LiveShift: Ein Ansatz für zeitversetzte P2P Multimedia-Übertragungen *

Fabio V. Hecht

Universität Zürich, Institut für Informatik (IFI)
Binzmühlestrasse 14, 8046 Zürich, Schweiz
hecht@ifi.uzh.ch

Abstract: Der Peer-to-Peer (P2P) Ansatz verbessert die Skalierbarkeit und verringert die Infrastruktur- und Verwaltungskosten von Multimedia-Übertragungen, weil die Nutzer sich an der Bereitstellung des Dienstes beteiligen. Dieser Dissertationsabriss führt LiveShift ein, einen neuartigen Ansatz, der die nahtlose Kombination von P2P-Direktübertragungen und zeitversetzter Übertragungsfunktionalität in einem einzigen System ermöglicht und der die dafür notwendigen Protokolle und Regeln umsetzt. LiveShift erfüllt die Anforderungen der Netzwerk- und Dienstverwaltung, indem neue Dienste in voll dezentralisierter Art und Weise und mit minimaler Beeinflussung des Betreibers angeboten werden. Drei Kernaspekte, die sich aus dem Ermöglichen eines solchen neuartigen Anwendungsfalles ergeben, werden genauer betrachtet: (a) Ein voll verteiltes P2P-*Protokoll* für den Austausch von direkten wie von zeitverögerten Video-Übertragungen, (b) die Untersuchung von passenden *Regelsätzen zur Wiedergabesteuerung* in verschiedenen Szenarien und unter Berücksichtigung verschiedener Parameter, und (c) der Entwurf und die Auswertung eines neuartigen, *vollständig verteilten P2P-Trackers*, der gleichermassen Effizienz und Lastausgleich verbessert. Das Aufzeigen der Brauchbarkeit der erwähnten Mechanismen durch deren Integration in einer einzigen, offen und frei verfügbaren *Anwendung*.

1 Introduction

Der Peer-to-Peer (P2P) Ansatz verbessert nachweislich die Skalierbarkeit und verringert die Infrastruktur- und Verwaltungskosten von Multimedia-Übertragungen [HLL+07], da die Bereitstellung eines Dienstes teilweise an Benutzer delegiert werden kann. Dieser Dissertationsabriss führt LiveShift [Hec] ein, einen neuartigen Ansatz, der ein P2P-Overlay sowohl für die Übermittlung von Multimedia-Übertragungen als auch für die diesbezügliche Datenhaltung benutzt. Dies ermöglicht es Peers, Übertragungen zwecks zukünftiger Verbreitung zu speichern, so dass gleichermassen nahtlose Direktübertragungen, zeitversetzte Übertragungen und Video auf Abruf (Video-on-Demand, VoD) ermöglicht werden. Damit erhalten Benutzer die Freiheit, alle verfügbaren Programme ohne vorgängige Aufnahme von jeder beliebigen Position in der Übertragung aus zu starten und uninteressante Teile der Übertragung zu überspringen, bis die Direktübertragung eingeholt wurde.

*Englischer Titel der Dissertation: “LiveShift: A Time-Shifted P2P Multimedia Streaming Approach”

Das System kann beispielsweise auf einer mit einer Festplatte ausgestatteten Settop-Box oder als Anwendung auf einem Rechner installiert werden. Da das System P2P-basiert ist, sind keine wesentlichen Infrastruktur- oder Verwaltungsinvestitionen auf Anbieterseite notwendig. Die Benutzer können selbst in Übertragungs- und Speichergeräte investieren, so wie es auch der Fall ist bei digitalen Videorekordern (Digital Video Recorder, DVR). Zusätzlich haben Benutzer mit LiveShift die Möglichkeit, gespeicherte Übertragungen mit anderen zu teilen, was die Anzahl verfügbarer Programme massgeblich vergrössert.

Die Dissertation [Hec, liv] bearbeitet mehrere Aspekte, die diesen Anwendungsfall möglich machen. Die vier Hauptbeiträge der Dissertationen sind die folgenden: (a) Ein einziges, vollständig verteiltes P2P-Protokoll [HBC+11], das fähig ist, Direktübertragungen und zeitversetzte Übertragungen schnell und integriert zu lokalisieren und zu verteilen, wird detailliert beschrieben und ausgewertet; (b) Regelsätze für die Wiedergabe (sogenannte Playback-Policies), die bestimmen, ob das System die Wiedergabe anhält oder ob es Blöcke übespring, wenn Inhalte nicht verfügbar sind, werden untersucht, verglichen und bezüglich Benutzererlebnis klassifiziert [HBSS12]; (c) ein neuartiger, vollständig verteilter P2P-Tracker – B-Tracker [HBS11] genannt – wird eingeführt als ein Tracker, der Vorteile in Bezug auf Effizienz und Lastausgleich gegenüber existierender, verteilter P2P-Tracker aufweist; und (d) eine umfassende Implementation von LiveShift [HBM+08] wurde umgesetzt für die Durchführung von realistischen Experimenten und Vorführungen.

