Hauptseminar Telematik
SS 2004

TECHNISCHE UNIVERSITÄT ILMENAU

Aufbau und Organisation von Multicast-Trees

Ronny Schöbel
24044

Inhalt

1 Einleitung

Neben Medien wie Radio oder Fernsehen setzt sich das Internet auch immer mehr als Medium zur Verteilung von Multimediadaten durch. Die Verfügbarkeit breitbandiger Internetzugänge, wie beispielsweise DSL, trägt zur Popularität solcher Anwendungen bei. Durch das große Volumen von Multimediadaten bietet es sich an, die Daten über das Internet zu streamen, um zusätzliche Downloadzeiten zu vermeiden. Besonders für das Live-Streaming von wichtigen Ereignissen an viele verschiedene Nutzer scheint das Multicast-Modell zur Verteilung der Daten sehr geeignet.

Gleichzeitig gewinnen auch Peer-to-peer Anwendungen immer mehr an Bedeutung, besonders da sie es erlauben ungenutzte Ressourcen von Internet-Hosts dem Netz zur Verfügung zu stellen und somit eine Lasterleichterung für Server und deren Netzwerkverbindungen erreichen können. So ist es auch ohne teure Serverrechner möglich, beliebige Daten - darunter auch gestreamte Multimedia-Inhalte - einem großen Kreis von Nutzern zur Verfügung zu stellen.

Hier soll anhand von zwei Beispielen (*Zigzag* und *SplitStream*) ein kurzer Überblick gegeben werden, wie sich eine Multicast-Struktur innerhalb von Peer-to-Peer Netzen aufbauen und verwalten lässt. Daneben werden die verschiedenen zu lösenden Probleme näher betrachtet.

2 Das Multicast-Modell

Multicast ist beschrieben durch ein sehr einfaches Modell, in dem keine Annahmen über die zu Grunde liegende Netzwerktopologie oder die vorhandenen Transportsysteme gemacht werden [1].

Multicast bedeutet, dass ein Sender eine Gruppe von Hosts durch das Senden nur eines Pakets erreichen kann. Die Empfängergruppe wird durch eine Multicast-Adresse identifiziert, an welche die Pakete versendet werden. Hosts teilen dem Netzwerk den Beitritt zu einer Multicast-Gruppe mit, wobei anzumerken ist, dass der Sender selbst kein Mitglied der Gruppe sein muss. Dies ermöglicht ein Senden an die Gruppenmitglieder von verschiedenen Punkten im Netzwerk. Der Transport der Multicast-Pakete erfolgt synchron, das heißt, alle Teilnehmer erhalten die Daten quasi gleichzeitig. Die Verteilung der Pakete an die einzelnen Hosts einer Multicast-Gruppe erfolgt durch die im Netzwerk vorhandenen Router, welche auch die eventuell notwendige Duplizierung der Pakete übernehmen.

Dieses sehr einfache Modell bietet eine Reihe von Eigenschaften, die Vorteile für die verschiedensten Applikationen bieten. Multicast ist für den Sender ein vollkommen transparenter Vorgang. Für die sendende Applikation besteht kein Unterschied zwischen dem Senden per Unicast bzw. per Multicast. Daraus leitet sich ab, dass dem Sender die Mitglieder der Multicast-Gruppe nicht bekannt sein müssen. Der Vorteil hier ergibt sich besonders wenn eine große Anzahl von Teilnehmern erreicht werden soll. Weiterhin erlaubt das Modell einen schonenden Umgang mit den vorhandenen Netzwerkressourcen. Da ein Sender nur ein Paket senden muss, um eine ganze Gruppe von Hosts zu erreichen und das gesendete Paket erst an den beteiligten Routern repliziert wird, transportiert jeder physikalische Netzwerklink immer nur eine einzige Kopie jedes Pakets.

Die Verteilung der Pakete per Multicast ergibt im Netzwerk eine Baumstruktur, deren Wurzel der Sender darstellt. Die inneren Knoten des Baums werden durch die am Multicast-Transport beteiligten Router, die Blätter von den Hosts der Multicast-Gruppe gebildet.

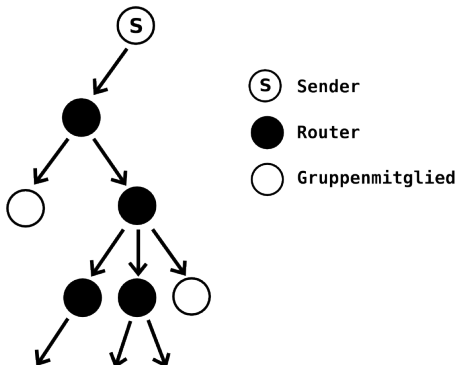

Abbildung 1: Entstehender Multicast-Tree

3 IP-Multicast

Eine Spezifikation für Multicast im Internet bietet das in [8] standardisierte IP-Multicast. Integriert in den Network-Layer des ISO/OSI-Referenzmodells bietet IP-Multicast allen Applikationen einen einheitlichen Zugang zu Multicast-Funktionalitäten.
IP-Multicast nutzt die im Internet bereits vorhandene Infrastruktur (Router, Backbones) zum Transport von Paketen, das Management der Gruppenzugehörigkeit von Hosts wird durch das Internet Group Management Protocol (IGMP) übernommen.
Trotz der Vorteile die dieser Ansatz bietet, stehen einer breiten Nutzung von IP-Multicast im Internet noch verschiedene Probleme im Wege. Ein erstes besteht im eingeschränkten Adressraum der Multicast-Anwendungen zur Verfügung steht. Verschiedene Bereiche des Adressraums sind durch die IANA (Internet Assigned Numbers Authority) statisch verschiedenen Protokollen bzw. Applikationen zugeteilt [9]. Innerhalb des frei verfügbaren Adressraumes kann es besonders bei Ad-Hoc-Anwendungen zu Adresskollisionen zwischen verschiedenen Applikationen kommen. Noch gravierendere Probleme ergeben sich im Zusammenhang mit dem Multicast-Routing und Gruppenzugehörigkeiten. Zum einen unterstützen bei weitem noch nicht alle IP-Hosts bzw. Router die IP-Multicast Erweiterungen oder implementieren sie unvollständig. Daneben bereiten die Routing-Algorithmen und Routerkonfiguration die meisten Schwierigkeiten [1].

4 Application Layer Multicast

Durch die ungelösten Probleme im Umfeld von IP-Multicast konzentrierte sich die Forschung der letzten Jahre darauf einen praktikablen Ausweg zu finden, der Applikationen ermöglicht die Vorteile einer Datenverteilung per Multicast im Internet zu nutzen ohne auf eine IP-basierte Multicast-Unterstützung angewiesen zu sein. So rückten besonders Peer-to-peer basierte Overlay-Netze ins Zentrum des Forschungsinteresses.

