechnern zu Gruppen ermöglichen, die idealtypischerweise aus lauter gleichwertigen Rechnern bestehen.

Beispielsweise könnte ein Auto nach einer Bremsung versuchen, mit benachbarten Fahrzeugen Kontakt aufzunehmen, und den Zusammenschluß ebenfalls gebremster Automobile in einer Gruppe initiieren [7]; dies würde die Erkennung eines Staus oder schlimmstenfalls gar einer Massenkarambolage ermöglichen, sodaß andere Autofahrer rechtzeitig vorgewarnt werden könnten - entweder über eine Meldung im Radio oder indem der Bordcomputer eine alternative Route zum Ziel berechnet.

Bei Bekanntwerden eines Unfalls oder Verbrechens könnten in der Nähe befindliche Einsatzfahrzeuge [4] automatisch zum Ort des Geschehens gelotst werden.

Im Katastrophenschutz und in der Aufklärung sind mobile Ad-hoc-Netzwerke sowie die Kommunikation in Gruppen gewiß von unschätzbarem Vorteil, einfach deshalb, weil es in solchen Situationen kein (funktionsfähiges) Festnetz gibt, auf das man zurückgreifen könnte. Bereits ein großflächiger Stromausfall (wie in Nordamerika im September dieses Jahres) reicht aus um das Telephon- und Handynetz lahmzulegen, da dieses ebenfalls fest verdrahtete Sendestationen benötigt.

Das Paradigma der Gruppenkommunikation umfaßt nicht nur die Gruppenbildung, sondern darüber hinaus natürlich auch die Kommunikation, welche sich meistens an eine größere Menge von Teilnehmern richtet. Die Unterstützung von Multicast-Services ist im Besonderen auch für Multimediaapplikationen von Bedeutung. Hier geht es darum, die Übertragung großer Datenmengen durch Aufteilung der Datenströme auf mehrere Rechner in den Griff zu bekommen. Beispiele für solche Anwendungen wären Video On Demand (Video auf Abruf), Videokonferenzen und andere Dialoganwendungen („Collaborative Computing“) [RS].

Auch Filesharing-Tools (vgl. Kazaa) scharen Rechner bezüglich eines angeforderten Datums und deren Verfügbarkeit. Die Möglichkeiten, die sich dem durchschnittlichen Internetnutzer derzeit bieten, sind leider sehr begrenzt, da die meisten Provider zwar ein großes Downstreamvolumen zum Herunterladen von Daten aber nur einen kleinen Upstream (64kbit/s~Modem) erlauben. Für Videoconferencing oder Filesharing wäre ein symmetrisches Verhältnis von Vorteil. Ein weiteres praktisches Problem ist die Konsistenz, da sich durch die wiederholte Replikation des Datums vor Erreichen des Bestimmungsortes allmählich Kopierfehler einschleichen und das Ergebnis verfälschen oder schlimmstenfalls die Zurückholung des Datums aus dem Rechnerverbund unmöglich machen, wenn nämlich die einzelnen Teile nicht mehr zusammenpassen. Für Fehlererkennung und -korrektur gibt es zwar gut ausgereifte Verfahren, bei derzeitigen Implementierungen kommt es aber trotzdem immer wieder zu Problemen, vor allem dann, wenn einzelne Benutzer gezielt

fehlerhafte Fragmente (mit korrekter Prüfsumme) einschleusen, aber auch diesem Problem könnte man mit einer lokalisierenden Fehlererkennung für das gesamte angeforderte Datum leicht begegnen.

Ebendieser Ansatz der verteilten Speicherung von Daten ist im Besonderen für mobile Ad-hoc-Netzwerke bestens geeignet; hier ist es allerdings notwendig, vermehrt lokale Daten einzubeziehen. Gruppenbasierte Systeme können hier rechtzeitig entstehende Partitionierungen vorhersehen, in Bearbeitung befindliche Transaktionen abschließen, stornieren oder bis zur nächsten Vereinigung verzögern und evtl. im Vorhinein Datenbestände zumindest an strategisch wichtigen Positionen verteilen, sowie die notwendigen Operationen zum Abgleich der inzwischen gesammelten Informationen bei einer Wiedervereinigung entsprechender Partitionen anstoßen. Somit lassen sich die zu Beginn angesprochenen Anforderungen (Verfügbarkeit, Sicherheit, Konsistenz…) auch in hochdynamischen mobilen Ad-hoc-Netzen, wo es keine festen Verbindungen gibt, gewährleisten.

1.2 Begriffsbestimmung und erläuternde Vorbemerkungen

1.2.1 Mobilität in Netzwerken

[2] unterscheidet folgende Arten von mobilen Netzwerken:

- Cellular Network (statisches Bezugspunktnetz, Fixpunktnetz)
- Virtual Cellular Network (relozierbares Bezugspunktnetz, Inertialnetz)
- Ad-hoc Network

Im sog. „Cellular Network“ läuft die Kommunikation mobiler Hosts stets über fest verankerte Basisstationen, außer sie befinden sich in direkter Reichweite zueinander. Beispiele für solche Fixpunktnetze wären das Mobilfunknetz oder das WLAN (Wireless Local Area Network).

Das „Virtual Cellular Network“ unterscheidet sich vom „Cellular Network“ dadurch, daß die Basisstationen nach Bedarf verlagert werden können; Sende- und Empfangseinrichtungen könnten z.B. in einem LKW installiert werden. Die Basisstationen verhalten sich aber immer noch so, als ob sie bezüglich eines Inertialsystem ruhen würden; d.h. es werden dieselben Routingalgorithmen wie in einem Netz mit fixen Bezugspunkten verwendet, sodaß sich an der Netztopologie im Laufe der Zeit nichts Grundlegendes ändern darf. Kommt es dennoch so weit, müssen die Basisstationen händisch rekonfiguriert werden.

In beiden Arten von Bezugspunktnetzen ist es nicht notwendig, daß einzelne Hosts Routingfunktionalität oder ähnliches übernehmen können: Solche Aufgaben werden an die Basisstationen delegiert, welche eine hierarchisch darüberliegende Ebene darstellen. In den vorhin besprochenen Netzen, nämlich Fixpunkt- und Inertialnetzen, kommunizieren die Hosts wenn nicht direkt stets über Basisstationen und ebendiese sind fest vorkonfiguriert.

1.2.2 Client/Server-lose Kommunikation in Ad-hoc-Netzwerken

Anders jedoch in einem Ad-hoc-Netzwerk: Dieses besteht aus einer beliebig angeordneten Menge von Computern, welche sich von selbst konfigurieren können. Die beteiligten Computer hören ihr Umfeld ab, erkennen andere Teilnehmer in ihrer direkten Reichweite und beginnen sodann Informationen über weiter entfernte Rechner auszutauschen. Wollen zwei entfernte Teilnehmer miteinander kommunizieren, übernehmen zwischenliegende Hosts die Vermittlung. Darüber hinaus stellt jeder Host einer Gruppe, der eigentlich primär an der Inanspruchnahme von Diensten interessiert ist, in Ermangelung einer zentralisierten Infrastruktur auch selbst Serverfunktionalitäten zur Verfügung. Die Organisation in Gruppen dient dabei, wie schon vorhin erwähnt, zur organisierten Verteilung der Datenbestände an bedeutungsvollen und sicheren Orten. Jeder Teilnehmer übernimmt dabei gleichzeitig die Rolle des Klienten und des Servers.

Kennzeichnend für diese Art von Netzen ist die Abwesenheit einer fest definierten Hierarchie. Idealtypisch besteht ein Ad-hoc-Netzwerk aus einer Menge uniformer Hosts mit gleicher Reichweite. Die Abwesenheit jeglicher Hierarchie führt aber allzuleicht zu einer schlechten Skalierbarkeit, weswegen unterschiedliche Reichweiten durchaus sinnvoll sein können (In kleinerem Maßstab läßt sich diesem Problem auch mit der bevorzugten Weiterleitung von Nachrichten, welche an weit voneinander entfernte Bestimmungsorte gerichtet sind, begegnen [7]). Gleich inwieweit das Netz homogen oder heterogen aufgebaut ist, solange das Verhältnis zwischen Sendereichweite und Empfindlichkeit des Empfängers dasselbe ist, sind die Knoten des Netzwerkes beidseitig voneinander erreichbar. Sendet hingegen A doppelt so stark wie B ohne gleichzeitig über einen empfindlicheren Empfänger zu verfügen, so erreicht A unter Umständen B, aber B nicht A. Befindet sich eine Störquelle X in der Nähe von B, so kann dies ebenfalls, obwohl A und B über gleichwertige Sende- und Empfangseinrichtungen verfügen, dazu führen, daß A B erreicht, aber nicht umgekehrt.

So gesehen müßte der Erreichbarkeitsgraph gerichtet sein, obgleich viele Modelle der Einfachheit halber lediglich einen ungerichteten Graphen verwenden und dadurch implizit voraussetzen, daß alle Verbindungen vollduplex, d.h. beidseitig, sind.

1.2.3 Multicasts und Routing

„Von einem einzelnen Paket, das an eine Multicastadresse abgesendet worden ist, stellt das Netzwerk eine Kopie dieses Pakets an jeden Host einer Gruppe aus. Den Hosts steht es dann frei, dieser Gruppe beizutreten oder sie zu verlassen, ohne sich mit anderen Hosts der Gruppe synchronisieren oder Vereinbarungen treffen zu müssen. Ein Host kann auch gleichzeitig in mehr als einer Gruppe Mitglied sein.“, merkt [66] an; prinzipiell kann man daraus eine rudimentäre Definition von Gruppe ableiten, sodaß jeder Multicast einen Fall von Gruppenkommunikation darstellt. Das Paradigma der Gruppenkommunikation, wie ich es hier vorstellen werde, umfaßt aber weit mehr, nämlich Mechanismen zur Gruppenbildung sowie die bereits angeschnittene verteilte Verwaltung von Daten, welche ohne strikte Trennung von Client und zentralem Server auskommt.

Neben Übermittlungen, welche sich an alle Gruppenmitglieder richten, sollten natürlich auch Meldungen an einzelne Rechner möglich sein. Auf das dafür nötige Routing (Wegberechnung) in mobilen Ad-hoc-Netzwerken wollen wir aber hier nicht näher eingehen, obwohl für bestimmte Zwecke der Austausch von Einzelmeldungen zwischen Vertretern unterschiedlicher Gruppen oder zwischen Anführer und Hosts einer Partition notwendig sein kann.

1.3 Probleme, Konzepte und Prinzipien

1.3.1 Problemstellungen

Soll ein Gruppenkommunikationsservice aufgesetzt werden, so muß dieses eine Reihe von Aufgaben und Anforderungen erfüllen können:

- Gruppenbildung
- Gewährleistung des Zusammenhaltes (aktiv/ passiv)
- Führen gemeinsamer Datenbestände
- verteilte Berechnung

Die Bildung von Gruppen beruht auf dem eigenständigen Zusammenschluß von Rechnern, welcher nach bestimmten Aspekten ausgerichtet ist:

- ***funktionale Aspekte*** beziehen sich auf den Zweck der Gruppenbildung (Stauerkennung, Aufklärung, Rettungseinsatz, …). Im einfachsten Fall

gibt der Bezeichner einer Gruppe ihren Zweck wieder; meistens sind aber mehrere Aspekte wie u.a. Ressourcennutzung, Fähigkeiten und Interessenslage mit einzubeziehen.

- ***lokale Aspekte*** sind vor allem in mobilen Netzwerken von großer Bedeutung. Beispielsweise wäre die Entfernung eines Einstazfahrzeuges vom Unfallsort ein lokaler Aspekt. Dabei können durchaus verschiedene Größen wie der kürzeste, der schnellste Weg und die Luftlinie eine Entfernung charakterisieren. Lokale Aspekte sind jedoch nicht nur für die physische Erreichbarkeit der Gruppenmitglieder verantwortlich sondern auch für eine reibungslose und störungsfreie drahtlose Kommunikation.

Die Hosts einer Gruppe sind auf die Fähigkeiten und Leistungen ihrer Mitglieder angewiesen, wenn die Gruppe nicht überhaupt zur Lösung einer gemeinsamen Aufgabe gebildet worden ist. Deshalb muß Wert auf den Zusammenhalt, der gegenseitigen Erreichbarkeit aller Mitglieder, gelegt werden. Unter aktiven Zusammenhalt versteht man die strategische Positionierung aller Teilnehmer, bspw. einer Aufklärunseinheit, um ein Gebiet möglichst gut und lückenlos (zur Beobachtung) abdecken zu können ohne daß der Funkkontakt dabei abbricht. Die meisten Systeme verfügen allerdings nur über eine Art von passivem Zusammenhalt, welcher garantieren soll, daß der Kontakt zu keinem Mitglied unerwartet abreißt, indem alle Teilnehmer in kritischen Situationen vorgewarnt werden. In Systemen mit aktivem Zusammenhalt gibt also der Rechner die Bewegung vor (z.B. Roboter wird bewegt), während in Systemen mit passivem Zusammenhalt Positionsänderungen zwar erfaßt aber nicht beeinflußt werden können (Mensch als Benutzer).

Sinn und Zweck der Gruppenbildung und des Zusammenhaltes ist es den Austausch von Neuigkeiten sowie das führen gemeinsamer Datenbestände und die Aufteilung von rechenintensiven Aufgaben auf mehrere auch in naher Zukunft sicher erreichbare Rechner zu ermöglichen. Damit kein Mitglied wichtige Neuigkeiten verpaßt, müssen Nachrichten zwischengespeichert werden können, falls einzelne Gruppenmitglieder zeitweilig nicht erreichbar sein sollten. Der in 1.5.3 vorgestellte total geordnete partitionsunabhängige Multicast soll genau dies gewährleisten.

1.3.2 Konzepte

Eines der wichtigsten Konzepte für Gruppenkommunikation in mobilen Ad-hoc-Netzwerken ist das der ***Partitionierung***. Können sich zu irgendeinem Zeitpunkt nicht alle Mitglieder gegenseitig erreichen, so zerfällt die Gruppe in disjunkte Teilmengen von Mitgliedern, in sogenannte Partitionen. Die Mitglieder einer Partition können sich einander alle erreichen (d.h.jeder jeden). Treffen sich die vorher getrennten Mitglieder

zweier unterschiedlicher Partitionen wieder, so werden die in der Zwischenzeit gesammelten Informationen wechselseitig abgeglichen. Die Partitionierung ist primär für Systeme mit passiven Zusammenhalt gedacht, kann aber auch in solchen mit aktiven Zusammenhalt zur Behandlung von Sonder- oder Ausnahmezuständen eingesetzt werden.

Befinden sich zwei Hosts in einem ***sicheren Abstand*** voneinander, so kann mit größter Wahrscheinlichkeit davon ausgegangen werden, daß die Verbindung zwischen ihnen innerhalb einer wohldefinierten Zeitspanne Δt nicht abreißt. Sichere Abstände fungieren als Mindestabstände zwischen den Teilhabern einer Partition oder Gruppe, um gewährleisten zu können, daß diese nicht auseinanderbricht bevor konsistenzerhaltende Vorkehrungen getroffen werden können. Die Einhaltung sicherer Abstände muß bei der Gruppenbildung, beim Zusammenschluß von Partitionen und vormals getrennter Gruppen sowie im laufenden Betrieb überwacht werden. Das Konzept der Einhaltung sicherer Abstände stellt eine Abstraktion realer örtlicher Gegebenheiten dar, weil darin nur die Abstände in Luftlinie nicht aber Hindernisse oder etwa Signallaufzeiten Berücksichtigung finden.

Die ***Latenz*** (auch: maximale Signallaufzeit) ist ein Resultat der Netztopologie und gibt die maximale Dauer bei der Weiterleitung zweier Pakete an, welche von unterschiedlichen Enden des Netzes stammen können. Geringe Latenzen sind für das Zeitverhalten bei oftmaligem Zusammenwarten (***Flush***) der Gruppenmitgliedern von Vorteil. Flushes dienen als Synchronisationspunkte u.a. bei der Spaltung oder Vereinigung von Partitionen oder nach einem Commit.

Die Gruppenbildung basiert auf funktionalen und lokalen Aspekten deren gewichtigster gewiß die Einhaltung eines sicheren Abstandes ist. Eine Möglichkeit die anhand von Aspekten wiedergegebenen Anforderungen an künftige Gruppenmitglieder zusammenzufassen ist die Verknüpfung anhand logischer Junktoren; allen voran *und* bzw. *oder*. Beispielsweise kann gefordert sein, daß bestimmte Ressourcen an den teilhabenden Rechnern verfügbar sein müssen und daß Daten in Form von Film, Ton oder Text zur Verfügung stehen müssen. Eine Möglichkeit der Umsetzung bietet hier die verteilte Errechnung von Mengenvereinigung und Durchschnitt aus Kapitel 1.4.1.

1.3.3 Prinzipien

Es mag verschiedenste Ansätze für Design und Implementierung geben; es ist jedoch sinnvoll sich im Zweifelsfall bei der Umsetzung von bestimmten Grundgedanken und Prinzipien leiten zu lassen: So beruht die Funktionsfähigkeit mobiler Ad-hoc-Netzwerke auf *dezentraler Organisation* und *spontaner*

Selbstorganisation. Eine Besonderheit ist hierbei die Vereinigung von Dienstanbieter und –geber (Client und Server) in einem Programm und Rechner. In Kapitel 1.4.1. ist die Realisierung einiger Operationen auf Partitionen mittels Partitionsanführer beschrieben, was dem Ansatz der Dezentralisierung entgegengerichtet zu sein scheint; all diese Aktionen können jedoch gleichfalls mittels dynamisch gewählter Vertreter realisiert werden.

Ein bedeutungsvolles Prinzip ist das der *Antizipation* (Vorwegnahme). Können unangenehme Ereignisse wie die Abtrennung von Partitionen vorausgesehen werden, so spart dies einigen Aufwand und bietet Vorteile gegenüber einer nachträglichen Problembehebung, weil damit bspw. einem Host vergebliche Kontaktaufnahmeversuche weitgehend erspart bleiben sollten und dispositive Enscheidungen (u.a. Wahl der Hauptpartition) getroffen werden können.

Ein weiterer Grundsatz ist die *Entlastung des Übertragungsmediums* von unnotwendigen Statusmeldungen. Mehrere solcher Meldungen können zusammengefaßt und mit Nutzdaten huckepack versandt werden. Grundsätzlich ist es vorteilhaft lieber mehr zu speichern und weniger zu verschicken, weil die Übertragungsbandbreiten von WLan- und Funkverbindungen im Vergleich zum verfügbaren Festwertspeicher sehr begrenzt sind. In näherer Zukunft ist hier eher mit einer weiteren Verschiebung zugunsten der Kapazität Festplatten zu rechnen.

Zerfällt eine Gruppe in Partitionen, welche später wieder zusammentreffen, so müssen die Daten abgegichen werden, um von nun an wieder auf einer gemeinsame Basis weiterarbeiten zu können. Die in der Zwischenzeit in isolierten Partitionen angestoßenen Transaktionen können erst konsolidiert werden (commit), wenn alle vorherigen durchgeführt worden sind. Das Prinzip der *Fairness* versucht sicherzustellen, daß Transaktionen aus keiner Partition zu kurz kommen oder bevorzugt werden. Meistens wird jedoch einer *ehebaldigsten Konsolidierung* der Vorrang gegeben, welches den Transaktionen einer ausgezeichneten Hauptpartition ein sofortiges Festsetzen erlaubt, sodaß sich jene aus Nebenpartionen hintanstellen müssen.

1.4 Architekturmodelle für Gruppenkommunikationsdienste in mobilen Ad-hoc-Netzwerken

Der Zusammenschluß in sogenannten *Proximity-Groups* beruht neben funktionalen auch auf lokalen Aspekten. Von den erreichbaren Geräten werden beispielsweise nur solche aufgenommen, die Interesse haben und sich innerhalb eines sicheren Abstandes von den übrigen Mitgliedern befinden, sodaß anzunehmen ist, daß sie in nächster Zeit in Reichweite bleiben. Andererseits kann z.B. die Gruppenzugehörigkeit zur Koordination der Bewegung einer automatisierten Aufklärungseinheit verwendet werden, um ein bestimmtes Gebiet möglichst gut

abzudecken. Schließlich sind noch Kommunikationsprimitive für die Übermittlung von Nachrichten an ein oder mehrere Gruppenmitglieder zur Verfügung zu stellen.

Die Implementierung folgt wie üblich dem Schichtenmodell. Jede Schicht bietet eine wohldefinierte Schnittstelle an und verwendet für deren Implementierung nur Funktionalitäten der direkt darunterliegenden Schicht. In der Literatur finden sich verschiedenste Vorschläge zu Spezifikation und Architektur von Rechnergruppen; hier ein recht allgemein gehaltener Ansatz in Anlehnung an [2].

Architekturvorschlag für Group Communication Systems (erweitertes Referenzmodell):

- Anwendungsschicht (z.B. ein Filesharing-Tool)
- Multicast Layer (Übermittlungsschicht)
- Membership Layer (Mitgliedschaftsschicht)
- Proximity Layer (Ortsbestimmungsschicht, Vermittlungsschicht)
- Data Link und Physical Layer

Hier der vom klassischen Internet her bekannte Aufbau (vgl. ISO/OSI-Referenzmodell):

- Anwendungs- und Darstellungsschicht (HTTP, SMTP(email), FTP, …)
- Transport- und Sitzungsschicht (UDP, TCP; Ports; Sitzungen)
- Vermittlungsschicht (IP; Routing)
- Sicherungs- und Bitübertragungsschicht (Ethernet, FDDI, Backbonenetze, …)

Die vier Schichten des Internet finden ihre Entsprechung auch im erweiterten Referenzmodell; vollkommen neu ist nur der Membership Layer. Er bestimmt, welche Rechner in die Gruppe aufgenommen werden sollen. Proximity- und Multicast Layer erweitern eigentlich nur die Funktionalität von Vermittlungs- und Transportschicht.

