Optimierung der Kommunikation in einer Peer-to-Peer Echtzeitanwendung

Micha Trautweiler, tmicha@student.ethz.ch

24. August 2003

1 Zusammenfassung

Ausgehend von PeerBlast [1], einem rasanten P2P Spiel, beschreiben wir zwei Verbesserungen der Kommunikationsarchitektur der unterliegenden P2P Plattform. Einerseits das Gitter als effiziente Organisationform der Peers, andererseits Multicast zur Ausnutzung der Topologie des physischen Netzes (z.B. Ethernet). Diese beiden Massnahmen verringern die Nachrichtenkomplexität drastisch, wodurch der Cluster erheblich besser skaliert.

2 Einleitung

Dieser Bericht entstand im Rahmen der Semesterarbeit mit dem Titel "P2P Communication" am Institut für Pervasive Computing der ETH Zürich. Ziel war es, einfache Ergänzungen zu realisieren, die die Skalierung der Kommunikationsarchitektur verbessern.

In der Laborarbeit PeerBlast [1] wurde das Spiel Bomberman [2] (auch als XBlast bekannt) gewählt, um die Anforderungen und Probleme einer Echtzeitanwendung in einer Peer-to-Peer Architektur kennen zu lernen. In XBlast steuern die Spieler von ihren Rechnern aus ihren lokalen Spieler über das gemeinsame Spielfeld. Aktionen von Spielern anderer Peers werden im P2P Netz verteilt, auf dem lokalen Peer nachgerechnet und in Echtzeit dargestellt.

Die Organisationsform der Peers im Cluster, das logische P2P Netz (nachfolgend Topologie genannt) wurde dabei als kompletter Graph realisiert. Jeder Peer kennt also jeden und kann jedem direkt Nachrichten zukommen lassen.

Die plattformunabhängige Implementation erfolgte in vier funktional getrennten Schichten: Der Benutzerschnittstelle, der Spiellogik, dem P2P Cluster sowie die Netzverbindungen (TCP-Sockets). Letztere zwei Schichten wurden als Serviceschnittstellen designt, die weitere Applikationen verwenden können. Die Topologie wird als Plugin in die P2P Schicht eingefügt, was sie einfach austauschbar macht (siehe Abbildung 1).

Die Arbeit ist wie folgt gegliedert: Kapitel zwei geht auf das Konzept und die Implementation der Gitter-Topologie ein. Die Integration von Multicast in die bestehende Netz Schicht wird anschliessend in Kapitel drei erläutert. Abschliessend werfen wir in Kapitel vier einen kritischen Blick auf die Arbeit und geben einen Ausblick auf weiterführende Aufgaben.