Für Application Layer Multicast übernehmen die Applikationen selbst große Teile der Aufgaben, die innerhalb des OSI-Referenzmodells eigentlich tieferliegenden Schichten zugeordnet sind [10]. Zu nennen sind hier besonders das Multicast-Routing sowie die Verwaltung der Gruppenzugehörigkeit einzelner Hosts. Der Pakettransport erfolgt per Unicast, da nicht mehr von einer multicast-fähigen Netzwerkschicht ausgegangen wird. Auf diese Weise wird effektiv eine zweite Netzwerktopologie über der bereits vorhandenen Netzwerkstruktur aufgebaut.

Da in einem solchen Overlay-Netz die Applikationen selbst die Kontrolle über die Multicast-Funktionen besitzen, lassen sich Ad-Hoc-Netzwerke einfacher realisieren, da vor allem das Problem der Adresszuteilung und Adresskollision deutlich gemindert wird.

Allerdings entstehen in ALM-Overlays auch neue Probleme.

Durch den Unicast-basierten Transport der Pakete wird nun zunächst die verfügbare Netzwerkbandbreite ein möglicher Engpass. Da die Paketreplikation nun von End-Hosts übernommen wird, sind auf physikalischen Netzwerklinks eventuell mehrere Kopien eines Datenpakets unterwegs. Die Anzahl dieser Kopien hängt dabei stark von der Topologie des vorhandenen wie auch des Overlay-Netzes ab. Besonders für Applikationen, die große Datenmengen wie beispielsweise Mediadaten transportieren kann dies leicht zu Service-Unterbrechungen führen, besonders wenn verschiedene Hosts über unterschiedliche Netzanbindungen verfügen. Solche Unterschiede bestehen oftmals nicht nur in der gesamten zur Verfügung stehenden Bandbreite, sondern auch in asymmetrischen Bandbreitenverhältnissen zwischen Up- und Downstream. Damit kann nicht jeder Hosts dem System im gleichen Maße seine Ressourcen zur Verfügung stellen.

Weitere Einschränkungen der Bandbreite entstehenden durch Signalisierungsaufwand zwischen den Hosts des Overlay-Netzes wie auch durch Konkurrenz mit weiterem Netzwerkverkehr. So kann die wirklich zur Verfügung stehende beträchtliche Schwankungen

aufweisen und ist damit von Applikationen schwer einschätzbar.
In direktem Zusammenhang mit der Bandbreite steht die Skalierbarkeit des Netzes. Skalierbarkeit in einem kooperativen System betrifft vor allem die Anzahl der möglichen Netzteilnehmer. Mit steigender Zahl nehmen Signalisierungsaufwand und die transportierende Nutzdaten zu. Daher bedeutet Skalierbarkeit auch, einen möglichst effizienten Umgang mit verfügbaren Netzwerkressourcen.
Das dritte große Problem besteht in der Zuverlässigkeit des Overlays. Kooperative Systeme nutzen die Ressourcen der End-Hosts für ihren Betrieb. Diese sind im Allgemeinen keine dedizierten Server. Kommen und Gehen einzelner Hosts können ein stochastisches und damit nicht vorhersagbares Verhalten aufweisen oder beteiligte Hosts können ohne Vorwarnung ausfallen. Aber auch durch Überlastung (congestion) einzelner Netzwerklinks kann die Zuverlässigkeit des Netzes stark beeinträchtigt werden. Um einen reibungslosen Betrieb des gesamten Overlays, möglichst ohne Service-Unterbrechungen für die Nutzer sicherzustellen müssen diese Probleme beim Design des Systems Beachtung finden.
Damit lassen sich für ALM-Overlays allgemein die folgenden Anforderungen formulieren:

- *Gute Abbildung auf vorhandene Netzwerktopologie*

Das entstehende Overlay sollte die darunterliegende vorhandene Netzwerkstruktur gut abbilden. Das bedeutet, Nachbarschaftsbeziehungen von Hosts sollten möglichst erhalten bleiben, um ein effizientes Paketrouting zu ermöglichen und die Duplikation von Paketen auf physikalischen Links zu reduzieren.

- *Kreisfreiheit*

Der entstehende Multicast-Tree muss kreisfrei sein, das heißt von der Wurzel ausgehend wird entlang der Kanten des Baums kein Host mehrfach besucht. Kreise innerhalb des Overlays können zu Verstopfungen von Netzwerkverbindungen oder falschen empfangenen Daten in der Applikation führen.

- *Anpassungsfähigkeit an unterschiedliche Bedingungen*

Durch Kommen und Gehen oder Ausfall von Hosts, durch Überlastung oder Ausfall von

Netzwerkverbindungen können unvorhergesehene Bedingungen innerhalb des Overlays entstehen, die den Netzbetrieb beeinträchtigen. Das System sollte die Fähigkeit besitzen solche Situationen bzw. Fehler zu erkennen und so weit wie möglich zu korrigieren. Dazu zählt besonders eine dynamische selbstgesteuerte Neuorganisation des Netzes.

- *Flacher, balancierter Multicast-Tree*

Für Echtzeit-Systeme, wie das Streaming von Multimediadaten ist es wichtig, den Multicast-Tree möglichst flach und balanciert zu halten. So werden kurze Transportwege geschaffen, die Laufzeit von Paketen und Verzögerungen beim Empfang verringern sich. So können beispielsweise verloren gegangene Pakete leichter wieder angefordert werden, ohne dass es beim Nutzer zu spürbaren Unterbrechungen kommt.
Diese Forderung steht allerdings in eventuellem Gegensatz zur ersten Forderung. Um die Skalierbarkeit des Netzes zu sichern, muss in einem sehr flachen Baum ein einzelner Host eine große Anzahl weiterer Hosts mit Daten versorgen. Dadurch steigt unweigerlich der Verbrauch an knapper Netzwerkbandbreite, da viele Paketduplikate versendet werden müssen.
Im Allgemeinen ist es kaum möglich alle Forderungen in gleichem Maße zu berücksichtigen. Daher entstanden im Laufe der letzten Jahre verschiedene Systeme und Prototypen, die je nach Anwendungsgebiet den Schwerpunkt auf unterschiedliche Eigenschaften legten. Die nächsten Abschnitte dieser Arbeit sollen zwei Systeme näher beleuchten und ihre Eigenschaften und Design-Prinzipien aufzeigen und verdeutlichen.

4.1 Zigzag

Entwickelt an der University of Central Florida, Orlando, soll Zigzag ein effizientes, skalierbares und fehlertolerantes Peer-to-peer basiertes Multimedia-Streaming-System schaffen. Zigzag ist in [2] detailliert beschrieben.
Zigzag erzeugt einen Multicast-Tree, dessen Wurzel die Quelle der Multimediadaten bildet (Server). Um End-zu-End Wartezeiten zu verkürzen, soll der Baum möglichst flach gehalten werden. Dabei balanciert Zigzag mit festen Regeln den entstehenden Baum. Mit N als Anzahl der am Service teilnehmenden Hosts und einer Konstanten k wird die Tiefe des Multicast-Trees auf $O(\log_k N)$ beschränkt. Um das System dennoch skalierbar zu machen, und die

Bandbreite von Clients nicht zum Flaschenhals werden zu lassen, ist innerhalb des gesamten Baums der Grad jedes Knotens konstant nach oben auf $O(k^2)$ beschränkt.
Anmeldung und Verlassen des Netzes wird Clients einfach gemacht, so dass eine schnelle Verfügbarkeit des gewünschten Service möglich wird. Um Bandbreite innerhalb der Netzstruktur zu sparen, bedient sich Zigzag eines eigenen Kontrollprotokolls mit geringem Overhead zum Austausch von Meta-Informationen zwischen den Hosts.
Die herausragende Eigenschaft von Zigzag besteht in der Trennung von Administration und Mediendistribution. Dafür baut Zigzag zunächst eine administrative Struktur innerhalb des Overlays auf, über die regelbasiert der Multicast-Tree gelegt wird. So soll eine höhere Fehlertoleranz durch Trennung von Aufgaben (Vermeidung von *Single point of failure*) und eine effizientere Nutzung der Ressourcen von Client-Hosts erreicht werden.