Der *Proximity Layer* setzt ähnlich dem Network Layer (Vermittlungsschicht) direkt auf der Hardware auf. In mobilen Ad-hoc-Netzwerken sendet jeder Host periodisch sog. Baken-Frames (eng.: beacon) aus, welche Aufschluß über sein Vorhandensein, seine momentane Position und evtl. seine IP-Adresse geben. Eine andere Möglichkeit ist es, jeden Teilnehmer auf ein Probe-Signal warten zu lassen und daraufhin in einem Probe-Response die gewünschten Informationen zurückzuliefern. In hochmobilen Ad-hoc-Netzwerken mit Fähigkeit zur Gruppenkommunikation wird aber meistens ersterer Ansatz gewählt, weil die sich ständig ändernden Positionen der Hosts dazu führen können, daß der Kontakt zwischen einzelnen Gruppenmitgliedern abreißt. Positionsänderungen werden daher stets an die darüberliegende

Mitgliedsschaftsschicht gemeldet, um den Zusammenhalt der Gruppe überwachen zu können.

Der momentane Aufenthaltsort eines Mitgliedes kann über ein Sattelitennavigationssystem (GPS, Galilieo (europ.), Glonas (russ.)) bestimmt werden. Diese Ebene sammelt alle für das Routing notwendigen Informationen, weshalb das Routing auch gleich hier implementiert werden kann.

Im Prinzip ist es möglich, ein vollkommen mobiles Netzwerk auch ohne IP-Adressen, nur über die ohnehin eindeutigen, vom Hersteller der Netzwerkkarte vorkonfigurierten MAC-Adressen,(Media Acess Layer) zu betreiben; die Verwendung einer hardwareunabhängigen Adressierung ermöglicht jedoch den Zusammenschluß von Rechnern mit unterschiedlichster Hardware und dadurch auch die Anbindung ans Festnetz, indem die Statusmeldungen Hardwareadresse, IP-Adresse und Aufenthaltsort des Absenders oder weiterer ihm bekannter Rechner enthalten. Im Festnetz gibt die IP-Adresse Aufschluß über den Ort, an den die Nachricht weitergeleitet werden soll; dies ist für mobile Hosts nicht besonders sinnvoll; eine Adreßvergabe, in der sich funktionale oder topologische Aspekte wiederspiegeln, wäre natürlich dennoch möglich.

Im ARP (Adress Resolution Protocol) dient die gleichzeitige Übermittlung der Adreßinformationen anderer Hosts ausschließlich der Entlastung des Übertragungsmediums von Statusmeldungen, in einem Funknetzwerk gibt sie auch Aufschluß über nicht direkt erreichbare Teilnehmer. Sendet jeder Host periodisch Informationen über alle ihm bekannten Hosts in seiner Gruppe aus, so erreichen einen die Informationen eines n Hops entfernten Host nach n Weiterleitungen. Das Wissen über die Positionen lediglich indirekt erreichbarer Computer ist nicht nur für die korrekte Weiterleitung von Einzelmeldungen unverzichtbar, sondern liefert auch der darüberliegenden Mitgliedsschaftsschicht wichtige Basisinformationen.

Auf dem Proximity Layer setzt der *Membership Layer* auf. Er bestimmt, wer dazugehört und wer nicht. Den funktionalen Aspekt einer Gruppe gibt zumeist ihre Benennung wieder. Daneben impliziert die Funktionalität auch viele Aufnahmekriterien. Hier kann es beispielsweise eine Einschränkung auf ein gewisses Areal wie den Umkreis um einen festen oder beweglichen Mittelpunkt, die aktuelle und künftig erwarteten Erreichbarkeit sowie die maximale Anzahl an Gruppenmitgliedern geben, entweder weil zur Lösung einer gewissen Aufgabe nicht mehr Rechner benötigt oder weil viele Implementierungen von Diensten die Kenntnis aller Mitglieder voraussetzen und daher nicht sonderlich gut skalierbar sind.

Diese Schicht muß aufgrund der vielen denkbaren Aufnahmekriterien entweder vielseitige Möglichkeiten bieten oder austauschbar sein. Letzteres ist durch das Designparadigma der Schichtung garantiert. Dazu müßte es das Betriebssystem dem Anwendungsprogramm gestatten, seine eigene Mitgliedsschaftsschicht an dieser Stelle

einzuschieben oder ein Callback anzugeben, was aber einer Aufweichung des Schichtenmodells gleichkäme.

Im laufenden Betrieb ist der Membership Layer dafür zuständig, den Zusammenhalt der Gruppe zu überwachen und dabei den einzelnen Mitgliedern eine konsistente Sicht (*View*) zu garantieren. Eine Gruppe kann in mehrere Partitionen zerfallen, nämlich genau dann, wenn die Verbindung zwischen mehreren disjunkten Mengen von Mitgliedern vorübergehend abreißt. Alle Hosts einer Partition müssen dieselbe Sicht teilen. Der Proximity Layer liefert jedem Computer die Menge aller Teilnehmer, deren Signale ihn direkt oder indirekt erreichen. Sind alle Verbindungen bidirektional, so ist die Menge der erreichbaren Hosts genau jene, die einen erreichen kann. In diesem Fall entspricht das Netz einem ungerichteten Graphen, und es kann die vom Proximity Layer gelieferte Menge an Hosts ziemlich direkt als View (Sicht) eingerichtet werden, wobei hiermit automatisch gewährleistet ist, daß alle Rechner einer Partition dieselbe Sicht teilen. Ansonsten muß die zurückgelieferte schwache Zusammenhangskomponente des gerichteten Netzgraphen auf eine starke reduziert werden, wobei natürlich der eigene Knoten enthalten bleiben muß.

Schließlich setzen auf der Mitgliedschaftsschicht neben den traditionellen Ende-zu-Ende–Protokollen (TCP, UDP) noch ein oder mehrere Implementierungen für Multicasts auf. Diese bilden den *Multicast*-Layer und stellen Primitive für die Kommunikation von einem oder mehreren Mitgliedern mit mehreren anderen zur Verfügung (one-to-many, many-to-many).

Multicasts werden bereits hardwaremäßig und darauf aufbauend im Internet auch auf IP-Ebene unterstützt und können mittels UDP von den Anwendungen genutzt werden. Einzelne Hosts treten einer Gruppe bei, indem sie die Netzwerkkarte oder den nächstgelegenen Router so konfigurieren, daß er Pakete mit der Multicastadresse, welche die gewünschte Gruppe eindeutig bezeichnet, an sie weiterleitet. Die Mitgliedsschaftsschicht könnte natürlich diese Konfiguration direkt vornehmen. Dem ist aus mehreren Gründen nicht so:

Zum Einen sollen neben dem so implementierbaren Datagramm-Multicast (IP/UDP) auch noch andere Durchsagearten unterstützt werden, zum Anderen reicht es nicht, seine Absichten dem für die aktuelle Domain zuständigen Router mitzuteilen, weil es einen solchen in mobilen Ad-hoc-Netzwerken gar nicht gibt und sich daher alle Teilnehmer an der Weiterleitung beteiligen müssen.

Datagrammdurchsagen eignen sich beispielsweise für die Übermittlung von multimedialen Datenströmen wie Film. Das Nachsenden von Bildinhalten verlorengegangener Frames ist bei einer Echtzeitübertragung nicht sinnvoll, da so viele Einzelbilder zu spät ankommen würden. Die Ordnung, in der die einzelnen Bilder decodiert werden können, muß nicht der Anordnung entsprechen, in der sie abgesandt worden sind, sodaß die Umordnung am besten von der Anwendung selbst

vorgenommen wird. Es gibt zwar Vorschläge für eigene Multimediaprotokolle (RTP, Real Time Transport Protocol), dennoch ermöglicht ein Datagrammservice vielfältigste Anwendbarkeit, indem es Aufgaben der Umordnung und der Erkennung verlorener Pakete der Anwendung überantwortet.

Andere Situationen erfordern wiederum die Übertragung eines geordneten verlustfreien Bytestromes an eine Menge von Mitgliedern der eigenen Partition. Handelt es sich dabei um eine einzige One-to-many-Verbindung, könnte man dieselbe Funktionalität auch durch den Aufbau von TCP-Verbindungen vom Sender zu allen Adressaten erreichen, was jedoch sehr ineffizient wäre. Sind mehrere Sender erlaubt, so kann eine absolute, für alle Empfänger idente Ordnung aller auf den Strom gemultiplexter Nachrichten erforderlich sein. Ein solches Service könnte in einem Chatroom für mehrere Benutzer oder zur Übermittlung von Befehlen eines Anwendungsprotokolls (vgl. FTP,…) an mehrere Adressaten benutzt werden.

Schießlich sollte auch ein Protokoll zur Verfügung stehen, das sich zur Übermittlung von Transaktionscodes eignet. Bei dem hier vertretenen dezentralen Ansatz der Datenspeicherung sollte am besten jeder Host den gesamten Datenbestand innehaben, um optimale Verfügbarkeit zu gewährleisten; jedenfalls sollte in jeder Partition zu jedem Zeitpunkt mindestens ein solcher Rechner vorhanden sein. Zu Beginn, wenn die Datenmenge noch gering ist, kann sie jeder Rechner aufnehmen, später, wenn sie soweit anwachsen sollte, daß einzelne Computer nicht mehr genügend freien Platz haben, kann der Datenbestand zunächst auf zwei, dann auf vier usf. Rechner aufgeteilt werden, währenddessen jeder Host weiterhin die wichtigsten Daten beibehält.

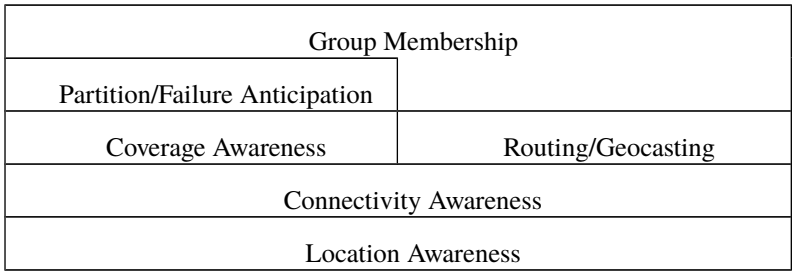
Nachrichten, die aufgrund einer Partitionierung nicht sofort zugestellt werden können, müssen bis zur nächsten Wiedervereinigung zwischengespeichert werden. Schlußendlich muß bei der Verschmelzung auch auf den in voneinander getrennten Partitionen initiierten Transaktionen eine totale Ordnung gefunden werden, um sicherzustellen, daß auf allen ursprünglich gleichen Datenbeständen dieselben Transaktionen in derselben Reihenfolge angewendet werden. Im Folgenden noch einmal eine Zusammenfassung der Anforderungen an ein solches total geordnetes, partitionsunabhängiges Multicast-Protokoll:

- zuverlässige, geordnete Verbindung
- Nachrichtenbasiert (nicht Bytestrom) eine Nachricht für eine Transaktion
- Speicherung von Nachrichten, bis Empfänger erreichbar
- totale Ordnung auf allen Nachrichten

Ein solches Protokoll benötigt klarerweise den Membership Layer, um Informationen über die Abspaltung und Vereinigung von Partitionen zu erhalten.

Damit wären die Aufgaben aller drei neu definierten Schichten festgelegt. Abschließend möchten wir aber noch das Schichtungsmodell aus [3] präsentieren, um zu zeigen, daß es hierzu durchaus unterschiedliche Ansätze gibt.

Die Connectivity Awareness steht dem Proximity Layer gegenüber, wobei das Routing und Geocasting auf eine darüberliegende Schicht ausgelagert worden ist. Daneben existiert noch ein sogenanntes Coverage Awareness, welches überwacht, ob das einer Gruppe zugeordnete Gebiet möglichst gut abgedeckt wird. Die Partition/Failure Anticipation („Teilungs- und Fehlervorwegnahme“) ist hier als eigenes Segment des Group Membership Layers dargestellt.



Dieser Ansatz mag sogar logisch besser nachvollziehbar sein als der anfangs vorgestellte, in der Praxis hingegen ist es oft angenehmer, an sich unterschiedliche Funktionen auf derselben Schicht zu implementieren, da eine zu rigide Kapselung nur zusätzlichen Aufwand und Performance-Einbußen mit sich bringen würde. Ähnlich wurden Anwendungs- und Darstellungsschicht sowie Transport- und Sitzungsschicht des ISO/OSI-Referenzmodells im Internet zusammen implementiert.

1.5 Implementierung und Algorithmen

1.5.1 Einrichtung einer konsistenten Sichtweise für gerichtete Netzwerke

Ganz so einfach ist es nun doch nicht, daß jeder Host nur periodisch Informationen über alle ihm bekannten Hosts in seiner Gruppe weiterzusenden braucht und die Hostmenge, die ihn erreicht, gleich als View eingerichtet werden kann. Einerseits soll, wie schon erwähnt, nicht unbedingt jeder erreichbare Host in die Gruppe aufgenommen werden, andererseits sagt die Tatsache, erreichbar zu sein, direkt noch nichts über jene Rechner aus, welche man selbst erreichen kann.

In diesem Abschnitt soll gezeigt werden, wie man in Netzwerken mit der Topologie eines ungerichteten Graphen starke Zusammenhangskomponenten erkennen kann; außerdem gelte es daraufhin unterschiedliche Vorstellungen über die Mitgliedschaft auf einen gemeinsamen Nenner zu bringen. Abschließend will ich noch die Berechnung des minimalen Spannbaums diskutieren, welchen man unter anderem zur Garantie eines sicheren Meistabstandes aller Gruppenmitglieder benötigt.

Eine Teilmenge von Knoten eines gerichteten Graphen heißt starke Zusammenhangskomponente, wenn zwei beliebige Punkte auf einem Kreis liegen, also in beiden Richtungen miteinander verbunden sind, für schwache Zusammenhangskomponenten ist es bereits hinreichend, wenn zwischen ihnen irgendeine Verbindung existiert, und diese Verbindung darf sogar aus einander entgegen gerichteten Kanten bestehen. Alle Partitionen einer Gruppe sollen stark zusammenhängend (d.h. jeder erreicht jeden), disjunkt und von maximaler Größe sein. Ferner müssen alle Hosts einer Partition dieselbe Sicht, genauer dieselbe Auffassung darüber, wer dazugehört und wer nicht, teilen.

Neben der Menge jener Hosts γ^-, deren Signale einen erreicht haben, konstruiert der gerade vorgestellte Algorithmus noch die Menge jener Hosts γ^+, zu welchen eine Rundreise möglich ist. Jeder Knoten bringt mit jeder neu empfangenen Statusmeldung seine Erreichbarkeitsmengen γ^- und γ^+ auf neuesten Stand und sendet periodisch sein eigenes γ^- und γ^+ zusammen mit seiner Identität ab:

G(V,E) *; -* $E \subseteq V \times V$, $V=\{v_1,\ldots,v_n\}$, *Weg*(a...e) = $\{(v_1,v_2),(v_2,v_3), \ldots ,(v_{n-1},v_n) \mid v_1=a \wedge v_n=e\} \subseteq E$

$n \in a.\gamma^- \Rightarrow \exists$ Weg(n…a)	*Menge der Knoten, die a erreichen*
$n \in a.\gamma^+ \Rightarrow \exists$ Weg(a…n…a)	*Knoten für die es von a aus eine Rundreise gibt*
$a \in (\gamma^- \cap \gamma^+)$	*Jeder Knoten ist von sich selbst erreichbar*
Anfangs gelte: $a.\gamma^- := a.\gamma^+ := \{a\}$	$\forall a \in V$ *Initialisierung*
x sendet an a:	
$a.\gamma^- := a.\gamma^- \cup x.\gamma^-$	$\forall n \in x.\gamma^- (\exists$Weg(n…x),(x,a)$\in E \rightarrow \exists$Weg(n…a) $\rightarrow n \in a.\gamma^-)$
$a \in x.\gamma^- \rightarrow a.\gamma^+ := a.\gamma^+ \cup x.\gamma^+$	

$a \in x.\gamma^- \quad \Rightarrow \quad \exists \text{Weg}(a \ldots x \ldots a) \quad \wedge$
$\forall m \in x.\gamma^+ : \exists \text{Weg}(m \ldots x) \wedge \exists \text{Weg}(x \ldots m)$
$\Rightarrow \forall m \in x.\gamma^+ : \exists \text{Weg}(a \ldots x \ldots m \ldots x \ldots a) = \text{Weg}(a \ldots m \ldots a)$
$\Rightarrow \forall m \in x.\gamma^+ : m \in a.\gamma^+$

Dieser Algorithmus terminiert, wenn sich die Menge keiner der Knoten mehr ändert. In der Praxis wird er weiterlaufen, solange die einzelnen Knoten in Bewegung bleiben. Wendet man ihn auf einen Graphen an, in dem alle Kanten beidseitig sind, so kennt bei einem Durchmesser d jeder Knoten nach spätestens d Schritten sein Einzugsgebiet γ^- und nach 2d Schritten alle erreichbaren Knoten γ^+, was auch unserer Erwartung entspricht, denn um an den entferntesten Knoten und wieder zurück zu senden, braucht es nun mal 2d Hops. Ungerichtete Verbindungen stellen zwar oft nur Ausnahmesituationen dar, werden aber auch erkannt. Leider weist der eben vorgestellte Algorithmus hier nicht immer so gute Laufzeiten auf, denn bis alle Knoten eines Kreises mit Umfang U (Kreis mit U Knoten und Kanten) von der Erreichbarkeit aller anderen Knoten auf dem Kreis wissen, muß der Kreis U-1 mal durchlaufen werden, weil pro Durchlauf immer nur der unmittelbare Vorgänger eines jeden Knotens in γ^+ aufgenommen werden kann. Nachdem für jeden Durchlauf die Nachrichten U mal durchgereicht werden müssen, bis sie wieder beim Absender ankommen, sind U^2 Hops für U Durchläufe notwendig, sodaß die Komplexität $O(n^2)$ beträgt, wenn der Graph aus einem einzigen großen Kreis besteht.

Es ist zwar möglich, den Algorithmus so zu erweitern, daß alle Hosts auf einem Kreis bereits nach dem ersten Durchlauf voll informiert sind, während zu jedem in γ^- aufzunehmenden Knoten auch die Menge seiner Vorgänger vermerkt wird. Dies macht jedoch die Datenstrukturen sperrig und den Algorithmus komplizierter, was man nur in Kauf nehmen sollte, wenn dies auch angebracht erscheint.

Eine Möglichkeit, in der verschiedene Mitglieder ihre Vorstellungen über die Mitgliedschaft anderer Knoten einbringen können, ist es, die Mengenvereinigung bzw. den Mengendurchschnitt aller Vorschläge zu bilden, indem man seine aktuelle Vorstellung mit den empfangenen vereinigt bzw. schneidet und wieder aussendet, bis alle von Nachbarn empfangenen Knotenlisten Ober- bzw. Untermenge der eigenen sind. [8] schlägt vor, die Bildung der Mengendisjunktion zur Einrichtung von Partitionen in Netzwerken mit gerichteten Verbindungskanten zu verwenden. Nachteil dabei ist, daß die nicht im Durchschnitt enthaltenen Hosts dabei alleine stehenbleiben. Ein Teilnehmer könnte zudem, ob in übler Absicht oder nicht, ein Frame mit leerer Menge aussenden und würde dadurch den Gruppenbildungsprozeß unmöglich machen.

Deshalb sollte jeder Empfänger nachsehen, ob er selbst in der neuen Menge enthalten ist und, wenn nicht, statt den Durchschnitt mit der eigenen Sichtweise zu bilden die Hosts der konkurrierenden Sicht von seiner eigenen abziehen. Ein Host, der

die leere Menge aussendete, würde somit alleine stehenbleiben und die übrigen Rechner bei der Gruppenbildung nicht weiter stören.

Diese Methode kann auch statt des obigen Ansatzes der gleichzeitigen Berechnung von Einzugsgebiet und Erreichbarkeit auf Ebene der Ortsbestimmungsschicht (Proximity Layer) verwendet werden, um in einer zweiten Phase auf Ebene der Mitgliedschaftsschicht den starken Zusammenhang jeder Partition sicherzustellen, während nur Informationen über das Einzugsgebiet, also jene Hosts, die einen erreichen können, von der darunter liegenden Schicht benötigt werden.

Die beiden vorhin gezeigten Ansätze dienen u.a. zur aktiven Erkundung der Erreichbarkeit; diese könnte aber auch aufgrund technischer Daten wie der Reichweite des Senders, welche in den auf Ebene des Proximity Layer ausgetauschten Statusmeldungen mitgesendet würden, ex ante errechnet werden, ohne daß natürlich dabei der Einfluß von nicht weiter vorhersehbaren Unterbrechungen oder Störungen Berücksichtigung finden könnte. Da auch Hindernisse im Weg sein können, wäre dies sicher kein leichtes Unterfangen.

Abgesehen von derart diffizilen Unterschieden wie Erreichbarkeit und Erreichtwerden stellt zuletzt angeführtes Prinzip der Vereinigung sowie der fallweisen Durchschnitts- und Komplementbildung einen wichtigen Ansatz zur verteilten Mitgliederaufnahme dar, welcher auch mit unterschiedlichen Vorstellungen einzelner Hosts umgehen kann.

Neben Kriterien, die einzelne Hosts betreffen, sind für die Bildung von Rechnergruppen auch Eigenschaften wie Signallaufzeiten, Zusammenhalt und die Anzahl der Mitglieder von Bedeutung.

Die maximale Signallaufzeit vom einen äußersten Ende der Gruppe zum anderen wird im graphentheoretischen Netzwerkmodell durch den sog. Durchmesser wiedergegeben, welcher den längsten aller kürzesten Wege zwischen zwei beliebigen Punkten bezeichnet. Dieser kann mit einer Rekursion über den längsten Weg vom Punkt x und dem Durchmesser eines auf x gebildeten Teilgraphen gelöst werden, worauf ich aber hier nicht näher eingehen möchte. Eine weitere Möglichkeit ist es, für zyklenfreie ungerichtete Graphen den längsten Weg von einem Endpunkt aus und von dem dadurch ermittelten gegenüberliegenden Endpunkt aus noch einmal den längsten Weg zu ermitteln.