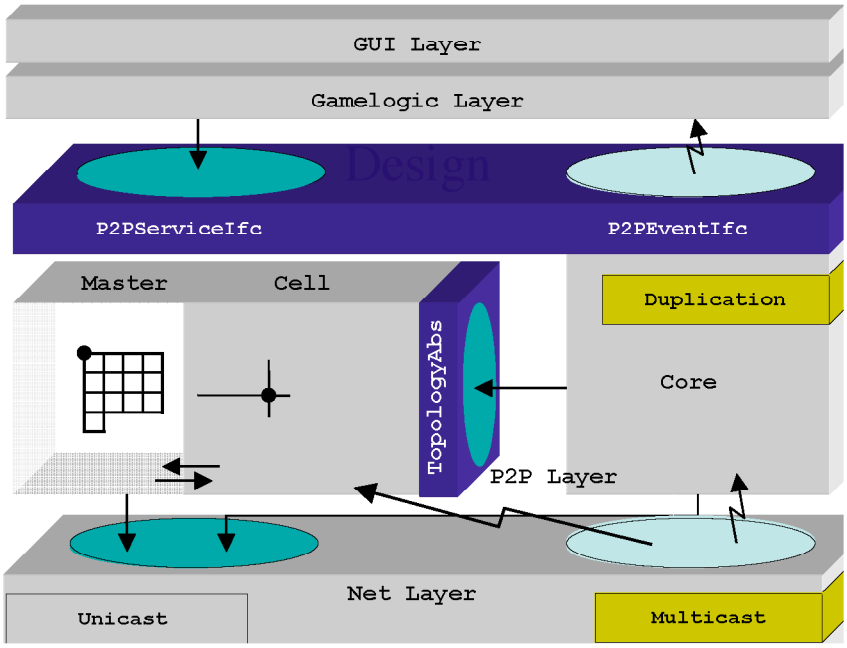

Figure 1: Architektur Übersicht des PeerBlast Projekts

3 Die Gitter-Topologie

In diesem ersten Teil werden wir das Gitter als eine weitere Topologie neben dem kompletten Graphen erläutern. Insbesondere gehen wir detailliert auf die Implementation der Leave-Operation ein.

3.1 Konzept

Die vollständige Vermaschung der Peers, wie im Falle des kompletten Graphen, skaliert schlecht: In einem Cluster mit n Peers werden $O(n^2)$ Verbindungen benötigt (n-1 pro Peer) und ebenso viele Nachrichten, um einen Spielzug im Cluster zu verbreiten. Diesem Nachteil kann grundsätzlich abgeholfen werden, indem der Vermaschungsgrad reduziert wird. Um trotzdem einen zusammenhängenden Graphen zu erhalten sowie den Echtzeitanforderungen gerecht zu werden, darf andererseits die Verzögerung nicht zu gross werden. Als ausgewogene Lösung dieses Tradeoffs stellen wir hier das Gitter vor.

Die Peers (*Cells* genannt) werden dabei in Zeilen und Spalten angeordnet, wobei jede Cell genau diejenigen Cells kennt, die in derselben Zeile oder Spalte liegen (Zeilen- bzw. Spaltennachbarn), wie in Abbildung 2. Die Dimension *dim* des Gitters bezeichnet die maximale Anzahl Cells aller Zeilen und Spalten. Gitter aus n Cells mit n MOD *dim* = 0 heissen *vollständig*.

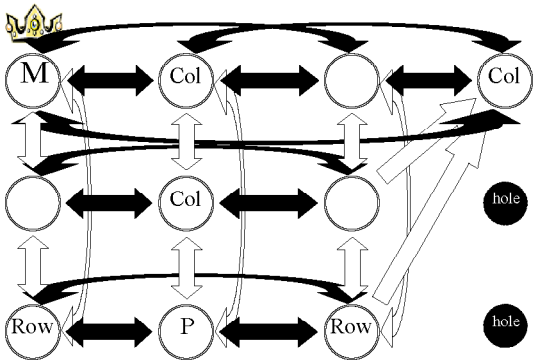

Figure 2: Als Gitter organisierter Cluster aus 10 Peers

Die Gitter-Topologie reduziert auf Kosten eines zusätzlichen zweiten Hops die Anzahl Verbindungen bei n Cells auf $O(\sqrt{n})$ ($2\times\sqrt{n-1}$ pro Cell) und dieselbe Anzahl Nachrichten pro Aktion, was eine asymptotische Verbesserung um den Faktor $O(\sqrt{n})$ bedeutet. Theoretisch verdoppelt sich die Verzögerung dabei jedoch mindestens. Für den Anwender in unseren Tests[1] blieb diese jedoch unbemerkbar klein.

3.2 Der Master

Die gitterförmige Struktur korrekt aufzubauen und zu unterhalten ist Aufgabe eines einzigen speziellen Peers, nämlich dem Gründer des Clusters (nachfolgend als *Master* bezeichnet).

Diese zentrale Administration ist zwar einfach, aber wenig fehlertolerant und die Asymmetrie birgt zudem die Gefahr einer Überlastung dieses Masters. Im vorliegenden Szenario eines infrastrukturbasierten Clusters liegt es jedoch nahe, mindestens einen bekannten Rechner im Cluster zu haben, welcher die Spieler als Treffpunkt nutzen, um ein laufendes Spiel zu finden.