2 LiveShift-Protokoll und Policies

Die LiveShift-Architektur besteht aus einem flexiblen Protokoll und initialen Policies, die den Gebrauch des Protokolls definieren. Der geschilderte Anwendungsfall gibt Benutzern die Möglichkeit zum Kanalwechsel und dem zeitversetzten Sehen. LiveShift übernimmt den Mesh-Pull-Ansatz [HLL+07], weil er sich besser an dynamische Netzwerkbedingungen und an Dynamik im P2P-Overlay (Churn) anpasst als Baum-basierte Protokolle [MR07]. Mesh-pull teilt den Datenstrom in Stücke (Chunks) auf, die zwischen den Peers ohne fixe Struktur ausgetauscht werden. LiveShift benutzt zwei Stufen der Stückelung – ein *Segment* ist eine Adressierungseinheit, die sich aus mehreren, kleineren *Blöcken* zusammensetzt. Segmente und Blöcke werden rein zeitbasiert adressiert und identifiziert – sie sind von wohldefinierter und fixer Länge. Dies macht es trivial herauszufinden, aus welchem Block und Segment die zur Zeit abzuspielende Video-Übertragung besteht. Die Unterscheidung von Segmenten und Blöcken ist wichtig, um (mittels grösseren Segmenten) die Anzahl von Tracker-Operationen zu reduzieren, währenddem Peers Blöcke von unterschiedlichen Peers herunterladen können, so dass sich das System (wegen der kleineren Blöcke) im Fehlerfall schnell erholt. So wird im DT nur die Verfügbarkeit von Segmenten eingetragen; die Verfügbarkeit von Blöcken innerhalb eines Segments wird im direkten Austausch zwischen Peers bekannt gemacht.

Während die Kanalliste und die dazugehörigen Kanalinformationen in der DHT gespeichert werden, ist der DT für das Abbilden von Segmenten auf gewisse Anbieter – Peers die mindestens ein Segment eines Blocks haben – verantwortlich. Für die Funktion des DT wird B-Tracker, ein voll verteilter Tracker, welcher von allen Peers im Netzwerk aufrecht

erhalten wir. B-Tracker wird im detail im Unterkapitel 4 behandelt.

Ein Hauptunterschied zwischen LiveShift und anderen P2P Systemen resp. Protokollen besteht in der Behandlung der Asymmetrie von Interessen. Die Unterstützung von zeitversetztem Sehen, wie es in LiveShift vorgesehen ist, setzt voraus, dass Peers gespeicherte Blöcke liefern obwohl sie nicht zwingend an Inhalten, welche der herunterladende Peer anbietet, interessiert sind. Das verlangt nach einer Aufteilung zwischen Nachbarn und Abonnenten. Sobald ein Peer r dem System beitritt, ruft er die Kanalliste aus der DHT ab. Nachdem r eine $kanalId$ und eine $startZeit$, ausgewählt hat, konsultiert er den DT um einen Satz K_r Kandidaten, die einen Block im entsprechenden Segment angepriesen haben, zu erhalten. Alternativ können `PeerVorschlag`-Nachrichten, welche Vorschläge von Kandidaten enthalten, von anderen Peers empfangen werden. Darauf hin kontaktiert Peer r eine Zahl von Kandidaten $p \in K_r$ indem er ihnen eine `Abonnieren` Nachricht schickt, welche den $SegmentId$ und die deklarierte Upload-Kapazität enthält.

Erhält ein Peer $p \in C$ eine `Abbonieren`-Nachricht von einem Peer r, so versucht er r in seine Abonnentenliste A_p hinzuzufügen. Falls $|S_p| < \overline{S}_p$ ist, ist die Abonnentenliste noch nicht voll und eine `Abonniert`-Nachricht, welche eine Tabelle mit den Blöcken, die p gespeichert hat, und eine Abfallzeit Z_S enthält, wird an r geschickt. Peer r wird nun über Änderungen in der zugehörigen Blocktabelle mittels `Haben`-Nachrichten auf dem laufenden gehalten. Gilt $|S_p| = \overline{S}_p$, so überbrüft p ob ein anderer Peer $q \in S_p$ eine tiefere Priorität als r hat. Wenn dem so ist, wird p bevorzugt und q wird aus S_p entfernt. In jedem Fall erhält einer der beiden Peers, q oder r, eine `NichtAbonniert`-Nachricht. Die Grösse von $|S_p|$ zu limitieren ist notwendig, da wenn jeder Peer $|S|$ Abonnenten für ein Segment hat so ist die Anzahl an `Haben`-Nachrichten für einen neuen Block im gesamten P2P-System $|S|^2 - |S|$. Werden `Haben`-Nachrichten an Peers, die den betreffenden Block bereits haben, eingespart, so reduziert sich die Zahl der Nachrichten lediglich auf $(|S|^2 - |S|)/2$. Man beachte, dass sich der vorgeschlagene Algorithmus in zwei wichtigen Aspekten von bereits existierenden Mesh-Pull unterscheidet: Erstens, die Beziehung zwischen zwei Peers ist asymmetrisch und zweitens, das sofortige bevorzugen von Peers resultiert in einer schnelleren Reaktion des Systems als wenn die Peer auswahl in fixen Zeitintervallen vorgenommen wird, was die allgemeine Wiedergabeverzögerung reduziert.