Maßgebend für den Zusammenhalt ist gewiß der minimale Spannbaum eines ungerichteten Graphen. Er verbindet die Knoten mit den kürzesten zur Verfügung stehenden Kanten. Die derzeit oder in nächster Zukunft voraussichtlich längste Kante des Spannbaumes (z.B. bei Annahme gleichförmig bewegter Hosts) laufe dabei am ehesten in Gefahr, die Verbindung zwischen den beiden verbleibenden Komponenten zu trennen, unterdessen noch längere Mehrfachverbindungen vernachlässigt bleiben dürfen. Der Spannbaum wird gewöhnlich auf ungerichteten Graphen berechnet, was

aber hier kein großes Problem darstellen sollte, denn die Partitionen einer Gruppe bilden starke Zusammenhangskomponenten.

Berechnet werden kann der minimale Spannbaum auf verteiltem Wege durch Meldungen zwischen einzelnen Hosts oder einfacher mittels Greedy- (“gierigem“) Algorithmus [JN]. Dieser ist anwendbar, weil kreisfreie Partialgraphen, welche durch das Weglassen von Kanten vom Ausgangsgraphen entstehen, ein spezielles Unabhängigkeitssystem, nämlich ein sogenanntes Matroid bilden. Unabhängig sei jeder Partialgraph, der keine Kreise enthält; Basen sind maximal unabhängig; somit bildet jede maximale kreisfreie Kantenmenge eine Basis. Der Greedy-Algorithmus ist anwendbar, weil in jedem Unterraum des Unabhängigkeitssystems, bestehend aus einem induzierten Subgraphen (entsteht durch das Weglassen von Knoten), alle Basen aus gleich vielen Kanten bestehen; d.h. jeder Spannbaum hat wie jeder andere Baum eben auch bei n Knoten n-1 Kanten. Soviel zum mathematischen Hintergrund.

Die Anwendung des Greedy-Algorithmus ist hingegen vergleichsweise einfach. In den zunächst kantenlosen Graphen werden solange Kanten aufgenommen, welche keinen Kreis mit bereits bestehenden Kanten bilden dürfen, bis die Aufnahme jeder weiteren Kante einen Kreis schlösse, also die Unabhängigkeit zerstörte. Können mehrere Kanten aufgenommen werden, wird gierigerweise stets die kürzeste genommen. Die Kantenlänge des entstehenden Baumes ist deshalb minimal, weil ein Baum mit n Knoten immer n-1 Kanten besitzt, egal in welcher Reihenfolge diese aufgenommen werden.

Den Kreisschluß beim Aufnehmen einer neuen Kante entdeckt man dadurch, daß eine solche Kante zweimal die gleiche Zusammenhangskomponente verbindet, anstatt zwei verschiedene zu vereinen. Die Kanten einer Zusammenhangskomponente könnten als einfach verkettete Liste gespeichert werden, wobei jeder Knoten zusätzlich auf das erste Element der Liste zeigt. Der Zeiger zum Listenkopf wird benötigt, um möglichst schnell festzustellen, ob zwei Kanten in derselben Komponente sind. Beim Verschmelzen zweier Zusammenhangskomponenten wird die kürzere Liste an die längere angehängt. Damit ergibt sich eine Worst-Case-Komplexität von $(\Sigma i=1..n \bullet i-1) = (n-1)(n-2)/2 = (n^2-3n-1)/2 \in O(n^2)$

1.5.2 Überwachung des Zusammenhalts

[1] Durch das regelmäßige Weiterleiten der Ortsinformationen in Form von Statusmeldungen kenne jeder Knoten die Menge jener Hosts, die er erreichen kann. Dies bedeutet aber für diesen Knoten noch nicht, daß sich all diese Hosts tatsächlich innerhalb eines sicheren Abstandes von ihm befinden, geschweige denn, daß Sender und Empfänger zu jedem Zeitpunkt die gleiche Auffassung der Gruppentopologie

haben, denn bis die Informationen eines n Hops entfernten Hosts wieder zurückkommen, braucht es 2n Zeitintervalle. Um etwa eine fehlerfreie Transaktionsverarbeitung implementieren zu können, ist es daher sinnvoll, Vereinigung und Abspaltung von Teilgruppen als atomare Ereignisse einzuführen. Eine möglicherweise Transaktionscodes beinhaltende Nachricht sollte noch im selben Gruppenzustand gesendet und empfangen werden können, indem Zustandsübergänge entweder ganz bevor oder erst nachdem alle laufenden Einzelübertragungen abgeschlossen sind, erfolgen. Eine Meldung, die in einer stark zusammenhängenden Gruppe gesendet wurde, sollte also inklusive dem zurückgelieferten ACK noch an ihrem Ziel ankommen, bevor sie in mehrere Partitionen zerfällt; schlimmstenfalls müßte die Nachricht verworfen werden, um zu verhindern, daß sie jemand in einem anderen Zustand empfängt, als sie gesendet worden ist, was man durch die Einhaltung von sicheren Abständen, welche nicht zu groß sein dürfen, zu erreichen trachtet.

Bisher haben wir nur den Fall betrachtet, daß sich Partitionen bei der Gruppenbildung in direkter Verbindung miteinander stehender Mitglieder abspalten und wiedervereinigen können; die im folgenden diskutierten Mechanismen könnten aber genauso zur Vereinigung vollkommen unabhängig voneinander entstandener Gruppen eingesetzt werden. Semantisch gesehen müßte das aber wohl ganz unterschiedliche Ereignisbehandlungen mit sich ziehen. Jedenfalls könnte statt einer expliziten Gruppenbildung zum Zeitpunkt x auch jeder Knoten als eigene Gruppe starten und durch fortgesetzten Zusammenschluß mit benachbarten Gruppen Teil einer größeren Gruppe werden.

Jede Partition oder stark zusammenhängende Gruppe wählt nach [1] einen Anführer, welcher einer nahekommenden Gruppe gegebenenfalls eine Vereinigung vorschlägt und gleichfalls die Abspaltung einer sich entfernenden Partition bekanntgibt. Als eindeutige Kennung der Gruppe kann dabei einfach die Kennung (Adresse) des Anführers dienen. Dem Anführer sind Positionen und Gruppenzugehörigkeit aller Rechner im eigenen Umkreis zu melden. Bei Verwendung des im letzten Kapitel eingeführten Algorithmus sind diese ohnehin jedem Teilnehmer bekannt.

Kommt eine neue Gruppe in Reichweite, so sendet der Gruppenleiter einen sog. „Merge-Request“ an den Leiter der gegenüberliegenden Gruppe, welcher entweder mit einem ACK oder, wenn dieser gerade beschäftigt (u.a. mit anderem Merge) ist, mit einem NAK antwortet. Daraufhin bereiten sich die Mitglieder beider Gruppen auf die Zusammenlegung vor, indem sie die Nachrichtenübertragung sowie sich gerade in Bearbeitung befindliche Transaktionen abschließen und daraufhin ein „Flush“ zum Gruppenleiter senden, welcher den Vollzug der Vereinigung, sobald von allen

Mitgliedern ein Flush eingetroffen ist, bekanntmacht (die Flushes könnten auch gebroadcastet per Broadcast verteilt werden, wenn jeder Teilnehmer selbst mitzählte).

Hosts, welche über längere Zeit nicht reagieren, können vorübergehend aus der Gruppe fallen. Statt einer asynchronen Verarbeitung kann man auch versuchen, das System mit Echtzeitanforderungen zu betreiben, was verhindern würde, daß jemand zu Unrecht ausgeschlossen werden würde; den unvorhergesehenen zeitweisen Wegfall eines Mitgliedes sollte das System trotzdem verkraften können, da auch Abstürze immer wieder vorkommen, wenngleich der entsprechende Host daraufhin möglichst bald mit dem Recovery beginnt. Sendet ein Host, der hinausgefallen ist, an einen Host der Gruppe, so erkennt dieser, daß der Absender der Nachricht nicht in seiner Mitgliederliste enthalten ist und gibt statt eines ACK einen Merge-Request zurück.

Die Abspaltung von Partitionen läuft nach demselben Muster ab. Den Empfang einer Mitgliederliste merken nämlich auch jene, die nicht mehr darin enthalten sind. Verfügten beide Anführer stets über vollständig aktuelle Daten, würden sie sich gleichzeitig und wechselseitig einen Abspaltungsbescheid erlassen. Zerfällt die Gruppe, bevor jeder die Abspaltung bestätigt hat, kann an noch auf Rückmeldungen aus der anderen Gruppe wartende Hosts eine Fehlermeldung ausgesandt werden. Grundsätzlich ließe sich das Protokoll vereinfachen, indem man auf Bestätigungen verzichtet und jeder Host selber prüft, ob sein Kommunikationspartner noch online ist, sofern jedem einzelnen Host alle Positionsangaben unterbreitet werden.

Zwischen den Hosts einer Gruppe darf der Abstand den sicheren nicht überschreiten, sonst wird eine Partitionierung eingeleitet. Dies überprüft der Leiter durch den Vergleich der größten Kante im minimalen Spannbaum mit dem sicheren Abstand. Welche Abstände als sicher erachtet werden sollen und welche nicht, hängt von der Mitgliederaufnahmestrategie (group membership policy) ab.

Am einfachsten handhabbar sind statische Ansätze, welche nur eine maximale Geschwindigkeit v_{max} für jedes Gruppenmitglied festlegen. Zwei Mitglieder können sich folglich mit max. $2 \cdot v_{max}$ voneinander entfernen, wenn sie sich beide in die entgegengesetzte Richtung bewegen. Die Zeit, welche von der Ankündigung bis zur tatsächlichen Abspaltung vergehen darf, heiße T_{trans}. Sei R die Reichweite (Sendestärke mal Empfangspräzision) und r der sichere Radius, so gilt:

$$r_1 = R - (2 \cdot v_{max} \cdot T_{trans})$$

Es ist auch möglich, die momentane Geschwindigkeit v einzubeziehen und die maximale Beschleunigung α_{max} zu beschränken, sodaß ein Überschallflugzeug, das knapp an einer stehenden Gruppe vorbeifliegt, erst gar nicht aufgenommen wird.

$$r_2 = R - T_{trans} \cdot \max\{ \; |v_i + v_j| + 2\alpha_{max}T_{trans} \; | \; i, j \in \text{Gruppe} \; \}$$

$$\begin{aligned} r_2' &= R - T_{trans} \cdot \max\{ \; |v_i| + |v_j| + 2\alpha_{max}T_{trans} \; | \; i, j \in \text{Gruppe} \; \} \\ &= R - T_{trans} \cdot (\max_1(V) + \max_2(V) + 2\alpha_{max}T_{trans}) \leq r_2 \end{aligned}$$

$$wobei\ V = \{\ |v_i| \quad |\quad i \in Gruppe\},$$

$$max_2(X) \ldots zweitgrößter\ Wert$$

r_2' ist leichter berechenbar als r_2, berücksichtigt aber nur den Betrag der Geschwindigkeit und nicht die Richtung

Schließlich könnte man beide Ansätze kombinieren, indem man sowohl die Geschwindigkeit als auch die Beschleunigung begrenzte, wobei man das Maximum aus r_1 und r_2 nimmt, sodaß aber Concordflüge erst nicht berücksichtigt wären.

Wiedervereinigung und vormalige Trennung habe ich anhand eines Gruppenanführers besprochen, wenngleich es sogar effizienter wäre, beide Vorgänge über Vertreter, welche möglichst nahe am Geschehen sind, abzuwickeln. Die Vereinigung könnten jene beiden Hosts initiieren, die einander am nächsten kommen, und bei gleich nahen jene mit der niedrigsten Kennung, obgleich von zwei der anderen Seite gleich nahe gerückten Hosts wohl jeder für sich anfangs glauben würde, daß er selbst der Nächstgelegene wäre, weil die Information über die Bewegung des jeweils anderen nicht sofort ankommt. So weit so gut; im Falle unterschiedlicher Abspaltungsbescheide zweier vermeintlicher Vermittler könnte einfach jener mit höherer Erzeugerseriennummer ignoriert werden. Insgesamt bietet die Ortskenntnis sowie die Ereignissteuerung von Vereinigung und Spaltung die Basis für partitionsorientierte oder gruppenbezogene Multicast-Services, auf welche ich jetzt näher eingehen will.

1.5.3 Multicasts

Letztendlich sollen aufbauend auf Gruppenbildung und -zugehörigkeit noch Protokolle zur Kommunikation zwischen Gruppenmitgliedern definiert werden. Dabei spielen neben der traditionellen Host-zu-Host-Kommunikation eben auch verschiedene Formen von Multicasts eine Rolle. Ich habe bereits im Abschnitt über die Architektur drei verschiedene Durchsageformen beschrieben und will nun auf die Implementierung von total geordneten partitionsunabhängigen Multicasts eingehen. Zuvor jedoch noch ein paar Worte über Datagramm-Multicasts und verläßliche geordnete Durchsagen, welche die Basis für die zuletzt beschriebenen total geordneten, partitionsunabhängigen Multicasts bilden.

Die einfachste Möglichkeit, eine Nachricht an mehrere Empfänger zu schicken, sieht das Fluten des gesamten Netzes vor, während nur jene, die Interesse haben, diese

auch tatsächlich lesen. Dabei leitet jeder Rechner alle ihm noch nicht bekannten Nachrichten, welche er empfängt, an alle seine Nachbarn weiter. Dabei werden leider einige Nachrichten mehrmals an denselben oder an einen Host weitergeleitet, der gar nicht daran interessiert ist.

Eine bessere Lösung stellt die Übertragung der Nachrichten über die Kanten eines minimalen Spannbaumes dar. Die Kantengewichtung gibt hierbei die Kosten für eine Übertragung auf der jeweiligen Verbindung wieder, sodaß die Übertragung entlang des minimalen Spannbaumes die geringsten Kosten verursachen sollte. Trotzdem erzielen in der Praxis Protokolle, welche Informationen über mehrere Wege zum Ziel schicken [6], bessere Erfolge, da sich die Weiterleitungskosten durch übermäßige Belastung einer einzigen Leitung erhöhen.

Für mobile Ad-hoc-Netzwerke ist demgegenüber die direkte Abbildung von Hosts auf Knoten und Verbindungen auf Kanten kein sehr passendes Modell, da mit dem Aussenden einer Information automatisch alle in Reichweite befindlichen Rechner mithören. Kanten müßten demgemäß einen Host mit allen umliegenden verbinden, auch wenn Kanten in einem ungerichteten Graphen per definitionem nur an zwei Knoten inzidieren dürfen. Diesen Umstand könnte man durch die Einführung künstlicher Abzweigungsknoten umgehen. Auch dann, wenn diese Überlegungen keine Berücksichtigung finden, spart der Einsatz minimaler Spannbäume immerhin den Endknoten unnotwendige Weiterleitungsversuche. Nun kann man versuchen, für jeden Knoten die Menge jener Hosts, welche dieser erreichen kann, aufzuschreiben und das so gestellte Mengenüberdeckungsproblem zu lösen, doch ist dieses NP-vollständig ($O(2^n)$) und nicht ohne weiteres effizient lösbar oder approximierbar.

Wir wollen uns jetzt aber lieber nicht an die optimale Lösung von Datragramm-Multicasts in mobilen Netzwerken wagen, sondern besser Ausschau halten nach der Herstellung eines verläßlichen geordneten Multicasts. Übermittelt ein Host seine Nachricht an ihre Zielgruppe, so muß jeder Host aus der Zielgruppe ein ACK zurückgeben. Es würde wohl zu einem heillosen Chaos führen, wenn dies anhand von lauter Einzelmeldungen zurück an den Absender geschehen würde, denn dann müßte der Absender für eine Durchsage an 1000 Empfänger 1000 ACKs entgegennehmen können. Auch die Ersetzung von ACKs durch NAKs kann die Worst-Case-Komplexität nicht verbessern. Deshalb ist es sinnvoll, jeden Knoten nur für die Verteilung an seine direkten Nachbarn verantwortlich zu machen. Erhält ein Rechner eine Multicastnachricht, so speichert er diese, während er sie an alle Nachbarn im Spannbaum weiterzugeben versucht. Er darf den Speicherplatz erst wieder freigeben, wenn er von all seinen Nachbarn ein ACK erhalten hat. Durch die strikte Übermittlung aller Nachrichten entlang eines Spannbaumes ginge die Ordnung, in der die Pakete abgesandt worden sind, eigentlich nicht verloren, es sei denn, wir befinden uns in einem mobilen Netzwerk, bei dem das Absenden einer Nachricht nicht nur dem eigentlichen Adressaten, sondern stets allen benachbarten Hosts zu Gehör kommt. In

diesem Fall kann die Verwendung von absenderspezifischen Ordnungszahlen, wie dies bei TCP Anwendung findet, von Nutzen sein.

Jetzt können wir uns endlich an total geordnete partitionsunabhängige Multicasts wagen. Diese ermöglichen es einer Gruppe von Servern, kongruente Datenbestände beizubehalten. Dies ist vor allem in hochmobilen Ad-hoc-Netzwerken, in denen Verbindungen zwischen einzelnen Hostgruppen zeitweise abreißen können, eine wirkliche Herausforderung. Wenn wir uns beispielsweise die Leistungsdaten eines heute erhältlichen Notebooks mit 120 GB Festplatte und einer WLAN-Karte für 10 Mbit/s ansehen, so müssen wir feststellen, daß zur Übertragung des gesamten Festplatteninhaltes in unkomprimierter Form fast 1,2 Tage notwendig sind ($120*1024^3*8\text{bit/Byte} \div 10^7\text{bit/s} \div 3600\text{h/s} \div 24\text{h/d}$). Angesichts dieses Umstandes wird jeder Rechner versucht sein, so viele Daten wie möglich selbst zu speichern. Bildet eine Menge von gleichwertigen Computern ein Netzwerk, so ist es folglich nicht unbedingt sinnvoll, nur ganz bestimmte Computer als Server auszuzeichnen und alle übrigen zu Proxies zu degradieren. Hier ist ein client/server-loser Ansatz, nämlich jener der Gruppenkommunikation, gefragt!

Kommen wir zum total geordneten partitionsunabhängigen Multicast zurück. Total geordnet bedeutet nicht nur, daß die Pakete, welche von einem einzelnen Host abgesendet worden sind, untereinander geordnet sind, sondern auch, daß eine totale Ordnung auf allen Nachrichten, welche von den unterschiedlichsten Hosts erzeugt worden sind, existiert. Sollen eine Abbuchung von 100€ und eine Einlage von 200€ auf ein Konto von 50€ erfolgen, so scheitert die Abbuchung, wenn sie vor der Einlage vorgenommen werden soll, solange es keinen Überziehungsrahmen gibt, wovon wir hier einmal ausgehen wollen. Will ein Netzteilnehmer 100€ abbuchen, so schickt er allen Gruppenmitgliedern, welche Kontodaten verwalten, eine Nachricht. Nun soll der Umstand, daß die Abbuchung glückt, nicht davon abhängen, welche der beiden Nachrichten (Abbuchung oder Einlage) früher bei den einzelnen Kontoverwaltungsrechnern ankommt, sondern von einer totalen Ordnung auf beiden Nachrichten. Diese totale Ordnung kann beispielsweise aus einem Zeitstempel und der eindeutigen Kennung des Absenders bestehen, sodaß eine Nachricht vor der anderen kommt, wenn sie früher oder von einem Host mit kleinerer Ordnungszahl erzeugt worden ist. Gäbe es keine solche totale Ordnung, so würden nämlich einige Kontoverwaltungsmaschinen die Abbuchung ausführen, andere jedoch einen Fehler zurückgeben, was die Inkonsistenz der Datenbestände als Folge hätte.

Kommt die Abbuchung also früher an und hat sie eine größere Ordnungszahl, so muß ihre Verarbeitung mindestens bis zum Eintreffen der Einzahlung hintangehalten werden. Die maximale Verzögerung, genannt Jitter, in einer Partition ist aus dem Netzdurchmesser und den Queuing-Delays bei der Weiterleitung gegeben. Wartete jeder Host die maximale Verzögerung vor dem Versuch eines Commit ab, so verfügte

er zu diesem Zeitpunkt auch schon über die richtige Reihenfolge der zu konsolidierenden Transaktionen, solange keine Pakete auf dem Weg verlorengegangen sind. Für die Durchführung des Commit warten alle Hosts zusammen, bis jeder seine Flush-Message abgegeben hat. Es empfiehlt sich hierbei, die Ordnungszahlen der zu bestätigenden Transaktionen mitzusenden, weil nur der Durchschnitt aller angeführten Transaktionen bestätigt werden darf. Nachrichten mit kleinerer Ordnungszahl als der zuletzt einem Commit unterzogenen müssen verworfen werden.

Eine Alternative hierzu ist es, sich für jede Meldung vor dem Abschicken eine eindeutige Sequenznummer vom Gruppenleiter zu holen, während Nachrichten bereits verarbeitet werden dürfen, sobald alle Nachrichten mit vorhergehenden Sequenznummern eingetroffen sind. Commits sind deshalb leider trotzdem nicht ganz überflüssig, weil Datenbankserver auch abstürzen können und beim Wiederanlauf sichergestellt sein soll, daß die bestätigten Transaktionsnachrichten erhalten bleiben.

