Technische Universität Ilmenau

Fakultät für Informatik und Automatisierung

Hauptseminar Telematik

Sommersemester 2004

Thema:
Scheduling

Bearbeiter : Alexander Müller
Studiengang : Informatik 97
Matrikelnummer : 26171

Inhaltsverzeichnis

1. Einführung

Diese Ausarbeitung zum Thema Scheduling: Welche Fragmente eines Streams werden von welcher Quelle angefordert beschäftigt sich mit Schedulingalgorithmen für Peer-to-Peer Netzwerke im Allgemeinen. Dabei kann auf ganz verschiedene Probleme eingegangen werden und versucht werden, durch geeignete Schedulingverfahren, Lösungen zu finden. Da diese Thematik schnell sehr komplex wird, werden sich hier neben dieser Einleitung, in der versucht wird die immense Größe des Problems etwas zu erläutern, nur einige Verfahren wieder finden.
Zunächst stellt sich die Frage: warum benötigt man Scheduler? Dazu schaut man sich ein herkömmliches Netzwerk oder das Internet an und stellt sehr bald fest, dass die unterschiedlichen Voraussetzungen der Hardware verschiedener Rechner in Einklang gebracht werden müssen. Im Internet gibt es immer noch Nutzer von Analogmodems, welche nur geringe Bandbreite für ein Peer-to-Peer Streaming beisteuern würden, während auf der anderen Seite Anwender, die z.B. in einem Universitätsnetz einen Rechner haben, bis zu 100 MBit Anbindungen haben. Doch nicht nur die unterschiedlichen Netzzugangsmöglichkeiten fordern einen Scheduler, es existieren noch viele weitere Probleme, auf die Rücksicht genommen werden kann und wo man versuchen kann, durch Betrachtung dieser, das ganze Peer-to-Peer Streaming System effizienter zu gestalten.
Neben der grundlegenden Tatsache, dass jeder Schedulingalgorithmus schnell abzuarbeiten sein soll, steht man im allgemeinen vor folgenden Problematiken: Da in einem Peer-to-Peer System von einem oder auch mehreren Server-Peers Daten empfangen werden können, muss es eine effiziente Zuweisung der Server an den Clienten geben, wobei wiederum verschiedene Sachen wie Entfernung der Server zum Client, Upstream Bandbreite der zur Verfügung stehenden Server oder deren Beeinflussung untereinander, eine Rolle spielen. Weiterhin sollte der Ausfall eines Servers kompensiert werden, bevor der Stream abbricht und jeder einzelne Peer im Netzwerk soll einem Serververhalten von herkömmlichen Server-Clienten Systemen nicht ausgesetzt sein. Eine weitere Problematik in einem Peer-to-Peer System ist sicher das Verhältnis zwischen den bereitgestellten Datenfragmenten und den angeforderten. Um dieses Verhältnis schnell zu vergrößern, ist es hilfreich, schnell

vielen Peers die Daten zu übermitteln, damit diese dann wiederum als Server-Peers mitwirken können, anderen Nutzern die Daten zur Verfügung zu stellen. Es wird hierbei von einer schnellen Kapazitätserhöhung, die anzustreben ist, gesprochen. Dazu liefert das Verfahren Differentiated Distributed Admission Control (DAC) im Abschnitt 2.2. eine Lösung. Zuletzt noch ein sehr wesentlicher Teil, den ein Scheduler zu verwalten hat und welcher auch in dieser Arbeit eine wichtige Rolle spielt, die Stream-Fragmentzuteilung auf die verschiedenen Quellen. Hiermit beschäftigt sich auch der folgende Algorithmus Optimal Media Data Assignment (OTS).

2.1. spezielle Verfahren - Optimal Media Data Assignment Algorithm (OTS)