Erhält Peer r eine `Abonniert`-Nachricht von p, so fügt er p zu seinen Nachbarn N_r hinzu. Jetzt muss r periodisch sein Interesse überprüfen indem er die Schnittmenge zwischen den Blöcken die zum herunterladen markiert sind und denen, welche p mit seiner Blocktabelle und seinen `Haben`-Nachrichten angepriesen hat. Ist die Schnittmenge nicht leer, so fügt r p zu I_r hinzu und sendet p eine `Interessiert`-Nachricht, was wiederrum p dazu veranlasst r in $W_p \subset S_p$, die Warteschlange für Peers die auf einen freien Upload-Platz warten, hinzuzufügen. Dann antwortet p mit einer `Eingereiht`-Nachricht, die eine Abfallzeit enthält. Im anderen Fall, in welchem p keine interessanten Blöcke mehr hat, sendet r eine `NichtInteressiert`-Nachricht um aus Q_p ausgetragen zu werden.

Peer p hat eine gewisse Anzahl an Upload-Plätzen $\overline{U}_p$, aus welchen jeweils einer einem interessierten Peer $r \in Q_p$ zur zugewiesen wird. Wird einem Peer r ein Upload-Platz zugewiesen, so erhält dieser ein `Zugewiesen`-Nachricht, welche eine Abfallzeit Z_I enthält. Bleibt der Upload-Platz in der Zeit Z_I ungenutzt, wird der Upload-Platz einem anderen Peer zugewiesen. Ähnlich zum Fall, in dem Peers in S_p sofort durch Peers mit höherer

Priorität ersetzt werden.

Erhält r einen Upload-Platz von p, ist es r erlaubt `BlockAnfrage`-Nachrichten an p zu senden und Videoblöcke mittels `BlockAntwort`-Nachrichten zu empfangen. Jeder Upload-Platz reiht maximal zwei `BlockAnfrage`-Nachrichten nacheinander in seine Warteschlange um die Verzögerung zwischen dem senden einer `BlockAntwort`-Nachricht und der nächsten `BlockAnfrage`-Nachricht zu minimieren. Das geht so weiter bis (a) r eine `NichtInteressiert`- oder eine `TrennenHoch`-nachricht sendet, (b) p eine `Eingereiht` (falls ein anderer Peer bevorzugt wird) oder eine `TrennenHerunter`-Nachricht sendet, oder (c) der Upload-Platz die Abfallzeit Z_I erreicht.

Die zwei unteschiedlichen trennen Nachrichten repräsentieren die Asymmetrie der Interessen. `TrennenHoch` wird von einem anfragenden Peer gesendet, welcher nicht mehr am herunterladen eines bestimmten Segmentes interessiert ist, i.e. beim Kanalwechsel. Ein Peer p der eine `TrennenHoch` empfängt unterlässt das Hochladen zu r und entfernt r aus S_p, Q_p und U_p. Im gegensatz werden `TrennenHerunter`-Nachrichten von einem anbietenden Peer p, welcher das System verlässt, versendet. Die Bedeutung ist ähnlich derer von `NichtAbonniert`, dahingehend, dass r aus S_p, Q_p und U_p entfernt und r seinerseits p aus K_r, N_r und I_r entfernen muss. Der unterschied liegt darin, dass r vor einem erneuten Versuch in betracht zieht, dass p das System verlassen hat.

LiveShift wurde mit dem Ziel evaluiert, die aus seinen Protokollen und Standard-Policies resultierende Leistung unter realen Bedingungen anhand der objektiven QoE Metriken Wiedergabeverzögering und Anteil der übesprungenen Blöcke zu messen. Die Evaluierung berücksichtigt, dass P2P Umgebungen heterogen bezüglich der Peer-Upload-Kapazitäten [HLR08] sind, dass Peers Kanäle mit hoher Frequenz wechseln [CRC+08], und dass Peers dem Netzwerk beitreten oder dieses verlassen (churn). Daher wurde das System vollständig implementiert. Vier Szenarien mit unterschiedlichen Peerkapazitäten wurden gewählt, um zu zeigen, dass das System eine Vielzahl von Peers mit einer Upload-Kapazität, die geringer ist als die Bitrate der Video-Übertragung, unterstützen kann. Während Szenario s1 nur 60% Last aufweist (Verhältnis zwischen der gesamten Upload- und benötigter Download-Kapazität), weist Szenario s3 85% Last auf.