Dies kann z.B. der Webserver sein, von dem das Spiel bezogen wurde. Somit sind sowohl Verfügbarkeit wie auch genügend Ressourcen gegeben. Da PeerBlast auf J2SE [3] basiert, werden zudem nur Rechner mit grossen Ressourcen (PCs, Laptops) unterstützt, was die vorliegende Lösung rechtfertigt.

In einem symmetrischen Szenario müsste jeder Peer die administrativen Daten (Clustergrösse, Positionen, Nachbarschaften) verwalten, was einerseits zu Konsensproblemen (Election, Synchronisation) und andererseits auch zu einer höheren Gesamtbelastung des Clusters führe.

[1] Im LAN mit 100 Mbit/s, bis 16 Hosts

3.3 Routing

Da ein Peer keine globale Sicht auf den Cluster hat, also nicht mehr jeden Peer im Cluster kennt, müssen Nachrichten geroutet werden. Im ersten Hop verteilt der Sender seine Nachricht an seine Zeilennachbarn.

Die Aufgabe des Routers nimmt derjenige Peer wahr, der die Nachricht im ersten Hop erhält. Er sammelt alle von Zeilennachbarn erhaltenen Nachrichten (schwarze, horizontale Pfeile in Abbildung 2) und reicht sie gemeinsam periodisch an seine Nachbarn in der Spalte weiter (weisse, vertikale Pfeile).

Damit das Routing bei beliebiger Clustergrösse n funktioniert (insbesondere bei nichtganzzahligen $\sqrt{n}$ wie im unvollständigen Gitter), übernehmen die Peers in der unvollständig ausgefüllten Zeile oder Spalte das Routing (weisse, geneigte Pfeile) für die leeren Stellen am Gitterrand[2].

3.4 Join

Im Gegensatz zum kompletten Graphen muss sich bei der Gitter-Topologie jeder Peer beim Master anmelden (Join-Operation), um in den Cluster aufgenommen zu werden. Dieser weist ihm einen Platz im Gitter zu, indem er ihm seine Zeilen- und Spaltennachbarn bekannt gibt.

Der Master baut das Gitter möglichst quadratisch auf, um eine maximale Anzahl Verbindungen von $O(\sqrt{n})$ zu garantieren, indem er abwechslungsweise Spalten und Zeilen ans Gitter anhängt. Im Cluster der Grösse $n{=}dim_{alt}{\times}dim_{alt}$ wird erst die Dimension inkrementiert ($dim_{neu}{:=}dim_{alt}{+}1$). Anschliessend wird in jedem Fall eine zusätzliche Spalte mit Cells aufgefüllt, solange $n{\leq}dim_{neu}$ $\times(dim_{neu}\text{-}1)$ und im Falle der Gleichheit ein vollständiges Gitter entsteht. Weitere Peers erhalten die Plätze in einer zusätzlichen Zeile, solange $n{\leq}\mathrm{dim}_{neu}$ $\times\mathrm{dim}_{neu}$ ist. Damit ist das Gitter um eine Dimension und der Cluster um $2{\times}dim_{neu}\text{-}1$ Peers gewachsen. Diesen Algorithmus führt der Master iterativ über die Dimension dim_{neu} aus, begonnen mit $dim_{neu}{=}1$ mit dem Master als ersten und einzigen Peer.

3.5 Leave

Fällt ein Peer aus oder reisst eine Kommunikationsverbindung ab, wird der nicht mehr erreichbare Peer aus dem Cluster entfernt (Leave-Operation). Dabei werden folgende vier Phasen durchlaufen: 1. Erkennung, 2. Umleitung, 3. Reorganisation und 4. Umstellung. Diese sind zusammen mit den dabei versandten Nachrichten aus Abbildung 4 für den Beispielcluster aus Abbildung 3 ersichtlich.

3.5.1 Erkennung

Als Erstes muss sichergestellt werden, dass ein Ausfall im Cluster bemerkt wird. Über einen TCP-Verbindungsabbruch zu einem Nachbarn wird der lokale Peer von der Netz Schicht informiert, der dies umgehend dem Master weiterleitet. Dieser notifiziert daraufhin alle Peers im Cluster über den Ausfall. Die P2P Schicht schickt den Benutzern der Serviceschnittstelle einen Event, um (wie

[2]Der Routing-Algorithmus ist in [1] detailliert beschrieben.

