Reihe: Telekommunikation @ Mediendienste · Band 14

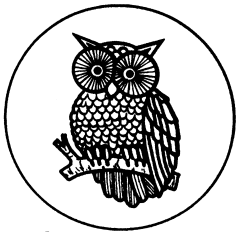
Herausgegeben von Prof. Dr. Dr. h. c. Norbert Szyperski, Köln, Prof. Dr. Udo Winand, Kassel, Prof. Dr. Dietrich Seibt, Köln, Prof. Dr. Rainer Kuhlen, Konstanz, Dr. Rudolf Pospischil, Brüssel, Prof. Dr. Claudia Löbbecke, Köln, und Prof. Dr. Christoph Zacharias, Köln

PD Dr.-Ing. habil. Martin Engelien
Dipl.-Inf. Jens Homann (Hrsg.)

Virtuelle Organisation und Neue Medien 2002

Workshop GeNeMe2002
Gemeinschaften in Neuen Medien

TU Dresden, 26. und 27. September 2002



JOSEF EUL VERLAG
Lohmar · Köln

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Virtuelle Organisation und Neue Medien 2002 / Workshop GeNeMe 2002 – Gemeinschaften in Neuen Medien – TU Dresden, 26. und 27. September 2002. Hrsg.: Martin Engelien ; Jens Homann. – Lohmar ; Köln : Eul, 2002
(Reihe: Telekommunikation und Medienwirtschaft ; Bd. 14)
ISBN 3-89936-007-9

Josef Eul Verlag GmbH
Brandsberg 6
53797 Lohmar
Tel.: 0 22 05 / 90 10 6-6
Fax: 0 22 05 / 90 10 6-88
http://www.eul-verlag.de
info@eul-verlag.de

Printed in Germany
Druck: RSP Köln

Bei der Herstellung unserer Bücher möchten wir die Umwelt schonen. Dieses Buch ist daher auf säurefreiem, 100% chlorfrei gebleichtem, alterungsbeständigem Papier nach DIN 6738 gedruckt.

Technische Universität Dresden
Fakultät Informatik • Institut für Angewandte Informatik
Privat-Dozentur Angewandte Informatik

PD Dr.-Ing. habil. Martin Engelien

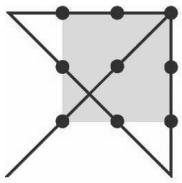
Dipl.-Inf. Jens Homann

(Hrsg.)

an der
Fakultät Informatik der Technischen Universität Dresden

in Zusammenarbeit mit der
Gesellschaft für Informatik e.V.,
GI-Regionalgruppe Dresden

gefördert von der Klaus Tschira Stiftung
gemeinnützige Gesellschaft mit beschränkter Haftung



am 26. und 27. September 2002
in Dresden
http://pdai.inf.tu-dresden.de/geneme
Kontakt: Thomas Müller (geneme@pdai.inf.tu-dresden.de)

B.2. Strukturbildung in P2P-Network-Communities

Markus Wulff

Dr. Herwig Unger

Fachbereich Informatik

Universität Rostock

1. Einleitung

Peer-to-Peer-Netzwerke (P2P) und -Communities sind in der Vergangenheit nicht nur durch die Popularität von File-Sharing Systemen wie Gnutella [4] oder Freenet [1] zum Gegenstand umfangreicherer Forschungen geworden. In einem P2P-System ist jeder Computer zugleich Anbieter und Konsument von Informationen aller Art. Es existiert kein zentraler Server mehr, der Nutzinformationen oder Informationen über die Netzwerkstruktur bereithält. Es können jederzeit Teilnehmer hinzukommen oder entfernt werden, ohne daß die Funktionalität des Gesamtsystems beeinträchtigt wird. In solchen Netzwerken finden sich Nutzer zusammen, die gleiche Interessen, wie z.B. den Austausch bestimmter Daten, haben (*Communities*). Trotz oder gerade wegen dieser dynamischen, dezentralen Architektur haben solche System einige signifikante Vorteile gegenüber den herkömmlichen Client-Server-Lösungen [5]. Eine zentrale Instanz ist z.B. eine Schwachstelle, wenn es um Sicherheit und Zuverlässigkeit geht. Nicht nur mögliche technische Probleme können mit dem Server alle von diesem angebotenen Dienste unerreichbar machen, sondern auch böswillige Angriffe von außen. Zentrale Datenbestände sind zudem meistens auch sehr umfangreich und somit nur mit großen Aufwand zu pflegen.

Aber auch P2P-Systeme haben einige Nachteile, die in ihrer Natur begründet sind. So hat u.a. kein Teilnehmer genaue Informationen über alle anderen Nutzer. Es sind jedem Einzelnen meistens nur wenige Nachbarrechner bekannt. Dieser Umstand erschwert im Gegensatz zum Client-Server-System, die Verbreitung und die Suche von Informationen im P2P-Netz. Trotzdem muß es möglich sein, daß jeder Teilnehmer alle verfügbaren Informationen finden und selbst Informationen an andere Nutzer in einer vertretbaren Zeit verteilen kann.

Zu diesem Problem sind in der Vergangenheit bereits Untersuchungen angestellt worden. PlanetP [2] z.B. baut auf jedem Knoten des Netzes lokale Kataloge auf, die durch „Weitersagen“ aktualisiert werden. Auch andere Lösungen setzen an dem zentralen Punkt Kommunikation an.