Der Algorithmus Optimal Media Data Assignment wurde an der Purdue University in West Lafayette, Indiana, USA entwickelt und beschäftigt sich mit der Zuteilung der Fragmente eines Streams auf die verschiedenen Quellen, wobei diese Quellen vorher schon zugewiesen wurden. Dazu werden folgende Vereinbarungen getroffen. Ein Client kann an einer Streaming Session nur teilnehmen, wenn seine Downstream Bandbreite mindestens der Bandbreite entspricht, welche für das Abspielen des Stream nötig ist. Diese wird im Folgenden mit R_0 bezeichnet. Alle teilnehmenden Server Peers nehmen nur an maximal einer Streaming Session teil und werden in n unterschiedliche Klassen nach ihrer Upstream Bandbreite eingeteilt. Die Klassenhierarchie wird wie folgt festgelegt. Ein Server-Peer, der eine Upstream Bandbreite von mindestens der Hälfte der zum Abspielen benötigten Bandbreite liefern kann, wird in Klasse 1 eingeteilt. Server-Peers, die eine Bandbreite zwischen einem Viertel und der Hälfte der zum Abspielen benötigten Bandbreite liefern, in Klasse 2.

$$\left\{\frac{R_0}{2^1};\frac{R_0}{2^2};...;\frac{R_0}{2^n}\right\} \equiv Klasse1;...;KlasseN$$

Wenn alle Quellen auf solche n Klassen zugeteilt wurden, kann ein optimaler Fragmentzuweisungsalgorithmus schnell gefunden werden.

In Bild 1a wird ein Scheduler gezeigt, der auf Klasse 1 Server-Peer P^1, auf Klasse 2 Server-Peer P^2 und auf Klasse 3 Server-Peers P^3 und P^4 eine Aufteilung trifft, indem die Fragmente einfach der Reihe nach ihrer Benennung zugewiesen werden. Das heißt P^1 spielt jedes 8k, 8k+1, 8k+2, 8k+3 Fragment ab, P^2 jedes 8k+4, 8k+5 Fragment. Server P^3 und P^4 spielen jedes 8k+6 bzw. 8k+7 Fragment des Streams. Das Abspielen des Streams auf dem Server kann hier