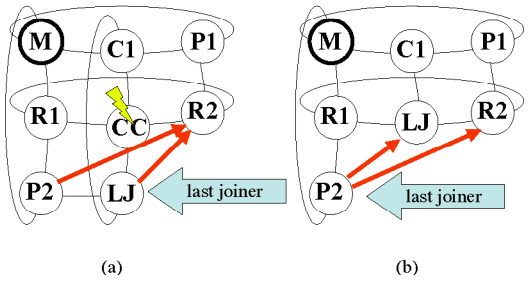

Figure 3: Das Gitter bevor (a) und nachdem (b) der Peer CC den Cluster verlässt (dim = 3)

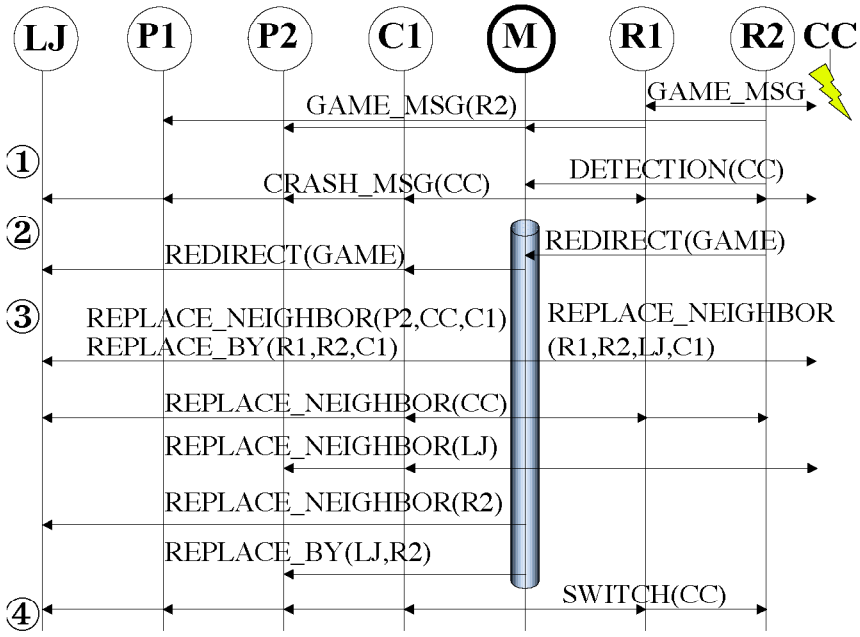

Figure 4: Die vier Phasen des Leave Protokolls mit den jeweils versendeten Nachrichten am Beispiel des Gitters mit 10 Peers (vor dem Leave von CC)

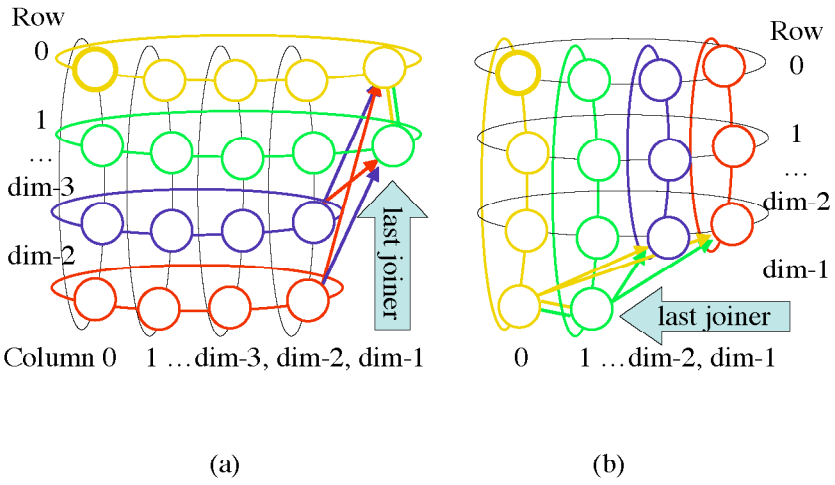

Figure 5: aktuelle Pseudonachbarn in der Spalte im Fall (a) mit (dim-1)< n < (dim-1) × dim und Zeile (Fall b) mit dim × (dim-1) < n < dim × dim am Beispiel dim = 5 (a) bzw. dim = 4 (b)

im Beispiel von XBlast) den betreffenden Spieler aus dem laufenden Spiel zu entfernen.