Die Kommunikation in P2P-Netzwerken erfolgt meistens über Nachrichten (Message Chains), die von einem Rechner zum nächsten weitergegeben werden. Wie schon in früheren Arbeiten gezeigt werden konnte, sind diese ein universelles und leistungsfähiges Werkzeug zur Kommunikation und Informationssuche in verteilten Systemen [6]. Es können aber auch durch eine gezielte Strukturierung von P2P-Netzen weitere Verbesserungen hinsichtlich des Informationsaustausches und der Informationssuche erzielt werden.

P2P-Netze sind zumeist relativ unstrukturiert und besitzen oft eine sog. “small world” Struktur [3]. Das heißt im wesentlichen, daß die Verbindungen zu Knoten in der direkten Nachbarschaft dichter sind als die zu entfernten Knoten. Der Vorteil ist der relativ kleine Durchmesser dieser Netzwerke. Es ist möglich, jeden Knoten mit nur einer, gegenüber der Anzahl der Knoten kleinen Anzahl von Schritten (Hops) zu erreichen. Durch eine gezielte Anpassung der gespeicherten Links zu den Nachbarrechnern kann relativ einfach eine Struktur aufgebaut werden, auf deren Basis dann effiziente Algorithmen zur Kommunikation und Informationssuche eingesetzt werden können.

Im Folgenden werden zwei verschiedene Verfahren zur dynamischen Strukturbildung in dezentralen P2P-Netzwerken vorgestellt und erste Simulationsergebnisse gezeigt, welche die Funktionsweise und Leistungsfähigkeit der Verfahren veranschaulichen.

2. Strukturbildung

Strukturen von P2P-Communities sollten möglichst ohne komplizierte, zeitraubende Algorithmen erstellt werden und sich an die sich relativ schnell ändernde Netzwerkstruktur anpassen können. Da in einem P2P-System keine globale Sicht auf das Gesamtsystem existiert, müssen die Strukturierungsalgorithmen so beschaffen sein, daß sie lokal auf jedem Knoten arbeiten und mit dem dort vorhandenen Wissen über andere Teilnehmer die Struktur aufbauen können. Der bereits vorgestellte Hypercube ist eine Möglichkeit zur Strukturbildung [7]. Es handelt sich dabei um eine relativ starre Struktur, die aber schon mittels eines völlig dezentralen, lokal arbeitenden Algorithmus aufgebaut und angepaßt werden kann. Änderungen in der Netzwerkstruktur kann die vorgestellte Hypercubestruktur dagegen nur unzureichend schnell folgen. Andere, flexiblere Algorithmen zur Strukturbildung in P2P-Netzwerk-Communities werden also benötigt, die den Anforderungen dieser dynamischen Netzwerke gerecht werden.

2.1 Ameisen

Die erste Möglichkeit ist die Strukturierung unter Verwendung von Algorithmen, die ein ameisenähnliches Verhalten nachbilden. Hierbei wird vor allem die Eigenschaft der Tiere genutzt, Wege mittels Duftstoffen (Pheromone) zu markieren und sich selbst an vorhandenen Duftpfaden zu orientieren.

Diese Strukturbildung geschieht ohne direkte Kommunikation zwischen den *Ameisen* (reaktive Strukturbildung), sondern ausschließlich über Umweltparameter (Pheromonkonzentration), wobei jedoch eine Anpassung insbesondere an Größe und Verkehr in der Community erfolgen kann. Im Folgenden wird gezeigt, wie eine völlig dezentrale, auf Kooperation beruhende Suchmaschine auf dieser Basis realisiert werden kann. Diese kann die Zeit für eine Suche nach benötigten Informationen in der Community reduzieren, falls mehr als eine Maschine eine bestimmte Information suchen. Dabei verfolgen die Ameisen zu verschiedenen Zeitpunkten unterschiedliche Strategien. Eine Informationssuche erfolgt durch parallel arbeitende Ameisen, während eine Konzentration von Informationen und eine Abfrage des gesammelten Wissens entlang eines automatisch gebildeten Kreises hoher Duftstoffkonzentration erfolgt.

Auch wenn nicht wirklich mit Ameisen gearbeitet wird und auch hier eigentlich Wanderer genutzt werden, soll dieser Begriff beibehalten werden, um eine bessere Abgrenzung zu dem in Abschnitt 2.2 vorgestellten Konzepten zu erreichen.

Die im folgenden gezeigten Abbildungen wurden mit Hilfe von Simulationsergebnissen erstellt. Dabei wurde das beschriebene Verhalten der Ameisen in einem P2P-System mit 2048 Knoten nachgebildet. Immer wenn eine Ameise einem Pfad folgt, verstärkt sie die Pheromonkonzentration. Wird ein Pfad nicht genutzt, zerfallen die Duftstoffe nach und nach. Diese Abnahme der Pheromonkonzentration über der Zeit folgt in der Simulation einer exponentiellen Funktion.

Um in einem P2P-System Informationen zu suchen, ist es notwendig, alle Knoten zu besuchen. Dies kann man erreichen, indem jeder Knoten allen ihm bekannten Nachbarn eine Anfrage sendet. Dies führt aber in größeren Systemen zu einer hohen Netzlast. Zudem muß berücksichtigt werden, daß mehrere Teilnehmer gleichzeitig Suchanfragen abschicken können. Eine mögliche Alternative sind *Minimum-Ameisen*. Diese folgen nicht dem Pfad mit der höchsten Konzentration an Duftstoffen, sondern dem mit der niedrigsten. Dadurch weichen die Ameisen einander aus und besuchen so Knoten, die vorher noch nicht besucht worden sind. Es ergibt sich das in Abbildung 1 gezeigte Bild.