Ist die Einführung einer totalen Ordnung geglückt, kann man sich an die Behandlung von Partitionierungen machen. Sind nicht alle Empfänger in der aktuellen Partition zu finden, so speichern alle an der Weiterleitung Beteiligten die Nachricht bis zur nächsten Wiedervereinigung, ohne natürlich auf das ACK der abwesenden Hosts zu warten. Ist es dann so weit, wird der Multicast anhand des neu aktualisierten Spannbaumes wiederaufgenommen, indem der der aufzunehmenden Partition nächstgelegene Host die Übertragung startet. Solange nicht alle Adressaten ihr ACK abgeben konnten, sollte also noch nicht an die Freigabe des Paketes gedacht werden, da statt einem Zusammenschluß auch eine weitere Aufteilung folgen kann und in diesem Fall gleich beide Untergruppen über die Nachricht verfügen.

Die Frage, die sich jetzt noch stellt, ist, wie man eine totale Ordnung in einem solchen partitionierbaren System etablieren kann. Gibt es einen Koordinator, der Ordnungszahlen austeilt, so müßten Nutzer, welche sich nicht in derselben Partition wie der Koordinator befinden, bis zur nächsten Wiedervereinigung mit dieser warten und dürften erst dann jene Nachrichten sequenzieren und versenden, die sie schon immer aufgeben wollten. Bevor eine Aufspaltung eintritt, kann der Koordinator noch schnell in eine andere Partition wechseln, um auch in Zukunft eine größtmögliche Anzahl an Hosts versorgen zu können; eine laufende Verlagerung in die Mitte der Gruppe, also dorthin, wo mit ausreichend kurzen Spannbaumkanten möglichst viele Hosts erreicht werden können, wäre natürlich auch denkbar.

Kommen Partitionierungen häufig vor, ist auch diese Lösung recht unbefriedigend. Es wäre wünschenswert, jeder Untergruppe ihre eigene konsistente Sichtweise zu geben, bis sie auf die Hauptgruppe mit ihrem Koordinator trifft. Dazu können in jeder Teilgruppe Ordnungszahlen ausgegeben werden, welche für alle Mitglieder der Teilgruppe eine totale Ordnung bilden. Schließen sich zwei Gruppen zusammen, müssen sie die beiden (für sich genommen totalen) Partialordnungen in einer einzigen Reihe total anordnen. Diese Zusammenführung kann durch einfaches

Hintereinanderstellen, durch Verschmelzung nach dem Reißverschlußprinzip oder dem Zeitstempelverfahren geschehen, welches früher initiierte Nachrichten weiter vorne einreiht.

1. Hintereinanderstellen: [x;y] & [a;b] -> [x;y;a;b]
2. Reißverschlußprinzip [x;y] & [a;b] -> [x;a;y;b]
3. Zeitstempelverfahren [1:a; 2:w] & [1:x; 2:c; 3:a] -> [1:a; 1:x; 2:c; 2:w; 3:a]

Zeitstempelverfahren und Reißverschlußprinzip können die Fairness bei der Vereinigung zweier Nebenpartitionen garantieren, während bei der Vereinigung mit der Hauptpartition sich die Nachrichten von Nebenpartitionen hintanstellen müssen. Transaktionen aus der Hauptpartition sollen nämlich weiterhin laufend konsolidiert werden können, während solche aus Nebengruppen mit ihrem Commit bis zur Vereinigung mit der Hauptgruppe warten müssen. Solange sie nicht mit einem Commit bestätigt worden ist, steht auch das endgültige Ergebnis einer Transaktion nicht fest.

Auch [5] beschäftigt sich mit den dort sog. „Totally Ordered Broadcasts in the Face of Network Partitions“: Hier erhält jede Nachricht einen der Zustände rot, gelb oder grün, je nachdem ob sie nur in einer Nebenpartition bekannt ist, in der Hauptpartition bekannt ist, aber ihre Anordnung noch nicht endgültig feststeht, oder ob sie fertig für das Commit ist.

Somit wären alle zuvor geforderten Eigenschaften des total geordneten partitionsunabhängigen Multicasts erfüllt. Partitionsunabhängig soll hier bedeuten, daß eine Nachricht, sobald dies möglich ist, alle Gruppenmitglieder erreicht, unabhängig davon, in welcher Partition sie aufgegeben wurde. Es bietet sich ferner die Möglichkeit, ein solches Service basierend auf partitionsbezogenen relativen Sequenznummern (1,2,3,…) oder auf quasi absoluten Größen wie der Kombination aus Zeitstempel und Adresse einzurichten, wobei stets beachtet werden muß, daß die Nachrichten der Hauptpartition vor allen anderen einzureihen sind.

1.6 Zusammenfassung

Für die Kommunikation in Gruppen gibt es sicherlich vielfältigste Anwendungen, ausgehend von der Vernetzung einzelner Automobile für Verkehrsleitsysteme oder zur Erkennung von Stausituationen bis hin zur verteilten Speicherung von Datenbeständen. Das Paradigma der Gruppenkommunikation eignet sich hervorragend für den Einsatz in mobilen Ad-hoc-Netzwerken und bietet einen ersten Ansatz, um sich selbst organisierende autonome Systeme verwirklichen zu können, so wie diese in der Natur überall zu finden sind.

Ich habe in dieser Arbeit versucht schrittweise darzustellen, wie ein solches gruppenbasiertes System arbeiten kann und bin - von der Erfassung der Ortsinformationen über die Mitgliedschaft in Gruppen bis zur tatsächlichen Unterstützung der Kommunikation aller Mitglieder untereinander - die einzelnen Ebenen anhand des zuvor beschriebenen geschichteten Architekturmodells schrittweise durchgegangen. Wenn ich hier auch nicht alles erschöpfend behandeln konnte, so hoffe ich doch einen guten Überblick über die sich bietenden Möglichkeiten gegeben zu haben.

Um ein mobiles Ad-hoc-Netzwerk tatsächlich implementieren zu können, sind natürlich weitere Wissensgebiete in die Betrachtung miteinzubeziehen: das Routing einzelner Nachrichten, welches die Voraussetzung für viele gruppenbasierte Services bildet, sowie eine Reihe weiterer Kommunikationsparadigmen wie das Publish/Subscribe-Modell, Event-Based Communication oder Gossip- und Epidemic Dissemination.

Die Gruppenkommunikation bietet jedenfalls neben ihrem Beitrag zur Erschließung völlig neuer Gebiete wie der Kommunikation in mobilen Ad-hoc-Netzwerken auch neuartige Lösungsansätze für bereits bestehende, derzeit aber andersartig gelöste Probleme wie der Kommunikation zwischen Client und Server an. Insgesamt betrachtet handelt es hier um eine aktive Forschungsdisziplin, welche zwar derzeit eher selten angewandt wird, in Zukunft aber ein hoch innovatives Potential bereitstellen könnte.

2 Publish/Subscribe

2.1 Einführung

2.1.1 Publish/Subscribe: ein Kommunikationsparadigma

In diesem Abschnitt soll eine Einordnung des Publish/Subscribe-Paradigmas innerhalb unterschiedlicher Kommunikationsparadigmen getroffen werden. Hierbei wird auf charakteristische Merkmale hinsichtlich zeitlicher und referentieller Kopplung der einzelnen Paradigmen eingegangen.

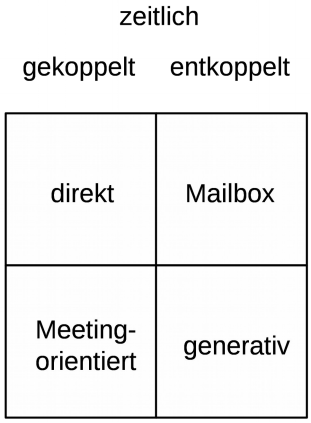


Abb. 1: Überblick über Kommunikationsparadigmen hinsichtlich zeitlicher und referentieller Kopplung (nach [TaSt02])

Abb. 1 soll einen Überblick über die einzelnen Kommunikationsparadigmen geben. Dabei spricht man von **direkter Kommunikation**, wenn der Sender den Empfänger direkt adressiert (referentielle Kopplung) und beide zur gleichen Zeit aktiv sein müssen (zeitliche Kopplung).
Man spricht von **Mailbox-Kommunikation**, wenn der Sender den Empfänger kennen muss (referentielle Kopplung), jedoch nicht beide zur gleichen Zeit aktiv sein müssen

(zeitlich entkoppelt). Der Sender kann seine Nachricht für den Empfänger in der „Mailbox“ hinterlassen.

Meeting-orientierte Kommunikation bedeutet dass sich die Kommunikationspartner nicht explizit kennen (referentielle Entkopplung), jedoch zur selben Zeit aktiv sind (zeitlich gekoppelt). Die Kommunikation erfolgt dadurch, dass sich die einzelnen Partner in Form eines Meetings begegnen (die Kommunikation erfolgt dann beispielsweise über eine zentrale Stelle).

Generative Kommunikation (beispielsweise vom System Linda implementiert) heißt, dass sich die Kommunikationspartner weder gegenseitig kennen (referentiell entkoppelt) noch dass sie zeitlich gekoppelt sind. Kommunikationspartner kommunizieren über einen gemeinsamen persistenten Speicher. Die von einem Prozess in diesen persistenten Speicher geschriebenen Daten können zu einem späteren Zeitpunkt von beliebigen anderen Prozessen gelesen werden [TaSt02].

Im Folgenden wird referentielle Entkopplung synonym mit räumlicher Entkopplung verwendet.

Bei gewöhnlicher generativer Kommunikation über einen persistenten gemeinsamen Speicher ist es erforderlich, dass der Empfänger einer Nachricht im synchronen Modus arbeitet. Das bedeutet, dass der Empfänger (im main-thread) wartet, bis die Nachricht im gemeinsamen Speicher steht. Erst nach dem Lesen der Nachricht nimmt der Empfänger seine Arbeit wieder auf [EFGK03].

Das Paradigma **Publish/Subscribe** ist nun eine spezielle Art der generativen Kommunikation. Publish/Subscribe zeichnet sich dadurch aus, dass es neben zeitlicher und referentieller Entkopplung auch noch Datenfluss-Entkopplung aufweist. Das heißt, dass nun auch der Empfänger im asynchronen Modus arbeitet. Publish/Subscribe bietet also größtmögliche Entkopplung, was Abhängigkeiten zwischen den einzelnen Kommunikationsparteien verringert und somit die Skalierbarkeit des Systems verbessert. Publish/Subscribe ermöglicht Subscriber-Prozessen, sich für eine bestimmte Art von Informationen bzw. ein Event i.A. zu registrieren und beim Eintreten dieses Events (auf der Seite der Publisher-Prozesse) informiert zu werden. Ist der Subscriber zum Zeitpunkt der Benachrichtigung vom Netz getrennt, so werden die Events beim Event Service zwischengespeichert und zu einem späteren Zeitpunkt ausgeliefert [EFGK03].

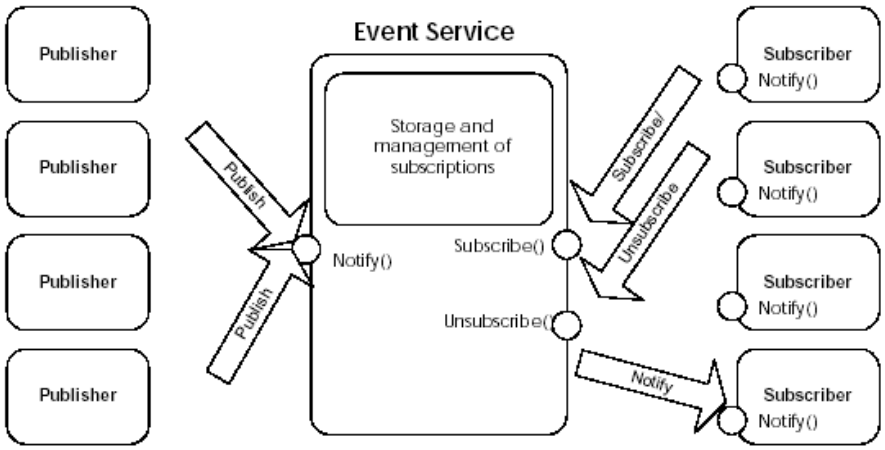


Abb. 2: Schematische Darstellung eines Publish/Subscribe-Systems (Quelle: [EFGK03])

Die obenstehende Abbildung verdeutlicht die Funktionsweise der Basisvariante von Publish/Subscribe. Eine Menge von Subscriber-Prozessen registriert sich bei einem Event Service für bestimmte Events, indem sie die Methode S*ubscribe* des Event Service aufrufen. Sie können ihre Interessen an diesem Event mit der Methode *Unsubscribe* wieder stornieren. Dabei wird die Information der Subscriber nur im Event Service gespeichert und nicht zu den Publisher-Prozessen weitergegeben. Die Publisher-Prozesse veröffentlichen Informationen, indem sie die Methode *Notify* des Event Service aufrufen. Sie informieren also nicht direkt die Subscriber. Schlussendlich informiert das Event Service die Subscriber über die eingegangene Information, indem sie die *Notify* Funktion der Subscriber aufrufen.

2.1.2 QoS bei Publish/Subscribe

2.1.2.1 Persistenz

Persistenz ist ausschlaggebend für Publish/Subscribe-Systeme, da Nachrichten oft über mehrere Stunden lang gespeichert werden müssen, bevor sie an die jeweiligen Subscriber weitergegeben werden. Diese lange Zeitspanne ist vor allem in Ad-hoc-Netzen relevant, da hier Subscriber oft über längere Zeit hinweg offline sind, und sich erst danach wieder in das Netz einloggen. Somit muss gewährleistet sein, dass – unabhängig von der Architektur des Publish/Subscribe-Systems – Nachrichten über eine bestimmte Zeit hinweg gespeichert bleiben.

2.1.2.2 Prioritäten

Ein weiterer Dienstgüte-Faktor für Publish/Subscribe-Systeme ist die Möglichkeit der Prioritätszuteilung zu Events. Damit ist es beispielsweise möglich, Events unterschiedlicher Wichtigkeiten unterschiedlich schnell an Subscriber zu senden.

2.1.2.3 Transaktionen

Transaktionen werden in Publish/Subscribe-Systemen dazu verwendet, um Nachrichten bzw. Events zu Blöcken zu gruppieren. Dabei besteht der Sinn darin, dass der Subscriber entweder einen ganzen Block (also eine ganze Folge von Nachrichten), oder gar keine Nachricht erhält. Das ist beispielsweise dann sinnvoll, wenn der Subscriber nur eine gesamte Folge von Nachrichten, die sich auf ein bestimmtes Thema beziehen und nur als geschlossene Folge sinnvoll sind, erhalten soll [EFGK03].
Handelt es sich um ein Publish/Subscribe-System, das Replikation[1] von Events verwendet, so sind weiters noch folgende QoS-Merkmale interessant:

2.1.2.4 Ordnung

Es ist wichtig, dass die Events in der richtigen Reihenfolge beim Subscriber eintreffen. Richtige Reihenfolge heißt, dass die Events eines Publishers beim Subscriber in der gleichen Ordnung eintreffen, wie sie beim Publisher generiert wurden [HuGa01].

2.1.2.5 Konsistenz

Es soll ein doppelter Empfang ein und desselben Events ausgeschlossen sein [HuGa01].

2.1.2.6 Vollständigkeit

Vollständigkeit bedeutet, dass der Subscriber alle Events erhalten muss, für die er sich registriert hat [HuGa01].

2.1.3 Arten von Publish/Subscribe

Abhängig von der Information bzw. den Events unterteilt man Publish/Subscribe in drei Arten, die anschließend kurz erläutert werden [EFGK03].

2.1.3.1 Themenbasiert (topic-based)

Hier können sich Subscriber für bestimmte Themen registrieren. Somit erhalten die Subscriber dann die Informationen bezüglich dieses Themas. Anstatt zu Themen ist

---

1 Genaueres zum Thema Publish/Subscribe mit Replikation später.

auch eine Registrierung zu Gruppen möglich. Dann bekommen alle Subscriber innerhalb einer Gruppe dieselben Informationen [EFGK03].

2.1.3.2 Inhaltsbasiert (content-based, property-based)

Inhaltsbasiertes Publish/Subscribe ermöglicht eine Registrierung zu Informationen/Events mit bestimmten Inhalten. Dies ermöglicht eine feinere Unterteilung der unterschiedlichen Informationen in interessante bzw. uninteressante Informationen für den Subscriber [EFGK03].

2.1.3.3 Typbasiert (type-based)

Hier besteht die Möglichkeit, dass sich Subscriber nur für Events eines bestimmten Typs registrieren [EFGK03].

Neben dieser Unterteilung kann auch eine Unterteilung vorgenommen werden, die sich auf die Architektur des Publish/Subscribe-Systems bezieht. Eine Unterteilung der Publish/Subscribe-Systeme nach unterschiedlichen Architekturen wird in folgendem Abschnitt diskutiert.

2.2 P/S-Architekturen für Fest- und Mobilnetze

In diesem Abschnitt sollen unterschiedliche Architekturen von Publish/Subscribe-Systemen beschrieben werden. Ausgehend von Grundarchitekturen, die in festverdrahteten Netzwerken auftreten werden anschließend Weiterentwicklungen dieser Architekturen für den Einsatz in mobilen Netzen und Ad-hoc-Netzen besprochen. Im Folgenden werden die Publisher als Event Sources (ES), die Subscriber als Event Displayers (ED) und das Event Service als Event Broker (EB) bezeichnet [HuGa01].

2.2.1 Zentralisiertes Publish/Subscribe

Bei der zentralisierten Variante des Publish/Subscribe gehen alle Events über einen einzigen Event Broker. Das heißt, dass sich alle EDs beim gleichen EB registrieren und dass alle ESs ihre Events über den gleichen EB verteilen. Dies ist zwar einfach in der Implementierung, jedoch bildet der EB einen Single Point of Failure. Beim Ausfall des EB ist das gesamte System nicht mehr funktionsfähig.
In einem festverdrahteten Netz befinden sich sowohl die ESs und EDs, als auch der EB an fixen, statischen Positionen innerhalb des Netzes. Bei Publish/Subscribe-Systemen in mobilen Netzen soll nach Möglichkeit versucht werden, den EB in ein festverdrahtetes Netz auszugliedern. Wenigstens jedoch soll sich der EB auf einem

eigenen Rechner befinden, und nicht am selben System wie ein ES oder ED. Der Grund dafür liegt darin, dass der EB bestimmte Operationen auszuführen hat, die mitunter ressourcenaufwändig sein können. So kann die Ermittlung der Subscriber, die sich für ein bestimmtes Event interessieren die Batterie eines mobilen EBs zu stark beanspruchen. Außerdem könnte es Probleme bereiten, wenn der EB ein mobiles Gerät ist, und sich zwischenzeitlich außerhalb der Reichweite der ESs und EDs befindet.
Da es in Ad-hoc-Netzwerken jedoch nur mobile und keine festverdrahteten Komponenten gibt, ergeben sich hier besondere Anforderungen für das Publish/Subscribe-System.
So kann es beispielsweise vorkommen, dass sich der EB plötzlich außer Reichweite der anderen Kommunikationsparteien bewegt. ESs müssen vor dem Publizieren ihrer Events überprüfen, ob ein EB erreichbar ist. Ist keiner erreichbar, so muss von den ESs ein neuer EB gewählt werden. Dies erfolgt durch Election-Algorithmen. Ebenso müssen auch die EDs in regelmäßigen Zeitabständen überprüfen, ob der EB noch existiert. Finden sie ihn nicht mehr, müssen sie in ihrer Umgebung nach einem neuen EB suchen und ihre Interessen registrieren (Subscription).
Tritt der Fall ein, dass ein zuvor offline gegangener (oder außer Reichweite bewegter) EB wieder zurück ins Netz kommt, nachdem ein neuer EB gewählt wurde, wird durch ein sogenanntes Merging-Protokoll ein neuer zentraler EB daraus geschaffen. Alternativ dazu könnten auch beide als EB zur Verfügung stehen, dabei würde man dann von einem verteilten Publish/Subscribe sprechen. Näheres hierzu aber in Kapitel unten f. [HuGa01].

2.2.2 Zentralisiertes Publish/Subscribe mit Quenching

Eine Verbesserung des gewöhnlichen zentralisierten Publish/Subscribe bietet das Quenching-Verfahren. Quenching (engl. für unterdrücken) ermöglicht eine Verringerung der Anzahl der zu übertragenden Events.
Ein zentrales Element dieses Verfahrens sind die sogenannten „combined active subscription expressions“. Dies ist ein logischer OR-Ausdruck folgender Form.

$$c_{all}(e) = c_1 \vee \ldots \vee c_n$$

Dabei steht c_i für das Interesse des ED_i an einem bestimmten Event e. Ist ED_i am Event e interessiert, so ist c_i=true, andernfalls ist c_i=false. Diese „combined active subscription expressions“ werden bei den ESs gespeichert. Sie geben ihnen keine Auskunft darüber, welcher Subscriber an einem bestimmten Event interessiert ist, sondern nur darüber, ob mindestens ein Subscriber (anonym) an einem Event interessiert ist. Demnach kann der ES schon im Vorhinein entscheiden, ob er sein

Event publiziert oder nicht. c_{all} ergibt true, sobald ein c_i true ist. D.h. dass ES sein Event an EB publiziert, wenn mindestens ein ED Interesse daran hat.
Für mobile Systeme und insbesondere Ad-hoc-Netzwerke bringt das den Vorteil, dass einerseits Bandbreite gespart wird, die bei Ad-hoc-Netzen eine besondere Einschränkung darstellt. Andererseits kann die Berechnung der „combined active subscription expression“ bei jedem auftretenden Event zu ressourcenintensiv für ein batteriebetriebenes mobiles Gerät sein. Somit muss man abwägen, was im Einzelfall wichtiger ist: Bandbreite oder Systemressourcen (wie Batterieladung) [HuGa01].