4.1.1 Administrative Struktur

Die administrative Struktur von Zigzag dient dem Management aller im System befindlicher Hosts. Aus ihr entsteht nach einem festen Regelwerk der Media-Distribution-Tree.
Zigzag organisiert alle Hosts (Peers) in einer Multi-Layer Struktur von Clustern. Mit der Anzahl der Layer H, und einer Konstanten $k > 3$, definiert sich die Struktur wie folgt.
Jeder Peer ist Mitglied von Layer 0. Jeder Layer $j < H - 1$ wird in Cluster der Größe $[k, 3k]$ partitioniert. Layer $H - 1$ (höchster Layer) enthält nur einen Cluster der Größe $[2, 3k]$.
Ein Peer eines Clusters in Layer $j < H$ wird zum *Head* dieses Clusters ernannt. Dieser *Head*

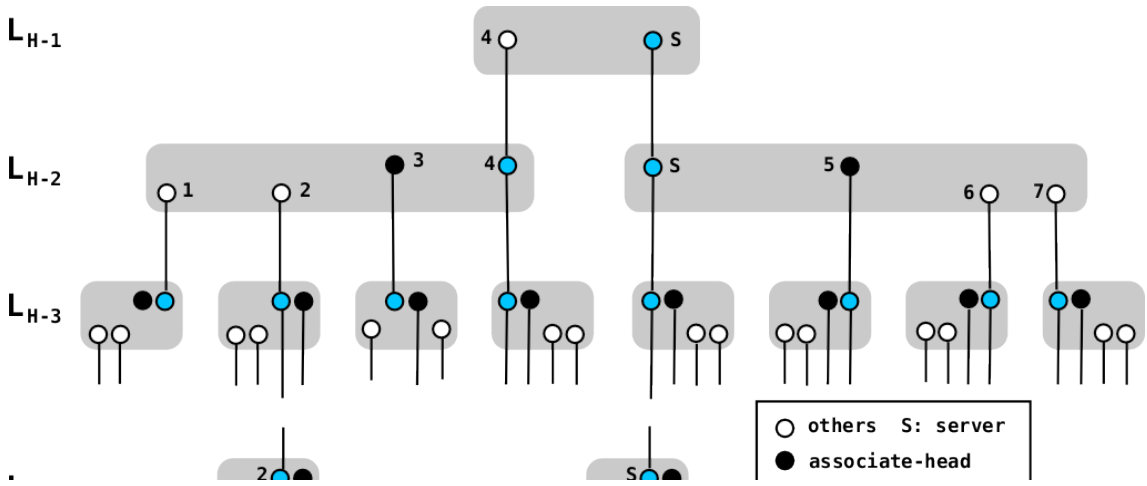

Abbildung 2: Administrative Struktur von Zigzag

wird Mitglied von Layer $j + 1$ falls $j < H - 1$. Der Server *S* ist *Head* jedes Clusters in dem er vertreten ist.
Ein Peer eines Clusters in Layer *j*, der nicht *Head* seines Clusters ist, wird zum *Associate-Head* des Clusters bestimmt. Eine Ausnahme bildet Layer $H - 1$, in dessen einzigem Cluster der Server *S* gleichzeitig *Head* und *Associate-Head* seines Clusters ist.
Diese Struktur wächst dynamisch mit der Anzahl der im Netz vorhandenen Peers, ausgehend von einem Layer mit nur einem Cluster. Mit dem Hinzukommen oder Wegfallen von Peers kann es notwendig werden, Cluster aufzuteilen oder zu verschmelzen um die definierten Clustergrößen zu erhalten. Die Beschränkung der Clustergröße auf 3k sorgt dabei für eine Art Hystereseverhalten, so dass nach dem Aufteilen eines Clusters nicht einer der beiden neu entstandenen Cluster Gefahr läuft eine zu geringe Anzahl an Peers zu enthalten, sollte eines seiner Mitglieder das Netz verlassen, bzw. nach dem Verschmelzen von 2 Clustern zu einem größeren Cluster besteht kaum die Gefahr, dass dieser wieder aufgetrennt werden muss, weil ein Peer hinzukommt.

4.1.2 Aufbau des Multicast-Trees

Der Multicast-Tree von Zigzag entsteht über der administrativen Struktur durch drei Regeln, die in Zigzag *C-Rules* genannt werden. Sie lauten im Original:

1. *A peer, when not at its highest layer, neither has a link out nor a link in.*
2. *Non-head members of a cluster must receive the content directly from its associate-head.*
3. *The associate-head of a cluster, except for the server, must get the content directly from a foreign-head.*

Diese Regeln garantieren, ausgehend von der administrativen Struktur jeder Peer die Streamingdaten genau durch einen Link erhält, wie auch einige weitere Eigenschaften von Zigzag.
Regel 1 verhindert zunächst die doppelte Zuweisung von Links an einen Peer und bestimmt das "grobe" Aussehen des entstehenden Baumes. Peers die in der administrativen Hierarchie weit oben angesiedelt sind, erhalten so auch ihre Daten früher, und reichen sie weiter nach unten.

Regel 2 bestimmt für jeden Peer eines Clusters woher dieser Peer seine Daten erhält und sichert damit die Versorgung mit Daten für jeden Peer eines Layers, ausgenommen alle *Associate-heads* dieses Layers.
Regel 3 schließt auch diese letzte Lücke. Ein *Associate-head X* auf Layer *j* erhält damit seine Daten von einem Peer, der sich mit dem *Head* von *X* auf Layer *j* + 1 in einem Cluster befindet.