Die Ameisen verteilen sich gleichmäßig über die gesamte Community und können so alle verfügbaren Informationen sammeln. Nun müssen die gefundenen Informationen an bestimmten Punkten abgelegt werden, um den Zugriff zu beschleunigen.

Bewegen sich die Ameisen entlang dem Pfad mit der höchsten Konzentration an Pheromonen, bildet sich nach relativ kurzer Zeit ein stabiler zyklischer Pfad heraus, dem alle Ameisen folgen. Diese Ameisen sollen *Maximum-Ameisen* genannt werden.

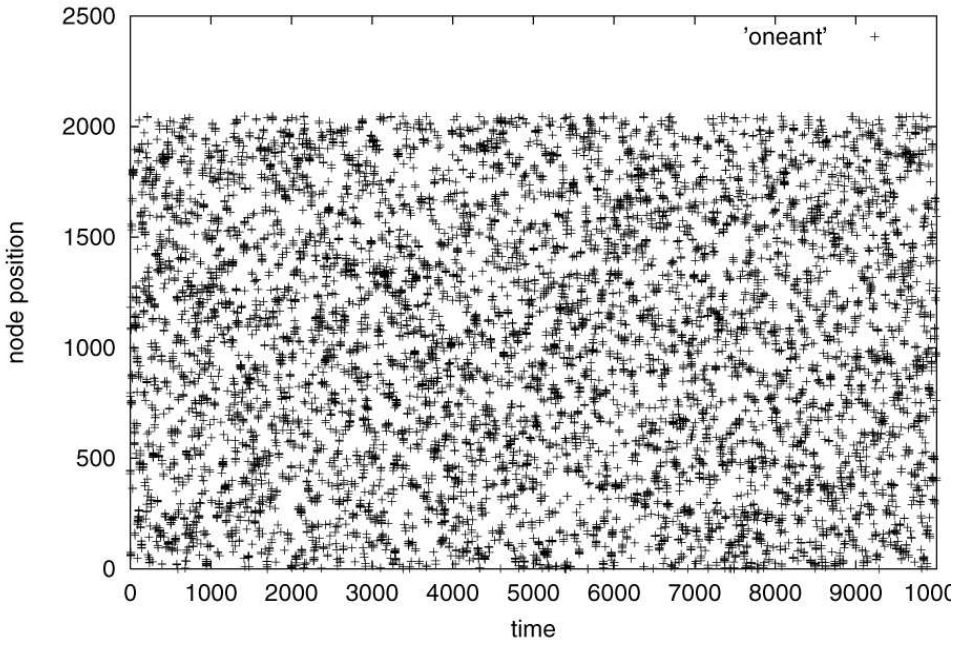


Abb. 1: Minimum-Ameisen folgen dem Pfad mit der niedrigsten Pheromonkonzentration.

Da sich auf diese Art und Weise die Netzlast konzentriert, würde dieses Vorgehen nur für kleine Systeme mit wenigen Knoten anwendbar sein. Führt man einen Maximalwert für die Konzentration ein, bis zu dem die Ameisen dem Pfad nur folgen dürfen, entstehen mehrere solcher Kreispfade, was in Abbildung 2 gezeigt ist. Durch den Einsatz dieser *Maximum-2-Ameisen* wird das System skalierbar und somit anwendbar für große Communities.

Damit ist es nun möglich, die von den Minimum-Ameisen gefundenen Informationen auf den Knoten entlang eines starken Pheromon-Pfades zu verteilen. Diese Knoten fungieren hier als Informations-Cache.

Es ist leicht einzusehen, daß es günstig ist, wenn eine Ameise beide Strategien beherrscht. Die *MinMax-Ameise* kann sowohl als Maximum-Ameise als auch als Minimum-Ameise arbeiten und wechselt periodisch zwischen den beiden Strategien. In der Abbildung 3 ist dieses Verhalten zu erkennen.

Nach einer bestimmten (festgelegten) Periode, in der die Ameise Informationen gesammelt hat, beginnt sie, die Daten auf ausgewählten Knoten entlang eines Pfades mit hoher Konzentration an Duftstoffen zu verteilen. Es ist in der Abbildung zu sehen, daß sich dieser Pfad erst langsam herausbildet, dann aber nach einiger Zeit stabil wird. Nun kann er auch von anderen Ameisen erkannt und genutzt werden.