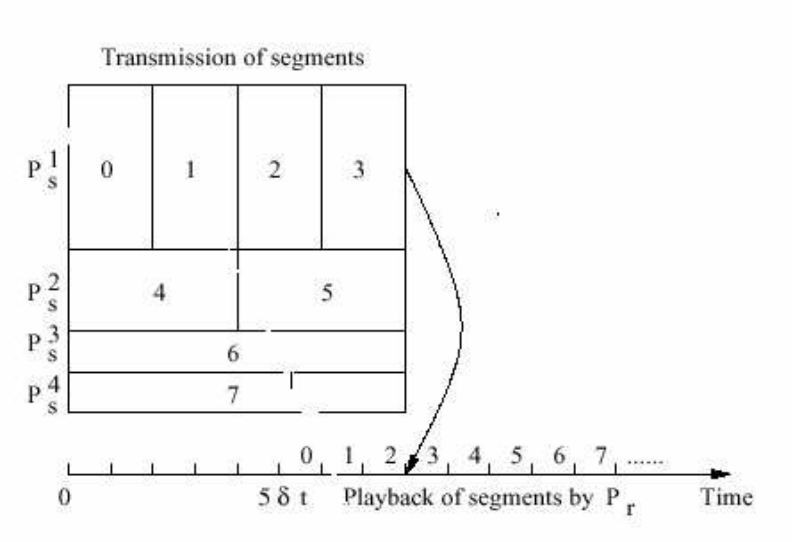


Bild 1a

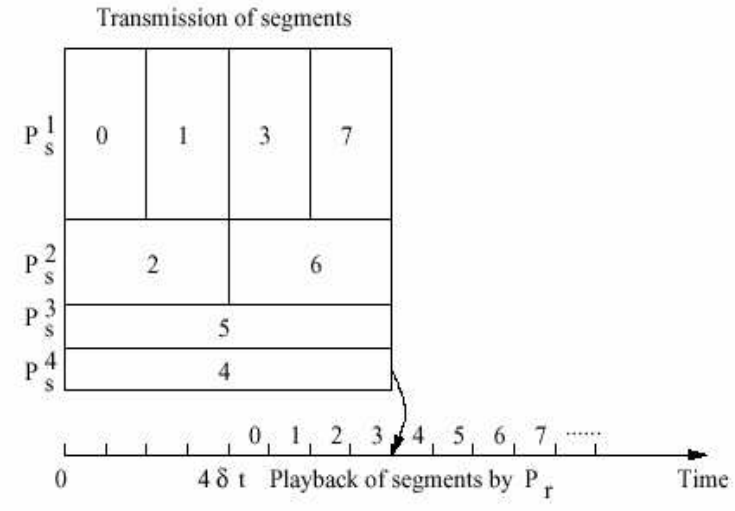


Bild 1b

nach fünf Zeiteinheiten beginnen, ohne dass danach noch einmal eine Pause eingelegt werden muss und der Stream ab da vollständig und kontinuierlich beim Client ankommt. In Bild 1b, welches den OTS zugrunde legt, beginnt das Abspielen des Streams bereits nach 4 Zeiteinheiten. Dies wurde lediglich durch eine Umordnung der Fragmente erzielt und ist auch bewiesenermaßen optimal. Die Anwendung von OTS bringt also immer eine optimal kleine Verzögerungszeit. Bild 1c zeigt den Pseudocode des Algorithmus OTS, wobei hier m die Anzahl der verwendeten Server sind und n die Anzahl der Klassen, auf die diese aufgeteilt wurden.

```
OTS_p2p(P_r, {P_s^1, P_s^2 ... P_s^m}) {
    i = 2^n − 1;
    while (i ≥ 0) {
        for j = 1 to m
            if the assignment to P_s^j is not complete {
                Assign segment i to P_s^j;
                i = i − 1;
            }
    }
}
```

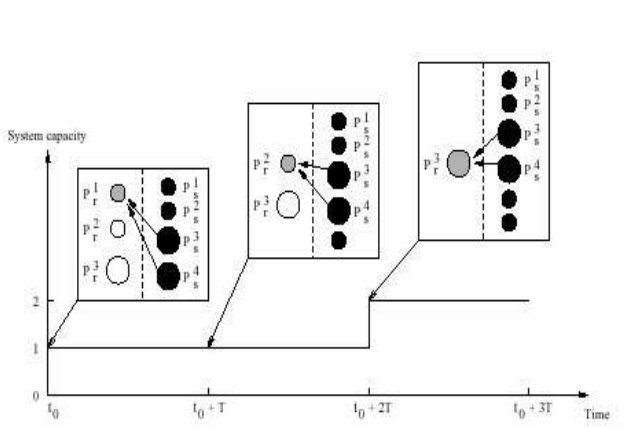
Bild 1c

Der Optimal Media Data Assignment Algorithm ist schnell auszuführen und läuft auf dem Client, welcher dann die jeweiligen Fragmentanforderungen an seine Server überträgt. Nun stellt sich natürlich die Frage ob OTS das Problem der Fragmentzuteilung, welches ja ein wesentlicher Bestandteil des Schedulings im allgemeinen Sinne darstellt, immer optimal löst. Das wesentliche an diesem Algorithmus ist, dass die Upstream Bandbreiten der Quellen auf für eine Rechnung günstige Werte herabgerundet werden, so dass das NP-schwere Problem der optimalen Fragmentaufteilung auf Server mit unterschiedlichen Bandbreiten auf ein Problem der Fragmentaufteilung auf Servern mit gerundeten Bandbreiten reduziert wird und somit das Problem schnell gelöst werden kann. Bezahlt wird das schnelle

Lösen des Problems mit der Nichtverwendung der Bandbreite über den Klassengrenzen, welche die Server ja eigentlich noch zur Verfügung hätten.

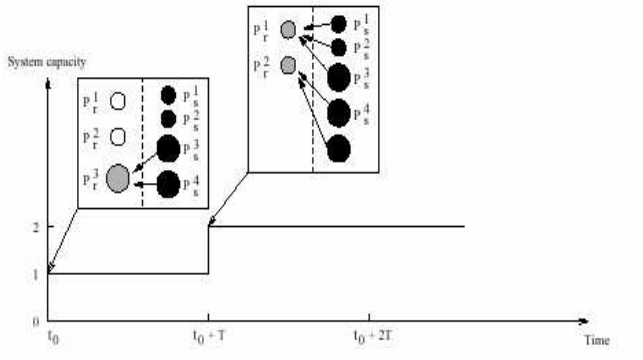
2.2. spezielle Verfahren - Distributed Differenciated Admission Control (DAC)