3.5.2 Umleitung

Jeder Nachbar des ausgefallenen Peers merkt also lokal, dass seine Verbindung abgerissen ist und leitet Nachrichten, die an den ausgefallenen Peer (*Crashed Cell*) addressiert sind (z.B. lokale ausgehende Nachrichten, falls die Crashed Cell in derselben Zeile sitzt), stattdessen an den Master weiter.

Dieser ermittelt aus Absender und Daten (Identifikation der Crashed Cell) die Empfänger und lässt ihnen die Nachricht über eine direkte Verbindung zukommen (z.B. den Spaltennachbarn der Crashed Cell). Auf diese Weise wird eine unterbruchsfreie Zustellung der Nachrichten bis zum Abschluss der Leave-Operation sichergestellt.

3.5.3 Reorganisation

Sobald der Master über einen Ausfall informiert ist, lokalisiert er die logische Position der Crashed Cell im Gitter (Zeile und Spalte). Der zuletzt eingefügte Peer (*Last Joiner*) ganz am Rand des Gitters wird an die freigewordene Stelle im

Gitter verschoben und ersetzt somit die Crashed Cell. Der Last Joiner eignet sich besonders gut, da die übrigen Cells so ihre Position im Gitter behalten können (also nicht "nachrutschen" müssen).

Der Algorithmus des Masters, um das Gitter zu schrumpfen, lehnt sich an denjenigen zum Wachsen an (siehe [1]). Die Schritte laufen jedoch in umgekehrter Sequenz ab, da auf diese Weise die Fallunterscheidungen denjenigen im wachsenden Gitter entsprechen: 1. Nachbarschaftsupdate 2. Das Gitter wird geschrumpft, d.h. die Anzahl Cells wird dekrementiert (und falls dadurch ein vollständiges Gitter entsteht auch die Dimension). 3. Der Last Joiner wird im geschrumpften Gitter neu zugewiesen.

Der erste Schritt, die Nachbarschaftsupdates, wird im Folgenden genauer erklärt: Als Erstes werden die aktuellen Nachbarn in der Zeile und Spalte (a) der Crashed Cell sowie (b) des Last Joiners bestimmt. Daraus generiert der Master Nachrichten, die die betroffenen Peers (a) die Crashed Cell als Nachbarn durch den Last Joiner (der ja dessen Position übernehmen wird) ersetzen sowie (b) den Last Joiner entfernen lassen.

Dabei kann es zu Überschneidungen kommen: Diese werden von jeder Cell lokal erkannt. Zusätzlich wird versucht, die Crashed Cell zu erreichen um ihr alle Nachbarn zu streichen (sinnvoll bei temporärem Ausfall, Verbindungsunterbruch oder Netzüberlastung, um den Peer ganz auszuschliessen).

Nach den bidirektionalen (wie im kompletten Graphen) werden auch die unidirektionalen Nachbarschaftsbeziehungen (Pseudo-Nachbarn) an das schrumpfende Gitter angepasst. Diese treten im nicht-vollständigen Gitter der Dimension *dim* (mit n Cells) zwischen den äussersten beiden Spalten (a: $(dim\text{-}1)^2 < n < (dim\text{-}1) \times dim$) oder Zeilen (b: $dim \times (dim\text{-}1) < n < dim^2$) auf (vgl. Abbildung 5).

Der Last Joiner befindet sich in der äussersten (a) Spalte oder (b) Zeile und ist damit (a) Ausgangspunkt eines unidirektionalen Kanals oder (b) selbst Pseudonachbar. Da er die Position der Crashed Cell übernimmt, werden (a) seine Pseudonachbarn oder (b) die Kanäle ausgehend von Cells, die ihn als Pseudonachbar referenzieren, entfernt.