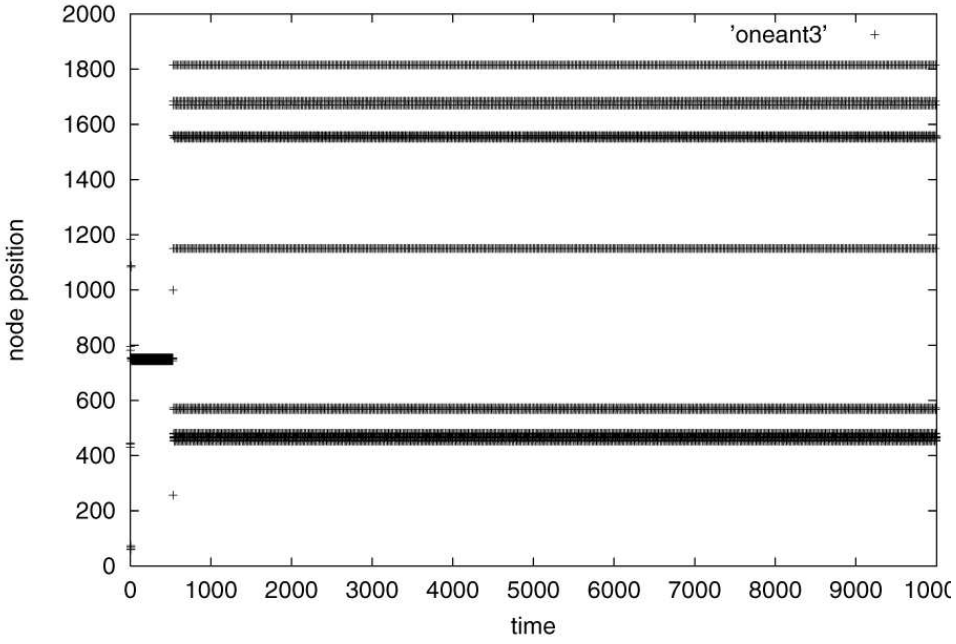


Abb. 2: Maximum-2-Ameisen folgen dem Pfad mit der höchsten Pheromonkonzentration, bis diese einen bestimmten Maximalwert erreicht hat.

Die Suche nach Daten unter Nutzung der beschriebenen Ameisen geschieht nun wie folgt: Wenn ein Knoten irgendwelche Informationen benötigt, generiert er eine neue Ameise. Diese verfolgt nun ersteinmal die Maximum-2-Ameisen Strategie. So werden bereits vorhandene Pfade mit hoher Pheromonkonzentration gefunden. Auf Informationen, die diesem Pfad entlang abgelegt sind, kann nun schnell zugegriffen werden. War die Suche bereits erfolgreich, werden die gefundenen Informationen direkt zu dem Knoten gebracht, der die Suche ausgelöst hat.

Wurden die gesuchten Informationen in diesem Cache nicht gefunden, folgt die Ameise dem Weg mit der geringsten Pheromonkonzentration (Minimum-Ameise). Nun wird die gesamte Community nach den entsprechenden Daten abgesucht. Durch die indirekte Kommunikation über die Pheromone wird erreicht, daß parallel arbeitende Ameisen gleichmäßig über die Community verteilt werden. Die Dauer dieser Suchperiode wird durch die Kapazität der Ameise und eine eingebaute Zeitbegrenzung limitiert.

Nach dieser Suche werden die gefundenen Informationen zum einen in dem durch den starken Pheromonpfad gebildeten Cache abgelegt und natürlich zu dem anfragenden Knoten gebracht.

Der Pfad mit der hohen Pheromonkonzentration ist ein gemeinsames soziales Gedächtnis für die ganze Community. Informationen, die häufig von verschiedenen Knoten gesucht werden, können hier mit hoher Wahrscheinlichkeit schnell gefunden werden. Informationen, die selten gesucht werden, sind wahrscheinlich nicht in dem Cache gespeichert und müssen wie beschrieben gesucht werden.

Die Simulationen haben gezeigt, daß ein solcher Pfad hoher Pheromonkonzentration relativ schnell aufgebaut wird und stabil ist. Auf Veränderung in P2P-Netzwerk reagiert dieses System tolerant. Der Ausfall von einzelnen Knoten oder das Hinzukommen neuer Knoten wird zuverlässig ausgeglichen. Das gesamte System der Duftpfade ist ohnehin dynamisch und muß von den Ameisen ständig erneuert werden.

Im Folgenden wird nun noch eine weitere Strategie zur Strukturbildung in P2P-Netzwerken vorgestellt, die nicht durch ein reaktives Vorgehen realisiert wird.

2.2 Wanderer

Der zweite Ansatz zur Strukturierung von Communities beruht auf dem Einsatz planender *Wanderer*. Wanderer sind spezielle Nachrichten, die ständig zwischen den Knoten zirkulieren. Sie enthalten keinen Programmcode. Zusätzlich zu der Nutzinformation transportieren sie Meta-Daten, die von den Knoten zur Steuerung der Wanderer verwendet werden.

Wanderer haben hier zum einen die gleiche Hauptaufgabe wie die Ameisen, nämlich in einem bestimmten Gebiet Informationen zu sammeln, dienen zum zweiten aber auch dem Zwecke der Gebietsaufteilung nach einem deterministischen Algorithmus. Der wesentliche Unterschied hierbei ist das kognitive Vorgehen der Wanderer; d.h. sie

besitzen einen Plan, nach dem sie sich im Netzwerk bewegen und der den jeweiligen Erfordernissen entsprechend angepaßt werden kann. Hierbei werden im Folgenden zwei Strategien unterschieden und diskutiert.

Die Anzahl der von einem Wanderer besuchten Knoten richtet sich nach den zeitlichen Anforderungen der Knoten. Es muß gewährleistet sein, daß jeder Knoten innerhalb eines von ihm bestimmten Zeitintervalls von einem Wanderer besucht wird.

2.2.1 Strukturbildung mit überlappenden Zyklen

Der erste Algorithmus erzeugt eine Cluster-ähnliche Struktur, bei der die jeweiligen Cluster nicht disjunkt sind. *Cluster* soll hier die Menge von Knoten genannt werden, die von *einem* Wanderer besucht werden. Die Knoten, die in mehr als einem Cluster enthalten sind, werden für den Informationsaustausch zwischen den Clustern benutzt. Der Austausch von Informationen ist notwendig, um eine Kommunikation zwischen allen Mitgliedern der Community zu ermöglichen.

Die Wanderer haben einen Plan, der ihren Weg, also die zu besuchenden Knoten, beinhaltet. Dieser kann von jedem Knoten geändert werden, damit er den Anforderungen des jeweiligen Knotens entspricht.

Im Folgenden wird der Algorithmus im Detail beschrieben.