Wie schon in der Einleitung erwähnt ist das schnelle Erhöhen der Kapazität eines Streaming Systems günstig für das weitere Streamen, denn mit einer höheren Kapazität in einem System wird die Wahrscheinlichkeit, dass ein Client nicht bedient werden kann minimiert. Die Kapazität zu einem Zeitpunkt ist davon abhängig, welche Clienten bis zu diesem Zeitpunkt bedient wurden. Intuitiv ist es für ein schnelles Kapazitätswachstum von Vorteil, wenn zunächst Clienten mit hohen Upstream Bandbreiten bevorzugt behandelt werden, weil diese dann ja schnell weitere Clienten bedienen können. Bild 2b verdeutlicht dies. Im Gegensatz zu der Reihenfolge im Bild 2a wird hier zunächst der erstklassige Peer P^3 bedient um dessen großen Upstream im weiteren Verlauf zu nutzen



(a) Admitting $P_r^1 \rightarrow P_r^2 \rightarrow P_r^3$

Bild 2a



(b) Admitting $P_r^3 \rightarrow (P_r^1 + P_r^2)$

Bild 2b

Bei Systemen mit vielen Peers führt das Bevorzugen von Peers mit hohem Upstream allerdings zu dem Problem, dass Peers mit geringem Upstream völlig vernachlässigt werden. Es wird hier von Starvation gesprochen. Peers mit niedrigem Upstream müssen sehr lange warten und haben somit eine sehr große Verzögerungszeit. Um dies zu vermeiden, aber dennoch die Vorteile einer schnellen Kapazitätserhöhung zu nutzen kann der Algorithmus DAC, welcher auch an der Purdue University, Indiana, USA entwickelt wurde, verwendet werden. Er schließt einen Kompromiss zwischen der totalen Bevorzugung von Peers mit höheren Upstream Bandbreiten und dem Bedienen der Clienten, die für das weitere Streamen im System nicht so wichtig sind. Dies geschieht mittels Wahrscheinlichkeitsberechnungen. Jeder Peer wird nach der oben genannten Art wieder in Klassen aufgeteilt und diesen Klassen werden Wahrscheinlichkeitswerte zugeteilt, wobei Klasse 1 Peers höchste Wahrscheinlichkeitswerte bekommen und umso höher die Klasse desto niedriger die Wahrscheinlichkeitswerte. Nun kann jeder Server-Peer bei Anfrage verschiedener Clienten über eine Berechnung entscheiden, welcher Peer bedient wird. Die Wahrscheinlichkeitswerte der Klassen werden während des Laufens des Systems dynamisch angepasst. Dieser ganze Prozess setzt natürlich voraus, dass DAC auf allen teilnehmenden Peers ausgeführt wird. Wird ein Client durch DAC nicht bedient, obwohl er schon einen Request gesendet hat, also das System ausgelastet ist, kann der Client seinen Servern einen Reminder senden, welcher die Server davon abhält Clienten mit niedriger Upstream Bandbreite früher zu bedienen als den Reminder sendenden Client. Weiterhin muss im ganzen System vorausgesetzt werden, dass jeder teilnehmende Peer seine Upstream Bandbreite bekannt gibt. Allerdings beinhaltet DAC ja einen Anreiz dazu, da Peers mit höheren Upstream Bandbreiten immer noch bevorzugt behandelt werden.
DAC wurde mit guten Resultaten an der Purdue University getestet. Dabei wurde in einem 144 Stunden Test eine Testumgebung mit 100 Klasse 1 Anfangsservern, 50000 Clienten, davon jeweils 10% Klasse 1 und 2 und 40% Klasse 3 und 4, aufgebaut und eine 60 Minuten Videodatei gestreamt. Jeder Client bekam Anfangs acht zufällig ausgewählte Server gestellt.

Zum Test hier ein paar Ausschnitte. Zunächst zur Kapazitätserhöhung im Streaming System. Bild 3 zeigt, dass die Kapazität unter Verwendung von DAC schneller steigt als bei Nichtverwendung.