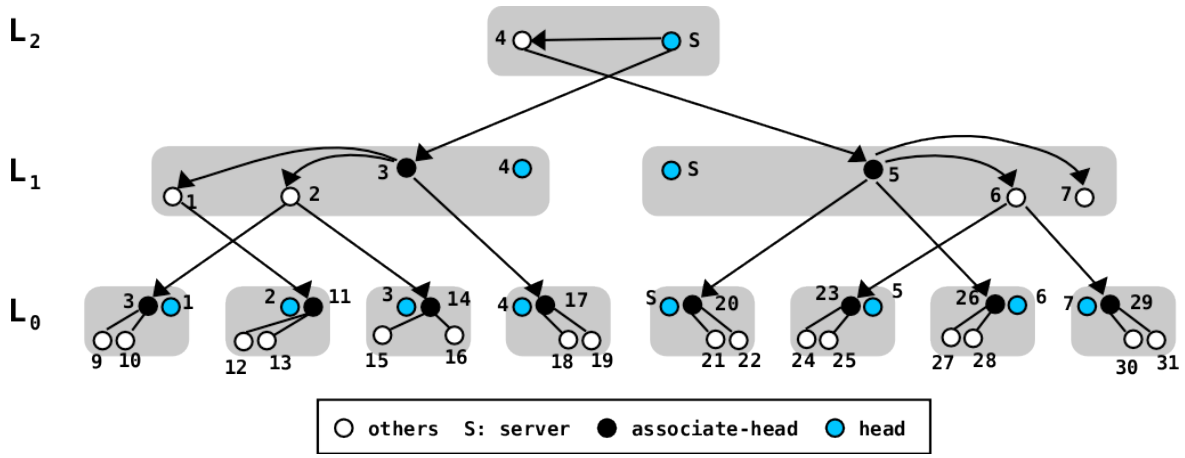

Abbildung 3: Multicast-Tree für Administrative Struktur aus Abbildung 2

Zigzag setzt die *C-Rules* nach Aufbau des Baums auch unter dynamischen Bedingungen wie Ausfall von Peers, oder Hinzukommen neuer Peers konsequent durch. Dafür sorgt das eigens entwickelte Kontrollprotokoll, über welches die einzelnen Peers Stausinformationen austauschen. Um möglichst wenig Overhead während dieser Kommunikation zu erzeugen, tauschen Peers nur mit einer begrenzten Anzahl weiterer im Netz befindlicher Hosts Daten aus. Zu diesen zählen eigene Clustermitglieder sowie Eltern und Kinder innerhalb der Multicast-Hierarchie. Die gesendeten Daten beinhalten gemessene End-zu-End-Wartezeiten der Multicast-Daten vom Server sowie Informationen über Erreichbarkeit von Layer-0 Peers und freie Plätze in erreichbaren Layer-0-Clustern. Mit erreichbar ist in diesem Zusammenhang ein vorhandener Weg innerhalb des Multicast-Trees vom Kontrolldaten sendenden Peer zu einem Layer-0-Peer gemeint.

4.1.3 Join und Leave Operationen

Möchte ein neuer Peer dem Zigzag-Overlay beitreten, kontaktiert er zunächst den Server. Dieser reicht die erhaltene Anfrage anhand der empfangenen Informationen über freie Plätze

entlang des Multicast-Trees nach unten, bis ein Layer-0-Cluster gefunden wird, der den neuen Peer aufnehmen kann. Sollten mehrere Peers zur Wahl stehen werden beim Weiterleiten der Anfragen Peers bevorzugt, die minimale End-zu-End-Wartezeiten zum Server sowie zum hinzukommenden Peer aufweisen. Sollte der Layer-0-Cluster, welcher den neuen Peer aufnimmt anschließend seine Maximalgröße von $3k$ überschreiten wird er in zwei gleichmächtige Teilcluster aufgeteilt. Zigzag sieht dabei unterschiedliche Algorithmen für eine Aufteilung von Clustern auf Layer 0 bzw. auf höheren Layern vor. Dies erfolgt zum einen mit dem Ziel auf Layer 0 die Datentransportwege kurz zu halten und auf höheren Layern möglichst wenige Umgruppierungen innerhalb des Multicast-Trees zu erreichen. In allen Fällen werden die *C-Rules* strikt eingehalten. Die Ausführung der Aufteilung eines Clusters obliegt dem *Head* des betroffenen Clusters, welcher die Ergebnisse der Neuberechnungen an alle beteiligten Peers. sendet.

Sollte ein Peer X willentlich oder durch Ausfall das Netz verlassen, sind je nach Rolle von X unterschiedliche Aufgaben zu erledigen um den Netzbetrieb zu sichern.

Im trivialsten Fall ist X ein Peer auf Layer 0. Sollte X der *Associate-Head* seines Clusters gewesen sein, ist der *Head* dieses Clusters dafür verantwortlich ihn durch einen anderen Peer zu ersetzen, wobei ein Peer selektiert wird, der den minimalen Abstand zu X hatte um die Folgen des Ausfalls möglichst gering zu halten. Ansonsten sind keine weiteren Aktionen nötig.

Falls X auf Layer j mit $j > 0$ vertreten war, müssen weitere Aufgaben erfüllt werden. Alle Cluster auf den Layern $0 \ldots j - 1$ deren *Head* bisher X war, benötigen nun einen neuen *Head*, wofür im Cluster auf Layer 0 ein zufälliger Peer (ausgenommen dem *Associate-Head*) gewählt wird, der X vollständig ersetzt. Für alle bisherigen Kinder Y von X muss ein neuer Peer gefunden werden, der nun die Bereitstellung der Daten übernimmt. Der *Head* des Clusters von Y ist dafür verantwortlich und wählt auf Layer j einen Peer - ausgenommen dem *Head* und *Associate-Head* - mit minimalem Ausgangsgrad.

Sollten durch Ausfälle Cluster eine Größe von k unterschreiten, werden kleine Cluster innerhalb der administrativen Struktur ausgehend vom höchsten Layer zu neuen Clustern zusammengefasst. Falls Cluster U auf Layer j zu klein ist, wird auf diesem Layer der kleinste Cluster V mit dem gleichen Supercluster wie U gesucht und mit U vereinigt.

Auf Layer $j + 1$ sind dabei einige Änderungen notwendig, da einer der beiden *Heads* von U bzw. V nach dem Zusammenschluss nicht mehr auf Layer $j + 1$ präsent ist.

Der *Associate-Head* des größeren der beiden Cluster U, V wird auch *Associate-Head* des

neuen zusammengeschlossenen Clusters.

4.1.4 Leistungsoptimierung

Zigzag bietet einige Möglichkeiten die Leistung des Netzes zu verbessern und dynamischen Netzwerkbedingungen anzupassen. Da verfügbare Bandbreit oftmals das größte Problem für Streamingsysteme darstellt, hängt die Leistung eines Zigzag-Netzes zunächst von der Wahl von k ab. Der entstehende Multicast-Tree wird mit steigendem k flacher und breiter, was im Allgemeinen eine gewünschte Eigenschaft ist.
Mit steigendem k wächst jedoch die Last die jeder *Associate-Head* im Netz und jeder *Head* auf Layer $j > 0$ zu tragen hat an, da ihr Ausgangsgrad ebenfalls ansteigt. So muss k so gewählt werden, dass ein Kompromiss zwischen hohem Ausgangsgrad bei großem k und langen Transportzeiten bei kleinem k gefunden wird.
Darüber hinaus bietet Zigzag einen Algorithmus zur Verteilung der Last von Peers innerhalb eines Clusters. Auf einem Layer $j > 0$ können Peers Verbindungen an andere nicht ausgelastete Peers abgeben. Dieser Algorithmus sollte auf jedem Peer ausgeführt werden, wenn sein Ausgangsgrad einen festgelegten Schwellwert überschreitet. Zu häufige Optimierungsversuche können zu ständigen Verbindungswechseln einzelner Peers führen und so zu Service-Unterbrechungen führen.
Eine alternative Optimierung der Tiefe des Multicast-Trees besteht in der Anwendung der C-Rules des ersten Zigzag-Entwurfs. In diesem nun als Direct-Zigzag bekannten System entfällt die Rolle des *Associate-Head*. Alle Mitglieder eines Clusters empfangen die Daten direkt vom *Head* eines anderen Clusters auf ihrem Layer. Die Transportwege werden dadurch nochmals verkürzt, da jeder Layer nur genau einmal durchlaufen wird. Der Grad jedes Peers kann hierbei jedoch höher sein als im indirekten *Zigzag*.

