Fairness in Überlastsituationen mittels Proof-Of-Work Funktionen

vorgelegt von
Diplom-Informatiker
Sebastian Golze
aus Berlin

von der Fakultät IV – Elektrotechnik und Informatik
FG Kommunikations- und Betriebssysteme (KBS)
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. Seifert
Gutachter: Prof. Dr. Heiß
Gutachter: Prof. Dr. Geihs

Tag der wissenschaftlichen Aussprache: 29. Juni 2009

Berlin 2009

D 83

Zusammenfassung

In der vorliegenden Arbeit wird gezeigt, dass die Aufgabe der Überlastvermeidung auch trotz zukünftiger Hardwareentwicklungen fortbestehen wird. Die besondere Schwierigkeit, eine faire Überlastvermeidung umzusetzen, besteht darin, dass in der heutigen Internetstruktur Klienten nicht mehr über ihre IP Adresse zuverlässig identifiziert werden können. Als Lösung werden Proof-Of-Work (POW) Funktionen vorgeschlagen. Dabei wird dem Klienten eine Rechenaufgabe pro gewünschten Zugriff auf den Server auferlegt. Die Anforderungen an POW Funktionen werden im Detail beleuchtet sowie die in der Literatur bekannten Verfahren erläutert. Mittels der im Rahmen dieser Arbeit entwickelten HashCashLin Funktion wird die Parametrisierbarkeit der POW Funktionen entscheidend verbessert.

Es wird ein dreischichtiger Regler entwickelt, der es erlaubt, das POW dynamisch zu parametrisieren und so die Überlast zu kontrollieren. Anhand einer Simulation wird die Funktion des Verfahrens untersucht und gezeigt, dass es deutlich fairer arbeitet als herkömmliche Ansätze. Daraufhin wird ein Prototyp erstellt, der demonstriert, wie der Ansatz ganz praktisch umgesetzt werden kann. Dieser Prototyp wird in Anlehnung an die Simulation ebenfalls erprobt und auch hier zeigt sich eine deutlich verbesserte Fairness. Somit wurde das Ziel der Arbeit, ein besonders faires Verfahren zur Überlastvermeidung zu entwickeln, erreicht.

Abstract

This work shows that overload handling will always be necessary regardless of future improvements in hardware performance. The most difficult part when constructing a fair overload handling system is that in today's internet infrastructure one can no more identify a single client by its IP address. The proposed solution is based on so called Proof-Of-Work functions. Thus, a client willing to send a request to the server in overload has to provide the solution to a computational puzzle which brakes down malicious clients and guarantees a fairer distribution of resources. The related work concerning Proof-Of-Work functions is discussed and the HashCashLin function is developed as an answer to weaknesses concerning parameterization among existing POW functions.

A three layer controller is designed which allows to adapt the hardness of the POW in order to precisely control the overload of a server. Using a simulation the functionality of the approach is studied. The results show that the POW system works much fairer than

traditional approaches. Finally, a prototype is implemented to show how the POW system can be integrated in real live web applications. This prototype is also tested and the obtained results back up the simulation data. Thus, the defined goal of this work to guarantee more fairness in overload situations has been attained.

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Trotz stetig steigender Rechenleistung von Servern entstehen immer wieder Situationen, in denen sich ein Dienstanbieter einer starken Überlastsituation ausgesetzt sieht. Die beiden hauptsächlichen Gründe für derartige Lastspitzen sind einerseits Denial-Of-Service Angriffe und andererseits ein temporär besonders gesteigertes Interesse durch die Nutzer des Dienstes. Bei einem temporär sehr stak belasteten Webserver kann der Grund z. B. ein lanciertes Sonderangebot oder Ähnliches sein. In den meisten Fällen ist es nicht möglich, genügend Infrastruktur vorzuhalten, um nur sehr kurzfristig auftretende Lastspitzen voll bedienen zu können. Daraus folgt, dass zwangsläufig eine Überlastsituation entsteht. Entweder der Dienst bricht daraufhin komplett zusammen oder er muss seinen Service massiv einschränken und Anfragen ablehnen. Das Ergebnis ist in beiden Fällen für den normalen Nutzer gleichermaßen unerfreulich. Entweder ist der Dienst überhaupt nicht mehr erreichbar und der Nutzer bekommt eine Fehlermeldung angezeigt oder viele seiner Anfragen werden willkürlich abgelehnt, was für den Benutzer ebenfalls nach einer Fehlfunktion aussieht. Je nach verwendetem Dienst ist ab einer bestimmten Quote von abgelehnten Anfragen der Dienst faktisch nicht mehr nutzbar.

Dieses Ergebnis ist insbesondere unerfreulich, weil es aggressive Klienten bevorzugt, welche dauerhaft Anfragen senden und so eine bessere Chance haben, ihre Anfragen beantwortet zu bekommen. Solche aggressiven Klienten können sowohl bösartige Absichten haben oder sich nur einen Vorteil beim Zugriff auf den Dienst verschaffen wollen.

Dieser Zustand ist nicht zufriedenstellend und deshalb wird in dieser Arbeit ein Verfahren entwickelt, welches in einer Überlastsituation eine faire Verteilung der knappen Ressourcen ermöglicht. Fair bedeutet hier, dass jeder Klient identische Chancen hat, auf den Dienst zuzugreifen, ohne dass sich ein aggressiver bzw. bösartiger Klient illegitime Vorteile verschaffen kann. Dies wird mittels so genannter Proof-Of-Work (POW) Funktionen erreicht, die auch unter den Begriffen Working Functions oder Delaying Functions bekannt sind.