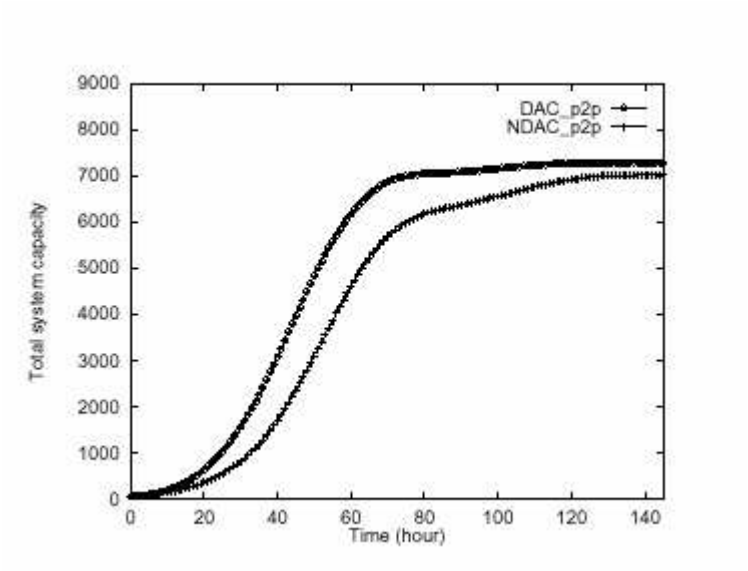
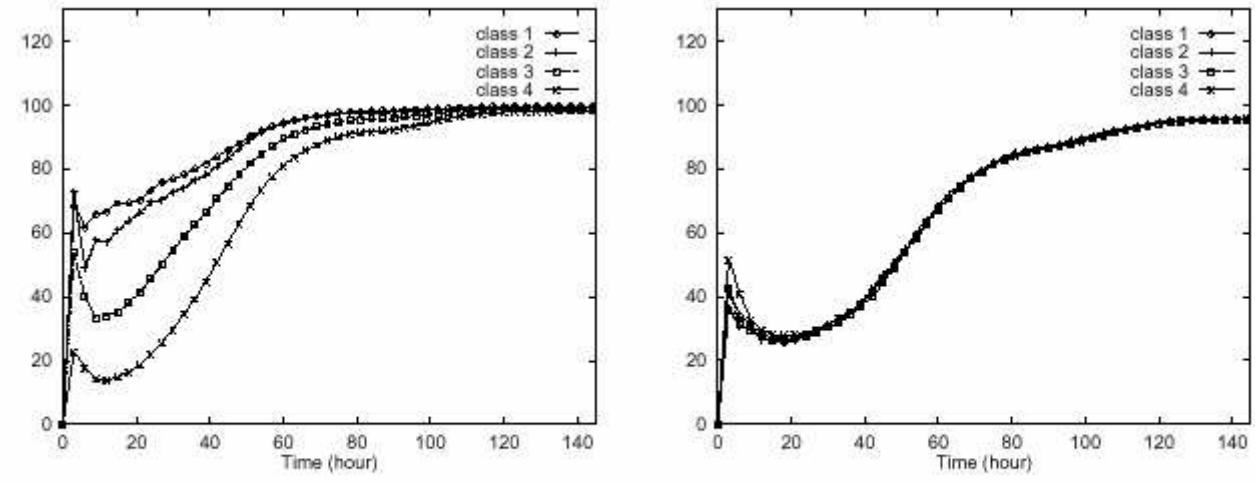


Bild 3

Im Bild 4 sind die Bedienraten der jeweilig klassigen Client-Peers dargestellt, wobei in Bild a DAC verwendet wurde und in Bild b nicht.

Bild 4



(a) Arrival Pattern 2, using DAC_{p2p}

(b) Arrival Pattern 2, using $NDAC_{p2p}$

Client-Peers der Klassen 1,2 und 3 werden bei Verwendung von DAC schon von Beginn an immer besser bedient als beim Nichtverwenden von DAC. Client-Peers der Klasse 4 werden Aufgrund ihrer niedrigen Wahrscheinlichkeitswertzuteilung bei DAC in den ersten Stunden schlechter bedient. Die Bedienung der Client-Peers mit höheren Upstream Bandbreiten kommt den Klasse 4 Clienten allerdings später auch zu Gute, weil diese dann durch die höhere Gesamtkapazität des Systems besser bedient werden können. Letztendlich begünstigt DAC alle Peers im Streaming System.

2.3. spezielle Verfahren – Peer-to-Peer Adaptive Layered Streaming (PALS)

Ein weiteres Verfahren, welches ein fortlaufendes Streaming sichern soll, ist das Peer-to-Peer Adaptive Layered Streaming (PALS). Bild 5a zeigt den allgemeinen Aufbau eines solchen Systems.