Die Peers wurden konfiguriert sich wie echte Nutzer zu verhalten, indem sie Kanäle zeitversetzt ansahen und häufig wechselten. Die Kanalpopularität war annähernd exponentiell und die Verweildauer pro Kanal meist kurz (in 55.97% der Fälle betrug sie weniger als 10s) während Kanäle aber auch länger nicht gewechselt wurden (In 3.89% der Fälle wurde ein Kanal über eine Stunde nicht gewechselt). Die benutzten QoS Metriken waren die von Peers wahrgenommene und gemeldete Wiedergabeverzögerung ab dem Zeitpunkt an dem die $(channelId, startTime)$ gewählt wurde, sowie der Anteil der übersprungenen Blöcke.

Abbildungen 1 und 2 zeigen die von Nutzern wahrgenommene Wiedergabeverzögerung in ausgewählten Szenarien. Während Low-Upstream-Peers (LU) nur 50% der Stromrate heraufladne können, besitzen High-Upstream-Peers (HU) die fünffache Kapazität. Eine geringere Wiedergabeverzögerung bedeutet eine geringere Anlaufverzögering, weniger Anhalten der Wiedergabe und eine grössere Nähe zu dem, was der Nutzer initial sehen wollte. Die Ergebnisse gewähren die folgenden Einsichten, während weitere Ergebnisse

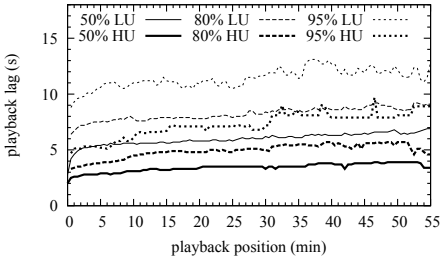

Abbildung 1: Wiedergabeverzögerung in s1

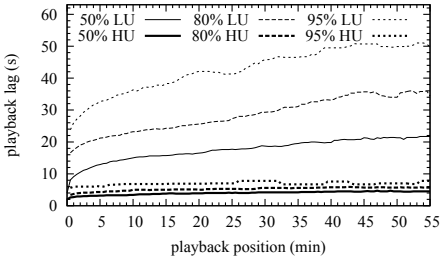

Abbildung 2: Wiedergabeverzögerung in s3

in der Dissertation zu finden sind: a) Die Wiedergabeverzögerung erhöht sich nur gering, wenn Nutzer einen Kanal länger schauen, was impliziert, dass Nutzer kein langes Anhalten der Wiedergabe wahrnehmen; b) sogar im ungünstigsten untersuchten Fall, nehmen 95% der HU Peers eine Wiedergabeverzögerung von weniger als 10s wahr, was eine akzeptable Performanz ist (viele live TV Übertragungen weisen eine ähnliche Verzögering auf); und c) LU Peers sind wesentlich anfälliger für hohe Wiedergabeverzögerungen, insbesondere in Szenarien mit Churn oder weniger verfügarer Bandbreite. Beispielsweise zeigt Abbildung 2, dass die schlechtesten 5% der LU Peers eine Wiedergabeverzögerungen von über 50 s haben, nachdem sie einen Kanal über 40 min geschaut haben.

3 Playback-Policies

Die Dissertation identifiziert zwei verwandte aber orthogonale Aspekte: die Definition von Protokollen und Poilicies zur direkten sowie zeitversetzten Multimedia-Übertragungen. Die *Playback-Policy* definiert ob ein Peer nicht-verfügbare Blöcke überspringt oder die Wiedergabe anhält, bis diese gefunden und heruntergeladen sind. Während einige Policies ausgiebig untersucht wurden, sind Playback-Policies nicht hinreichend untersucht. Dieser Abschnitt beschreibt und generalisiert vier Playback-Policies basierend auf verwandten Arbeiten und einer neuen Catchup Policy.

Skip/Stall Policy. Die Skip/Stall (`sk`) Policy wird durch $P_{sk} = (\ell, \alpha, \beta)$ definiert, wobei ℓ die Grösse des Wiedergabepuffers (in Blöcken) angibt und $\alpha \in (0, 1]$ den Füllstand des Puffers angibt, der erreicht sein muss, damit mit der Wiedergabe begonnen wird. Fehlende Blöcke werden unmittelbar übersprungen, allerdings wird bei leerem Puffer die Wiedergabe angehalten und wieder mit dem initialen Puffern begonnen. Wenn ein Block bei $b_e(t)$ fehlt und der Puffer nicht leer ist, hält die Wiedergabe an, bis ein Anteil von $\beta \in [0, 1]$ des Wiedergabepuffers gefüllt ist; erst dann werden fehlende Blöcke übersprungen.