Durch den Positionswechsel kann der Last Joiner (LJ) auch die Routingfunktion für Nachrichten von Cells in seiner (ehemaligen) Zeile (Z) an seine (ehemalige) Spalte (S) nicht mehr wahrnehmen. Deshalb müssen dies die betroffenen Cells selbst tun, nämlich mit einem direkten, unidirektionalen Kanal.

Im Fall (a) nimmt genau eine Cell $\in$ Z alle verbleibenden Cells $\in$ S $\setminus$ LJ auf Geheiss des Masters als Pseudo-Nachbarn in ihre Spalten-Nachbarschaft auf. Im Fall (b) fügt jede verbleibende Cell $\in$ Z $\setminus$ LJ eine Cell $\in$ S als Pseudo-Nachbar zu ihre Zeilen-Nachbarschaft hinzu.

Um die Korrektheit während der Reorganisation sicherzustellen, geschehen die Nachbarschaftsupdates nicht sofort und unkoordiniert, sondern das Gitter wird solange in der alten Form (mit Umleitungen) belassen, bis alle Update-Nachrichten erhalten wurden.

3.5.4 Umstellung

Nach Abschluss der Reorganisation kennen alle Peers ihre neuen sowie diejenigen bestehenden Nachbarn, die nicht mehr gebraucht werden. Auf ein spezielles Signal des Masters an alle Peers passen diese ihre Nachbarschaftsbeziehungen

an und lösen die temporäre Umleitung auf. Somit ist das Schrumpfen des Gitters abgeschlossen und der ausgefallene Peer aus dem Cluster entfernt.

4 Multicast

Die zweite Massnahme, um die Anzahl Nachrichten im Cluster zu reduzieren, ist die Ausnutzung der Multicastunterstützung [4] durch das physische Netz (z.B. Ethernet).

4.1 Konzept

Diese Erweiterung ist funktional in der Netz Schicht angesiedelt. Dies bedeutet, dass sie unabhängig von wie auch keine Kenntnis über die darüberliegende P2P Schicht hat, insbesondere nicht über die verwendete Topologie.

Multicast wird transparent für die höherliegenden Schichten automatisch verwendet, wenn das lokale Netz es unterstützt. Empfänger im selben LAN erhalten die Nachricht direkt als Multicast (UDP), die übrigen separat per Unicast (TCP). Multicast erweist sich als besonders wirkungsvoll, da jeder Peer seine Nachrichten, die er über die Serviceschnittstelle erhält, an alle anderen Peers im Cluster verteilt.

Befindet sich eine Teilmenge p von n Peers im gleichen LAN (wie häufig bei Netzwerkspielen), können diese p ihre Spielzüge mit $1+(n\text{-}p)$ Nachrichten im kompletten Graphen verteilen, die übrigen $n\text{-}p$ Peers ausserhalb des LANs benötigen dafür nach wie vor deren n-1 (worst case). Ist im best case hingegen (p=n) genügt eine einzige Nachricht, um mit Multicast alle Peers zu erreichen.

4.2 Bilden von Multicastgruppen

Beim Initialisieren der Netz Schicht wird eine Discovery-Nachricht über einen allen Peers bekannten Multicast-Socket versandt, um Rechner im eigenen LAN zu finden, die die PeerBlast (genauer: dieselbe Netz Schicht) bereits am Laufen haben.

Diese fügen den Sender der Discovery-Nachricht in ihre Multicastgruppe ein und antworten ebenfalls mit einer Discovery-Nachricht, jedoch per UDP-Unicast an die in der Nachricht enthaltene Portnummer. Die aufstartende Netz Schicht registriert ihrerseits alle eingehenden Antworten als per Multicast erreichbare Peers.

Bei Erkennung des Ausfalls eines Peers wird dieser lokal aus der Gruppe ausgeschlossen. Auf diese Weise werden die Gruppen laufend aktualisert.