Um die praktische Relevanz dieser Fragestellung zu verdeutlichen, werden an dieser Stelle zwei praktische Beispiele vorgestellt, in denen das Überlastproblem unzureichend gelöst wurde und die von den Ergebnissen dieser Arbeit profitieren könnten.

Das erste Beispiel betrifft die Webseite der Deutschen Bahn AG www.bahn.de. Hier werden wiederholt Sonderangebote zu bestimmten Terminen, wie z. B. das „Herbst-

spezial“, angeboten. Aus eigener Erfahrung, und wie dies auch in der Presse [SP] geschildert wird, geraten die Server der Bahn zu diesen Terminen immer wieder in Überlast. Diese Überlast wird unzureichend behandelt, so dass sich die Antwortzeiten stark verlängern oder die Anfragen des Benutzers schlicht mit Fehlermeldungen abgewiesen werden. In diesem Fall kann sich ein technisch versierter Benutzer ohne weiteres einen deutlichen Vorteil verschaffen, indem er den Server manuell oder automatisch mit einer Unzahl an Anfragen überhäuft, um die Chance zu verbessern, mit einer der Anfragen erfolgreich zu sein. Diese Möglichkeit ist aber nicht erwünscht, da dies nicht einer fairen Gleichbehandlung aller Benutzer entspricht und die Überlastsituation als Ganzes durch entsprechende Verhaltensweisen noch verschärft wird. Darüber hinaus wird der Server durch zusätzliche Benutzer belastet, die unabhängig vom Sonderangebot in der Zukunft liegende Reisen planen und dazu generelle Fahrplanauskünfte einholen etc. Optimalerweise bringt man diese Benutzer dazu, solche Aktivitäten auf einen späteren Zeitpunkt zu verschieben, um so den Verkauf des Sonderangebots besser abwickeln zu können. In einer realen Schalterhalle würden die meisten dieser Probleme durch eine einfache Warteschlange gelöst. Benutzer ohne gesteigertes Interesse am akuten Fahrkartenkauf würden die Schlange meiden und alle anderen würden sich anstellen, ohne dass einzelne Benutzer diverse Plätze in den Warteschlangen gleichzeitig besetzen könnten etc. Die Aufgabe besteht dementsprechend darin, das Äquivalent einer solchen fairen Warteschlange technisch umzusetzen.

Das zweite Beispiel, welches das bearbeitete Problem verdeutlicht, ist der Online-Ticketshop zur Fußball-Weltmeisterschaft 2006. Da immer wieder Restkontingente in den Verkauf kamen, wurde ein Onlinesystem eingerichtet, mit dem diese kleinen Ticketmengen verkauft werden konnten. Wie zu erwarten war, überstieg die Nachfrage das Angebot bei weitem. Die Kernaufgabe eines Marktes besteht darin, Angebot und Nachfrage in Einklang zu bringen, dementsprechend hätte man ohne weiteres ein Auktionssystem einrichten können, welches den Markt ausgeglichen hätte. Aus politischen Gründen wollte man aber die Tickets zum normalen Preis verkaufen, so dass ein first come first serve Verfahren gewählt wurde.

Abbildung 1: Warteschlange des Online Buchungssystems

Als Konsequenz entstand eine massive Last auf dem Buchungssystem und insbesondere auf der Webseite, welche die aktuell zur Verfügung stehenden Tickets anzeigte. Um möglichst schnell reagieren zu können, pollten findige Nutzer diese zum Teil mit vollautomatischen Programmen im Sekundentakt. Noch kritischer schien die Last das nachgelagerte Buchungssystem zu treffen, so dass hier eine Warteschleife von 30 Sekunden vorgeschaltet wurde. Die getestete Version bestand aber nur aus einem JavaScript Programm, welches die 30 Sekunden in Schritten von einer Sekunde mittels der setTimeout() Funktion herunterzählt und die Anfrage dann weiterleitet. Diese Warteschleife kann technisch mit minimalem Aufwand umgangen werden. Abbildung 1 zeigt einen Screenshot der verwendeten Warteschleife und Abbildung 2 den entscheidenden Teil des JavaScript Programms, welcher nach 30 Sekunden den Link für einen erneuten Verbindungsaufbau angezeigt hat.

```
function ZeitAnzeigen() {
   […]
   window.setTimeout('ZeitAnzeigen()',1000);

   if (relSekunden>0) {
        [Hier wird die Anzahl verbleibender Sekunden angezeigt]
   } else
   {
        [Hier wird der weiterführende Link angezeigt]
   }
}
```

Abbildung 2: Unzureichende Implementierung der Warteschlange in JavaScript

In diesem Fall hatte man das Problem der Überlast zumindest erkannt und eine Lösung implementiert. Die technische Umsetzung war aber unzureichend und verschaffte aggressiv

anfragenden Klienten sogar einen weiteren Vorteil, da alle anderen Klienten auf eine Anfrage pro 30 Sekunden heruntergebremst wurden, während diese Klienten weiterhin mehrmals pro Sekunde Zugriffsversuche starten konnten.

Diese beiden ausführlichen Beispiele haben gezeigt, dass das untersuchte Problem der Fairness in Überlastsituationen in der Praxis existiert und einer adäquaten Lösung bedarf.