4.1.5 Ungelöste Probleme

Einige Probleme von ALM-Overlays bleiben in Zigzag unter andrem durch Design-Entscheidungen und Fokussierung auf bestimmte Eigenschaften ungelöst.
Zigzag ist abhängig von einem stabilen Server, der die zu streamenden Daten in das Netz

einspeist und gleichzeitig die Anfragen von Hosts die dem Overlay beitreten wollen beantwortet. Dadurch bildet der Server einen Flaschenhals und Zigzag wird anfällig gegen Flash-crowds, einen große Anzahl von Hosts die zu ungefähr gleicher Zeit dem Netz beitreten. Weiterhin wird in Zigzag davon ausgegangen, dass Hosts genügend ausgehende Netzbandbreite verfügen um mehrere andere Peers mit Daten zu versorgen. Selbst bei minimaler Wahl von $k = 3$ müssen einige Peers damit in der Lage sein bis zu 15 weitere Peers mit Daten zu versorgen.
Auch die darunterliegende Netzwerktopologie wird nur bedingt abgebildet. Jeder Peer sammelt statistische Informationen über Paketlaufzeiten und teilt diese über das Kontrollprotokoll seinen Nachbarn im Netz mit. Diese Informationen können als Indizien für topologische Nähe der Hosts zueinander benutzt werden, da Zigzag jedoch bei einem neu hinzukommenden Peer zuerst versucht Cluster aufzufüllen, und Leistungsdaten des Netzes erst als Zweitkriterium verwendet ist die Abbildung nicht optimal. Der Split eines bereits vollen Clusters könnte einem Peer vielleicht erlauben Verbindungen zu Peers die ihm eventuell physikalisch näher sind aufzubauen.

4.2 SplitStream

In Zusammenarbeit zwischen der Rice University, Houston und Microsoft Research, Cambridge wurde SplitStream als Peer-to-peer Streaming System zur Verteilung verschiedener Inhalte, u.a. auch von Multimedia-Daten entwickelt [3, 4].
Die grundsätzliche Idee von SplitStream basiert auf der Beobachtung, dass in ALM-Overlays nur ein relativ kleiner Prozentsatz von Peers an der Verteilung der Daten beteiligt ist, und damit zur Funktion des Netzes beiträgt. In Multicast-Systemen sind alle Peers entweder innere Knoten oder Blätter des Multicast-Trees. Bereits in einem binären, voll balancierten Baum sind 50% aller Knoten Blätter. In Mehrweg-Bäumen wie sie im Allgemeinen in einem Multicast-System entstehen steigt dieser Anteil an Blättern sehr schnell. Da nur innere Knoten die Multicast-Daten verbreiten bleibt viel Kapazität innerhalb des Netzes ungenutzt. SplitStream geht hier einen anderen Weg und versucht mehrere überlagerte Multicast-Trees aufzubauen, so dass jeder Peer möglichst nur in genau einem Multicast-Tree einen inneren

Knoten bildet.

Erreicht werden soll so eine bessere Lastverteilung, mehr "Gerechtigkeit" innerhalb des Overlays und eine höhere Robustheit des gesamten Netzes gegen Ausfall einzelner Peers. Um diese Eigenschaften zu verwirklichen teilt SplitStream die zu verteilenden Daten in mehrere Datenströme - *Stripes* - auf. Jeder einzelne *Stripe* muss dabei gewissen Anforderungen genügen. Die gesamten Daten werden so zerteilt, dass alle *Stripes* (ungefähr) die gleiche Bandbreite zum Transport benötigen. Weiter ist verlangt, dass jeder *Stripe* die gleiche Menge an Information enthält, und sich sinnvoll nutzbare Daten auch aus nur einem empfangenen *Stripe* rekonstruieren lassen. Für das Codieren und Decodieren der Daten macht SplitStream keinerlei Vorgaben. Die Applikation selbst dafür verantwortlich. So lassen sich beispielsweise Multimedia-Daten mit Hilfe eines Multiple Description Coding in einzelne Stripes zerlegen.

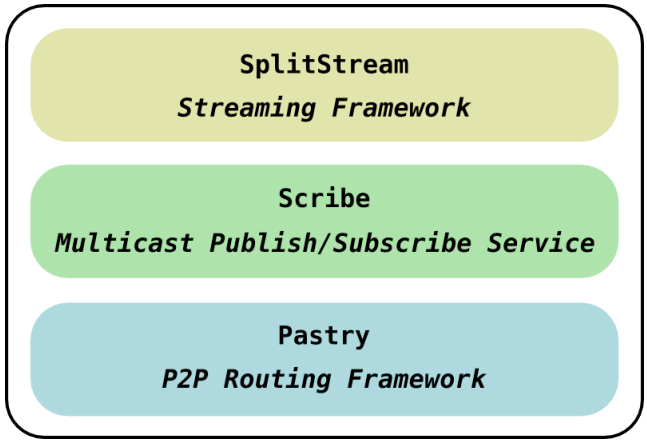

Abbildung 4: SplitStream Architektur

SplitStream erlaubt jeder Node des Netzes selbst zu kontrollieren wie viel Netzwerkbandbreite sie dem System zur Verfügung stellen kann bzw. möchte. Die einzige Anforderung an jede Node besteht darin, dass sie sich verpflichtet, wenigstens einer weiteren Node einen *Stripe* zu senden.

Um auch auf dynamische Vorgänge innerhalb des unter dem Overlay liegenden Netzes reagieren zu können, ist SplitStream weitgehend selbstorganisierend. Das bedeutet, dass Alternativrouten für Pakete gewählt werden können, sollte dies z.B. durch Ausfall einer Node notwendig werden.

Zur Umsetzung dieser Eigenschaften ist SplitStream kein monolithisches System, sondern besitzt eine Architektur die sich grob in drei Schichten teilt. Die unterste Ebene bildet *Pastry*, ein selbstorga-nisierender Peer-to-peer und Routing Framework. Darauf setzt *Scribe*, ein Gruppenmanagement System auf, während das eigentliche Streamen der Daten und die Verwaltung der *Stripes* sowie verschiedene Richtlinien für das gesamte Netz von *SplitStream* erledigt werden.

4.2.1 Pastry

Pastry bildet als Peer-to-peer Framework die Grundlage des gesamten SplitStream-Systems. Seine Aufgaben beinhalten den Aufbau und die Verwaltung des Overlays sowie das Routing von Paketen innerhalb des Netzes [5].