2.2.3 Verteiltes Publish/Subscribe mit Broadcast

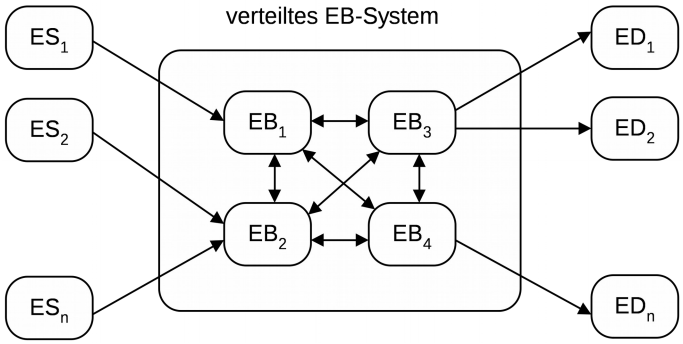


Abb. 3: Schematische Darstellung des verteilten Publish/Subscribe-Broadcast-Systems

Das größte Problem bei zentralisierten Architekturen besteht im Single Point of Failure. Dieses besteht bei verteiltem Publish/Subscribe nicht, da hier eine Menge von EBs existiert. Fällt bei dieser Architektur ein EB aus, zieht das keinen totalen Stillstand des Publish/Subscribe-Systems nach sich. Ein weiterer Vorteil dieser Architektur besteht darin, dass nun nicht mehr ein einzelner EB den Vergleich der Events mit den Subscriptions der Subscribers vornehmen muss, sondern dass diese Aufgabe nun über mehrere EBs verteilt ist.
Jeder dieser EBs ist für eine gewisse Anzahl von Subscribers zuständig. Jeder der EBs erreicht eine Menge anderer, in seiner Reichweite liegender, EBs (in mobilen Netzen). Die gegenseitige Erreichbarkeit kann anhand eines Graphen (Erreichbarkeitsgraph) veranschaulicht werden.
Der Publish/Subscribe-Vorgang geht folgendermaßen vor sich: Die einzelnen EDs registrieren sich bei einem der EBs. Jeder ES veröffentlicht seine Events bei einem (meist dem nähesten) EB. Daraufhin verschickt der EB das erhaltene Event an alle anderen (erreichbaren) EBs. Daher kommt die Bezeichnung Broadcast. Weiters

vergleicht der EB jedes erhaltene Event mit den bei ihm gespeicherten Abonnements (Subscriptions) der EDs und leitet gegebenenfalls ein Event an die interessierten EDs weiter [HuGa01].

2.2.4 Verteiltes Publish/Subscribe mit Multicast

Publish/Subscribe mit Broadcast zieht nach sich, dass die Verbindungen im Netzwerk zu stark ausgelastet werden. Das ist vor allem aufgrund der beschränkten Bandbreite in mobilen (bzw. ad hoc-) Netzen ein besonderes Problem. Daher entwickelte man Publish/Subscribe-Verfahren auf Multicast-Basis. Hier verteilen die EBs die Events nicht an alle erreichbaren EBs, sondern nur an ausgewählte. Die Auswahl hängt davon ab, ob das jeweilige Event für einen bestimmten EB von Interesse ist.
So leiten EBs ein Event e nur an solche EBs weiter, die über mindestens eine Subscription eines Subscribers für genau dieses Event e verfügen. Das bedeutet, dass nun jeder EB auch über die Subscription-Informationen der von ihm erreichbaren EBs Bescheid wissen muss. Kann ein EB alle anderen EBs erreichen, so muss er die gesamten Subscription Informationen speichern, was eine erhebliche Ressourcenbeanspruchung bedeuten kann. Außerdem kann der Vergleich des Events mit den gespeicherten Subscription Informationen (also die Ermittlung, wer sich für dieses Event interessiert) ebenfalls aufwändig sein.

Bei verteilten Publish/Subscribe-Systemen im mobilen Bereich und besonders in ad hoc Netzen, wo keinerlei Komponenten in einem festverdrahteten Netz liegen, ergeben sich besondere Eigenheiten. Da sich die EDs meist in Bewegung befinden, kommt es vor, dass sie sich außerhalb der Reichweite von EBs bewegen oder aufgrund ihrer begrenzten Batterieladung gezwungenermaßen oft offline gehen. Nachdem sie sich wieder in das Netz einloggen, können sie sich an einer anderen Position befinden und sich demnach bei einem anderen EB registrieren. Der neue EB muss nun nicht nur die Interessen des ED in Form der Subscription-Informationen erfahren, es muss auch gewährleistet werden, dass der zuvor offline gewesene ED jene Events nachgeliefert bekommt, die er während seiner „Abwesenheitszeit“ versäumt hat. Diese Events werden im Sinne des Publish/Subscribe zwischengespeichert. Die zwischengespeicherten Nachrichten werden sodann vom alten EB auf den neuen übertragen. Dies erfolgt in Form eines „Handoff-Protokolls“. Anschließend ist es möglich, dass der neue EB den Subscriber ED mit den zuvor versäumten Events versorgt.
Ebenfalls ist der Fall vorstellbar, dass durch Verbindungstrennung vom Netz und neuerliche Verbindungswiederaufnahme ein ED bei zwei oder mehreren EBs registriert ist. Dabei muss darauf geachtet werden, dass der ED beim alten EB abgemeldet wird und die Events nur von einem EB erhält. Andernfalls können durch doppelt erhaltene Events Inkonsistenzen auftreten. Um diese Probleme zu verhindern

und ein Abonnement eines Events bei einem einzigen EB sicherzustellen, gibt es besondere Mechanismen[2] [HuGa01].

2.2.5 Publish/Subscribe mit Replikation

Während es in den bisher besprochenen zentralen und verteilten Publish/Subscribe-Varianten weder erwünscht ist, dass ein ED zur gleichen Zeit bei mehreren EBs für ein Event registriert ist, noch dass ein ES ein Event an mehrere EBs veröffentlicht, ist das der Kernpunkt von Publish/Subscribe mit Replikation.

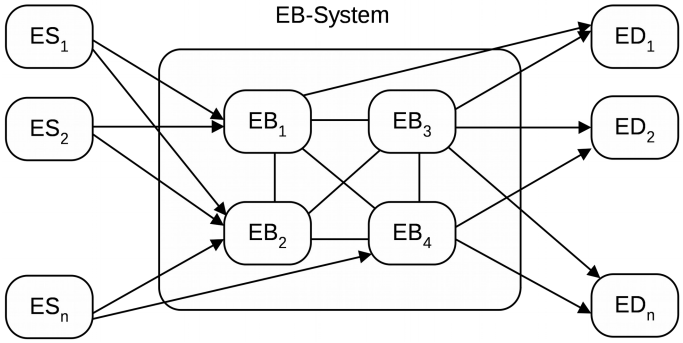


Abb. 4: Schematische Darstellung eines auf Replikation basierenden Publish/Subscribe-Systems

Replikationsbasierte Verfahren gehen besonders auf die Schwächen von mobilen Netzen ein. Hier registriert jeder ED seine Interessen bei mehreren EBs. Weiters publiziert jeder ES seine Events bei mehreren EBs. Dies hat den Vorteil, dass ein ES seine Events auch noch veröffentlichen kann, wenn ein EB außer Reichweite gerät. Ebenso besteht für EDs der Vorteil, dass sie ihre abonnierten Events auch noch erhalten, wenn ein EB außer Reichweite gerät, ausfällt, o.ä.
Jedoch treten mit Replikation gleichzeitig auch Konsistenzprobleme auf: Es muss verhindert werden, dass ein ED ein und dasselbe Event von mehreren EBs erhält [HuGa01].

2.3 Multicast-Bäume in MANETs

In folgendem Abschnitt werden Verfahren zur Generierung von Multicast-Bäumen, die fürs Multicast-Publish/Subscribe in Mobilen ad hoc Netzwerken (mobile ad hoc networks, MANETs) benötigt werden, beschrieben.

---

2 Näheres zu diesen Mechanismen in [HuGa01b]

2.3.1 Einleitung und Kostenmetriken

Wohingegen bei zentralisierten Publish/Subscribe-Verfahren die zentrale Instanz die Events an alle Subscriber ausliefert, sind bei verteilten Architekturen besondere Überlegungen hinsichtlich der Auslieferung bzw. Weiterleitung der Events nötig. So sind Algorithmen nötig, die ermöglichen, dass Events vom Publisher über einen möglichst „optimalen“ Weg an die Subscriber gelangen. Optimal bedeutet hierbei entweder möglichst kurz (minimale Anzahl der Hops bzw. minimale Anzahl gesamt beteiligter Instanzen) oder möglichst schnell (Route mit maximaler verfügbarer Bandbreite). Da ein Event nicht nur an einen, sondern an mehrere Subscriber ausgeliefert werden kann, ist die Route ein Baum (Multicast-Baum, Publish/Subscribe-Tree - PST). Ziel der Baumgenerierungs-Algorithmen ist es, einen möglichst optimalen Multicast-Baum zu berechnen.
Zur Berechnung des optimalen PST gibt es unterschiedliche Ansätze. So erfolgt die Berechnung beim OPT Algorithmus (siehe Kapitel 2.3.2.1) vollständig in einer zentralen Instanz, bei ShopParent (siehe Kapitel 2.3.2.2) hingegen vollständig verteilt.
Im Folgenden wird der Einfachheit halber angenommen, dass ein Event nur von EINEM Publisher (genannt Root) publiziert wird. Dies ist aber einfach erweiterbar. Man kann mehrere Publisher behandeln, indem für jeden Publisher ein PST berechnet wird. Ein anderer Ansatz besteht darin, dass die weiteren Publishers ihre Events zuerst zum Root-Publisher schicken [HuGa03].

Die im Folgenden beschriebenen Algorithmen verwenden als Gütekriterium für einen Baum die Effizienz des Verbreitens eines Events zu den einzelnen Subscribers. Dabei ist die Effizienz umso höher, je weniger Events umsonst weitergeleitet werden. Die Effizienz wird dadurch gesteigert, dass ein Knoten ein Event nur dann an seine in Reichweite befindlichen Knoten (Kinder) sendet, wenn sie daran interessiert sind. Daher muss ein Knoten nicht nur seine eigenen Subscriptions, sondern auch die der Kinder speichern, wie bereits erwähnt. Genauer gesagt speichert ein Knoten drei Subscriptions: S_i, s_i, s'_i [HuGa03]. Dies geht aus Tab. 1 hervor.

s_i	Inhärente Subscription	eigene Subscriptions des Knotens i
s'_i	Proxied Subscription („bevollmächtig“ für Kinder)	Subscriptions der Kinder von i
S_i	Effektive Subscription	Subscriptions von i und (effektive) Subscriptions der Kinder von i

Tab. 1: Arten von Subsciptions

Je nachdem, welchen Subscriptions ein Event in einem Knoten entspricht, werden sie entweder nur empfangen (received, r), verarbeitet (processed, p) oder weitergeleitet (forwarded, f), was in Tab. 2 verdeutlicht wird.

Event erfüllt	interessant für	empfangen	verarbeiten	weiterleiten
$s_i \wedge \neg s'_i$	i	ja	ja	nein
$\neg s_i \wedge s'_i$	Kinder von i	ja	nein	ja
$s_i \wedge s'_i$	i und Kinder von i	ja	ja	ja
$\neg S_i$	weder i noch Kinder von i	nein	nein	nein

Tab. 2: Veranschaulichung der im Knoten i ausgeführten Aktionen bei Eintreffen eines Events, das bestimmte Kriterien erfüllt (nach [HuGa03])

Ausgehend von diesen unterschiedlichen Subscriptions kann man nun die Kosten berechnen, die entstehen, um ein Event vom Publisher zu den Subscribers auszuliefern. Diese werden auch die Kosten des Baumes genannt, und bilden sich aus der Summe der Kosten der einzelnen Knoten. Die Kosten der Knoten können aufgeschlüsselt werden in Kosten von Empfang, Bearbeitung und Weiterleitung eines Events. Der Aufwand für Empfang sei r Arbeitseinheiten, für Bearbeiten p Arbeitseinheiten und für Weiterleiten f Arbeitseinheiten. Die Arbeitseinheiten können beispielsweise den Zeitaufwand, Speicheraufwand, Prozessoraufwand, Batterieaufwand o. dgl. beschreiben.
Die Berechnung der Kosten in Arbeitseinheiten ist in MANETs (Mobilen Ad Hoc Netzwerken) deswegen relevant, da beschränkte Ressourcen (also beschränkte Arbeitseinheiten) vorhanden sind. Im Folgenden wird angenommen, dass r=p=f gilt, obwohl vorstellbar ist, dass etwa das Weiterleiten eines Events aufwändiger ist als dessen Empfang.
Die Formel für die Berechnung der Kosten C eines Knotens i für die gesamte Verarbeitung des Events E lautet:

$$C_i(E) = (r+p) * \left|E(s_i \wedge \neg s'_i)\right| + (r+f) * \left|E(\neg s_i \wedge s'_i)\right| + (r+p+f) * \left|E(s_i \wedge s'_i)\right|$$

Dabei steht E(α) für die Menge der Events, die der Signatur α entsprechen.
|E(α)| steht für die Anzahl der Elemente der Menge E(α).
Folglich belaufen sich die Kosten zur Auslieferung des Events E in einem Baum auf

$$C(E) = \sum_i C_i(E)$$

Anhand der Kosten C(E) können nun zwei Bäume miteinander verglichen werden. Der Baum mit den geringeren Kosten ist zu bevorzugen.

Wenn man sich dieses Kostenmaß C_i genauer überlegt, stellt man fest, dass die Komponenten $(r+p)*|E(s_i \wedge \neg s'_i)|$ und $(r+p)*|E(s_i \wedge s'_i)|$ in jedem Baum gleich bleiben. Das liegt daran, dass (egal wie und wo Events durch den Baum geschickt werden) jeder Knoten seine eigenen Events immer empfangen und bearbeiten muss.

Was sich von Baum zu Baum ändert, sind die Kosten, die durch die Weiterleitung an Kinder entstehen. Diese Kosten, die nur auf der Arbeit beruhen, die ein Knoten für die Kinderknoten ausführen muss, nennt man Overhead, kurz O. Der Overhead setzt sich folgendermaßen zusammen:

$$O_i(E) = (r+f)*|E(\neg s_i \wedge s'_i)| + f*|E(s_i \wedge s'_i)|$$

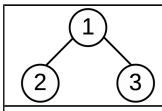
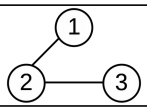
Analog zu C(E) ist der Overhead des gesamten Baumes zur Auslieferung des Events E gleich

$O(E) = \sum_i O_i(E)$

[HuGa03]

2.3.1.1 Beispiel

Im folgenden Beispiel sind alle drei Knoten gegenseitig erreichbar.

	
$O_2(E)=2*0+1*0=0$ (da Knoten 2 keine Kinder hat) $O_3(E)=2*0+1*0=0$ (da Knoten 3 keine Kinder hat) $O_1(E)=2*2+1*0=4$	$O_2(E)=2*1+1*0=2$ $O_3(E)=2*0+1*0=0$ (da Knoten 3 keine Kinder hat) $O_1(E)=2*2+1*0=4$ (da Knoten 2 auch Subscription von Knoten 3 enthält)
O(E)=4 → besserer PST	O(E)=6 → schlechterer PST

(nach [HuGa03])

2.3.2 Baumgenerierungs-Algorithmen in MANETs

Dieses Effizienzmaß O wird nun von Baumgenerierungs-Algorithmen verwendet, um einen optimalen PST zu finden. Ein optimaler PST weist einen minimalen Overhead auf.

2.3.2.1 Algorithmus OPT

OPT ist ein Algorithmus, der einen optimalen PST garantiert. Somit wird der Baum mit dem geringsten Overhead ermittelt. OPT ist ein zentralisierter Algorithmus, d.h. er läuft auf einem einzigen Knoten. Auf diesem Knoten ist das Wissen über das gesamte System vorhanden, sprich die Subscriptions aller Knoten und der Erreichbarkeitsgraph. Nachdem dieser Knoten den optimalen PST errechnet hat, verteilt er diesen an alle anderen Knoten.

Der Algorithmus errechnet den optimalen PST folgendermaßen:

1. Berechnung aller aufspannenden Bäume aus dem Erreichbarkeitsgraphen
2. Berechnung des Overheads für jeden aufspannenden Baum
3. Wahl des aufspannenden Baumes mit geringstem Overhead als PST

Zu bemerken ist hierbei, dass der resultierende PST kein minimal aufspannender Baum im grafentheoretischen Sinn ist, sondern auf der Minimalität des Overheads beruht.
Der Nachteil des OPT Algorithmus besteht darin, dass die gesamte Menge der Spannbäume durchsucht wird. Dies erfordert u.U. viele Ressourcen, was besonders in MANETs zum Problem wird. Weitere Nachteile, die ihren Ursprung in der Zentralität des Algorithmus haben, liegen im Single Point of Failure und der schlechten Skalierbarkeit [HuGa03].

2.3.2.2 Algorithmus ShopParent

ShopParent garantiert im Gegensatz zu OPT kein optimales Ergebnis, ist dafür aber verteilter Natur. Es handelt sich hierbei um einen Greedy-Algorithmus, daher wird nur eine suboptimale Lösung ermittelt. Beim ShopParent Algorithmus braucht kein Knoten globales Wissen zu haben, sondern nur bestimmte Informationen über die Eltern- und Kindknoten. Dabei wird in jedem Knoten eine Instanz des Algorithmus ausgeführt. Der Algorithmus wird im Folgenden beschrieben.

Kommt ein neuer Knoten d zum ad hoc Netzwerk hinzu, so muss dieser ermitteln, wer sein Elternknoten (parent) im PST sein soll. (Die im Folgenden ausgeführten Aktivitäten werden in Abb. 5 dargestellt.) Dazu sendet d eine Parent-Probe Nachricht in das Netz. Wie in einem Funknetz üblich, erreicht diese Nachricht alle in Reichweite befindlichen Knoten von d. Alle Nachbarsknoten, die bereits einen Weg zum Root-Knoten r gespeichert haben, senden nun eine Antwort zurück an d. Diese Antwort ist die Parent-Advertise Nachricht. So senden die Knoten a und c (Nachbarknoten von d, die bereits eine Route zu r gespeichert haben) eine Parent-Advertise Nachricht an d.

Diese enthält die Route von a bzw. c zu r, die ID von a bzw. c und die effektive Subscription von a bzw. c.
Der Knoten i wählt nun aufgrund aller erhaltenen Parent-Advertise Nachrichten jenen Knoten j als seinen Elternknoten, der die beste Routing-Metrik ergibt (beispielsweise den kürzesten Weg zu Root). Weiters muss i seine eigenen Subscriptions an j schicken, der nun seine daraus resultierende neue effektive Subscription an seinen Parent-Knoten schicken muss. Eventuell muss j nun nach einem neuen Parent-Knoten suchen (mittels Parent-Probe), wenn der alte Parent-Knoten die neue effektive Subscription nicht erfüllt [HuGa03].

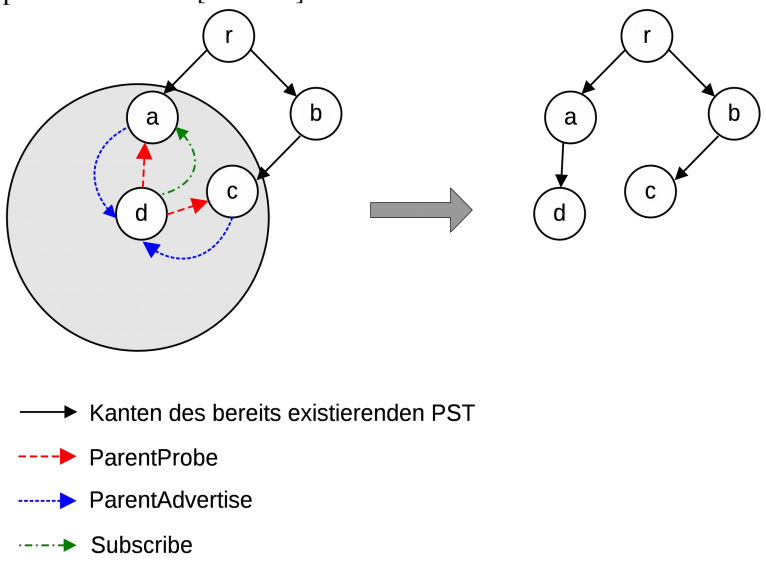


Abb. 5: Ausführung des ShopParent Algorithmus im Knoten d, nachdem dieser in die Reichweite von a und c kommt

In einem mobilen Ad-hoc-Netz können sich aufgrund von Reichweitenüberschreitungen, Batterieladungsabfall u. dgl. ständig Änderungen im Erreichbarkeitsgraphen ergeben. Daher ist es notwendig, dass die Knoten den PST regelmäßig aktualisieren. Hierzu sendet jeder Knoten in periodischen Zeitabständen eine Parent-Advertise Nachricht aus, um den anderen mitzuteilen, dass er noch online bzw. verfügbar ist. Weiters hört jeder Knoten auf solche Parent-Advertise Nachrichten seiner Nachbarn. Nun können zwei Fälle eintreten:

1. Der Knoten i empfängt keine Parent-Advertise Nachrichten und muss sich deshalb mittels Parent-Probe Nachrichten auf die Suche nach einem neuen Parent-Knoten machen.
2. Der Knoten i empfängt Parent-Advertise Nachrichten

a. Erkennt i, dass es einen Nachbarsknoten n mit besseren Routing-Metriken gibt als jene vom alten Parent, so wählt er n als neuen Parent und benachrichtigt seinen alten Parent-Knoten dahingehend, dass dieser die Weiterleitung von Events an i einstellen kann.
b. Erkennt i, dass der alte Parent-Knoten noch immer die beste Routing-Metrik aufweist, so behält er diesen als Parent.

[HuGa03]

In Abb. 6 wird nun ein konkretes Beispiel veranschaulicht, in dem Knoten a ausfällt.