In den sich anschließenden Kapiteln wird zuerst im Detail auf das Phänomen Überlast und seine Ursachen eingegangen. Danach wird der Begriff der Fairness betrachtet und der für diese Arbeit maßgebliche Fairnessbegriff hergeleitet. Im Folgenden werden verschiedene Ansätze diskutiert, die verwendet werden könnten, um diese Fairness durchzusetzen, und es wird erklärt, warum der Proof-Of-Work (POW) Ansatz gewählt wurde. Die verschiedenen Typen von POW Funktionen sowie generelle Anforderungen an POW Funktionen werden ebenfalls vorgestellt. Im Anschluss wird die im Rahmen dieser Arbeit entwickelte HashCashLin POW Funktion präsentiert. Da eine POW Funktion nur effizient arbeiten kann, wenn ihre Schwierigkeit auf den aktuellen Anwendungsfall abgestimmt ist, beschreiben wir verschiedene Verfahren, die zur Parametrisierung der POW Funktionen verwendet werden können. Daraufhin wird insbesondere auf den reglerbasierten Ansatz der POW Überlastvermeidung eingegangen. Das Verfahren wird sowohl mittels einer Simulation als auch eines realen Prototypen getestet. Die Arbeit schließt mit einer Zusammenfassung der Ergebnisse und einem Ausblick auf weitere Fragestellungen.

2 Entstehung von Überlast

Die Entwicklung der Leistungsfähigkeit von CPUs ist seit Jahrzehnten von exponentiellem Wachstum gekennzeichnet. Moores Gesetz beschreibt diesen Zusammenhang [MOORE65]. Auch die Leistungsfähigkeit der Netzwerkverbindungen zwischen den Rechnern wächst stark. Daraus könnte man schlussfolgern, dass sich das Phänomen Überlast mit der Zeit von alleine lösen könnte. Es wäre denkbar, dass die wachsende Leistung der Systeme zu einem solchen Überangebot an Kapazitäten führt, dass generell immer alle Anfragen optimal bedient werden können. Obwohl dies natürlich für die Zukunft nicht gänzlich ausgeschlossen werden kann, zeigt die bisherige Erfahrung, dass Leistungsgewinne relativ schnell durch neue und aufwändigere Funktionen bzw. Dienste wieder zunichtegemacht werden. Diese Beobachtung wird gestützt durch das Wirth'sche Gesetz, welches besagt, dass Software in kürzerer Zeit langsamer wird als Hardware schneller wird [WIRTH95]. Dabei basiert Wirths Aussage wiederum auf dem Parkinson'schen Gesetz, welches besagt, dass Arbeit genau in dem Maße ausgedehnt wird, wie Zeit zu ihrer Erledigung verfügbar ist [PARK57]. Somit kann vorausgesetzt werden, dass die Überlastproblematik auf andere Weise gelöst werden muss als durch das reine Bereitstellen weiterer Ressourcen.

Im Rahmen dieser Arbeit handelt es sich bei Überlast um den Zustand eines Systems, das mit einer Menge Anfragen konfrontiert ist, welche die zur Verfügung stehenden Mittel übersteigt. Das bedeutet, unter Überlast wird hier gewissermaßen eine Art Datenstau verstanden. Alternativ kann Überlast auch einen Zustand beschreiben, in dem die Betriebsparameter einer Anlage überschritten werden, so dass diese beschädigt bzw. zerstört werden kann. Dieser Aspekt spielt hier aber keine Rolle. Es werden nur Systeme betrachtet, bei denen im Fall einer Überlast ein Einbruch der Systemleistung eintritt, was verhindert werden soll. Diese Definition entspricht dem englischen Begriff *congestion*.

Die Entstehung einer typischen Überlast erklärt sich insbesondere durch den Zusammenhang zwischen leistungssteigernden und leistungsmindernden Effekten bei einer zunehmenden Belastung des Systems [HEIß87]. Zuerst wird auf die steigernden Effekte eingegangen. Diese kann man auch als leistungsausnutzende Effekte bezeichnen. Je mehr Anfragen an ein System gestellt werden, desto besser kann das System seine Ressourcen auslasten und somit eine größere Gesamtleistung pro Zeiteinheit erbringen. Bezogen auf das Beispiel eines Rechnersystems bedeutet dies, je mehr Prozesse gleichzeitig laufen, ein desto größerer Teil des Prozessors und Arbeitsspeichers kann ausgelastet werden. Somit wird die Gesamtleistung des Systems bei steigender Nachfrage gesteigert. Diese Steigerung

setzt sich so lange fort, bis z. B. die Kapazität des Speichers erschöpft ist und hier keine weiteren leistungssteigernden Effekte mehr auftreten.

Parallel existieren die leistungsmindernden Effekte. Diese sind gewissermaßen die Verwaltungskosten des Systems. Wenn ein System einen gewissen Anteil seiner Ressourcen zur eigenen Verwaltung aufwendet, stehen diese für die Erbringung der Nutzleistung nicht mehr zur Verfügung. Im Fall eines Rechnersystems ist z. B. der Aufwand eines Prozesswechsels ein solcher leistungsmindernder Effekt.

Kritisch wird es nun insbesondere, wenn alle leistungssteigernden Effekte ausgeschöpft sind und die Last, also im Beispiel die Anzahl der Prozesse, weiter steigt. Von nun an steigen nur noch die leistungsmindernden Effekte an. Ohne Gegenmaßnahmen gelangt das System irgendwann an einen Punkt, an dem es vollständig mit seiner eigenen Verwaltung ausgelastet ist. Konkret wäre ein Rechner nur noch mit dem ständigen Wechseln der Prozesse beschäftigt, ohne dass die einzelnen Prozesse irgendeine Arbeit verrichten können. Ab diesem Punkt ist das System im schlimmsten Fall vollkommen blockiert und kann sich ohne äußeren Eingriff nicht mehr aus diesem Zustand befreien. Fast jedem Computernutzer sind Zustände dieser Art bekannt, die in der Praxis meistens durch eine Überlastung des virtuellen Speichersystems entstehen, so dass der Rechner fast vollständig mit dem Austauschen von Speicherseiten ausgelastet ist. Abbildung 3 verdeutlicht die beschriebenen Effekte anhand einer typischen Überlastkurve.