Remaining Download Time Policy. Die Remaining Download Time Policy (`rd`) hält die Wiedergabe an bis die verbleibende Downloadzeit t_d kleiner ist als die verbleibende Wiedergabezeit t_p. Die Policy wird durch $P_{rd} = (\ell, \alpha, \beta, t_p)$ definiert, wobei der gleiche Algorithmus wie für die `sk` Policy verwendet wird. Im Gegensatz zur `sk` Policy wird allerdings eine variable Puffergrösse ℓ' verwendet, die basierend auf den Parametern t_p

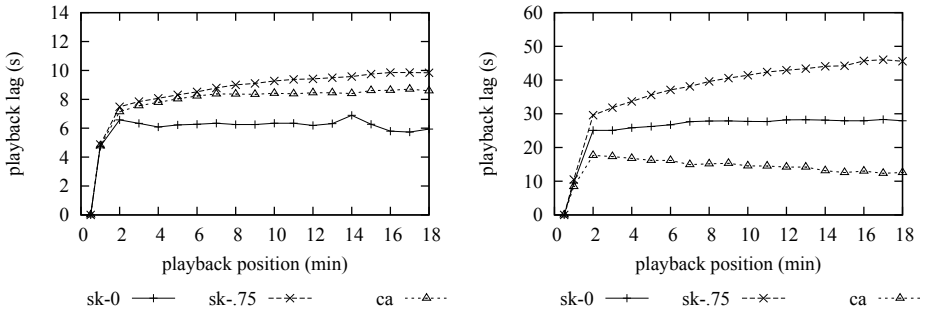

Abbildung 3: Playback-Policies, Wiedergabeverzögerung in s1 (links) und s4 (rechts)

und ℓ berechnet wird. Es sei i die Rate mit der Blöcke herunter geladen werden und L sei die Blocklänge, ausserdem sei r die relative Rate mit der Blöcke herunter geladen werden, also $r = i \cdot L$. Die variable Puffergrösse ℓ' ist daher durch $\ell' = \max\left(\frac{t_p}{L} \cdot \frac{(1-r)}{\alpha}, \ell\right)$ berechnet, wobei ℓ sowohl für das initiale Puffern als auch als eine allgemeine untere Schranke für die Puffergrösse (z.B. wenn $r \geq 1$) verwendet wird.

Retry Policy. Die Retry (`re`) Policy wird durch $P_{re} = (\ell, \alpha, T)$ definiert und hält nach dem initialen Puffern die Wiedergabe an, wenn ein Block an der Wiedergabeposition t nicht verfügbar ist. Es wird bis zu T mal versucht einen fehlenden Block abzuspielen. Sobald der Block verfügbar ist, wird er wiedergeben. Wird aber die Verzögerungsgrenze erreicht, wird er übersprungen.

Ratio Policy. Die Ratio (`ra`) Policy wird durch $P_{ra} = (\ell, \alpha, n)$ definiert, wobei die Bedeutung von ℓ und α unverändert bleibt. Nach dem initialen Puffern, wird ein verfügbarer Block an der Wiedergabeposition immer wiedergegeben. Es werden bis zu x aufeinanderfolgende fehlende Blöcke übersprungen, wenn direkt hinter diesen mindestens xn aufeinanderfolgende Blöcke verfügbar sind.

Catchup Policy. Die Policy wird durch $P_{ca} = (\ell, \alpha)$ definiert, wobei ℓ und α genau wie bei `sk` zum initialen Puffern verwendet werden. Nachdem die Wiedergabe begonnen hat, werden bei nicht-leerem Puffer alle fehlenden Blöcke übersprungen. Sollte der Puffer leer sein, wird die Wiedergabeposition zurückgesetzt, indem alle Blöcke, die für die Wiedergabeverzögerung verantwortlich sind, übesprungen werden, bis $b_e(t) = b_p(t)$.

Evaluation. Alle oben aufgeführten Playback-Policies wurden in LiveShift implementiert. Das Hauptziel war es zu vergleichen wie verschiedene Playback-Policies das Benutzererlebnis bei verschiedenen Leveln der Inhaltsverfügbarkeit beeinflussen. Bei den Experimenten wurde der gesamte LiveShift Code verwendet. Die Evaluationsparameter ähneln denen in Abschnitt 2. Jede Playback-Policy wurde mit vielen verschiedenen Werten für deren Hauptparameter untersucht. Aus Platzgründen werden in diesem Abschnitt nur Ergebnisse für die Skip/Stall mit Parametern $\beta = 0$ (`sk-0`) und $\beta = .75$ (`sk-.75`) und die Catchup Policy dargestellt. Alle Ergebnisse sind in [HBSS12] zu finden. Die Evaluationsmetriken umfassen Wiedergabeverzögering und übersprungene Blöcke.