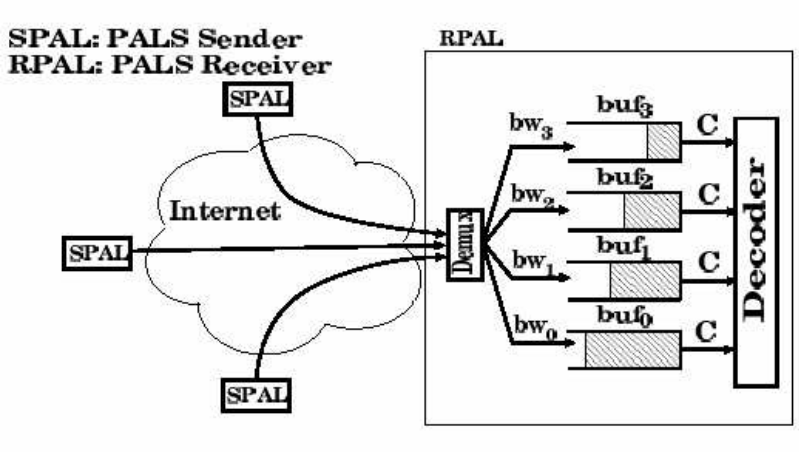


Bild 5a

Beim PALS ist der grundlegende Ansatz der, dass die Qualität des Streams geändert werden kann, wenn sich während der Laufzeit der Durchsatz der Quellen ändert. Dieser Quality Adaption Ansatz geht dabei davon aus, dass ein kurzzeitiger Abfall des Durchsatzes durch einen Puffer kompensiert wird und erst ein langfristiger Abfall

ein Herabsetzen der Streamqualität zur Folge hat. In einer Phase, in der der Durchsatz der Quellen höher ist als die Bandbreite, die für das Abspielen des Streams gebraucht wird, wird der Puffer gefüllt. Wenn diese Füllphase lang genug andauert und der Puffer einen bestimmten Wert erreicht hat, wird die Qualität des Streams verbessert. Ab hier wird ein höherer Durchsatz benötigt, um die Streamqualität aufrecht zu erhalten. In dem Fall, dass der Durchsatz geringer ist als die Bandbreite, die zum Abspielen des Streams benötigt wird, wird der Puffer zur Qualitätssicherung genutzt. Beim Unterschreiten eines bestimmten Wertes wird die Qualität des Streams verringert und nun wird auch nur ein geringerer Durchsatz nötig sein, um den Stream in aktueller Qualität abzuspielen.
Das anfordern der verschiedenen Fragmente des Streams von den Servern wird bei PALS über eine Sliding Window Methode praktisiert. Neben der Synchronisation der Sender wird dadurch auch verhindert, dass die Server noch ältere Daten oder nichts senden. In Bild 5b wird die Sliding Windows Methode erläutert.

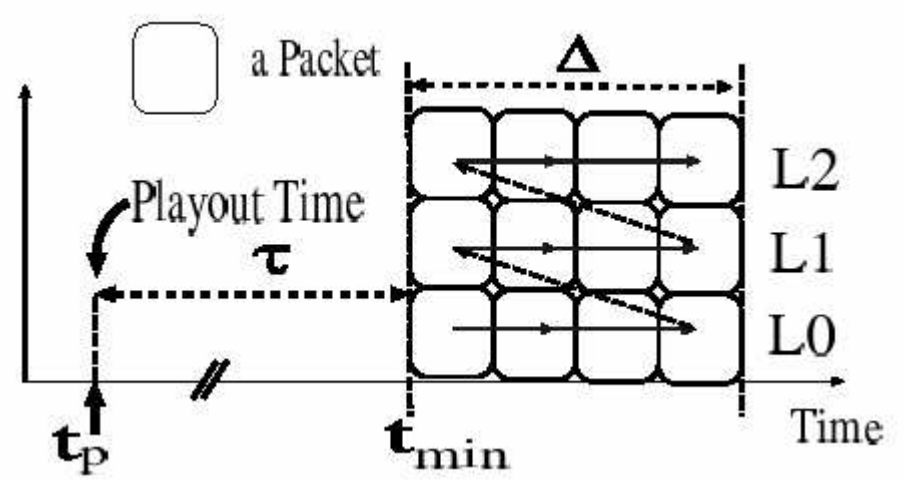


Bild 5b

Um T Sekunden vor der aktuellen Abspielzeit zu bleiben wird das Window periodisch alle Δ Sekunden nach vorn verschoben. Wenn der Durchsatz eines Senders dabei überschätzt wird, wird nach Δ Sekunden eine weitere Anfrage an den Sender gestellt und somit verhindert, dass dieser zu alte Daten sendet. Um zu vermeiden, dass ein Sender nichts tut, wird das komplette Senden überwacht und bei etwaigen IDLE