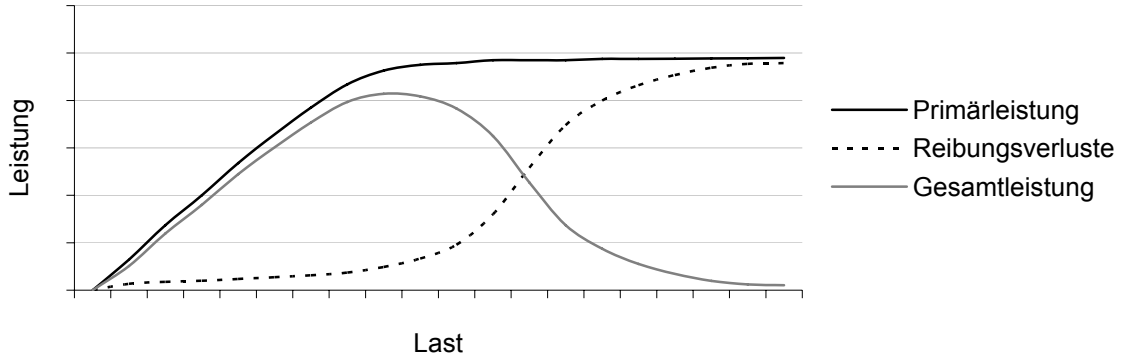


Abbildung 3: Typische Überlastkurve eines Rechnersystems

Neben einem einzelnen Rechnersystem können sich insbesondere auch Computernetze in einem Überlastzustand befinden. Die Überlast einer Netzwerkverbindung kann zwei Ausprägungen haben. Einerseits kann eine Point to Point Verbindung in Überlast geraten, was mit der Überlast eines einzelnen Rechnersystems vergleichbar ist, da der Endpunkt der

Verbindung die Überlast handhaben muss. Andererseits kann ein geteiltes Medium in Überlast geraten. Wird probiert, über das Medium zu viele Nachrichten zu versenden, entstehen immer mehr Kollisionen, bis keine Kommunikation mehr möglich ist. Hier kann die Überlast nicht durch einen einzelnen Rechner bekämpft werden, sondern alle Rechner müssen diese Aufgabe gemeinsam angehen wie z. B. mittels eines exponential backoff entsprechend dem ursprünglichen ALOHA System [ABRAM70].

Das Überlastphänomen als solches existiert auch in vielen anderen Bereichen außerhalb der Informatik. So kann ein Computersystem in Überlast mit einem klassischen Stau im Straßenverkehr verglichen werden. Sobald eine gewisse Anzahl von Fahrzeugen pro Fahrspur überschritten wird, behindern sich die Fahrzeuge gegenseitig so sehr, dass der Gesamtdurchsatz der Straße trotz steigenden Verkehrsaufkommens sinkt [MSRS04].

Angesichts dieser Breite des Überlastphänomens beschränkt sich die vorliegende Arbeit auf die Bekämpfung von Überlast bei Diensten in Rechnernetzen. Das bedeutet, es gibt einen Dienstanbieter, der z. B. eine Webseite oder auch eine Netzwerkverbindung zur Verfügung stellt und diesen Dienst vor Überlast schützen möchte. Das wiederum heißt, der Überlastfall eines geteilten Mediums wird nicht bearbeitet, aber jeder andere Dienst, der nach außen über einen oder mehrere Rechner zur Verfügung gestellt wird, kann nach dem entwickelten Verfahren vor Überlast geschützt werden.

Im Folgenden wird bei der Beschreibung einer Überlast die folgende Terminologie verwendet. Das System, welches belastet wird und somit in eine Überlastsituation geraten kann, wird als *Dienstanbieter* oder *Dienst* bezeichnet. Die Einheit, welche den vom Dienstanbieter zur Verfügung gestellten Dienst nutzt, wird als *Klient* bezeichnet. Die einzelnen Aufgaben, welche beim Dienst eintreffen, werden *Anfragen* genannt.

3 Fairness in Überlastsituationen

Im vorangegangenen Kapitel wurden Überlastsituationen aus der Sicht des Dienstanbieters betrachtet. Dieser hat primär ein Interesse daran, die Überlast zu kontrollieren, um den Dienst aufrechtzuerhalten. Unabhängig davon, welches Verfahren verwendet wird, basieren alle Ansätze darauf, den Zufluss an Anfragen zu drosseln. Das bedeutet im Umkehrschluss aus Klientensicht, dass unweigerlich Anfragen verzögert werden bzw. unbeantwortet bleiben und mit einer Fehlermeldung enden. So einfach es ist, die Last des Servers zu drosseln, so schwer ist es, eine sinnvolle Entscheidung zu treffen, welche Anfragen bearbeitet und welche abgewiesen werden. Das Ziel dieser Arbeit besteht darin, diesen Prozess möglichst fair zu gestalten. Im folgenden Abschnitt wird aus diesem Grund ein entsprechender Fairnessbegriff erarbeitet und daraus die konkretisierte Zielstellung der Arbeit hergeleitet.