Jede Pastry-Node erhält eine zufällig vergebene 128-bit lange *nodeId*, die als Ziffernfolge zur Basis 2^b (mit b als Konfigurationsparameter) vorstellbar sind. Geroutet werden *Messages*, applikationsabhängige Datenpakete anhand von *Keys*, die ihren Namensraum mit den NodeIds teilen. Pastry routet Messages innerhalb des Netzes zu der Node, deren NodeId sich numerisch am wenigsten vom gegebenen Key unterscheidet. Für ein effizientes Routing fließen Nachbarschaftsbeziehungen in der zu Grunde liegenden Netzwerkstruktur mit in die Routing-Entscheidungen ein.

Jede Node eines Pastry-Netzes hält in einer Tabelle verschiedene für das Routing und das Management des Overlays notwendige Informationen. Diese beinhalten eine Routing-Tabelle, ein *leaf set*, und ein *neighborhood set*.

Im *leaf set* L werden $|L|$ Nodes eingetragen deren NodeId die größte Übereinstimmung mit der eigenen NodeId aufweisen. Im Gegensatz dazu enthält das *neighborhood set* die NodeIds der physikalisch nahesten Nodes. Die Routing-Tabelle selbst ist organisiert in $\lceil log_{2^b} N \rceil$ Zeilen mit jeweils 2^b Spalten. Jede Zeile i enthält NodeIds die ein gemeinsames Präfix von i Ziffern mit der eigenen NodeId teilen. Die Ziffer an Position $i + 1$ der jeweiligen NodeIds bestimmt, in welche der $2b$ Spalten sie eingetragen wird. Felder zu denen keine bekannte Node passt bleiben leer.

Eine ankommende Nachricht wird geroutet, in dem ihr Key zunächst mit den NodeIds des *leaf set* verglichen wird. Fällt der Key numerisch in das Intervall der NodeIds des *leaf set* wird die Nachricht an die Node des *leaf set* weitergeleitet, deren NodeId die geringste Differenz zum Key der Nachricht aufweist.

Kann auf diesem Wege kein Match erzielt werden (Key außerhalb des Intervalls) konsultiert die Pastry-Node ihre Routing-Tabelle. Die passende Node wird hier mit einem Longest-Prefix-Match mit dem Key gefunden. Dabei werden zunächst Nodes betrachtet, deren NodeIds ein längeres Präfix mit der zu routenden Nachricht gemeinsam haben. Kann kein Match erzielt werden, wird die Suche auf Nodes ausgeweitet, deren NodeIds das gleiche Präfix wie die aktuelle Node mit dem Key der Nachricht teilen. Gesendet wird an die Node mit der numerisch kleinsten Differenz zum Key der Nachricht. Pastry garantiert das

Konvergieren des Routing-Verfahrens solange nicht große Teile des Netzes gleichzeitig ausfallen.

Um auch in der Praxis eine gute Routing-Performance zu erreichen fließen Lokalitätsbetrachtungen in die Pastry Routing-Entscheidungen ein. Eine Liste mit im unterliegenden Netzwerk nahen Nodes führt jede Pastry-Node in ihrem *neighborhood set*. Die darin enthaltenen Nodes können von anderen Nodes benutzt werden, um ihre Position im Netz zu bestimmen. Zusätzlich versucht jede Node in ihre Routingtabelle NodeIds von möglichst nahe liegenden Nodes aufzunehmen. Dies geschieht nicht nur beim Eintritt in ein Pastry-Netz sondern während des gesamten Betriebs werden Routing-Einträge überprüft und eventuell aktualisiert. Die Bestimmung der Distanz einer Node zu einer anderen übernimmt Pastry nicht selbständig, sondern erwartet von der nutzenden Applikation diese Funktionalität zu implementieren.

Möchte eine neue Node dem Netz beitreten, muss ihr wenigstens eine weitere (nahe gelegene) Node bekannt sein, welche den Auftrag erhält eine spezielle Nachricht durch das Netz zu routen, deren Key die NodeId der neuen Node bildet. Alle auf dem Weg der Nachricht besuchten Nodes senden daraufhin ihren Status zurück an die neue Node, so dass diese aus den erhaltenen Informationen ihre eigenen Tabellen initialisieren kann.

Pastry erreicht durch dieses Routingschema relativ gute Ergebnisse im Hinblick auf effiziente Routen. Innerhalb des Overlays lassen sich $log_2{}^bN$ Hops erwarten um eine Nachricht zu routen. Durch die vorhandenen Lokalitätsbetrachtungen von Pastry werden im darunterliegenden Netz Routen gefunden die im Schnitt die doppelte Distanz zur optimalen Route aufweisen. Längere Routen im unterliegenden Netz sind dabei zumeist auf die lokalen Routing-Entscheidungen zurückzuführen, die jede Pastry-Node trifft, ohne auf globale Auswirkungen zu achten. So wählt Pastry in jedem Routing-Schritt zwar einen optimalen Weg, kann aber die globale Optimalität einer Route nicht garantieren, und im Allgemeinen auch nicht erkennen.

4.2.2 Scribe

Als ein weiterer Bestandteil des SplitStream-Frameworks bildet Scribe ein auf Pastry basierendes Gruppenmanagement und Multicast-System [6]. Dafür nutzt Scribe viele Eigenschaften von Pastry um Multicast-Trees so effizient wie möglich zu generieren. Scribe

unterstützt innerhalb eines Pastry-Overlays mehrere Gruppen mit einer beliebigen Anzahl von Mitgliedern, wobei auch sehr dynamische Gruppenmitgliedschaften möglich sind.
Scribe-Nodes können - sofern sie notwendige Berechtigungen besitzen - Multicast-Gruppen erzeugen, bestehenden Gruppen beitreten und sie auf Wunsch wieder verlassen. Um an eine bestimmte Gruppe zu senden, müssen die Sender nicht notwendigerweise Mitglied der Gruppe sein.
Scribe nutzt pseudo-zufällige Pastry-Keys (*groupId)* zur Identifizierung der einzelnen Gruppen. Die Pastry-Node, deren NodeId numerisch die kleinste Differenz zu einer GroupId aufweist bildet einen Rendevouz-Punkt für das Senden von Multicast-Daten an diese Gruppe und gleichzeitig die Wurzel des entstehenden Baums. Um Node-Ausfällen entgegenzuwirken können die benötigten Informationen auf nahe gelegenen Nodes repliziert werden. Scribe-Nodes führen eine Liste von Gruppen, die ihnen bekannt sind und für jede dieser Gruppen eine Liste von unmittelbaren Nachbarn innerhalb des Pastry NodeId-Raums die Kinder innerhalb des Baums bilden. Jede Scribe-Node hat die Aufgabe, die Multicast-Daten der einzelnen Gruppen an die Liste ihrer jeweiligen Kinder zu verteilen.
Eine Scribe-Node tritt einer Multicast-Gruppe bei, indem sie eine *JOIN* Nachricht an die gewünschte GroupId sendet. Diese Nachricht wird von Pastry zu der als Wurzel für diese Gruppe agierenden Node geroutet. Alle auf dem Weg von der Nachricht besuchten Nodes werden *Forwarder* für diese GroupId. Jede Node nimmt dabei ihren Vorgänger (von dem sie die Nachricht direkt erhalten hat) in die Liste von Kindern für die Gruppe auf, sollte die betreffende Node dort noch nicht eingetragen sein. Falls die Gruppe selbst noch nicht bekannt ist, wird sie einfach der Liste bekannter Gruppen hinzugefügt.
Der Multicast-Tree für eine Gruppe entsteht so aus der Konvergenz der Routen aller Gruppenmitglieder zur als Rendevouz-Punkt für die Gruppe bestimmten Node.
Die Multicast-Daten werden auf umgekehrtem Weg von der Wurzel aus verteilt (reverse path forwarding). Dabei verhindern die Pastry Routing-Eigenschaften das Entstehen von Kreisen und stellen die Konstruktion eines korrekten Baums sicher.
Das Verlassen einer Multicast-Gruppe geschieht auf einem ähnlichen Weg wie der Beitritt auch, wobei die austretende Node den Baum endgültig nur verlassen kann, wenn sie keine anderen Nodes als Kinder mit Daten versorgen muss.
Ausfälle von Nodes oder Netzwerkverbindungen kann Scribe durch periodische Heartbeat-Signale innerhalb der Multicast-Strukturen erkennen. Nodes deren Parent ausfällt, treten der Gruppe durch erneutes Senden einer *JOIN* Nachricht wieder bei, wodurch eine neue Parent-