Ausgewählte Resultate in den Abbildungen 3 zeigen, dass abhängig von der Playback-Policy (und deren Parameter), unterschiedliche Wiedergabeverzögerung resultieren. Während die Skip/Stall Policy durch den Parameter β sehr flexibel ist, resultiert die neue Catch-up Policy in einer tieferen Wiedergabeverzögerung verglichen mit anderen Policies in Szenario s4. Beobachtungen von LiveShift mit verschiedenen Playback-Policies, verschiedenen Parametern, in Szenarion mit Unter- und Überprovisionierten P2P Netzwerken, haben gezeigt dass Playback-Policies das Nutzererlebnis beinflussen. Nicht nur die ausgewählte Policy, aber auch ihre Parameter beinflussen wichtige Metriken, wie zum Beispiel die Wiedergabeverzögerung oder die Anzahl der übersprungenen Blöcke.

4 B-Tracker

Trackers werden verwendet um Peers zu finden die eine spezifische Ressource anbieten. In LiveShift sind das Peers, welche einen Block in einem spezifischen Segment anbieten. Eines der Ziele des LiveShift Entwurfs ist es von einer voll-verteilten Architektur zu profitieren, um die Skalierbarkeit und Robustheit zu verbessern; daher müssen Peers ihre Tracker-Dienst effizient anbieten und gleichmässig verteilen, so dass die Last vom Tracker Service gerecht verteilt wird. Allerdings sind heutige Tracker-Ansätze nicht geeignet, da populäre Inhalte nicht gleich verteilt wereden [HBS11].

Primäre Trackers. Zuerst setzt B-Tracker eine verteilte Hash-Tabelle (DHT) Struktur für die Tracker Entdeckung ein. Um die DHT als Tracker verwenden zu können, muss die *put(key, value)* Funktion modifiziert werden um mehrere Wertepaare abzuspeichern unter dem selben Schlüssel, und die *get(key)* Funktion muss ebenfalls modifiziert werden um eine zufällige Anzahl von Wertepaaren zurückzuliefern. Die Zahl dieser primären Peers ist durch den Replikationsfaktor von r_p gegeben. Da die primären Peers mit hoher Wahrscheinlichkeit nicht Anbieter von einer bestimmten Ressource sind, unter berücksichtigung einer zufälligen Verteilung, und um Peers zu motivieren weitere, sogenannte sekundäre Trackers zu verwendn, gibt es eine Limite von b_p Anbietern pro Ressource.

Sekundäre Trackers. Wenn ein Peer die Anbieterliste von einem primären Tracker bezogen hat, werden weitere Anfragen an sekundäre Trackers gestellt, da diese ebenfalls Anbieter dieser Ressource sind. Das führt dazu, dass die Last gleichmässig auf den sekundären Trackers verteilt wird. Der Parameter n_p legt die Anzhal der primären Trackers fest, welche zuerst konsultiert werden bevor man die sekundären Tracker anfragt.

Verbesserung der Effizienz. Anfragen an primäre und sekundäre Tracker beinhalten einen Bloom filter mit bereits bekannten Anbietern. Anbieter, welche in diesem Filter gespeichert sind werden ausgeschlossen.

Updating Mechanism. Der Replikationsfaktor r_p bestimmt wie viele Einträge für primäre Tracker gespeichert werden, und r_s bestimmt wie viele Einträge für sekundäre Tracker gespeichert werden.

Veraltete Informationen. Die Tracker-Informationen veralten mit der Zeit, da Peers das Netzwerk verlassen oder eine Ressource nicht mehr anbieten ohne den zuständigen Tracker zu informieren. B-Tracker verwendet eine Lebenszeit für jedes Wertepaar.