Zur Zeit ist jeder Peer genau in einer Multicastgruppe vertreten, nämlich derjenigen des lokalen LANs (das evtl. nur den eigenen Rechner umfasst). Es könnten jedoch ohne Weiteres weitere Gruppen definiert werden. Deren Konfiguration (Elemente, Adresse, Portnummer) könnte z.B. in der Defaultgruppe verteilt werden, um ein LAN in mehrere Gruppen aufzusplitten.

4.3 Versand

Eine Nachricht, die der Netzschicht zum Versand überreicht wird, ist geeignet für Multicast, wenn mehr als ein Adressat spezifiziert ist oder gar keiner (was in der

Serviceschnittstelle einem Broadcast in den Cluster, also einer Nachricht an alle Peers entspricht). Um sicherzugehen, dass kein Peer eine Nachricht erhält, die er nicht erhalten sollte, wird geprüft, ob alle sich im LAN (in der gespeicherten Multicastgruppe) befindenden Peers auch als Adressaten spezifiziert sind, falls welche angegeben sind.

Ist dies nicht der Fall, wird die Nachricht einzeln per (TCP-) Unicast versandt. Befindet sich umgekehrt nur eine Teilmenge der Adressaten im LAN, werden diese per Multicast angesprochen, die Übrigen aussortiert und separat via TCP bedient.

4.4 Protokollerweiterungen

Multicast ist mit UDP realisiert, was den Nachteil mit sich bringt, dass keinerlei Garantien über den Erhalt sowie die Reihenfolge des Erhalts gegeben werden. Um denselben Grad an Zuverlässigkeit wie TCP zu erreichen, müsste ein Reliable Multicast Protokoll implementiert werden. Im vorliegenden Szenario wäre dies jedoch sicherlich übertrieben: Der Verlust einer (Multicast-)Nachricht ist nicht anwendungskritisch. Entscheidender ist die Echtzeitgarantie, die durch ein aufwändiges Protokoll [5] gefährdet würde.

Im Rahmen unserer Arbeit haben wir uns aus diesen Gründen auf eine einfache Implementation beschränkt. Diese erkennt Duplikate anhand von Sequenznummern auf der Netz Schicht und benutzt ein vereinfachtes Sliding-Window Protokoll [6], um die Auslieferung von Multicast-Nachrichten gleicher Sender in korrekter Reihenfolge zu garantieren.

Anstelle eines Acknowledgements (*ACK*), das den Vorteil des Multicasts in Bezug auf die Nachrichtenkomplexität zunichte machen würde, tritt ein negatives Acknowledgement (*NACK*). Falls eine Nachricht über Multicast empfangen wurde, deren Sequenznummer über dem Next Frame Expected (*NFE*) des Sliding Window dieses Senders liegt, wird dem Sender per Unicast mit einem NACK signalisiert, dass die Nachricht mit der Sequenznummer NFE fehlt. Die nochmalige Übertragung erfolgt per Unicast an den bei der Discovery bekannt gegebenen UDP Port und für jedes NACK einzeln.

Um den Sendepuffer beim Multicast beschränkt zu halten, haben wir zwei Alternativen vorgeschlagen: Erstens das Senden eines periodischen ACK (z.B. auf jede x. Nachricht, wobei x = kleinstes Byte der lokalen IP) und zweitens ein implizites ACK.

Die zweite Version haben wir realisiert, indem das Ausbleiben von NACKs zur Wiederverwendung des Pufferplatzes veralteter Nachrichten führt. Als veraltet gelten Nachrichten, die solange zurückliegen, dass eine für den Benutzer bemerkbare Verzögerung (während dem Warten auf die fehlende Nachricht), gefolgt von einem plötzlichen Schub von Ereignissen (ausgelöst durch das Eintreffen einer der vermissten Nachricht und damit der Auslieferung aller gepufferten Nachrichten mit höheren Sequenznummern) eintritt. Veraltete Nachrichten verletzen also die Echtzeitbedingung und werden daher senderseitig implizit gelöscht[3].