Node von Pastry gefunden wird.
Um den Aufbau des Multicast-Trees einer Gruppe effizienter zu gestalten existieren alternative Algorithmen in Scribe zum Aufbau eines Baums. Für viele kleine Gruppen kann *scribe collapse* benutzt werden, um die einzelnen Bäume effizienter zu gestalten. Dabei werden nach dem Aufbau eines Baums Nodes die nur als *Forwarder* dienen, d.h. selbst nicht Mitglied der Gruppe sind, und nur ein Kind mit Daten versorgen wieder aus der Struktur des Baums entfernt. Eine weitere Alternative besteht darin, ein ähnliches Verfahren bereits beim Aufbau des Baums anzuwenden. Dafür werden nicht alle nach dem Senden eines *JOIN* besuchten Nodes als *Forwarder* genutzt, sondern die erste besuchte Node, die bereits Mitglied der Gruppe ist, adoptiert die neue Node als Kind. Allerdings wird durch dieses Verfahren die Fähigkeit von Scribe eingeschränkt viele gleichzeitige Beitritte zu einer Gruppe sinnvoll zu behandeln, da einzelne Nodes stärker belastet werden.

4.2.3 SplitStream

Die oberste Schicht des SplitStream-Systems bildet die eigentliche SplitStream-Software. Sie ist für die Konstruktion und die Verwaltung je eines Multicast-Trees pro Stripe zuständig. SplitStream versucht dabei zu erreichen, dass jede Node nur in einem der Bäume einen internen Knoten bildet. Bandbreiteneinschränkungen von Nodes gilt es zusätzlich zu berücksichtigen [4]
Durch die Einschränkungen der Netzwerkbandbreite, die Nodes bereit sind dem System zur Verfügung zu stellen, kann SplitStream nicht in jedem Fall die Konstruktion eines korrekten "Waldes" garantieren. Eine notwendige Bedingung für die Möglichkeit zur Konstruktion von Bäumen mit disjunkten internen Knoten besteht in der Verfügbarkeit einer ausreichenden Forwarding-Kapazität im Netz. Das heißt, die Summe der Stripes die alle Nodes zu empfangen wünschen muss kleiner oder gleich der Summe der von jeder Node weitergegebenen Stripes sein. Allerdings stellt dies noch keine hinreichende Bedingung dar, da einzelne Stripes trotz Erfüllung dieser Bedingung eventuell überhaupt nicht weitergegeben werden. Daher muss noch dafür gesorgt werden, dass Nodes mit genügend ausgehender Bandbreite verschiedene Stripes empfangen, um dem Algorithmus zur Konstruktion der Bäume Möglichkeiten einzuräumen die Last im Netz zu verteilen, und damit zu bestimmen, welche Node welchen Stripe an andere Nodes weitergibt. Im Allgemeinen ist dies in einer kooperativen Umgebung erfüllt, da Nodes normalerweise egoistisch in dem Sinne handeln,

dass sie mehr Daten konsumieren als sie bereit sind weiterzugeben. Sind beide Bedingungen erfüllt, kann SplitStream mit hoher Wahrscheinlichkeit einen "Wald" von Multicast-Trees generieren, der die Bandbreitenzwänge der Nodes beachtet und dessen Bäume interne disjunkte Knoten besitzen. Um dies zu erreichen bedient sich SplitStream der Eigenschaft von Pastry, Pakete an Nodes zu routen, deren NodeId ein längeres Präfix mit dem Schlüssel der Nachricht teilen als die aktuelle Node.

Jeder Stripe wird als separate Scribe Multicast-Gruppen verwaltet. Um Bäume für k Stripes zu generieren erzeugt SplitStream k Scribe-Gruppen, deren GroupId (*stripeId*) sich möglichst bereits in der ersten Ziffer unterscheiden. Ideale Bedingungen lassen sich erreichen, wenn der Pastry-Parameter b so gewählt wird, dass $k = 2^b$. Durch eine gleichmäßige Verteilung der Pastry-NodeIds innerhalb des gesamten Raums der möglichen Ids lässt sich so erreichen, dass jeder Stripe StripeIds und NodeIds enthält, die sich in der ersten Ziffer gleichen und von der ersten Ziffer der NodeIds / StripeIds aller anderen Stripes unterscheiden. Damit ist das erste Ziel von SplitStream erreicht, Bäume mit internen disjunkten Knoten zu konstruieren.

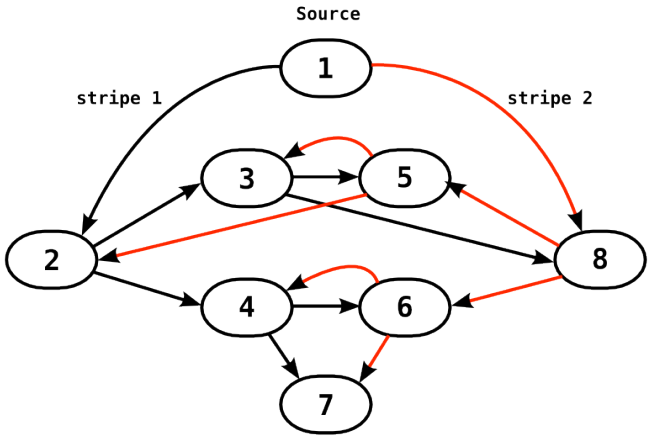

Abbildung 5: Multicast-Struktur von SplitStream

Um Überlastung von Nodes zu vermeiden und jeder Node die Möglichkeit zu geben, ihre ausgehende dem Netz zur Verfügung gestellte Bandbreite zu kontrollieren, kann jede Node die Anzahl ihrer Kinder begrenzen. Hat eine Node die Maximalzahl unterstützter Kinder erreicht, und erhält sie eine *JOIN* Nachricht einer weiteren neuen Node, wird die neue Node zunächst als Kind aufgenommen. Danach entscheidet die nun überlastete Node nach verschiedenen Kriterien von welchem ihrer Kinder sie sich trennt. Zunächst werden Kinder betrachtet, deren StripeId kein gemeinsames Präfix mit der eigenen NodeId teilen. Existiert kein solches Kind, verstößt die Node das Kind, dessen NodeId den kürzesten Präfix-Match mit der eigenen NodeId liefert. Das verstoßene Kind sucht daraufhin unter seinen bisherigen "Geschwistern" nach einer neuen Eltern-Node. Da jede Node den gleichen Algorithmus benutzt, setzt sich dieses Verhalten rekursiv den Baum entlang in Richtung der Blätter fort.