3.1 Fairness

Im Sport wird der Begriff Fairness verwendet, um ein regelkonformes und vorbildliches Verhalten zu bezeichnen. Hierbei ist insbesondere eine Handlungsweise gemeint, die dem Prinzip folgt, sich immer so zu verhalten, wie man es von seinem Gegenüber in einer vergleichbaren Situation auch verlangen würde. Demnach geht Fairness in vielen Aspekten deutlich über die bloße Befolgung der formalen Regeln hinaus.

Ein weiterer Bereich, in dem traditionellerweise der Fairnessbegriff verwendet wird, ist das Justizwesen. Insbesondere im Strafverfahren wird allgemein von einem fairen Verfahren gesprochen. Diese aus dem Angelsächsischen übernommene Begrifflichkeit bezeichnet ein Verfahren, das einer Reihe von rechtlichen Grundsätzen entspricht [EURO]. Der Angeklagte wird entsprechend dem Gleichheitsgrundsatz [GRUND] so behandelt, wie es jeder andere Beteiligte des Verfahrens in einer entsprechenden Lage auch erwarten würde.

Fair ist somit das Verhalten, welches jeder von seinem Gegenüber in einer vergleichbaren Situation auch erwarten würde. Das Erlangen eines Vorteils nach Methoden, die, würden sie alle anwenden, zum Zusammenbruch des Systems führen können, ist demnach explizit nicht fair. Dieser Denkansatz lässt sich bis zu Immanuel Kant zurückverfolgen, der ihn in seiner Kritik der praktischen Vernunft wie folgt formulierte: „Handle so, dass die Maxime deines Willens jederzeit zugleich als Prinzip einer allgemeinen Gesetzgebung gelten könnte" und „handle nur nach derjenigen Maxime, durch die du zugleich wollen kannst, daß sie ein allgemeines Gesetz werde" [KANT].

Das beschriebene Konzept wird klassischerweise auf die Fairness zwischen einzelnen Individuen angewandt. Komplexer ist die Frage der Fairness zwischen anderen Einheiten wie z. B. Familien, Arbeitnehmern und Anteilseignern, Organisationen oder ganzen Staaten. Hier stellt sich implizit immer die Frage nach der internen Referenz- oder Verrechnungseinheit, auf deren Grundlage Ressourcen fair verteilt werden können. In manchen Fällen müssen sogar unterschiedliche Aspekte miteinander aufgerechnet werden (z. B. Kapitaleinsatz, Arbeitseinsatz und Risiko) und in einem solchen Fall fällt es schwer, einen ähnlich allgemeingültigen Fairnessgrundsatz zu formulieren. Ganz praktisch wird diese Schwierigkeit bei der Verteilung der Stimmrechte im Rat der Europäischen Union deutlich. Während kleinere Staaten der Meinung sind, ihnen stünden in ihrer Eigenschaft als souveräner Staat bereits Stimmrechte zu, argumentieren die bevölkerungsreicheren Länder, die Stimmrechte müssten entsprechend der Anzahl der Einwohner verteilt werden.

3.2 Fairness in der Informatik

Klassischerweise wird das Thema Fairness von der Informatik im Rahmen des Scheduling betrachtet. Teilweise wird unter Fairness verstanden, dass kein Prozess dauerhaft blockiert wird [FRAN86]. Dieses Verständnis von Fairness wird auch als *freedom from starvation* bezeichnet. Von anderen Autoren wird über diese Forderung hinaus gegangen und gefordert, dass einzelne Prozesse gleiche Chancen haben, auf eine Ressource, wie z. B. die CPU, zuzugreifen. So lange nur ein Benutzer auf dem System arbeitet, stellt das Konzept der Fairness eher einen abstrakten Ansatz dar, der ein insgesamt sinnvolles Funktionieren des Systems ermöglichen soll.

Anders sieht es aus, wenn sich mehrere Benutzer eine Ressource, wie z. B. eine CPU, eine Netzwerkverbindung oder einen Server, teilen. In diesem Fall kommt die Fairness in ihrer klassischen Definition zum Tragen. Man erkennt hier, dass Fairness ein klienten- oder benutzerorientiertes Kriterium ist [MAEH91]. Dies bedeutet im Umkehrschluss, dass eine Lösung auf der Ebene der Klienten oder Benutzer arbeiten muss. Ein reiner Allokationsmechanismus auf Job- oder Prozessebene kann dies nicht leisten.

Nagle liefert mit seinem Fair-Queuing [NAGLE85] einen entsprechenden Lösungsansatz. Dabei formuliert er die Zielstellung wie folgt „More specifically, we would like, in the presence of both well-behaved and badly-behaved hosts, to insure that well-behaved hosts receive better service than badly-behaved hosts“. Im Fall des Fair-Queuing wird für jeden Klienten eine separate Warteschlange verwaltet, die wiederum im Round-Robin Verfahren abgearbeitet werden. Auf diese Weise ist es für einen einzelnen Klienten unmöglich, im

Überlastfall mehr Ressourcen zu beanspruchen, als ihm bei gleichmäßiger Aufteilung der Ressource an alle Klienten zustehen würde. Darüber hinaus führt ein besonders aggressives Verhalten eines Klienten zu einer Überlast in seiner persönlichen Warteschlange, was bedeutet, dass entsprechendes Verhalten nicht nur keinerlei Vorteil bringt, sondern darüber hinaus bestraft wird. Nagle erkennt dabei aber auch schon, dass sein Verfahren in dieser Form nur anwendbar ist, so lange die Klienten technisch gesehen einzeln identifizierbar sind.