Ein Knoten i benötigt einen Wanderer für sein Informationsmanagement und wartet einen bestimmte Zeit. Danach kann er einen eigenen Wanderer generieren, wenn dies nicht schon vorher geschehen ist. Jeder Knoten darf nur maximal einen eigenen Wanderer besitzen. Das Zeitintervall, in dem ein Wandererbesuch erwartet wird, hat die Länge $t_{avg,i}$ mit einer maximalen Abweichung Δt.

Der Plan eines neuen Wanderers enthält zu Beginn nur die Daten des „Heimatknotens“. Alle weiteren Planpositionen sind leer (Eintrag NIL). Folgende Laufzeitdaten sind weiterhin in einem Wanderer enthalten:

t_{cycle}: die mittlere Zeit, die der Wanderer für die Abarbeitung des gesamten Plans benötigt. Diese wird aus den letzten r Durchläufen ermittelt.

p: die momentane Position im Plan.

q: die Gesamtlänge des Plans (derzeit ein konstanter Wert); Die mittlere Hop-Dauer t_{hop} kann nun mit t_{cycle}/q berechnet werden.

Der Wanderer läuft nun durch das Netzwerk, indem er seinem Plan folgt. Das bedeutet, der Knoten an Position *p* sendet den Wanderer zu dem Knoten, der an Position *p*+1 im Plan des Wanderers verzeichnet ist. Wenn die Position *p*+1 eine NIL-Position ist, erfolgt die Weiterleitung zufällig. Der Wanderer wird an irgendeinen, dem aktuellen Knoten bekannten, Knoten gesandt. Eine bestimmte Anzahl *e* an Planpositionen kann reserviert sein und darf damit nicht von den besuchten Knoten verändert werden. Diese Positionen werden wie NIL-Positionen behandelt und ermöglichen es dem Wanderer, das Netzwerk außerhalb seines Weges zu erkunden.

Ein Knoten kann folgende Operationen auf einem Wanderer ausführen:

FORWARD:

Der Wanderer wird entsprechend dem oben beschriebenen Mechanismus, also abhängig vom Inhalt der Planposition *p*+1, zum nächsten Knoten gesendet. Jeder Besuch eines Wanderers setzt den Zähler *t*, der den Abstand zwischen zwei Wandererbesuchen auf einem Knoten speichert, auf 0 zurück.

KILL_OWN:

Ein Knoten kann, wenn die mittlere Zeit *t* zwischen den letzten *r* Besuchen kleiner ist als $t_{kill,i}$, den eigenen Wanderer wieder aus dem System entfernen.

SET_REQ:

Wenn ein Knoten nicht oft genug besucht wird, also die mittlere Zeit *t* zwischen den letzten *r* Besuchen größer ist als $t_{request,i}$, wird das REQUEST-Flag gesetzt. Wenn REQUEST gesetzt ist, versucht der Knoten den Plan des nächsten Wanderers so zu verändern, daß dessen Besuchsintervall verkürzt wird. Hierzu wird die Operation CHECK_PLAN(*ptr) benutzt. Ist der aktuelle Knoten dagegen noch nicht in dem Plan des Wanderers enthalten, wird er mittels ADD_TO_PLAN(l) eingefügt. Waren die beschriebenen Operationen innerhalb von $t_{try,i}$ nicht erfolgreich, wird mit Hilfe von GENERATE_OWN ein eigener Wanderer erzeugt. In jedem Fall wird das REQUEST-Flag zurückgesetzt.

GENERATE_OWN:

Ein Knoten generiert einen eigenen Wanderer nach Ablauf der Zeit $t_{try.i}$, wenn er noch keinen eigenen Wanderer hat und das REQUEST-Flag gesetzt ist.

Es gilt $t_{kill,i} << t_{avg,i} << t_{request,i} < t_{try,i}$

ADD_TO_PLAN(l):

RMV_FROM_PLAN(l),

CHECK_PLAN(*ptr):

Mit diesen Funktionen kann der Plan eines Wanderers auf die gewünschte Art und Weise an gegebener Position *l* modifiziert werden, um die Besuchshäufigkeit zu steuern.

Zusätzlich sind natürlich weitere Operationen notwendig, die für die Bearbeitung der vom Wanderer transportierten Nutzdaten nötig sind.

Nach einer bestimmten Zeit wird die Anzahl der Wanderer stabil und die Pläne werden nicht mehr geändert. Die Cluster bleiben somit unverändert, bis sich die Netzwerktopologie oder die Netzwerklast ändern. In Abbildung 4 ist dies am Beispiel eines Wanderers dargestellt.