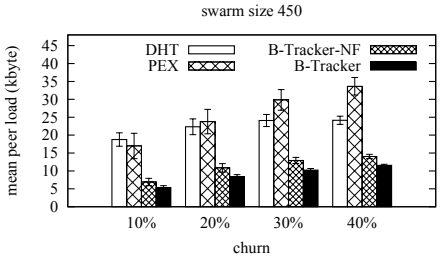

Abbildung 4: Effizienz

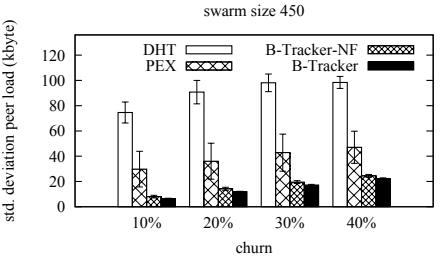

Abbildung 5: Lastausgleich

B-Tracker wurde in Java 1.6 implementiert und mit PEX und einem DHT-basierten Tracker Ansatz verglichen. Die Evaluation hat sich auf die Effizienz und den Lastausgleich fokussiert. Die Last berücksichtigt nur den Upload Traffic, da dies die knappste Ressource in eine P2P System ist. *Effizienz* ist definiert als das Mittel der Last pro Peer im Schwarm. Das heisst, je geringer die Last, desto besser die Effizienz. *Lastausgleich* ist definiert als die Standardabweichung der Last aller Peers im Schwarm. Das heisst je kleiner die Standardabweichung, desto besser der Ausgleich.

Ein P2P-System mit 1.000 Peers wurde wie folgt emuliert. Bei jedem Durchlauf wird ein Schwarm mit 50 oder 450 Peers erstellt. Jeder Peer im Schwarm erhält 35 Einträge aus der DHT. Die Messung beginnt erst, nachdem die ersten Einträge erhalten wurden, sodass ein Live-Schwarm emuliert werden kann. Um die Dynamik (churn) zu emulieren, werden Peers mit einer gewissen Wahrscheinlichkeit ausgeschaltet (10%, 20%, 30%, 40%). Die Peers, welche ausgeschalten werden, werden durch neue Peers ersetzt. Danach versuchen alle Peers 35 Einträge zu beziehen.

Effizienz. Die Abbildung 4 zeigt die durchschnittliche Belastung pro Peer für Schwarmgrössen 50 und 450. Ein DHT Ansatz ist wegen der vielen Routing-Nachrichten nicht effizient. PEX ist noch weniger effizient, weil Peers viele unnötige Meldungen versenden, um andere Peers über die Nachbaren zu informieren, ohne das dies nötig wäre. Der B-Tracker Ansatz zeigt eine bessere Effizienz als der B-Tracker-NF Ansatz wegen dem Einsatz von Bloom Filtern – obwohl diese Meldungen Grösser sind, da die Meldungen den Filter beinhalten, enthält die Liste meist keine unnnötigen Informationen.

Lastausgleich. Die Abbildung 5 zeigt den Lastenausgleich als Standardabweichung der Last von allen Peers im Schwarm. In allen untersuchten Szenarien wird die Last im B-Tracker-NF und B-Tracker Ansatz besser verteilt als mit dem DHT Ansatz. Im DHT Ansatz, werden Tracker stark ausgelastet sobald sich der Schwarm vergrössert. Der PEX Ansatz verbessert vor allem in einem grossen Schwarm den Lastausgleich. Der B-Tracker-NF Ansatz zeigt dass mit Hilfe von Bloom Filtern der Lastausgleich minimal verbessert wird. Der Lastausgleich verschlechtern sich sobald der Schwarm grösser wird. Dies kann dadurch erklärt werden, dass der DHT Ansatz verwendet wird, wenn PEX und sekundäre Tracker nicht die 35 Einträge bereitstellen können.

5 Zusammenfassung

Dieser Dissertationsabriss hat mehrere Schlüsselaspekte vorgeschlagen und untersucht, welche beim vollständig verteilten direkten und zeitversetzten P2P-Multimedia-Streaming involviert sind. Dieser Anwendungsfall war bisher grösstenteils unerforscht.

Der Fokus lag auf vier wichtigen Aspekten, namentlich (a) der Entwicklung und Evaluation eines vollständig verteilten Mesh-Pull-P2P-Protokolls, welches den Austausch sowohl von direkten als auch zeitversetzten Videostreams ermöglicht, (b) dem Untersuchen von Playback-Policies in verschiedenen Szenarien und unter Verwendung unterschiedlicher Parameter, (c) der Definition und Evaluation eines vollständig verteilten P2P Trackers, welcher sowohl die Effizienz als auch die Lastverteilung verbessert, und (d) der Demonstration der Praktikabilität obiger Mechanismen mittels Implementierung in einer OpenSource-Anwendung, welche in diesem Bericht nicht detailliert wurde.

Das LiveShift Protokoll wurde vollständig implementiert und unter höchst realistischen Bedingungen evaluiert, inklusive des schnellen Umschaltens zwischen zwischen Kanälen, einer grossen Zahl an Peers mit geringerer Upload-Kapazität als der Stream-Rate, sowie bei einer hohen Churn-Rate. Auswertungen basierend auf Traces zeigen, dass das System in der Lage ist, Inhalte und Upstream-Kapazität schnell genug zu finden, um eine geringe Wiedergabeverzögerung in Bezug zur Wiedergabeposition aufrechtzuerhalten.