3.3 Zielsetzung

Unabhängig vom Aspekt der Fairness ist es das Ziel dieser Arbeit, Überlast effektiv zu kontrollieren. Diese Aufgabe entspricht gewissermaßen der Pflicht, während die Fairness die Kür darstellt. So lange die Überlast nicht kontrolliert wird, ist es wahrscheinlich, dass das System kollabiert und sich die Frage der Ressourcenverteilung somit praktisch gar nicht mehr stellt. Aus diesen Überlegungen ergibt sich die erste Anforderung an den gesuchten Lösungsansatz:

(1a) $L = N$, falls $N < S$

(1b) minimiere $abs(L-S)$, falls $N \geq S$

L: Serverlast
N: Nachfrage der Klienten
S: Sollwerte der Last

Dies bedeutet, so lange die Nachfrage nach Ressourcen ohne Überlast bewältigt werden kann, soll die Last nicht beschränkt werden. Sollte die Nachfrage über dem Sollwert der Last liegen, soll die Abweichung der realen Last vom Sollwert minimiert werden.

Über die reine Überlastkontrolle hinaus soll die Ressourcenzuteilung in Überlastsituationen fair gestaltet werden. Dabei gehen wir von in etwa gleichartigen Anfragen der Klienten aus. Im Fall eines überlasteten Internetdienstes ist diese Annahme legitim, da die typischen Anfragen an den Dienst üblicherweise einen vergleichbaren Ressourcenverbrauch auf der Serverseite nach sich ziehen. Im Rahmen einer weiteren Untersuchung wäre aber auch eine Erweiterung des Modells mit gewichteten Anfragen denkbar.

Unter Fairness verstehen wir in diesem Zusammenhang, dass im Überlastfall kein Klient durch aggressive Anfragemuster mehr Ressourcen des Servers erhalten darf als ein Klient,

der auf normale Weise anfragt. Konkret bedeutet dies, dass die Ressourcenverteilung, vergleichbar mit dem Fair Queuing, den jeweiligen Klienten miteinbeziehen muss und nicht allein auf Anfrageebene arbeiten kann. Dieser Zusammenhang lässt sich wie folgt formalisieren:

(2a) $Z_i \leq N_i$

(2b) $Z_i \geq Z_j$, falls $Z_i < N_i$

Z_i, Z_j: Ressourcenzuteilen für die Klienten i bzw. j
N_i: Nachfrage nach Ressourcen des Klienten i

Forderung 2a besagt, dass kein Klient mehr Ressourcen erhalten soll, als er nachfragt. 2b besagt, sobald ein Klient i existiert, der weniger Ressourcen erhält, als er nachfragt, darf jeder andere Klient j nicht mehr Ressourcen erhalten als dieser Klient i.

Während diese beiden Punkte den Kern der gestellten Anforderungen darstellen, wären weitere weniger formale Eigenschaften wünschenswert. Einerseits soll das Verfahren benutzerfreundlich und gewissermaßen selbsterklärend sein. Ein Benutzer soll zumindest über die Überlast informiert werden, denn das simple Abweisen von Anfragen wird von Benutzern oft als Systemfehler gedeutet. Dies führt zu einer negativen Einschätzung des betroffenen Dienstes und unter Umständen dazu, dass durch wiederholtes Anfragen probiert wird, den Dienst doch noch zu erreichen. Eine klar kommunizierte Überlast mit einem klaren Warteszenario stellt hier den deutlich benutzerfreundlicheren Ansatz dar.

Andererseits ist in einer Überlastsituation auch ein gewisser Abschreckungseffekt von wenig motivierten Klienten wünschenswert. Klienten, die den Service nicht akut benötigen, werden bei einer im Vorhinein erkennbar langen Wartedauer davon Abstand nehmen, den Service zu nutzen, und zu einem späteren Zeitpunkt wieder anfragen. Dieses Verhalten ist im Überlastfall durchaus erwünscht, weil im Zweifel eher die Klienten mit einem besonders dringenden Anliegen bedient werden.

Nachdem die Anforderungen formuliert wurden, wird im nächsten Kapitel untersucht, inwiefern bereits zur Verfügung stehende Verfahren diese Bedingungen erfüllen.

4 Vergleich bekannter Verfahren zur Überlastvermeidung

In diesem Kapitel werden die bekannten Verfahren zur Überlastvermeidung besprochen. Dabei wird insbesondere auf die Fairness der einzelnen Ansätze eingegangen. Daneben wird auch auf die Frage eingegangen, ob und wie im Internet einzelne Klienten überhaupt als solche identifiziert werden können, weil diese Frage bei der Bewertung der Fairness der folgenden Verfahren eine zentrale Rolle spielt.

4.1 Flusskontrolle

Ein klassischer Ansatz zur Überlastvermeidung ist die Flusskontrolle, wie sie z. B. vom TCP Verfahren entsprechend RFC2581 verwendet wird [RFC2581]. An dieser Stelle betrachten wir exemplarisch die *slow start* und *congestion avoidance* Algorithmen. Wird eine TCP Verbindung neu aufgebaut, fehlt dem Sender ein ausreichendes Wissen über die Leistungsfähigkeit des zugrunde liegenden Netzes. Aus diesem Grund wird die Verbindung mit einem kleinen *congestion window* gestartet. Dies bedeutet, der Sender kann nur eine kleine Datenmenge senden, ohne dafür vom Empfänger eine Bestätigung zu erhalten. Sobald der Empfang von Paketen bestätigt wird und das *advertised window* des Empfängers groß genug ist, vergrößert der Sender sein Fenster und setzt somit die Datenrate herauf. Bei einem Paketverlust halbiert er wiederum die Größe seines Fensters. Paketverluste werden somit als Indikator für eine Überlastsituation des Netzwerks herangezogen. Bei besonders fehleranfälligen Übertragungsstrecken, wie z. B. dem Randbereich eines WLAN, fällt dadurch die Übertragungsrate stärker als gewünscht ab [PH07], im normalen Fall arbeitet dieses Verfahren aber relativ effizient. Wenn Router im Überlastfall Pakete verwerfen müssen, hat dies zugleich den Effekt, dass der Zustrom an neuen Paketen gedrosselt wird.