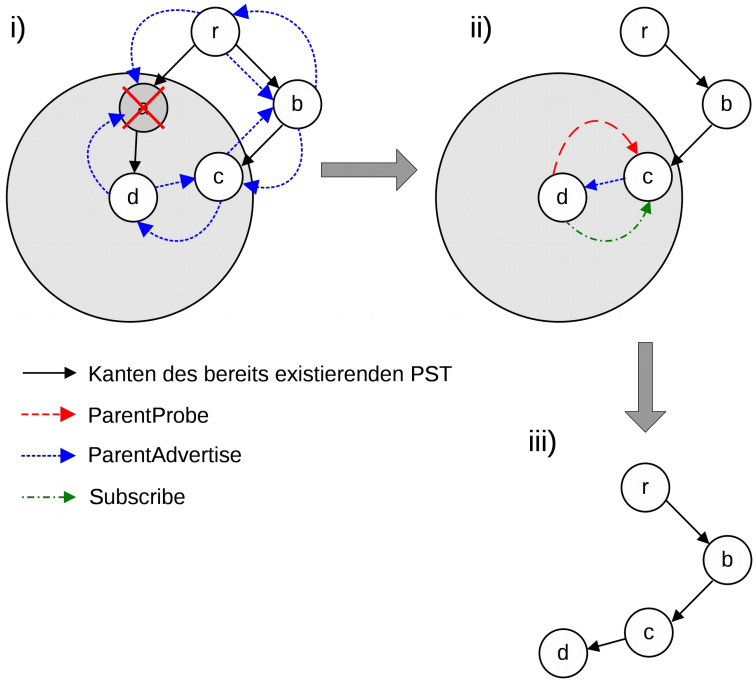


Abb. 6: Aussenden der Parent-Advertise Nachrichten zu regelmäßigen Zeitpunkten (i). Nachdem erkannt wird, dass Knoten a nicht reagiert, sucht sich d einen neuen Parent-Knoten (ii). Der neu entstandene PST wird in (iii) veranschaulicht.

Anhand der Routing-Metrik unterscheidet man folgende drei Varianten des ShopParent Algorithmus:

- **SP-NHOP**
  Die Variante SP-NHOP wählt die Länge des Weges bis zum Root-Knoten als Routing-Metrik. Die Länge des Weges ist die Anzahl der Hops. Dabei muss jeder

Knoten die Anzahl der Hops bis zum Root-Knoten speichern und bei den Parent-Advertise Nachrichten weiterschicken.

- **SP-OVHD**
  Bei SP-OVHD benützt jeder Knoten i als Routing-Metrik die Änderung des Overheads, die sich ergibt, wenn i einen Knoten als Kind aufnehmen würde.
- **SP-COMBO**
  SP-COMBO wählt als Routingmetrik das Produkt aus den Metriken aus SP-NHOP und SP-OVHD.

[HuGa03]

2.4 LIME – Linda in a Mobile Environment

Linda und LIME sind Modelle für die Kommunikation in verteilten Systemen. Die Kommunikation erfolgt dabei zeitlich, räumlich und (in Erweiterungen) datenflusstechnisch entkoppelt. Während Linda auf festverdrahtete Netze ausgerichtet ist, bietet LIME eine Erweiterung für mobile Netze und auch Ad-hoc-Netze. Dabei ist LIME – im Gegensatz zu den bisher besprochenen Publish/Subscribe-Modellen – nicht nur auf die Kommunikation mobiler Hosts, sondern auch auf die Kommunikation mobiler Agenten ausgelegt [MuPR03]. Mobile Hosts und mobile Agents unterscheiden sich hinsichtlich der Art der Mobilität.

- **Physische Mobilität („Roaming“)**
  Darunter versteht man die Möglichkeit, dass sich ein Knoten eines Netzwerkes (Host) physisch bewegen kann. Somit kann sich der Host außer Reichweite seines Access-Points zum Netzwerk bewegen. Daraus ergibt sich, dass der Host entweder nicht mehr erreichbar ist oder sich mit einem neuen Access-Point verbindet, der sich an einer anderen Position im Netzwerk befindet. Physische Mobilität wird auch als Roaming bezeichnet.

- **Logische Mobilität („Code Mobilität”)**
  Logische Mobilität wird häufig Code-Mobilität genannt, dabei bewegt sich ausführbarer Code von einem Knoten zu einem anderen. Somit kann beispielsweise Code zwischen physisch mobilen, aber auch zwischen physisch nicht mobilen Hosts hin und her geschickt werden.

[MuPr99]

Sowohl bei Linda als auch bei LIME funktioniert die Kommunikation über einen „Shared Tuple-Space“, einen sogenannten gemeinsamen Speicher. Da dieser Speicher persistent ist und die Adressierung des in diesem Speicher abgelegten Inhaltes inhaltsbasiert erfolgt, ist Linda/LIME zeitlich und räumlich entkoppelt. Inhaltsbasierte

Adressierung bedeutet, dass die gespeicherten Daten nicht über die Adressen der Erzeuger dieser Daten adressiert werden, sondern über inhaltliche Kriterien. So kann ein Kommunikationspartner nicht alle Daten, die von Knoten mit der Adresse a erzeugt wurden, abfragen, sondern nur etwa alle Daten mit Inhalt x. Das Lesen und Schreiben der Daten erfolgt dabei über vordefinierte Operationen „in“ und „out“. Mittels „out“ kann in den Tuple-Space geschrieben werden, mit „in“ wird etwas aus dem Tuple-Space gelesen. Dabei ist zu erwähnen, dass out nicht blockiert, in jedoch schon. D.h. dass ein Prozess, der aus dem Tuple-Space der keine (bzw. nicht die gewünschten) Daten enthält, lesen will, solange warten muss (blockiert) bis ein anderer Prozess diese Daten in den Tuple-Space geschrieben hat [MuPr99], [MuPr03].
Nun wäre Linda/LIME laut der im Kapitel 2.1.1 getroffenen Klassifikation ein Modell generativer Kommunikation. Durch die Erweiterung durch eine nicht-blockierende Leseoperation wird Linda/LIME um datenflusstechnische Entkopplung [MuPr03] ergänzt. Somit sind Linda und LIME als Publish/Subscribe-Systeme einzustufen.

Das Merkmal, das LIME von anderen Pubish/Subscribe-Varianten unterscheidet, ist die Art des Tuple-Space. Jeder Agent hat seinen eigenen Tuple-Space, der sich immer mit dem Agenten mitbewegt. So bewegen sich die einzelnen Agenten zwischen den Hosts und nehmen ihren Tuple-Space immer mit. Daher ist dieser auch noch verfügbar, wenn ein Host offline geht [MuPr03]. Agents brauchen immer einen Prozessor, um ihren Code (sich selbst) ausführen zu können. Dieser Prozessor wird von den einzelnen Hosts zur Verfügung gestellt. Somit ist ein Host eine Art Container für einen Agenten [MuPr01].
Befinden sich Agenten in Reichweite zueinander, so können diese Nachrichten (Events im Sinne des Publish/Subscribe) miteinander austauschen.
Zwei Agenten befinden sich in Reichweite, wenn sie entweder auf demselben Host, oder auf benachbarten Hosts ausgeführt werden. Benachbarte Hosts wiederum sind Hosts, die mit einem Netz (Kabel bzw. Funkverbindung) miteinander verbunden sind.
Der Nachrichtenaustausch erfolgt hierbei auf eine spezielle Art:
Kommen mehrere Agenten in Reichweite zueinander, so bilden sie eine Gruppe (LIME group). Dieser Vorgang nennt sich „engagement“. Die Inhalte der Tuple-Spaces der einzelnen Agenten werden dabei in einer abstrakten Form zusammengefügt. Daraus entsteht ein Tuple-Space mit lokalen Objekten und Remote-Objekten anderer Tuple-Spaces. Dabei scheint es dem Agenten, als würde er alle Daten lokal gespeichert haben. Somit kann jeder der Agenten auf alle Tuple-Spaces der in Reichweite befindlichen Agenten zugreifen, ohne selbst zu wissen, wo sich die Daten befinden (referentielle Entkopplung).
Der Vorgang, bei dem sich zwei Agenten voneinander entfernen und der Tuple-Space wieder getrennt wird, nennt sich „disengagement“ [MuPR03].

3 Gossip und Epidemische Verbreitung

3.1 Epidemic Algorithms

3.1.1 Einleitung

Epidemische Algorithmen eignen sich besonders zur Vernetzung vieler kleiner drahtloser Geräte, die von Batterien gespeist werden. Solche Systeme unterliegen besonderen Ressourcenbeschränkungen, wie Energie, Speicherkapazität und CPU-Leistung. Um trotzdem Kommunikation betreiben zu können, ist es entscheidend, sehr einfache Algorithmen für Routenfindung, das Routing selbst, das Multicasting und das Aufnehmen neuer Knoten in solche Netze zu entwickeln. Solche Algorithmen müssen lokal arbeiten, minimale Zustände haben, Topologieänderungen im Netzwerk verkraften und mit minimaler Kommunikation auskommen. Die im Weiteren beschriebenen Algorithmen (flooding, gossip) weisen diese Eigenschaften auf. Im Artikel [GKW+02] wird speziell auf Experimente mit Flooding-Algorithmen eingegangen und Auswirkungen auf Netzwerkschichten (Physical und Data Link Layer, MAC Layer, Network und Application Layer) beschrieben. In [NTSS99] werden Analysen allgemeiner Art (redundant broadcast, contention und collision) durchgeführt sowie Lösungsmöglichkeiten für Leistungssteigerungen geboten. Die Kommunikation findet auf einem Kanal statt und funktioniert mittels CSMA (carrier sense multiple access) und keiner CD (collision detection). [SACS03] beschäftigt sich speziell mit Broadcast basierend auf der Performanzsteigerung mittels wahrscheinlichkeitstheoretischer Ansätze.
Dieses Kapitel behandelt grundsätzliche Algorithmen aus den zuvor erwähnten Artikeln und stellt Probleme und Leistungssteigerungen allgemeiner Art vor.

3.1.2 Flooding

Epidemische Algorithmen, wie auch *flooding* einer ist, betreiben keine differenzierte Kommunikation, sondern verbreiten Informationen immer nur zu den nächsten Kommunikationspartnern, die ihrerseits nichts anderes tun. Auf diese Weise verbreiten sich Informationen über das gesamte Netzwerk. Der folgende Algorithmus, der aus [GKW+02] stammt, setzt diese Voraussetzungen um:

- Let S be local state of node and R a random number.
- If message M_i is received for the first time, then:

- Take local action based on M_i: $S \leftarrow f_1(M_i,S)$.
- Compose message $M_i' = f_2(M_i, S)$.
- Make Boolean retransmit decision $D = f_3(S, R)$.
- If D is yes, then
  Transmit M_i' to all neighbours.

Mit diesem verallgemeinerten Algorithmus wird dargelegt, dass jeder Knoten, der eine Nachricht erhält, aufgrund dieser seinen lokalen Zustand ändert. Wird die Nachricht zum ersten Mal empfangen, wird lokal entschieden, ob sie weitergeschickt wird oder nicht. Es wird auch dargestellt, dass die neue Nachricht nicht unbedingt die gleiche sein muss wie die empfangene. Dies wird durch die Funktionen *f1, f2* und *f3* verdeutlicht: Jeder Knoten besitzt einen lokalen Zustand, der verändert wird, sobald eine Nachricht eintrifft (gewährleistet durch *f1*(M_i,*S*)). Danach wird mit der Funktion *f2* eine neue Nachricht erzeugt, die aus Informationen des lokalen Zustandes des aktuellen Knotens und der ursprünglichen Nachricht besteht. *F3* ist eine Funktion, die basierend auf einer Zufallszahl und dem lokalen Zustand entscheidet, ob die neu erzeugte Nachricht weiter geschickt werden soll oder nicht.
Zwei konkrete Umsetzungen davon werden in [SACS03] beschrieben. Beide Algorithmen verändern die in der Nachricht inkludierte Information nicht. Der Unterschied besteht darin, dass im ALGORITHMUS 1 die Nachricht auf jeden Fall weitergeschickt wird und im zweiten Fall nur mit einer bestimmten Wahrscheinlichkeit.

ALGORITHMUS 1 ***flood (m)*****:**
Upon reception of message *m* at node *n*:

If message *m* received for the first time **then**
- Broadcast(*m*) {this is the basic local broadcast primitive to nodes within range only}

endif

ALGORITHMUS 2 ***p-flood (m,p)*****:**
Upon reception of message *m* at node *n*:

If message *m* received for the first time **then**
Broadcast(*m*) with probability *p* {this is the basic local broadcast primitive to nodes within range only}

endif

Abb. 7: Zwei konkrete Algorithmen zur Nachrichtenverbreitung mittels flooding aus [SACS03]

3.1.3 Redundante Übertragungswiederholungen *(redundant broadcast)*, Konkurrenzphasen *(contention)* und Kollisionen *(collision)*

Durch die Überschneidung der Reichweiten vieler Knoten, die räumliche Ausbreitung der Daten und das Fehlen einer Basisstation ist zu erwarten, dass viele Aussendungen,

die ein Knoten vornimmt, bereits von potentiellen Empfängern erhalten wurden. Daher sind diese Nachrichten für bestimmte Knoten redundant.
Durch räumliche Nähe der Knoten ergibt sich, dass bei neuerlichen Aussendungen eine starke Konkurrenz um den Kommunikationskanal besteht.
Durch das Fehlen von *collision detection* kann man davon ausgehen, dass man mehr Kollisionen hat, die mehr Schaden anrichten. Um diesen Effekt zu minimieren, wird *CA (collision avoidance)* betrieben [NTSS99].

3.1.3.1 Redundant broadcast

Wenn man von zwei Sendern (A, B) ausgeht, die eine bestimmte Kreisfläche abdecken, und Knoten A eine Nachricht nach B sendet, gibt es eine Überschneidung der Kreisfläche (vgl. mit „Abb. 8: Analyse der zusätzlichen Flächenabdeckung mit 2 Knoten“ darunter). Wird nun von Knoten B aus ein *broadcast* durchgeführt, so wird erneut die überschnittene Fläche abgedeckt und nur ein bestimmter Bereich zusätzlich. Der zusätzlich abgedeckte Bereich beträgt durchschnittlich 41% und maximal 61% (grauer Bereich in „Abb. 8: Analyse der zusätzlichen Flächenabdeckung mit 2 Knoten“).
Nimmt man zu den vorhandenen zwei Knoten einen weiteren Knoten C (vgl. mit „Abb. 9: Analyse der zusätzlichen Flächenabdeckung mit 3 Knoten“ darunter) auf, der seinerseits erneut ein *broadcast* durchführt, so deckt dieser im Durchschnitt nur noch 19% an zusätzlicher Fläche ab.
Werden weiterhin Knoten aufgenommen, so kann man für diese errechnen, dass ab einer Knotenanzahl k und $k \geq 4$ durchschnittlich nur mehr 0,05% zusätzliche Fläche abgedeckt wird.

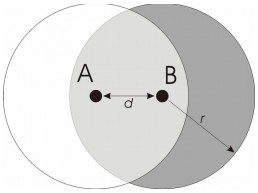


Abb. 8: Analyse der zusätzlichen Flächenabdeckung mit 2 Knoten

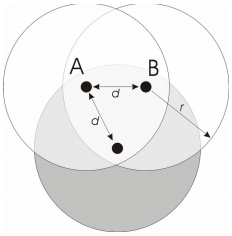


Abb. 9: Analyse der zusätzlichen Flächenabdeckung mit 3 Knoten

3.1.3.2 Contention

Als Szenario wird angenommen, dass wieder drei Knoten mit sich überlappenden Kreisflächen existieren. Knoten A ist der erste Sender, Knoten B ein potentieller zweiter und Knoten C, der in der Schnittmenge von A und B liegt und damit also die *broadcasts* von A und B wahrnehmen kann, ist ebenfalls ein potentieller zweiter Sender.

In diesem Szenario ist die Wahrscheinlichkeit sehr groß, dass Knoten B und C um den Sendekanal konkurrieren, da sie zur gleichen Zeit den *broadcast* von A empfangen und gleichzeitig beide selber einen *rebroadcast* versuchen. Die Wahrscheinlichkeit einer *contention* liegt dabei bei 59%.

Wenn man zusätzliche Knoten in den Sendebereich von Knoten A aufnimmt, kann man zwei Knotentypen ausmachen. Einerseits solche, die sich bei einem Sendeversuch nicht stören und andererseits andere, die auf den gleichen Kanal senden wollen. Es ist unschwer zu erkennen, dass bei einer Gleichverteilung der zusätzlichen Knoten im Bereich von A immer mehr *contention* auftritt und im Gegenzug dazu immer weniger konkurrenzfreie Knoten vorhanden sind. Bei einer Knotenanzahl n und $n \geq 6$ steigt die Wahrscheinlichkeit einer *contention* auf über 80%.

3.1.3.3 Collision

Bei Ad-Hoc-Netzen wird CSMA/CA verwendet. Dabei wird nach dem Versand von Daten, oder wenn erkannt wird, dass sich etwas am Kanal befindet, eine *Backoff*-Prozedur angestoßen. In dieser wird zu gewissen Zeitpunkten der Kanal abgetastet und, sofern dieser frei ist, eine zuvor gewählte Zufallszahl um eins erniedrigt, solange bis Null erreicht wird. Danach kann der Sender auf den Kanal zugreifen.

Dabei tritt das Problem auf, dass die in verschiedenen Knoten ablaufenden *backoff* Prozeduren ungefähr gleich aussehen und dadurch unter Umständen alle Knoten zum gleichen Zeitpunkt ihren *rebroadcast* beginnen. Eine andere negative Eigenschaft ist, dass während einer Sendung keine Kollisionen erkannt werden. Wenn eine Aussendung beginnt, dann wird auch bei Kollisionen das gesamte Paket gesendet. Je größer demnach Pakete sind, desto schlimmer wirken sich Kollisionen aus.

3.1.4 Mechanismen um Redundanzen, Konkurrenzphasen und Kollisionen einzuschränken

Eine Möglichkeit, um solche Probleme zu verhindern ist, dass einige Knoten daran gehindert werden, ihre Informationen zu senden. Ferner werden fünf solcher Ansätze aus [NTSS99] beschrieben.

3.1.4.1 Probabilistic Scheme

Dieses Verfahren beschreibt den zuvor in „Flooding“ angeführten ALGORITHMUS 2. Eine Nachricht wird mit einer Wahrscheinlichkeit p weitergeschickt und mit 1–p verschluckt. Wenn p gleich 1 ist, entspricht dieses Verfahren dem normalen Flooding. Um die Probleme, die in Kapitel „Contention“ beschrieben wurden, zu vermeiden, wird eine kleine, zufällige Verzögerung nach dem Backoff eingebaut (Damit beginnen nicht alle Sender ihr Rebroadcasting zur gleichen Zeit.).

3.1.4.2 Counter-Based Scheme

Dieses Verfahren beruht auf den in Kapitel „Redundant broadcast“ gemachten Erfahrungen in Bezug auf zusätzliche Flächenabdeckung, wenn eine Nachricht zuvor von vielen Knoten verbreitet wurde.
Dabei zählt ein Knoten, wie oft eine Nachricht, die er gerade erhalten hat, bereits ausgeliefert wurde, bevor er diese erneut verteilt. Ist sie schon ausreichend oft kommuniziert worden, wird das *broadcast* in diesem Knoten verhindert.

3.1.4.3 Distance-Based Scheme

Dieses Verfahren funktioniert ähnlich wie das obere, nur wird hierbei nicht die Anzahl der bereits ausgesendeten Nachrichten als Maß genommen, sondern der Abstand zweier Knoten. Je näher zwei Knoten einander sind, desto weniger zusätzliche Fläche kann bei einem *rebroadcast* abgedeckt werden. Ist die Entfernung E_1 zweier Knoten A und B geringer als eine vorher bestimmte minimale Entfernung E_2, bei der ein Knoten noch senden darf, wird ein *rebroadcast* in B unterdrückt.
Zur Ermittlung der Entfernung zweier benachbarter Knoten könnte die Signalstärke dienen.

3.1.4.4 Location-Based Scheme

Dieser Ansatz versucht exakte Ortsangaben zu benutzen, um festzustellen, ob es sinnvoll ist, eine Nachricht weiterzuleiten. Dies wird mittels *GPS (Global Positioning System)* umgesetzt.
Wenn ein Knoten von *k* anderen, die ihre Position mitschicken, Pakete erhält, so kann die zusätzliche Netzabdeckung errechnet werden und damit über das *rebroadcast* in diesem Knoten entscheiden. Die Kalkulation der zusätzlichen Abdeckung ist allerdings teuer, weshalb man sich mit konvexen Polygonen behilft:
Die Kommunikationspartner eines Knotens können mittels Linien miteinander verbunden werden, wobei sich ein konvexes Polygon ergibt. Liegt der Knoten, der ein *broadcast* durchführen will, innerhalb, kann die Aussage getroffen werden, dass er maximal 22% zusätzliche Fläche bei einem *broadcast* hätte abdecken können und damit keine Rundsendung mehr durchführen darf.

3.1.4.5 Cluster-Based Scheme

In diesem Verfahren werden nicht Berechnungen zum Zweck der zusätzlichen Netzabdeckung durchgeführt, sondern Graphen zur Leistungssteigerung verwendet. Der *Cluster Formation Algorithm*:
Ein Knoten sendet periodisch Pakete aus, um seine Präsenz bekannt zu geben. Jeder Kommunikationspartner verbindet sich selbstständig mit einem anderen. Alle Knoten führen eine eindeutige *ID* mit sich. Ein Knoten mit minimaler *ID* ist *head* eines *clusters*, damit also Kopf seiner verbundenen Nachbarn, die über die *ID* des Kopfes identifiziert werden können. Ein Knoten, der zwei *cluster* miteinander verbindet, wird als *gateway* bezeichnet. Wenn ein Knoten *A* mit niedrigerer *ID* als der *head H* ins Netz eintritt, dann tauschen *A* und *H* ihre Rollen.
Durch diese Einteilung des Netzes ergeben sich einige Konsequenzen:
Der Kopf eines *clusters* kann durch seine Rundsendung alle Knoten erreichen, sofern keine Kollision auftritt. Um Nachrichten in andere Netzwerke zu propagieren, können *gateways* benutzt werden. Für normale Mitglieder eines *clusters*
besteht keine Notwendigkeit mehr, Nachrichten zu *broadcasten,* da diese Aufgabe ausschließlich den *gateways* und den *heads* eines *clusters* obliegt.