[3]In der vorliegenden Implementation ist der Puffer auf 256 Nachrichten beschränkt

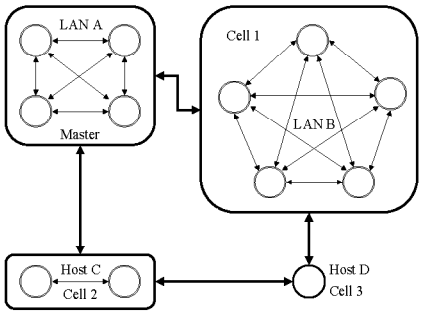

Figure 6: Vier komplette Graphen schliessen sich zu einem 2 × 2 Gitter zusammen

4.5 Multicast im Gitter

Idealerweise lässt sich Multicast ohne Änderungen auch im Gitter verwenden: Eine Teilmenge aller Peers wird mittels Multicast erreicht, die übrigen per Unicast. Dabei ist jedoch zu beachten, dass Nachrichten doppelt empfangen werden können. Da eine Multicast-Nachricht über logische Kanäle fliesst, die von der Topologie nicht benutzt werden, kann sie sowohl Zeilen- und Spaltennachbarn wie auch dem Absender unbekannte Cells im LAN erreichen.

Da beim Versand eines Broadcasts im LAN mangels Empfängerliste nicht überprüft werden kann, ob alle Cells im LAN auch adressiert werden, erhalten Spaltennachbarn wie auch die unbekannten Cells im LAN dieselbe Nachricht ebenfalls über TCP von den Zeilennachbarn des Adressaten (den Routern) zugestellt.

In der vorliegenden Architektur gelingt es nicht, auf der Netz Schicht eine Duplikatenkontrolle einzubauen ohne die Nachrichten auszupacken, weil die routenden Cells den Absender mit ihrer eigenen Adresse überschreiben, um sich dem Empfänger als Spaltennachbarn zu erkennen geben.

Die Duplikatenerkennung muss folglich oberhalb der Topologie erfolgen. Um zumindest gegenüber der Spielelogik Transparenz zu wahren, wurde der topologieneutrale Teil der P2P Schicht mit einer solchen Kontrolle (basierend auf Sequenznummern) erweitert.

Alternativ könnten mehrere Multicastgruppen verwendet werden, um Duplikate dieser Art zu vermeiden: Dabei hätte jede Spalte und jede Zeile ihre eigene Multicastgruppe, jede Cell wäre in zwei Gruppen vertreten.

5 Fazit

Neben den angesprochenen Duplikaten ergaben sich vor allem bei der Implementation der komplexen Leave-Operation Schwierigkeiten: Die Implementation der Erkennung und Notifikation sowie der kniffligen Umordnung der Nachbarschaftsbeziehungen vor allem im Fall eines nicht-vollständigen Gitters (die

Clustergrösse n ist weder Quadratzahl ($n = dim \times dim$) noch ist $n = dim \times (dim\text{-}1)$) erwiesen sich als fehleranfällig und zeitraubend.

Die vorliegenden Verbesserungen der Architektur befähigen Peer-to-Peer Anwendungen, die auf der P2P Schnittstelle von PeerBlast aufsetzen, zur theoretisch klar besseren Skalierung. Wie diese Vorteile in der Praxis ins Gewicht fallen, wird sich in experimentellen Messungen zeigen müssen.

Die Multicast-Fähigkeit von PeerBlast macht zudem komplexere Topologien, die sich aus kompletten Graph und Gitter aufbauen lassen, interessant. Denkbar wäre die Zusammenfassung der kompletten Graphen jedes LANs aus Peers zu einer Cell der Gitter-Topologie (siehe Abbildung 6).

6 Referenzen

1. [1] M. Cicolini, B. Huber, S. Schlachter, M. Trautweiler: "Entwicklung eines rasanten Peer-to-Peer Spiels", Laboratory in Distributed Systems, April 2003, Zürich
2. [2] http://www.xblast-center.com
3. [3] Java 2 Platform, Standard Edition, http://java.sun.com/docs
4. [4] Multicast Explained, http://www.linux.org/docs/ldp/howto/Multicast-HOWTO-2.html
5. [5] Reliable Multicast Links, http://www.nard.org/~tmont/rm-links.html
6. [6] Transmission Control Protocol Specification, RFC793, 1981