Zustand eine neue Anfrage geschickt. Das Überschreiben von alten Sendeanforderungen durch neue und das Überwachen wird durch ein Reverse Flow Control realisiert. Die Anforderungen der Daten von den Sendern erfolgt über ein Round Robin Prinzip und geschieht quantitativ proportional zu dem zuletzt erreichten Durchsatz des jeweiligen Senders. Ein weiteres Feature des PALS ist das Peer Selection. Die Server-Peers werden hier durch ein Zufallsprinzip ausgewählt. Wird ein zweiter Peer zu einem ersten schon vorher aus einer Liste gewählten Peer ausgewählt und erhöht sich der Gesamtdurchsatz, so wird dieser beibehalten und ein weiterer ausgewählt. Da hier nur der Gesamtdurchsatz als entscheidendes Kriterium für die Auswahl auftaucht, kommen natürlich eine Vielzahl von Erweiterungsmöglichkeiten in Betracht. Es können weiterhin noch die Einzeldurchsätze der Server-Peers und deren Beeinflussung untereinander überwacht werden. Der Grossteil dieses PALS Systems, insbesondere die Algorithmen zur Berechnung, Überwachung und Koordination werden auf den Client-Peers ausgeführt.

2.4. spezielle Verfahren - Distributed Streaming (Rate Allocation / Packet Partition)

Ein Distributed Streaming Verfahren der Berkeley University of California zeigt weitere Möglichkeiten der Herangehensweise beim Aufbau und Scheduling von Streaming Systemen. Das Verfahren soll mit seinen beiden wesentlichen Algorithmen
Rate Allocation und Packet Partition in diesem Abschnitt beleuchtet werden.
Die zur Verwendung stehenden Bandbreiten werden durch den komplett auf den Clienten implementierten Transmission Control Protocol friendly Rate Control Algorithmus (TFRC) abgeschätzt. TFRC benutzt dazu folgende Formel, wobei die Bandbreite B von der TCP Segmentgröße s in Byte, der erwarteten Roundtrip Time R in Sekunden, dem TCP Time Out T_{rto} und der erwarteten Verlustrate r abhängig ist.

$$B = \frac{s}{R\sqrt{\frac{2p}{3}} + T_{rto}(3\sqrt{\frac{3p}{8}})\,p(1+32p^2)}$$

Der Rate Allocation Algorithmus, der im Folgenden erläutert wird, ist dafür zuständig, die Fragmente den jeweiligen Quellen zuzuteilen. Er läuft nach auf den Client-Peers. Wie im Bild 6a zu sehen, ist bei diesem Verfahren die Verlustrate und nicht der absolute Durchsatz von einem Sender zum Empfänger im Vordergrund. Sender mit niedriger Verlustrate werden dabei beim jeweiligen Empfänger intuitiv bevorzugt behandelt.