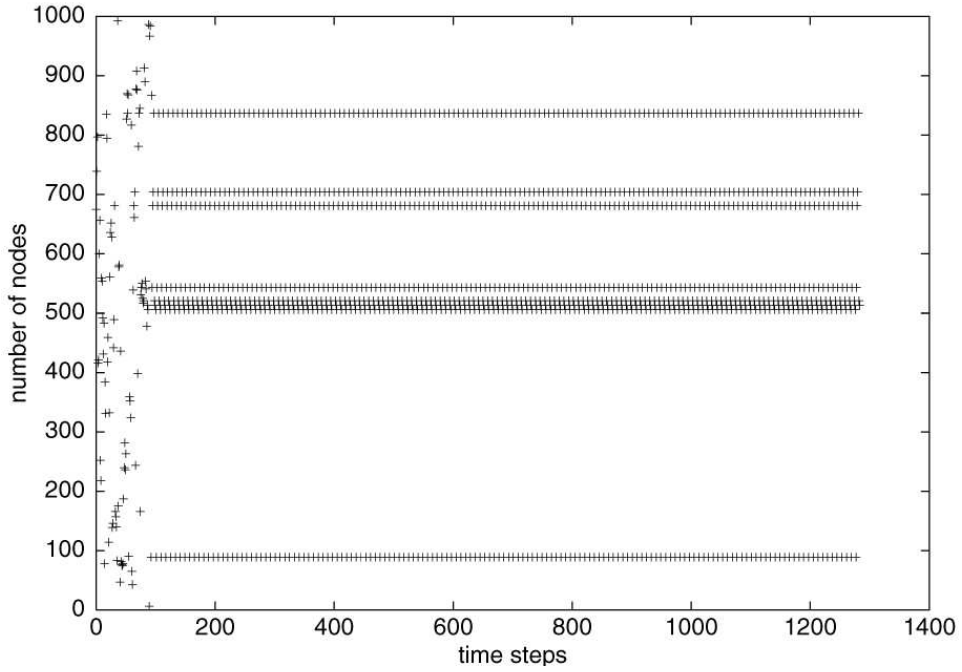


Abb. 4: Der Weg eines Wanderers über der Zeit.

Durch diesen ersten Wandereralgorithmus entstehen nicht-disjunkte Zyklen. Dadurch besteht die Möglichkeit für die Wanderer, an den Schnittpunkten der Wege Informationen auszutauschen, d.h. es entsteht unmittelbar eine Kooperation zwischen den Wanderern.

2.2.2 Strukturbildung mit disjunkten Kreisen

Der zweite Algorithmus arbeitet ähnlich wie der in Abschnitt 2.2.1 vorgestellte. Auch hier bewegen sich die Wanderer einem Plan folgend durch das Netzwerk. Die resultierenden Cluster und der Algorithmus zur Anpassung der Pläne unterscheiden sich jedoch grundlegend von dem oben beschriebenen. Jeder Knoten kann hier nur in den Plan eines einzigen Wanderers eingetragen werden. Dadurch entstehen komplett disjunkte Cluster und ein direkter Informationsaustausch zwischen den Clustern ist nicht mehr möglich. Deshalb werden Wanderer auf höherer Ebene eingesetzt, welche die Cluster der jeweils niedrigeren Ebene miteinander verbinden. Diese Wanderer arbeiten genau so wie die „normalen“ Wanderer, besuchen aber keine Knoten sondern Cluster. Der eigentliche Informationsaustausch findet auf einem bestimmten Knoten im Cluster statt.

Wie auch im vorhergehenden Algorithmus wird wieder davon ausgegangen, daß ein Knoten innerhalb des Zeitintervalls $t_{avg,i}$ mit einer maximalen Abweichung Δt von einem Wanderer besucht werden möchte. Dazu generiert dieser Knoten einen Level-0-Wanderer.

Die Wanderer für Ebene 0 und höhere Ebenen sind identisch. Jeder Level-*n*-Wanderer ist eine Nachricht, die einem Plan folgend, zwischen den Knoten im Netz zirkuliert. Der Plan enthält auch hier wieder eine Liste von zu besuchenden Knoten. Zu Beginn enthält der Plan nur den Eintrag des erzeugenden Knotens ($t_{avg,i}$). Desweiteren enthält jeder Wanderer folgende Laufzeitinformationen:

t_{cycle}: die mittlere Zeit, die für einen kompletten Plandurchlauf benötigt wird; berechnet aus den letzten *r* Durchläufen.

q: Anzahl der Knoten im Plan; die mittlere Hop-Dauer t_{hop} ergibt sich hieraus durch t_{cycle}/q

t_{run}: die bisher für den aktuellen Zyklus verbrauchte Zeit.

t_{change}: Zeitpunkt, zu dem der Plan des Wanderers zu letzten Mal geändert wurde.

Zusätzlich sind folgende Daten in einem Wanderer gespeichert:

n: Ebene (Level) des Wanderers

$t_{change,max}$: der maximale Wert für *tchange*

c, *d*, *k*: Konstanten zur Berechnung der Mittelwerte

r: Anzahl der Werte, aus denen ein Mittelwert berechnet wird

t_{min}: $t_{min} = min^{p}_{i=1} t_{avg,i}$

Wenn nach dem Abarbeiten des Plans $t_{run} < t_{min} - c \cdot t_{hop}$ gilt, wird der Wanderer solange zufällig weitergeleitet, bis $t_{run} = t_{min} - c \cdot t_{hop}$ erreicht ist.

Die Knoten, die von einem Wanderer besucht werden, können dessen Plan nicht verändern und werden auch nicht automatisch in dem Plan aufgenommen. Wenn sich zwei Wanderer auf einem Knoten treffen, werden ihre Pläne verschmolzen und nur ein Wanderer setzt den Weg fort. Dieser besucht nun auch die Knoten, die in dem Plan des anderen Wanderers enthalten sind. Wenn der Plan dadurch zu lang wird, d.h. der Wanderer nicht mehr in der Lage ist, die zeitlichen Anforderungen aller Knoten im Plan zu erfüllen, wird der Plan wieder geteilt. Dazu wird aber jeweils nur eine Planposition abgespalten und ein neuer Wanderer erzeugt, der diese als einzigen Eintrag in seinem Plan hat.

Die folgenden Operationen können auf einen Wanderer angewandt werden:

FORWARD:

Der Wanderer wird entsprechend seinem Plan bzw. zufällig weitergeleitet (siehe oben).