Der große Nachteil dieses Verfahrens besteht darin, dass eine aktive Mitwirkung der Beteiligten erforderlich ist. So existieren TCP Implementierungen, die im Fall von Paketverlusten nicht standardkonform vorgehen und ihre Datenrate nicht entsprechend drosseln [MWD07]. Auf diesem Weg kann sich ein einzelner Teilnehmer im Netz leicht einen erheblichen Vorteil verschaffen. Das bedeutet, die Flusskontrolle begrenzt streng genommen weder die Last zuverlässig, noch ist sie fair, da aggressives Klientenverhalten belohnt wird. Praktisch funktioniert dieses Verfahren bisher wohl nur deshalb relativ reibungslos, weil sich die großen Hersteller von TCP Implementierungen weitgehend an den Standard halten und die einzelnen Benutzer meistens technisch nicht in der Lage sind, entsprechende Veränderungen vorzunehmen. Aus dieser Betrachtung folgt, dass ein

Verfahren nach dem Vorbild der Flusskontrolle nicht die im Rahmen dieser Arbeit angestrebte faire Überlastkontrolle garantieren kann.

4.2 Identifikation von Klienten

Die meisten Verfahren, welche behaupten, fair zu arbeiten, machen diese Fairness daran fest, wie die Ressourcen auf die einzelnen Benutzer verteilt werden. Dabei wird allgemein angenommen, dass sich alle Anfragen eines Benutzers durch den Dienstanbieter leicht einander zuordnen lassen. Im Folgenden wird gezeigt, dass dies insbesondere auf Grund neuerer Entwicklungen der Netzwerktechnik nicht der Fall ist.

Um dies zu belegen, betrachten wir zuerst den Standardfall, welcher von den meisten Autoren implizit angenommen wird. Danach verfügt jeder Benutzer über einen Rechner und dieser Rechner verfügt über eine feste öffentliche IP Adresse. Heutzutage ist es aber so, dass aus unterschiedlichen Gründen nur noch ganz wenige Rechner, wie z. B. Arbeitsplatzrechner der TU-Berlin im Class B Netz 130.149.*.*, auf diese Weise angebunden sind.

4.2.1 Dynamische IP Adressen

Der erste Faktor, der es erschwert, Klienten bzw. Benutzer zu identifizieren, sind die dynamischen IP Adressen der großen Internetprovider. Bei jeder Einwahl erhält der Rechner eine neue IP Adresse aus dem Pool des Providers. Das bedeutet, durch ein kurzes Trennen und Wiederherstellen der Netzwerkverbindung können sich Klienten eine neue „Identität“ verschaffen. Im Rahmen der Strafverfolgung ist es unter Umständen möglich, die Benutzer anhand der Zeit und der zugewiesenen IP Adresse zu identifizieren [GKR07], aber für die Überlastvermeidung muss davon ausgegangen werden, dass diese Daten nicht zur Verfügung stehen.

4.2.2 IP Masquerading

Auf Grund der mehr oder weniger gescheiterten Einführung von IPv6 [RFC2460] und der daraus folgenden Knappheit an IPv4 [POSTEL81] Adressen sowie der stark verbreiteten gemeinsamen Nutzung von breitbandigen Internetzugängen durch mehrere Benutzer hat sich das Verfahren der Network Address Port Translation (NAPT) [SE01] / IP Masquerading stark durchgesetzt. Hierbei nutzen mehrere Rechner gemeinsam die Internetverbindung eines einzelnen Rechners. Dieser ist mit den restlichen Rechnern über ein Netzwerk mit privaten IP Adressen verbunden (typischerweise im dafür vorgesehenen Klasse B Netz

192.168.*.* [RE96]), nimmt Verbindungen entgegen und leitet sie ins Internet weiter, als seien es seine eigenen Verbindungen. Obwohl keine passende Statistik vorliegt, kann angenommen werden, dass insbesondere die Nutzer von DSL-Zugängen mit ihren Rechnern fast ausschließlich per Masquerading an das Internet angebunden sind. Laut [BROWN06] ist in einigen Ländern der Welt weniger als eine IP für tausend Einwohner verfügbar. Sollen diese Einwohner alle an das Internet angebunden werden, wird massives IP Masquerading unumgänglich sein.

Das bedeutet, über die IP Adresse kann nicht erkannt werden, ob Anfragen mit dieser Absenderadresse von einem oder mehreren Rechnern stammen. Insbesondere im Fall der immer größer werdenden privaten Netze, die komplett über eine IP auf das Internet zugreifen, stellt dies ein Problem dar, im Überlastfall trotzdem eine geeignete Fairness durchzusetzen.