3.2 *Gossip*

3.2.1 Einleitung

Gossip kann mit „Tratsch“ übersetzt werden und bedeutet in Bezug auf Netzwerke, dass sich benachbarte Knoten über ihre unzuverlässig ausgelieferten Informationen, die erhalten oder auch nicht erhalten wurden, unterhalten. Demnach kann man dieses Protokoll in 2 Phasen einteilen: Während der ersten Phase werden Informationen mittels eines unzuverlässigen Protokolls verteilt. Da damit allerdings noch nicht sichergestellt werden kann, dass alle Informationen auch bei allen Empfängern angekommen sind, wird in der zweiten Phase versucht, verloren gegangene Informationen wiederzugewinnen. Dies geschieht indem benachbarte Knoten miteinander Informationen austauschen und diese wechselseitig abgleichen. Im Weiteren werden Algorithmen besprochen, die spezielle Mechanismen zur Verfügung stellen, um diesen Abgleich effizient zu gewährleisten. Mit dem *gossip protocol* kann man nicht nur *reliable multicast* umsetzen, sondern auch *failure detection* und *garbage collection* betreiben [LIMM00]. In dieser Arbeit wird speziell auf *scalability* und *reliability* von *gossip* Protokollen eingegangen.

3.2.2 Merkmale von *Gossip* Protokollen

Eine sehr adäquate Zusammenstellung der im Folgenden diskutierten Merkmale von *gossip* entstammt aus [LIMM00] und wird mittels Informationen aus [VORB01] im Abschnitt „*Scalability* von *Gossip* im Vergleich mit anderen Protokollen“ ergänzt.
Die Relevanz von *gossip* in Ad-hoc-Netzen liegt in dessen Skalierbarkeit. Auch wenn neue Teilnehmer im Netz aufgenommen werden, sinkt dessen Performanz nicht rapide ab, wie das bei anderen Protokollen der Fall ist. Erwähnenswert ist auch, dass die Anzahl der von einem Knoten ausgesendeten Nachrichten konstant bleibt, auch wenn neue Kommunikationspartner im Netz dazu kommen. Der dabei verwendete Algorithmus ist einfacher Natur und basiert auf sich langsam ändernden Informationen. Solange demnach das Netz stabil bleibt, auch wenn neue Knoten ins Netz eintreten, ist *gossip* skalierbar.
Das Protokoll ist äußerst anpassungsfähig und das Eintreten und Verlassen von Teilnehmern wird mit dem *gossip* Protokoll selbst realisiert.
Für viele *broadcast* Protokolle gibt es einen Grenzwert *f* an Fehlern, mit denen das Protokoll gerade noch arbeiten kann. Wird dieser Grenzwert *((f+1)* Fehler) überschritten, funktioniert das Protokoll nicht mehr. Betrachtet man den Bereich von Null bis *f* Fehlern, so kann man solche Protokolle unterscheiden, die einerseits keinen gravierenden Verfall der Leistung bis zum Grenzwert *f* aufweisen (*degrade gracefully*) und andererseits solche, die mit zunehmender Fehleranzahl einen rasche Leistungsminderung erfahren. *Gossip* gehört zur ersteren Protokollgruppe und kann demnach auch noch mit hoher Fehlerrate eine gute Leistung erzielen.

3.2.3 *Scalability* von *Gossip* im Vergleich mit anderen Protokollen

In [AYRM03] wird ein kurzer Überblick über andere routing Protokolle gegeben: Erwähnt wird speziell das Protokoll *distributed core multicast* (DCM). Dieses Protokoll sieht ausgezeichnete Router vor, die sich an den Ecken der Backbones befinden. Solche dezidierte Router gibt es allerdings bei mobilen Ad-hoc-Netzen nicht, weshalb spezielle Protokolle wie zum Beispiel *AMRoute* (benutzt einen multicast Baum basierend auf einem vermaschten Netz), *ODMRP* (Ein Sender flutet auf Befehl eine bestimmte Anzahl von Hosts) oder *MAODV* (vgl. „3.2.5.1 Kurzbeschreibung von MAODV“) entwickelt wurden. Auch *Gossip* gehört zur zweiten Gruppe und kann als kombiniertes Zweiphasenprotokoll angesehen werden. Als Beispiel kann das später beschriebene *Anonymous Gossip* angeführt werden, welches in der ersten Phase Nachrichten in das Netz flutet und in der zweiten Phase das *MAODV*-Protokoll verwendet, um Informationen zwischen Hosts auszutauschen.

In [VORB01] werden Probleme von traditionellen *reliable multicast protocols* behandelt und ihre Schwächen bei Einbettung in große Internet-Umgebungen beschrieben.

Die meisten Protokolle, die *reliable group communication* umsetzen, haben einen signifikanten Einbruch des Datendurchsatzes, sobald mehrere Teilnehmer in diesem Netz eine Störung erfahren. Im Extremfall kann ein einziger Knoten, der durch beispielsweise lokalen Paketverlust verlangsamt wird, seine gesamte Gruppe beeinflussen (*Throughput Instability)*.
Wegen dem gerade adressierten Problem werden immer aggressivere Fehlererkennungsmechanismen angewendet und problembereitende Knoten von der Gruppe ausgeschieden *(Micropartitions)*. Da diese allerdings nicht ausgeschieden bleiben können, müssen sie später wieder in die Gruppe aufgenommen werden. Durch diesen Mechanismus werden sehr oft funktionierende Knoten ausgeschieden, um danach wieder in die Gruppe einzutreten. Durch diesen Versuch, das Netzwerk skalierbar zu halten, entsteht der Effekt, dass gerade das Gegenteil erreicht wird und mit steigender Teilnehmerzahl die Kosten für den Umgang mit langsamen Knoten steigen.
Um Skalierbarkeit handhabbar zu machen, werden große Systeme als hierarchische Baumstruktur dargestellt. Experimente haben gezeigt, dass gerade diese Struktur bei auftretenden Netzinstabilitäten die Eigenschaft hat, Datenverkehr stark zu erhöhen.
Zum Auffinden von lokalen Kommunikationspartnern tritt bei den herkömmlichen Protokollen oft der Effekt auf, dass bei der Kommunikation zwischen Sender und Empfänger das Netz mit Nachrichten, die von allen anderen Teilnehmern der Gruppe gelesen werden *(Request and Retransmission Streams)*, überschwemmt wird.

3.2.3.1 Skalierungsvorteile von gossip

Dieser Abschnitt dient nicht zur Beschreibung der in *bimodal multicast* und *anonymous gossip* verwendeten Algorithmen, die später beschrieben werden, sondern stellt lediglich die aus den Verfahren resultierenden Vorteile fest.
Die Kommunikation bei *gossip* benutzt ein gewichtetes Schema, das zum Ausgleich der Last auf Verbindungen zwischen Knoten dient. Durch die vorwiegend mit Nachbarknoten stattfindende Kommunikation ergibt sich der Vorteil, dass über Verbindungen übertragen wird, bei der die Latenzzeit sehr gering ist. Weiter entfernte und damit teure Kommunikation über Router wird weitestgehend vermieden.
Um die Nachrichtenmenge nicht zu groß werden zu lassen, wird nicht jede Nachricht gespeichert, sondern eine Hashtabelle über die Nachrichten gebildet. Damit wird sichergestellt, dass die am häufigsten verwendeten Nachrichten ausreichend verlässlich im Netzwerk zur Verfügung stehen. Der sich dadurch ergebende Vorteil ist, dass die durchschnittliche Datenlast bei einem Teilnehmer abnimmt, wenn das System größer wird.
Weiters ist hervorzuheben, dass jeder Knoten im Netzwerk eine konstante Last trägt, die Algorithmen einfach zu implementieren sind und diese auch kostengünstig in ihrer

Ausführung sind. Bei guter Konfiguration kann auch für Übertragungswege und Router eine günstige Auslastung erzielt werden.
Bimodal multicast stellt die Möglichkeit zur Verfügung, dass *reliability* für die benutzte Applikation angepasst werden kann.
Das Protokoll besitzt eine sehr stetige Datenübertragungsrate und ist daher äußerst vorhersehbar und variiert kaum hinsichtlich seines Durchsatzes. Diese Eigenschaft ist speziell für *realtime* Applikationen eine äußerst günstige Eigenschaft.

3.2.4 Bimodal Multicast (BM)

Die verwendete Basisliteratur für *bimodal multicast* ist [VORB01] und [BHO+99]. Dieses Protokoll ist statischer als das im Anschluss beschriebene *AG*. In *BM* betrachtet man eine abgegrenzte Zeitdauer *(execution periods)*, bei der die Anzahl der Mitglieder bekannt und stabil ist. Beim Verlassen oder Eintreten von Teilnehmern endet die alte Periode und eine neue beginnt.
Im Gegensatz zu *anonymous gossip* wird *bimodal multicast* nicht in drahtlosen Netzen verwendet, sondern kann in kritischen Anwendungen, wie Flugverkehrskontrollsystemen, basierend auf LAN oder WAN eingesetzt werden. Typischer Weise befinden sich in solchen Systemen nicht viel mehr als einhundert Knoten. Entstanden ist es aus dem alten NNTP Protokoll (bei *network news servers* im Einsatz).
Bimodal multicast benutzt *gossip,* um eine wahrscheinlichkeitstheoretische Aussage über die Verlässlichkeit von Netzwerken zu treffen (Anpassung der *reliability*), wie zuvor erwähnt.

3.2.4.1 Protokollphasen

Das hier verwendete Protokoll wird als *Pbcast* (*probabilistic broadcast*) bezeichnet und stellt eine konkrete Implementierung des *BM* dar..

1. OPTIMISTIC DISSEMINATION PROTOCOL
Diese erste Phase verteilt, basierend auf *IP multicast*, Nachrichten unzuverlässig im Netz, oder, falls dies nicht zur Verfügung steht, mit einem beliebigem anderen Protokoll. Im zweiten Fall wird ein Spannbaum über die Teilnehmer aufgebaut (vgl. *AG* mit MAODV).

2. TWO-PHASE ANTI-ENTROPY PROTOCOL
Diese zweite Phase dient zur Sicherstellung, dass mit bestimmter Wahrscheinlichkeit alle Knoten die Nachricht erhalten.
Dieser Teil des Protokolls kann wieder in zwei Phasen eingeteilt werden. Die erste erkennt, dass ein bestimmter Knoten eine Nachricht nicht erhalten hat und die zweite

korrigiert die Inkonsistenz im Netz. Dabei werden wahlweise Kommunikationspartner aus dem Netz herausgegriffen und der Nachrichtenbestand beider abgeglichen.
Beispiel: In „Abb. 10: Illustration des Pbcast Anti-entropy-Protokolls.“ illustriert: (weiße Fläche … *optimistic dissemination protocol*; graue Fläche … *anti-entropy;* schwarze Pfeile … gut gegangene Übertragungen; graue Pfeile … fehlgeschlagene Übertragungen). Alle Nachrichten in der ersten Phase – bis auf M_0 von P nach Q und M_1 von Q nach R – kommen an. Während der ersten *anti-entropy Phase* wird erkannt, dass Q die Nachricht M_0 nicht erhalten hat. In der zweiten *anti-entropy Phase* passiert das Gleiche mit M_1. Die fehlenden Nachrichten werden während den *anti-entropy* Phasen noch einmal übertragen. Durch die Abbildung verleitet, könnte man annehmen, dass die Phasen sequentiell abgearbeitet werden. Tatsächlich laufen die Phasen aber parallel ab, und die wiederholten Nachrichtenübermittlungen können von beliebigen anderen Kommunikationspartnern, als der tatsächlichen Quelle, kommen.

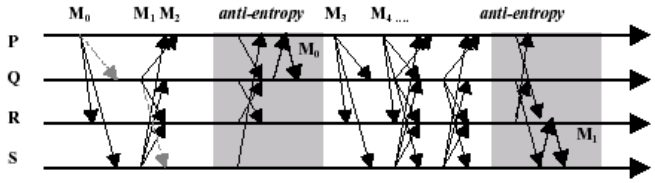


Abb. 10: Illustration des Pbcast Anti-entropy-Protokolls.

3.2.4.2 Optimierungen von Pbcast im Überblick

- *Soft-Failure Detection*
  Soft failure sind solche Fehler, die beim Empfangen einer Nachricht auftreten, auch wenn diese ordnungsgemäß abgeschickt wurde. Wird innerhalb einer Runde die Anforderung einer Neusendung nicht bedient, so wird diese Nachricht beim Einlangen nach der Runde verworfen. Daraus kann geschlossen werden, dass die Rundenzeit nicht korrekt konfiguriert ist.
- *Round Retransmission Limit*
  Die maximale Datenmenge, die während einer Runde ausgetauscht werden kann, ist limitiert. Wird das Limit erreicht, so hört der Sender auf, Nachrichten zu schicken, was zur Folge hat, dass ein ohnedies nachhängender Knoten zusätzlich verzögert wird. Eine Lösung dafür ist, dass über mehrere Runden hinweg von mehreren Sendern der Informationsbedarf des nachlaufenden Knotens abgedeckt wird.
- *Cyclic Retransmission*
  Um redundante Neuauslieferungen zu vermeiden, muss gespeichert werden, was in einer Vorrunde angefordert wurde, um in der nächsten Runde nicht erneut die

gleichen Nachrichten zu übermitteln. Es könnte ein Empfänger eine Nachricht gerade noch nicht empfangen haben und in der neuen Runde eine neuerliche Anforderung für diese absenden.

- *Most-Recent-First Retransmission*
  Die neuesten Nachrichten werden als erste behandelt, da bei dem *oldest-first* Verfahren momentan fehlerhafte Knoten nicht die Möglichkeit haben, sich zu erholen.
- *Independent Numbering of Rounds*
  In diesem Protokoll hat jeder Knoten seine eigene Runde (asynchron zu allen anderen). Die Anzahl der *gossips* einer Nachricht ist aber ausschlaggebend dafür, wann diese vom *gossip* ausgenommen werden kann. Daher kann nicht zentral entschieden werden, wann davon ausgegangen werden kann, dass eine Nachricht bereits von jedem Knoten gesehen wurde, sie folglich aus dem Netz entfernt werden kann. Die Lösung ist, dass jede Nachricht einen Zähler mit der Anzahl ihrer *gossips* mit sich führt.
- *Random Graphs for Scalability*
  Wenn man davon ausgeht, dass *gossip* mittels *IP multicast* umgesetzt wird, ergeben sich zweierlei Nachteile:
  1. Um überhaupt *gossip* betreiben zu können, muss jeder Knoten alle seine *multicast* Gruppen kennen. Diese Voraussetzung hat große Datenmengen zur Folge, die das Netz belasten.
  2. In WANs werden Verbindungen mit großer Verzögerung benutzt, wodurch sich die Rundenzeiten stark verlängern und auch der Pufferaufwand größer wird.

  Beide Probleme können aus diesem Grund vermieden werden: Nicht jeder Knoten muss mit jedem anderen verbunden sein, sondern nur mit einigen davon. WANs bestehen aus vielen kleineren LANs, die miteinander vernetzt sind. In WANs sind nur Endpunkte miteinander verbunden, und in LANs müssen sich nur lokale Teilnehmer kennen.
- *Multicast for Some Retransmissions*
  Es kann vorkommen, dass Nachrichten nicht nur bei einem einzigen Knoten, sondern in einer ganzen Region fehlen. Daher ist es in dieser Situation besser, nicht mit jedem anderen Knoten jeweils eine Punkt-zu-Punkt-Verbindung aufzubauen, sondern die Nachricht gleich zu verteilen.

3.2.5 Anonymous Gossip (AG)

Dieses Protokoll ist speziell für *nomadic wireless devices* angepasst und wird im Bereich der Ad-hoc-Netze eingesetzt. *AG* ist dem zuvor beschriebenen *BM* sehr ähnlich. Die Unterschiede liegen darin, dass bei *AG* nur sehr wenige Informationen über potentielle Kommunikationspartner im Netz bekannt sind. Der Grund dafür liegt

in der raschen Änderung der Topologie solcher Netzwerke. *AG* begegnet dieser Eigenschaft, indem *gossip messages* mit einer bestimmten Lebensdauer über einen zufälligen Pfad geschickt werden und eventuelle Empfänger auf diese Nachrichten antworten.
AG und *BM* sind derart konstruiert, dass mit hoher Wahrscheinlichkeit alle oder keine Knoten Nachrichten empfangen, aber mit sehr niedriger Wahrscheinlichkeit wenige Knoten eine Nachricht erhalten.
Wie in der „3.2.1 Einleitung“ bereits erwähnt, funktioniert *AG* in 2 Phasen: In der ersten Phase werden Informationen mit einem nicht zuverlässigen Protokoll über das Netzwerk verteilt. *Anonymous gossip* verwendet hierzu das MAODV *(Multicast Ad-hoc On-Demand Distance Vector Protocol)* als unzuverlässiges Protokoll, welches *multicast trees* erzeugt. In der zweiten Phase beginnen Knoten miteinander Kontakt aufzunehmen und gleichen ihren Informationsbestand gegenseitig ab.
Die in diesem Abschnitt verwendeten Informationen stammen aus [CHRB01] und [VORB01].

3.2.5.1 Kurzbeschreibung von MAODV

Jeder Knoten des Netzes verwaltet zwei Routing-Tabellen (*Route Table (RT) und Multicast Route Table (MRT)*).
In der *RT* werden die *next hops* für Routen zu anderen Kommunikationspartnern eingetragen. In dieser Tabelle werden folgende Informationen gespeichert:

- Ziel-IP-Adresse
- Zielsequenznummer: Hier wird der Neuigkeitswert der Route bis zum Ziel gespeichert.
- Anzahl der *hops* bis zum Ziel
- IP-Adresse des *next hop*
- Lebensdauer des Eintrages

Wenn bei einem Sender keine adäquate Route gefunden werden kann, wird eine Anfragenachricht (*RREQ*) ins Netz verteilt. Hat ein Knoten am Weg zum Empfänger einen aktuellen Eintrag der Route zum Zielknoten, so wird eine *unicast* Nachricht (*RREP*) zum Sender geschickt. Wenn kein Eintrag besteht, wird eine *RREQ* an alle seine Nachbarn verbreitet. Falls der neue Empfänger bereits der Zielknoten ist, antwortet dieser Knoten ebenso mit einer *RREP* Nachricht. Der Knoten, der die ursprüngliche Nachricht in das Netz gesendet hat, sucht sich aus den von seinen Kommunikationspartnern erhaltenen Nachrichten die kürzeste aus und speichert sie in seiner RT. Jene Knoten, die eine *RREQ* erhalten haben und keinen Eintrag in ihrer RT hatten, speichern ebenfalls den Weg zum Zielknoten in ihrer RT ab.

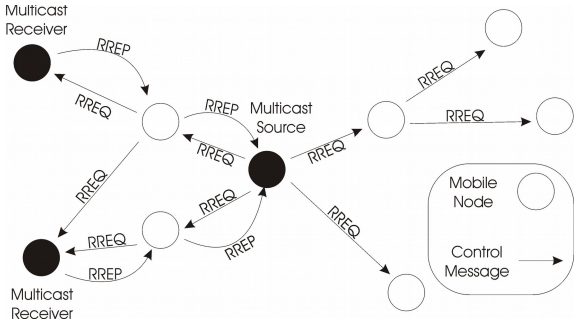


Abb. 11: MAODV Path Recovery [KUCH01]

Die *Multicast Route Table* enthält Einträge über die Gruppen, von denen der aktuelle Knoten ein Router ist. Diese Tabelle besteht aus den folgenden Informationen:

- *Multicast group IP* Adresse
- IP Adresse des Gruppenleiters
- Sequenznummer der Gruppe
- Anzahl der *hops* zum Gruppenleiter
- *Next hops*: Dies sind die Knoten im *multicast tree*, zu denen der Knoten verbunden ist. Es besteht die Möglichkeit, dass Knoten noch nicht angesprungen wurden, jedoch angesprungen werden können. Solche Knoten werden mit einem *enabled flag* vermerkt. Knoten, die näher zum Gruppenleiter sind als andere, werden als *upstream nodes* bezeichnet.
- Lebensdauer des Eintrages

Gruppeneintritte in *MAODV*:
Für Knoten, die nicht einer *multicast* Gruppe angehören, ist es möglich, einer solchen beizutreten. Ein solcher Knoten sendet eine spezielle *RREQ* (diese Nachricht besitzt ein *join flag*) in die gewünschte Gruppe. Jeder Knoten, der das *RREQ* empfängt, vermerkt sich den Ursprungsknoten in seiner MRT (das *enabled flag* auf *false* gesetzt) und kann mit einer *RREP* an den Ausgangsknoten antworten. Um sich eine Route durch seine neue Gruppe zum Gruppenleiter frei zu machen, wählt der eintretende Knoten eine *MACT* Nachricht entlang der kürzesten Route. Alle sich auf dieser Route befindlichen Gruppenmitglieder setzen das *enabled flag* auf *true*.
Verlassen von Gruppen in *MAODV*:
Um aus einer *multicast* Gruppe wieder ausscheiden zu können, wird von diesem Knoten eine *MACT* Nachricht mit gesetztem *prune flag* an den *upstream* Knoten – also dem näher zum Gruppenleiter liegenden – gesendet. Wird eine solche Nachricht empfangen, löscht der *upstream* Knoten den ersteren aus seiner MRT unter der Bedingung, dass der ursprüngliche Knoten ein Blatt im Baum ist. Wenn nicht, dann

verlässt der Knoten trotzdem die Gruppe, muss dann aber als Router für seine alte *multicast* Gruppe fungieren.