MERGE:

Wenn $t_{cycle} < t_{min} - c \cdot t_{hop}$ gilt, werden alle auf dem Knoten befindlichen Wanderer vereinigt.

DIVIDE:

Wenn nach *k* Durchläufen $t_{cycle} > t_{min} + c \cdot t_{hop}$ gilt, wird der Wanderer geteilt. Dazu wird die Planposition des Knotens abgespalten, der die kürzeste Zeit t_{avg} aufweist.

GENERATE:

Wenn nach $d \cdot t_{change,max}$ Durchläufen keine MERGE oder DIVIDE Operation auf dem Wanderer ausgeführt wurde und $q >= 2$, erzeugt dieser Level-*n*-Wanderer einen Level-(n+1)-Wanderer. Der neue Wanderer „gehört“ danach dem Knoten mit der längsten Zeit t_{avg}. Auf diesem Knoten erfolgt nun auch der Informationsaustausch zwischen den Ebenen *n* und *n*+1.

REMOVE:

Wenn eine MERGE oder DIVIDE Operation durchgeführt wurde, muß der betreffende Wanderer aus den Plänen der Wanderer der höheren Ebenen entfernt werden.

DIE:

Wenn der Plan eines Wanderers leer ist, wird der Wanderer entfernt.

Dieser Algorithmus erzeugt eine „atmende“, hierarchische Clusterstruktur, wenn das darunterliegende Netzwerk nicht stabil ist. Die Anzahl der Ebenen ändert sich, wenn sich die Anzahl der Knoten ändert.

Auch für diese Strategie wurden Simulationen durchgeführt, um das Verhalten dieses Algorithmus zu untersuchen. Es wurde eine Netzwerkgröße von 1024 bzw. 2048 Knoten angenommen. Abbildung 5 zeigt die Anzahl der Wanderer als Ergebnis der Simulation in einem stabilen Netzwerk.

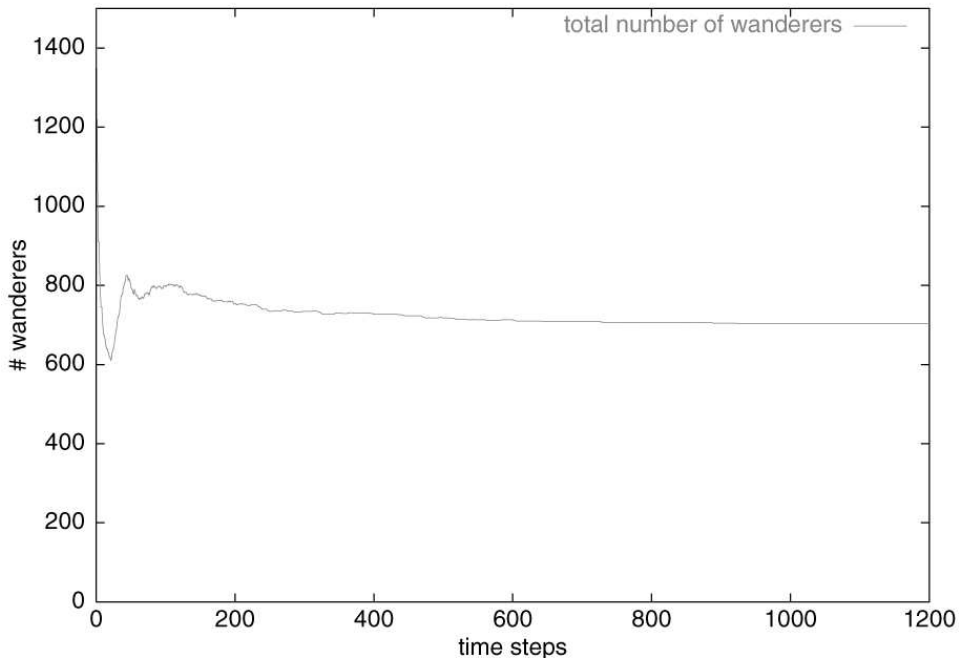


Abb. 5: Anzahl der Wanderer im System über der Zeit.

Zu Anfang erzeugt jeder Knoten einen eigenen Wanderer, wodurch sich viele Wanderer im System bewegen. Durch den Verschmelzungsprozeß nimmt die Zahl der Wanderer schnell ab. Nach einiger Zeit ist jedoch ein erneuter Anstieg der Wandererpopulation zu

verzeichnen. Dies ist dadurch zu erklären, daß nach einer Stabilisierung der Wanderer auf unterster Ebene der Aufbau der höheren Ebenen beginnt, also neue Wanderer erzeugt werden. Auch hier ist nach einiger Zeit wieder eine Abnahme der Wanderer zu beobachten.

Der Verlauf des Aufbaus der Hierarchie ist in Abbildung 6 dargestellt. Zur besseren Übersicht wurde die *y*-Achse logarithmisch geteilt, da auch die Anzahl der Wanderer auf der jeweils höheren Ebene etwa in diesem Maße abnimmt. Die Anzahl der Wanderer auf der nächst höheren Ebene ist Abhängig von der Plangröße der Wanderer dieser Ebene.

In dem Diagramm ist gut zu erkennen, wie die einzelnen Ebenen nacheinander aufgebaut werden. Es ist leicht einzusehen, daß die Anzahl der Wanderer auf den höheren Ebenen stark abnimmt. Auf der letzten Ebene bleibt in den allermeisten Fällen nur ein Wanderer übrig. Dieser wird von der darunterliegenden Ebene erzeugt und findet nun keinen Partner zum verschmelzen. Dies würde sich ändern, wenn in dem darunterliegenden P2P-Netzwerk Knoten hinzukommen würden.