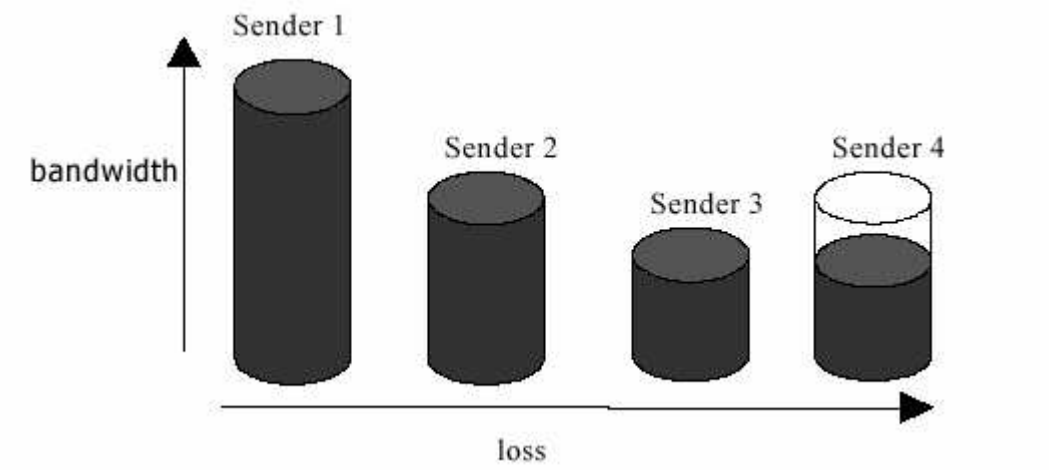


Bild 6a

Nachdem Sende- und Verlustraten für alle potenziellen Sender berechnet wurden, wird nach der Verlustrate aufsteigend sortiert, die Bandbreiten der Sender mit niedrigsten Verlustraten akkumuliert und diese dann beansprucht. Im Bild kann Sender 4, also derjenige mit höchster Verlustrate, unter seiner erwarteten Bandbreite operieren, weil die Summe der Bandbreiten der ersten 3 Sender und ein Teil von Sender 4 für das Streamen der Daten genügen.
Weiterhin wird ein Packet Partition Algorithmus auf allen teilnehmenden Sendern ausgeführt. Mit dem Ausführen auf allen Sendern soll vermieden werden, dass gleiche Pakete von mehreren Sendern gesendet werden. Zu jedem Zeitpunkt, an dem der Client die Senderate eines Senders angepasst haben möchte, empfängt der Sender ein wie in Bild 6b dargestelltes Kontrollpaket. Es enthält die Senderaten S_i für die jeweiligen i Sender, die Verzögerungen D_i und ein Syncronisationsfeld Sync.

Bild 6b

D1	D2	D3	D4	D5	S1	S2	S3	S4	S5	Sync

3. Fazit

Wie schon in den Verfahren deutlich wurde, mussten viele Einschränkungen und Vereinfachungen vereinbart werden, um die komplexe Natur des Schedulingproblems beherrschen zu können. Einen allumfassenden optimalen Algorithmus, der immer in kleinstmöglicher Zeit die beste Lösung liefert existiert nicht. Da so viele verschiedene Schedulingprobleme beim Peer-to-Peer Streaming entstehen, gibt es auch eine Vielzahl von Lösungsvarianten. Es ist somit notwendig, speziefische Algorithmen problemangepasst zu verwenden und diese etwaig auf das Problem zugeschnitten zu verändern. Die vorgestellten Algorithmen versuchen, einen Kompromiss aus optimalen Ergebnissen und schneller Berechenbarkeit einzugehen und liefern somit für spezielle Fälle sehr gute Lösungen. Abzuraten ist allerdings von der kompletten Übernahme der Verfahren, ohne diese noch einmal auf seine Bedürfnisse zugeschnitten verändert zu haben. Allerdings kann man auf die Grundkonzepte der Verfahren aufbauend sehr leicht problemangepasste Varianten finden. Weiterhin können verschiedene Algorithmen parallel oder miteinander ausgeführt werden und gerade mit OTS und DAC existieren zwei Algorithmen, die sehr gut parallel implementiert werden können. Dies wird auch beim Streamingverfahren GnuStream gemacht.

Quellen

„On Peer-to-Peer Media Streaming“ by Mohamed Hefeeda, Susanne Hambrusch, Bharat Bhargava, Dongyan Xu
Purdue University, West Lafayette, IN, USA

„Layered Peer-to-Peer Streaming“ by Yi Cui, Klara Nahrstedt
University of Illinois at Urbana-Champaign

„PALS: Peer-to-Peer Adaptive Layered Streaming“ by Reza Rejaie (University of Oregon)
Antonio Ortega (University of Southern California)

„GnuStream: A P2P Media Streaming System Prototype“ by Xuxian Jiang, Yu Dong, Dongyan Xu, Bharat Bhargava
Purdue University, West Lafayette, IN, USA

„Distributed Video Streaming over Internet“ by Thinh PQ Nguyen, Avideh Zakhor
Berkeley, CA, USA

„TCP-Friendly Unicast Rate-Based Flow Control“
by Jamshid Mahdavi & Sally Floyd

GnuStream @ http://www.cs.purdue.edu