Abgebrochene Verbindungen:
Wenn ein Knoten bemerkt, dass sein *upstream* Knoten nicht mehr vorhanden ist, sendet dieser eine *RREQ* Nachricht (mit der Information über die aktuelle *hop* Anzahl) an den Gruppenleiter. Jeder Knoten am Weg kann ein *RREP* an den Absender schicken, wenn bei diesem eine kleinere *hop* Anzahl eingetragen ist. Wenn kein anderer Knoten, auch nach wiederholten Versuchen, antwortet, wird ein neuer Gruppenleiter bestimmt, da angenommen werden muss, dass das Teilnetz vom gesamten Netz getrennt wurde.

3.2.5.2 Anonymous Gossip

Informationen werden natürlicherweise nicht nur ein einziges Mal im Netzwerk verteilt, sondern immer wieder. Daher wird auch die zweite Phase des Protokolls, also der Abgleich empfangener oder nicht empfangener Daten, iterativ wiederholt. Bemerkenswert an dieser Stelle ist, dass bei einer *gossip round* mehr als nur ein Datensatz ausgetauscht werden kann. Ein vereinfachter Algorithmus einer *gossip round* für *AG* sieht wie folgt aus:

1. Ein Knoten A sucht sich beliebig einen anderen Knoten B des Netzes aus.
2. A sendet Informationen über die in diesem Knoten gespeicherten Daten an B.
3. B vergleicht seine Daten mit den Informationen von A, um zu erkennen, was im Knoten B selbst vorhanden ist und was von Knoten A gebraucht wird. Knoten A und B tauschen die fehlenden Nachrichten miteinander aus.

Ein solcher Algorithmus wäre allerdings nicht sehr hilfreich, da er das primäre Ziel, nämlich keine zusätzlichen Staus in ohnedies staugefährdete drahtlose Netze zu bringen, verfehlen würde. Um diese Eigenschaft zu gewährleisten, wird der ursprüngliche Algorithmus verfeinert:
Da bei *AG*, im Gegensatz zu *BM*, keine regelmäßigen Nachrichten als Lebenszeichen in das Netzwerk gesendet werden und daher kein anderer Kommunikationspartner bekannt ist, wird die *gossip message*, eine zusätzliche Nachricht, als Kommunikationsmittel eingesetzt. Eine *gossip message* enthält folgende Informationen:

- Gruppenadresse der *multicast Group*
- die Quelladresse des Senders der *gossip message*
- Array an Sequenznummern von verloren geglaubten Nachrichten
- Anzahl dieser Nachrichten (*lengthOf(*Array))

- die Sequenznummer der als nächstes erwarteten Nachricht

Wie man in dieser Übersicht über die Nachricht erkennen kann, wird schon im Quellknoten über fehlende Sequenznummern die Aussage getroffen, welche Datensätze fehlen.
Der sendende Knoten wählt einen möglichen *gossip* Partner aus der im *MAODV* erstellten MRT aus und sendet diesem eine *gossip message*. Da auch bei einer zufälligeren Auswahl an Kommunikationspartnern lediglich ein kleiner Teil der vorhandenen Teilnehmer verwendet würde, ist diese Auswahl für die Protokollersteller legitim. Ein Empfänger der Nachricht entscheidet, ob er selbst seine Daten mit dem Quellknoten abgleicht oder diese Nachricht weitersendet. Die Antwort auf den *gossip* Wunsch wird über eine *unicast message* beantwortet. Indem Nachrichten entlang des *mutlicast trees* gesendet werden, wird gewährleistet, dass keine Zyklen in der Aussendung entstehen.
Jedem Knoten sind seine fehlenden Informationen bekannt, die er lokal in einer *lost table* speichert und auch in der *gossip message* als *lost buffer* mitschickt. Zudem wird in jedem Knoten eine weitere Tabelle gehalten, die die erhaltenen Nachrichten mit Sequenznummer und Absender speichert. Mit diesen Informationen kann der Abgleich durchgeführt werden.

3.2.5.3 Lokalität von gossip

Dass Kommunikation mit nahen Knoten billiger ist als mit weiter entfernten, ist einsehbar, da ein geringerer Kommunikationsaufwand betrieben werden muss. Aus diesem Grund kann man sagen, dass sich *gossip* auf lokale Bereiche beschränken sollte. Die Kommunikation allerdings so stark einzuschränken, hätte einen negativen Effekt, da Paketverlust vor allem innerhalb eines lokalen Bereiches auftritt. Um einerseits Paketverlust vorzubeugen und andererseits das Netzwerk nicht unnötig zu belasten, muss gewährleistet werden, dass mit großer Wahrscheinlichkeit mit direkten Nachbarn und mit ausreichender Wahrscheinlichkeit mit entfernten Knoten kommuniziert wird.
Damit dies zu gewährleisten ist, wird der *next hop* Eintrag in der MRT des MAODV um die Hopanzahl *(nearest member)* zu einem weiteren Knoten ergänzt. Wird von einem Knoten eine *gossip message* empfangen, so wird mit großer Wahrscheinlichkeit ein solcher *next hop* Nachbar gesucht, der einen kleinen *nearest member* Wert besitzt, und nur manchmal ein entfernter Knoten gewählt.
An dieser Stelle sei bemerkt, dass natürlich jedes Neueintreten, Verlassen oder ein Ortswechsel eines Knotens eine Korrektur des *nearest member* Eintrages zur Folge hat. Indikatoren dafür sind die aus dem *MAODV* resultierenden Nachrichten, die Knoten senden, um topologische Änderungen anzuzeigen (*MACT messages, prune messages)*.

3.2.5.4 Cached gossip

In Ad-hoc-Netzen mit sehr eng beieinander liegenden Knoten kann es vorkommen, dass die *gossip* Nachricht von A nach B einen längeren Weg durch den *multicast tree* nimmt, als der kürzeste Weg betragen würde. Da aber der Retourpfad *unicast* zur Zustellung seiner Nachrichten und damit den kürzeren Weg benutzt, kann diese Information für weitere *gossips* genutzt werden. Zusätzlich ergibt sich der Vorteil, dass diese *unicast* Wege auch in Funktion sind, wenn der *multicast tree* gerade repariert wird.
Um dies zu realisieren, wird in jedem Knoten des Netzes ein *member cache* eingerichtet. Bei jedem *gossip* Versuch muss mit einer gewissen Wahrscheinlichkeit bestimmt werden, ob mittels Standard-*Gossip* vorgegangen wird oder ob der Algorithmus einen zufälligen Knoten aus dem *gossip cache* nimmt und *unicast* seinen Partner sucht.

3.2.6 Übersicht über weitere *Gossip* Protokolle

In diesem Abschnitt sind zwei weitere erwähnenswerte Systeme, die auf *gossip* basieren, kurz umrissen:

3.2.6.1 Astrolabe

Die verwendeten Informationen wurden aus [RBDV02] entnommen.
Die Idee von Astrolabe ist die Speicherung und die Aktualisierung einer dynamischen Datenstruktur zur Verwaltung von Informationen. Diese Daten werden auf eine große Menge von Rechnern verteilt. Die Datenstruktur ist ein *Distributed Shared Memory*, das als hierarchische Tabelle dargestellt wird. Die Kommunikation basiert dabei auf *BM*.

3.2.6.2 Gravitational Gossip

Dieses Protokoll stellt eine Variation von *BM* dar, das zum Sammeln von Sensorwerten eingesetzt werden kann.
Mit diesem Protokoll ist es möglich, eine große Anzahl von Subgruppen zu realisieren. Einzelne Knoten des Netzes können dabei so konfiguriert werden, dass sie nur eine bestimmte Menge an Daten erhalten (100%, 50%, 20%, …). Diese Konfigurationsmöglichkeit wird als Gravitation bezeichnet.

3.2.7 Der Zusammenhang zwischen *Gossip* und *Flooding*

Dieser Abschnitt dient zur Herstellung der Verbindung zwischen gossip und flooding und basiert auf dem Artikel [LIHHun]. Um den Vergleich durchzuführen, wird ein einfaches *gossip* Protokoll (*GOSSIP1(p)*) verwendet:

- Ein Sender schickt eine Anfrage nach einem Weg mit der Wahrscheinlichkeit 1.
- Falls die Anfrage zum ersten Mal beim Empfänger eintrifft, dann:
  - Entweder wird die Nachricht mit einer Wahrscheinlichkeit *p* an den Rest der Gruppe verteilt, oder mit einer Wahrscheinlichkeit von *(1-p)* verworfen.
- Ansonsten wurde die Nachricht schon einmal empfangen.

Dieser ursprüngliche Algorithmus hat allerdings bei wenigen Nachbarn rund um die Quelle das Problem, dass eine Verhungerung am Anfang des *gossips* auftreten kann.
Daher wird diese Variante erweitert, indem innerhalb der ersten *k Hops* nach der Quelle immer noch mit einer Wahrscheinlichkeit von 1 gesendet. Dieses Verfahren wird mit *GOSSIP1(p,k)* bezeichnet. Die Leistung des Algorithmus hängt dabei wesentlich von den Parametern *p* und *k* ab.
Betrachtet man bestimmte Sonderfälle der *gossip*-Erweiterung, so kann man bestimmte Spezialfälle ausmachen:

- *GOSSIP1(p,0)* entspricht dem *GOSSIP1(p)* und
- *GOSSIP1(1,1)* ist äquivalent zu *flooding* und entspricht dem *GOSSIP1(1)*.

Konfiguriert man *GOSSIP(p,k)* mit *p* ausreichend hoch, um alle Knoten zu erreichen, kann man sich also (*1-p*) Nachrichten ersparen und trotzdem alle Knoten erreichen. In den meisten Fällen liegt *p* zwischen 65-75%, um eine Nachricht mit Sicherheit an alle Kommunikationspartner verteilen zu können, was bedeutet, dass man zwischen 25-35% weniger Nachrichten als bei *flooding* ausgesendet hat.

3.2.8 Heuristische Leistungsverbesserungen von *Gossip* Protokollen

An dieser Stelle werden weitere Optimierungen von *gossip* besprochen, die allerdings nicht auf ein spezielles Verfahren abzielen, sondern auf dem Algorithmus, der im Kapitel „3.2.7 Der Zusammenhang zwischen Gossip und Flooding“ eingeführt wurde, basieren und aus [LIHHun] stammen.

3.2.8.1 Ein System mit 2 Schwellenwerten:

Auch wenn mit *GOSSIP1(p,k)* erreicht werden kann, dass die Kommunikation am Anfang nicht ausstirbt, bleibt trotzdem das Problem bestehen, dass innerhalb eines Netzes ein Engpass zwischen zwei Teilnetzen bestehen kann, über den die *gossip message* möglicherweise nicht hinweg gelangt. Diese Variante heißt *GOSSIP2(p1,k,p2,n)*, wobei *p1<p2* und *p1* mit *k* aus dem vorher erklärtem *gossip* herrührt. Als Erweiterung gibt es die Parameter *p2* und *n*, die besagen, dass ein Knoten, wenn er weniger als *n* Nachbarn besitzt, mit einer Wahrscheinlichkeit von *p2* sendet.

GOSSIP2 ist in regulären Netzen von wenig Interesse, kann jedoch in beispielsweise Ad-hoc-Netzen einen positiven Einfluss haben, da dort leicht Engpässe entstehen können.

3.2.8.2 Verhindern von zu frühem Aussterben von gossip messages

Bei diesem Ansatz kennt ein bestimmter Knoten seine Nachbarn *n* und die Wahrscheinlichkeit *p*, mit der eine Nachricht weitergesendet wird. Mit dieser Information kann *pn*, also die Anzahl der Nachrichten, die ein Knoten erhalten sollte, errechnet werden. Werden also viel weniger als *pn* Nachrichten innerhalb einer bestimmten Zeit erhalten, so wird angenommen, dass die Nachricht ausstirbt, und der Knoten selbst schickt sie an alle seine Nachbarn weiter. Dieser Algorithmus wird mit *GOSSIP(p,k,m)* bezeichnet, wobei *m* die Anzahl von Nachbarn darstellt, von denen eine Nachricht erwartet wird. Wenn die Nachricht nicht *m*-mal erhalten wird, dann sendet der Knoten selbst die Nachricht weiter. Im Vergleich zu *GOSSIP1(p,k)* kann damit eine Leistungssteigerung von 8% erzielt werden. (vgl. „3.1.4.2 Counter-Based Scheme“)

3.2.8.3 Neuerliche Versuche

Wenn in Netzen, die *gossip* benutzen, ein Weg gefunden wurde, so wird immer diese Route zwischen zwei Knoten genutzt. Es kann aber sein, dass ein solcher Weg nicht gefunden wird, auch wenn er vorhanden ist. Die Lösung dafür ist ein neuerlicher Versand der *gossip message*, um den Weg von Sender zum Empfänger zu finden.
Mit diesem Ansatz erhöht sich zwar die Anzahl der Nachrichten nicht dramatisch, es entsteht aber vor allem in großen Netzwerken (durch lange *timeout*-Intervalle) eine bedeutende Verzögerung.
Man kann davon ausgehen, dass, falls fast alle Knoten im Netz die Nachricht erhalten haben, auch der gesuchte Knoten darunter ist, daher ein neuerlicher Versand nicht nötig ist. Falls aber nur wenige Knoten die Nachricht erhalten, ist ein neuerliches Versenden notwendig.
Um auch in großen Netzen zu wissen, ob fast alle Knoten die Nachricht erhalten haben, wird ein *hop count* verwendet, den jede Nachricht mit sich trägt. Durchläuft eine Nachricht das Netzwerk, wird an bestimmten Stellen (z.B. 15 *hops*) eine Nachricht mit einer bestimmten Wahrscheinlichkeit zum Sender geschickt, der wiederum diese Nachrichten zählt und eine Entscheidung über die Netzabdeckung dieser Nachricht treffen kann.

3.2.8.4 Zonen

Jeder Knoten bildet in unmittelbarer Umgebung eine Zone von Nachbarn, die er mittels *flooding* herausfindet, diese aber danach kennt. Wird später eine Nachricht an Zonenmitglieder verschickt, so kann das in direkter Weise geschehen. Jede Zone hat

Randknoten, die dazu dienen, *routing* Funktionen zu übernehmen, wenn Nachrichten an Teilnehmer gesendet werden, die nicht in der eigenen Zone liegen. Randknoten sehen dann nach, ob der Empfänger von ihnen aus zu erreichen ist oder nicht. Wenn ja, wird die Nachricht zugestellt, wenn nicht, wird über die Randknoten des Randknotens weitergesucht, usw. Die Auswirkungen sind weniger in großen, sondern vor allem bei kleinen Netzwerken spürbar. (vgl. „3.1.4.5 Fehler: Referenz nicht gefunden“)
Dieser Algorithmus wird mit *GOSSIP(p,k,k´)* bezeichnet, wobei *k´* dem Zonenradius entspricht.

Literaturverzeichnis

[EFGK03] Eugster, P.Th., Felber, P., Guerraoui, R., Kermarrec, A.-M.: The Many Faces of Publish/Subscribe, in: ACM Computing Surveys (CSUR), Volume 35, Issue 2 (June 2003), S. 114-131

[TaSt02] Tanenbaum, Andrew, S., van Steen, Maarten: Distributed Systems – Principles and Paradigms, Prentice Hall, 2002

[HuGa01] Huang, Yongqiang, Garcia-Molina, Hector: Publish/Subscribe in a Mobile Environment. In: Proceedings of the 2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE), 2001

[HuGa01b] Huang, Yongqiang, Garcia-Molina, Hector: Exactly-once semantics in a replicated messaging system. In: Proceedings of the 17th International Conference on Data Engineering, 2001

[HuGa03] Huang, Yongqiang, Garcia-Molina, Hector: Publish/Subscribe Tree Construction in Wireless Ad-hoc Networks. In: 4th International Conference on Mobile Data Management (MDM), 2003

[MuPR99] Murphy, A. L., Picco, G. P., Roman, G.-C.: LIME: Linda Meets Mobility. In: Proceedings of the 21st International Conference on Software Engineering, Los Angeles, California, p. 368-377, 1999

[MuPR03] Murphy, A. L., Picco, G. P., Roman, G.-C.: LIME: A Coordination Middleware Supporting Mobility of Hosts and Agents. Technical Report WUCSE-03-21, Washington University, Department of Computer Science and Engineering, St. Louis, Missouri, April 2003-11-24

[BHO+99] K.P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal Multicast," ACM Trans. Computer Systems, vol. 17, no. 2, pp. 41-88, May 1999.

[CHRB01] R. Chandra, V. Ramasubramanian, and K. Birman, “Anonymous gossip: Improving multicast reliability in mobile ad-hoc networks”, in Proc. 21st International Conference on Distributed Computing Systems (ICDCS), 2001, pp. 275-283.

[GKW+02] D. Ganesan, B. Krishnamurthy, A. Woo, D. Culler, D. Estrin, and S. Wicker. An empirical study of epidemic algorithms in large scale multihop wireless networks. http://www.cs.virginia.edu/~qc9b/cs851/ , 2002.

[KUCH01] Thomas Kunz and Ed Cheng, "Multicasting in ad-hoc networks: Comparing MAODV and ODMRP", *Proceedings of the Workshop on Ad hoc Communications*, Bonn, Germany, September 2001.

[LIHHun] L. Li, J. Halpern, Z. J. Haas, “Gossip-based Ad Hoc Routing”, http://www.cs.cornell.edu/courses/cs735/2001fa/papers/LiLi-GspInfo02.ps, 2001.

[LIMM00] M.-J. Lin, K. Marzullo, and S. Masini. “Gossip versus deterministic flooding: Low message overhead and high-reliability for broadcasting on small networks.” In

Proceedings of 14th International Symposium on DIStributed Computing (DISC 2000, pages 253--267, Toledo, Spain, October 4-6 2000.

[NTSS99] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. In Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking, pages 151–162, Aug 1999.

[RBDV02] R. van Renesse, K. P. Birman, D. Dumitriu, and W. Vogel. Scalable management and data mining using Astrolabe. In Proc. First International Workshop on Peer-to-Peer Systems (IPTPS '01), Cambridge, MA, Mar. 2002.

[SACS03] Yoav Sasson and David Cavin and André Schiper. Probabilistic Broadcast for Flooding in Wireless Mobile Ad hoc Networks. In Proceedings of IEEE Wireless Communications and Networking Conference (WCNC 2003), 2003.

[VORB01] Werner Vogels, Robbert van Renesse, and Ken Birman, "Using Epidemic Techniques for Building Ultra-Scalable Reliable Communications Systems". Workshop on New Visions for Large-Scale Networks: Research and Applications, Vienna, VA, March 2001.

[AYRM03] Ayman El-Sayed, Vincent Roca and Laurent Mathy, "A survey of proposals for an alternative group communication service". IEEE Network, January/February 2003.

[1] G.-C. Roman, Q. Huang, and A. Hazemi, "On maintaining Group Membership Data in Ad Hoc Networks,"
Washington University, St Louis, Technical Report WUCS-00-26, April 16, 2000.

[2] R. Prakash and R. Baldoni, "Architecture for Group Communication in Mobile Systems," presented at Symposium on
Reliable Distributed Systems, West-Lafayette (IN), USA, 1998.

[3] Rene Meier, Marc-Oliver Killijian, Raymond Cunningham, and Vinny Cahill: "Towards Proximity Group Communication".- Dublin

[4] M.-O. Killijian, R. Cunningham, R. Meier, L. Mazare and V. Cahill, "Towards group communication for mobile
participants", Proceedings of the 1st ACM Workshop on Principles of Mobile Computing (POMC), August 29-30, 2001,
Newport, Rhode Island, USA.

[5] L. Keidar and D. Dolev, Dependable Network Computing, D. Avresky Editor, chapter 3: "Totally Ordered Broadcast
in the Face of Network Partitions. Exploiting Group Communication for Replication in Partitionable Networks", Academic
Publications.

[6] Ayman El-Sayed and Vincent Roca, INRIA Rhone-Alpes Laurent Mathy: "A Survey of Proposals for an Alternative Group Communication Service" .- Lancaster University

[7] Linda Briesemeister: “Group Membership and Communication in Highly Mobile Ad Hoc Networks”. – Berlin, 2001

[8] O. Babaoglu, R. Davoli and A. Montresor, "Group Communication in Partitionable Systems: Specification and
algorithms", IEEE Transactions on Software Engineering, 27(4): 308-336, Apr. 2001.

[9] Imrich Chlamtac, Jason Redi: “Mobile Computing: Challenges and Potential” .- Texas, Boston, 1998

[10] David Powell: “Group Communication” .- Toulouse

[11] Rene Meier: “Communication Paradigms for Mobile Computing” .- Dublin

[12] A. Fekete and N. L. Shvartsman. "Specifying and using a partitionable group communication service", ACM
Transactions on Computer Systems, 19(2): 171-216, May 2001.

[13] Flaviu Cristian: “Synchronous Group Communication” .- San Diego

[14] Gruia-Catalin Roman, Qingfeng Huang, Ali Hazemi: “Consistent Group Membership in Ad Hoc Networks”.- Washington

[15] G. V. Chockler, L. Keidar and R. Vitenberg, "Group Communication Specifications: A Comprehensive Study", To
appear in ACM Computing Surveys.

[55] D. Jungnickel, Graphen, Netzwerke und Algorithmen, BI 1994.

[66] Peterson, Larry L.: “Computernetze; ein modernes Lehrbuch / Larry L. Peterson; Bruce S. Davie .- 1.Aufl.” – Heidlberg : dpunkt-Verl., 2000

[77] Steinmetz, Ralf: Multimedia-Technologie: “Grundlagen, Komponenten und Systeme/Ralf Steinmetz“. –
3., überarb.Aufl. – Berlin, Heidelberg; New York; Barcelona; Hongkong; London; Mailand; Paris; Singapur; Tokio: Springer, 2000
ISBN 3-540-67332-6