Findet die verstoßene Node auf diesem Wege keinen Anschluss, greift ein weiterer

SplitStream Mechanismus, der die Effizienz des Netzes erhöhen soll.
Neben den eigentlichen Gruppen für jeden Stripe richtet SplitStream eine weitere Scribe-Gruppe ein, die *Spare Capacity Group*. In dieser Gruppe finden sich alle Nodes, deren ausgehende Kapazität nicht vollständig ausgenutzt wird. Nodes dieser Gruppe nehmen ansonsten verwaiste Nodes auf, die keine Möglichkeit haben, bestimmte Stripes auf dem "regulären" Weg zu empfangen. An die Aufnahme sind bestimmte Bedingungen geknüpft, die sicherstellen, dass die Struktur des Overlays konsistent und beispielsweise kreisfrei bleibt. Die Besonderheit der Spare Capacity Group besteht darin, dass sie Nodes ermöglicht in mehr als nur einem Baum eine interne Node zu bilden, sollte sie genügend Ressourcen zur Verfügung haben.
Kann eine Node auch auf diesem Wege keine Eltern-Node finden, bleibt die Node vom Service ausgeschlossen, und die Applikation wird davon unterrichtet, dass keinerlei Forwarding-Kapazität im Netz zur Verfügung steht.
Ausfälle von Nodes werden von SplitStream ähnlich wie von Scribe behandelt. Alle Nodes, die Kinder ausgefallenen Node waren, versuchen durch erneute *JOIN* Nachrichten neue Eltern zu finden, oder weichen auf die Spare Capacity Group aus, sollte keine Eltern-Node gefunden werden.

4.3 Fazit

Beide hier vorgestellten Systeme nutzen grundlegend verschiedene Strategien und Algorithmen um ein zuverlässiges, skalierbares und effizientes Application Layer Multicast zu ermöglichen. Trotz unterschiedlicher Zielstellung beider Systeme müssen sie die gleichen Probleme lösen. Allerdings sind vorher getroffene Annahmen und Nutzungsszenarien mit in das Design jedes Systems mit eingeflossen, was sich auch in den Vor- und Nachteilen widerspiegelt.
Eine optimale Lösung aller beim Streamen von Multimediadaten per Multicast auftretenden Probleme kann keines der Systeme für sich in Anspruch nehmen. Der Aufbau der Multicast-Strukturen und deren Aufrechterhaltung im Betrieb ist nicht durch einen optimalen Algorithmus möglich und muss so von Fall zu Fall entschieden werden.

5 On-Demand Streaming ist anders

An ein On-Demand-Streaming System werden im Allgemeinen andere Anforderungen gestellt als an ein Live-Streaming System [8]. Dies liegt hauptsächlich in der Asynchronität der Streaming-Zugriffe begründet. Während in Live-Streaming Systemen alle Clients ihre Daten quasigleichzeitig erhalten, ermöglicht On-Demand-Streaming einen zeitlich wahlfreien Zugriff auf Datenbestände. Dies widerspricht grundsätzlich der Natur einer Multicast-Anwendung.

Dennoch gibt es Versuche auch in On-Demand-Streaming Systemen die Effizienz durch Multicast zu erhöhen. Da Daten per Multicast synchron übertragen werden, muss in solchen Systemen zunächst dafür gesorgt werden, dass mehrere Clients zeitgleich auf Daten zugreifen, um die Vorteile einer Multicast-Anwendung zu nutzen. Mehrere Alternative Verfahren stehen hierfür, je nach Anwendungsfall zur Auswahl.

Das traditionellste Verfahren besteht im *periodic broadcast*. Ein Server sendet mehrere zeitverschobene Multicast-Sessions in ständiger Wiederholung mit denen sich Clients verbinden können. *Batching* aggregiert mehrere Clients zu einer Multicast-Session. Beide Verfahren sind allerdings eventuell mit langen Wartezeiten für die Nutzer verbunden.

Eine Verbesserung kann durch *Patching* erreicht werden. Dabei wird Clients die Möglichkeit gegeben, zu einer Multicast-Session aufzuschließen. Die anfänglich fehlenden Daten empfängt der Client bis zum Anschluss an die Multicast-Gruppe per Unicast. Eine Weiterführung dieser Idee besteht im *Merging*, wobei kleine verschiedene Multicast-Sessions über die Zeit zu immer größeren Gruppen zusammengelegt werden. Caching-Strategien spielen hierbei eine herausragende Rolle um die Clients synchronisieren zu können. In Systemen wie oStream wurde die Nutzbarkeit von Application Layer Multicast für ein On-Demand-Streaming System näher untersucht.

6 Quellenverzeichnis

[1] Holbrook, H..
Multicast: Past, Present, Future.
In Talk at UCB, 21. Februar 2003

[2] D. A. Tran, K. A. Hua, and T. T. Do.
Zigzag: An efficient peer-to-peer scheme for media streaming.
In IEEE INFOCOM, 2003

[3] Castro, M., Druschel, P., Kermarrec, A., Nandi, A.,Rowstron, A. and Singh,A..
SplitStream: High-bandwidth content distribution in a cooperative environment.
In Proceedings of (IPTPS'03) (February 2003)

[4] Castro, M., Druschel, P., Kermarrec, A., Nandi, A.,Rowstron, A., Singh, A..
Splitstream: High-bandwidth multicast in cooperative environments.
In 19th ACM Symposium on Operating Systems Principles, 2003

[5] Rowstron, A.,Druschel, P..
"Pastry: Scalable, Decentralized Object Location and Routing for LargeScale
Peer-to-Peer Systems",
18 Conference on Distributed Systems Platforms, Heidelberg (D), 2001.

[6] Castro, M., Druschel, P., Kermarrec, A.-M., Rowstron A..
SCRIBE: A large-scale and decentralized application-level multicast infrastructure.
IEEE Journal on Selected Areas in communications (JSAC), 2002.

[7] Y. Cui, B. Li and K. Nahrstedt.
oStream: Asynchronous streaming multicast in application-layer overlay networks.
In IEEE JSAC special issue on Recent Advances in Service Overlay, 2003

[8] Deering, S.
Host Extensions for IP based Multicast
Request for Comments RFC 1112
http://rfc.net/rfc1112.html

[9] Internet Assigned Numbers Authority
nternet Multicast Addresses
http://www.iana.org/assignments/multicast-addresses