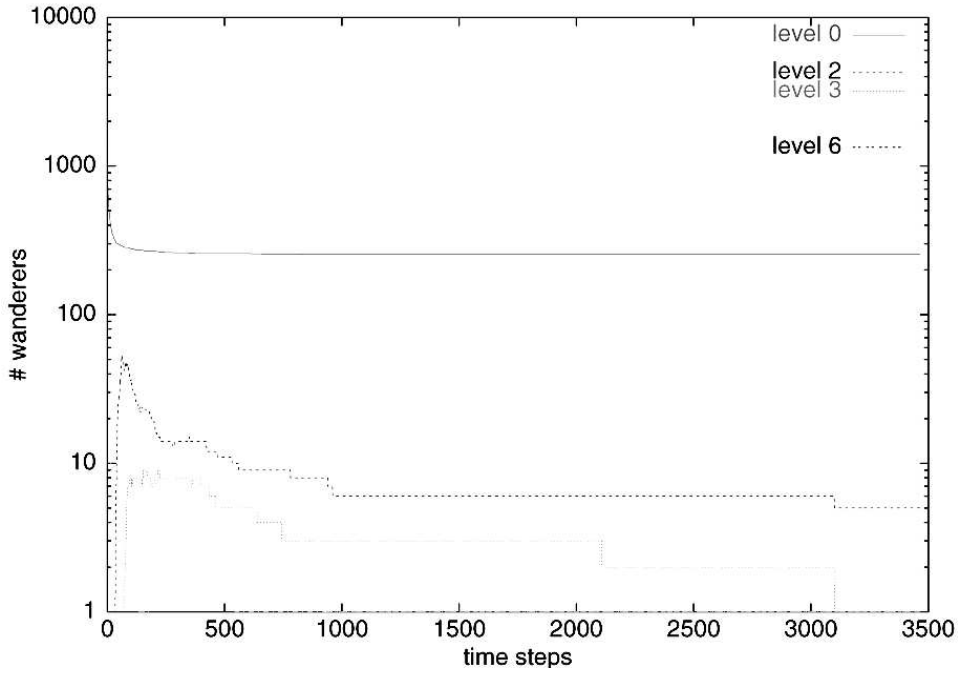


Abb. 6: Aufbau der einzelnen Ebenen über der Zeit.

3. Zusammenfassung und Ausblick

In diesem Artikel wurden unterschiedliche Ansätze zur Strukturierung von P2P-Netzwerken vorgestellt. Für diese Algorithmen gibt es verschiedene Einsatzmöglichkeiten. Im Fall der Ameisen wurde mit der dezentralen Suchmaschine bereits eine Anwendung diskutiert. Die Clusterstruktur, die im zweiten Teil beschrieben wurden, könnte die Grundlage zur Gewährleistung eines bestimmten Grades an Qualität (QoS) bezüglich der Versorgung mit Informationen in einer bestimmten Zeit sein.

Das Hauptziel ist es, durch die Strukturierung von P2P-Netzwerken deren Skalierbarkeit zu verbessern. Nach Auffassung der Autoren werden solche Netzwerke in Zukunft verstärkt für unterschiedlichste Anwendungen eingesetzt werden. Mit ihrer Flexibilität und Fehlertoleranz, z.B., besitzen P2P-Netzwerke wichtige Eigenschaften, mit denen sie für viele Einsatzgebiete klare Vorteile gegenüber herkömmlichen Systemen aufweisen. Dazu ist es allerdings notwendig, daß P2P-Netze besser beherrschbar werden und zu einem gewissen Grad zu administrieren sind. Als zentrales Problem steht hier, ein zuverlässiges Informationsmanagement zu gewährleisten, d.h. die Möglichkeiten zu Suche und Verbreitung von Informationen zu verbessern.

Weitere Arbeiten auf diesem Gebiet sind auf die nähere Untersuchung des Verhaltens der Verfahren in dynamischen und vor allem realen Umgebungen gerichtet. Insbesondere Möglichkeiten zur praktischen Anwendung derartiger Algorithmen müssen gefunden und hinsichtlich ihrer Eignung für die Realisierung mit Hilfe von P2P-Netzwerken untersucht werden.

4. Literatur

[1] Clarke, I., O. Sandberg, B. Wiley und T. Hong: *Freenet: A Distributed Anonymous Information Storage and Retrieval System*. In: *ICSI Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, 2000.

[2] Cuenca-Acuna, F. M., C. Peery, R. P. Martin und T. D. Nguyen: *PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities*. Techn. Ber. DCS-TR-487, Department of Computer Science, Rutgers University, Mai 2002.

[3] Deo, N. und P. Gupta:*WorldWideWeb: A Graph Theoretic Approach*. CS TR-01-001, University of Central Florida, 2001.

[4] Gnutella. www.gnutella.com, 2002.

[5] Unger, H., H. Unger und N. Titova: *Structuring of Decntralized Computer Communities*. In: *High Performance Computing 2002 (HPC 2002)*, S. 245–250, San Diego, CA, USA, Apr. 2002.

[6] Wulff, M. und H. Unger: *Message Chains as a New Form of Active Communication in the WOSNet*. In: Tentner, A. (Hrsg.): *ASTC High Performance Computing*, Washington, 2000.

[7] Wulff, M. und H. Unger: *Adaptive Datenverwaltung im Internet*. In: Engelien, M. und J. Homann (Hrsg.): *Virtuelle Organisation und Neue Medien 2001 (GeNeMe2001)*, Sep. 2001.