Anschliessend wurde das Verhalten von LiveShift bei Anwendung verschiedener Playback-Policies mit unterschiedlichen Parametern untersucht. Es wurde offensichtlich, dass verschiedene Playback-Policies das Nutzererlebnis in einem P2P Video-Streaming-System beeinflussen, sowohl was die Wiedergabeverzögerung als auch den Anteil übersprungener Blöcke betrifft. Das Verständnis des Nutzerverhaltens ist die Grundvoraussetzung, um die am besten geeignete Policy für das gewünschte Ergebnis auszuwählen. Dabei kann entweder die Wiedergabeverzögerung so gering wie möglich gehalten, das überspringen von zahlreichen Blöcken vermieden, oder ein Kompromiss der beiden Varianten erreicht werden. Die endgültige Entscheidung kann dem Benutzer überlassen oder vom übermittelten Inhaltstyp abhängig gemacht werden.

Des Weiteren wurden der Entwurf, die Implementierung, und die Evaluation von B-Tracker beschrieben. Bei B-Tracker handelt es sich um einen neuartigen P2P-Tracker, welcher den neusten Stand der Technik weiter verbessert. Die wichtigste Innovation von B-Tracker ist, dass sämtliche Peers als Tracker für diejenigen Resourcen agieren, welche sie selbst zur Verfügung stellen. Dabei konnte bestätigt werden, dass B-Tracker im Vergleich zu verteilten P2P-Trackern, welche den reinen DHT- und PEX-Ansatz verwenden, sowohl die Effizienz als auch die Lastverteilung verbessert.

Danksagung

Der Autor dankt T. Bocek, D. Dönni, A. Lareida, P. Poullie, M. Waldburger und B. Stiller von der Communication Systems Group (CSG) der Universität Zürich sowohl für ihre wertvolle Unterstützung während der Ausarbeitung dieser Arbeit, als auch für deren

Übersetzung.

Literatur

[CRC+08] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon und Xavier Amatriain. Watching Television over an IP Network. In *8th ACM SIGCOMM Conference on Internet Measurement (IMC '08)*, Seiten 71–84, New York, NY, USA, 2008.

[HBC+11] F. V. Hecht, T. Bocek, R. G. Clegg, R. Landa, D. Hausheer und B. Stiller. LiveShift: Mesh-Pull P2P Live and Time-Shifted Video Streaming. In *36th IEEE Conference on Local Computer Networks*, Seiten 319–327, Bonn, Germany, Oktober 2011.

[HBM+08] F. V. Hecht, T. Bocek, C. Morariu, D. Hausheer und B. Stiller. LiveShift: Peer-to-peer Live Streaming with Distributed Time-Shifting. In *8th International Conference on Peer-to-peer Computing (P2P'08)*, Aachen, Germany, September 2008.

[HBS11] F. V. Hecht, T. Bocek und B. Stiller. B-Tracker: Improving Load Balancing and Efficiency in Distributed P2P Trackers. In *11th IEEE International Conference on Peer-to-peer Computing (P2P'11)*, Seiten 310–313, Kyoto, Japan, September 2011.

[HBSS12] F. V. Hecht, T. Bocek, F. R. Santos und B. Stiller. Playback Policies for Live and On-Demand P2P Video Streaming. In *IFIP Networking*, Prague, Czech Republic, Mai 2012.

[Hec] F. V. Hecht. LiveShift: A Time-Shifted P2P Multimedia Streaming Approach. `http://www.csg.uzh.ch/staff/hecht/hecht-dissertation-notfinal.pdf`, last visited: November 2012.

[HLL+07] X. Hei, C. Liang, J. Liang, Y. Liu und K.W. Ross. A Measurement Study of a Large-Scale P2P IPTV System. *IEEE Transactions on Multimedia*, 9(8):1672–1687, December 2007.

[HLR08] X. Hei, Y. Liu und K. W. Ross. IPTV over P2P Streaming Networks: The Mesh-Pull Approach. *Communications Magazine, IEEE*, 46(2):86–92, March 2008.

[liv] LiveShift: P2P Live Video Streaming with Time Shifting Ability. `https://github.com/zegotinha/LiveShift/`, last visited: October 2012.

[MR07] N. Magharei und R. Rejaie. Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches. In *IEEE INFOCOM 2007*, Seiten 1424–1432, May 2007.

Fabio Victora Hecht has obtained a Doctoral degree at the Communication Systems Group at the Department of Informatics of the University of Zurich in December 2012 under supervision of Prof. Dr. Burkhard Stiller (University of Zurich, Switzerland) and Dr. Tobias Hoßfeld (Universität Würzburg, Germany). Formerly, he has obtained a five-year degree named "Bacharelado" – equivalent to a Swiss Diplom – in Computer Science from Universidade Federal do Rio Grande do Sul (UFRGS), Brazil.