4.2.3 Proxyserver

Durch Proxyserver entsteht ein mit dem Masquerading vergleichbares Problem. Es ist nicht ersichtlich, welche und wie viele Rechner bzw. Benutzer sich hinter den Anfragen verbergen. Im Fall der Proxy Server ist das Problem praktisch sogar noch gravierender, da diese meistens noch viel größere Mengen an Benutzern bündeln als das IP Masquerading. Große Internetprovider leiten einen Teil ihrer Anfragen durch wenige Server, so dass bei nicht für den Cache geeigneten Inhalten Anfragen tausender Benutzer beim Dienstanbieter unter der gleichen Absender IP erscheinen. Insbesondere im Fall des transparenten Proxy hat der normale Benutzer auch keine einfache Möglichkeit, diesen im Überlastfall zu umgehen. Besonders verbreitet ist diese Praxis bei UMTS Zugängen, selbst wenn die Provider nicht gerne darüber reden, da hier oft gleichzeitig über den Proxy Bilder nachkomprimiert werden, um das UMTS Netz zu entlasten [IH04]. Im Gegensatz zu IP Masquerading sind Proxy Server mit steigenden Bandbreiten der Zugänge und immer interaktiveren und somit schlechter zu cachenden Inhalten, bis auf wenige Ausnahmen wie den UMTS Zugängen, eher dabei, an Verbreitung zu verlieren.

4.2.4 Terminal-Server

Während in der Anfangszeit der Computertechnik Mainframe Rechner mit mehreren per Terminal angemeldeten Benutzern die Regel waren, hatte sich insbesondere mit dem Erscheinen des PCs eine Workstation und Personal Computer Landschaft durchgesetzt. In jüngster Zeit gibt es aber wieder Bewegungen zurück zum zentralisierten Rechner mit Thin

Clients [GLO07]. Auch die TU-Berlin hat einen Teil der Sun Workstations durch Sun Ray Thin Clients ersetzt. Diese Entwicklung führt aber wiederum dazu, dass viele Benutzer aus der Sicht eines Dienstes im Internet unter einer einzigen IP erscheinen und somit eine Ressourcenverteilung nach IP Adressen nicht als besonders fair angesehen werden kann.

4.2.5 Multi IP Hosts

Nachdem eine Reihe von Fällen aufgezählt wurde, die dazu führen, dass mehrere Benutzer unter einer identischen IP Adresse auftauchen, soll die Existenz des gegenteiligen Falls nicht verschwiegen werden. Dieser ist zwar seltener anzutreffen, aber nichtsdestotrotz gibt es Kontingente an faktisch nicht genutzten IP Adressen, welche z. B. von Firmen weiterhin gehalten werden. So verfügen teilweise selbst sehr kleine Firmen mit einer symmetrischen DSL Anbindung über Subnetze mit 32 Adressen und mehr. In einem solchen Fall ist es leicht möglich, mit nur einem Rechner unter einer Vielzahl von Adressen aufzutreten. Ein solches Vorgehen kann ganz praktisch, z. B. im Fall eines Fair Queuing auf IP Basis, einen erheblichen Vorteil bedeuten, wenn die Chance, Anfragen beantwortet zu bekommen, um ein Vielfaches steigt.

4.2.6 Zusammenfassung

Es wurde gezeigt, dass es in der momentanen Struktur des Internets nicht sinnvoll ist, für Fairnessbetrachtungen die IP Adresse des Klienten heranzuziehen. In zu vielen Fällen verbergen sich hinter einer Adresse viele unterschiedliche Rechner bzw. Benutzer (Masquerading, Proxy, Terminal-Server) und in anderen Fällen kann ein identischer Benutzer sich hinter wechselnden IP Adressen verstecken (dynamische IP Adressen, Multi IP Hosts).

Diese Feststellung bedeutet für alle im Folgenden untersuchten Verfahren, dass nicht nur klar sein muss, nach welchem Schlüssel die Aufteilung der Ressourcen im Überlastfall erfolgen soll, sondern sobald der Klient eine Rolle spielt, muss geklärt werden, wie die einzelnen „echten“ Klienten identifiziert werden können.

4.3 Warteschlangen

Wartenschlagen stellen sowohl in der Informatik als auch in anderen Lebensbereichen die gebräuchlichste Form der Überlastreaktion dar. Dabei erfüllen sie die folgenden Aufgaben:

- Es erfolgt eine Begrenzung des Zuflusses an Anfragen in das eigentliche Bediensystem und somit wird die Kernaufgabe jedes Systems zur Überlastvermeidung behandelt.

- Verbesserung der Auslastung des Systems: Je größer die Varianz der Eintreffraten, desto länger sollte die Warteschlange sein, um eine optimale Auslastung des Bediensystems zu bewirken.
- Abschreckung: Die Aussicht auf eine lange Wartedauer hilft, die Anzahl der neu eingehenden Anfragen zu reduzieren. Insbesondere tendenziell „unwichtige“ Anfragen werden so eventuell vom Kernsystem ganz ferngehalten.
- Überlast verwalten: Theoretisch kann sich eine Warteschlange dauerhaft weiter füllen und die Überlast wird so kontinuierlich als eine Art Bugwelle jeweils in die Zukunft verschoben.

Angewendet auf das im Rahmen dieser Arbeit untersuchte Problem der Überlast von Diensten in einer großen Netzwerkstruktur bedeutet dies, dass die Begrenzung des Zuflusses von Anfragen zweckmäßig ist. Die Verbesserung der Auslastung durch das Mitteln von Anfragen spielt hingegen in einer entsprechenden Situation kaum eine Rolle. In einem realen Überlastszenario dürfte eine typische Warteschlange schnell so weit gefüllt sein, dass es wegen clientseitigem Timeout keinen Sinn macht, noch weitere Anfragen anzunehmen. Das bedeutet, eine klassische Warteschlange kann ohne nachgeschaltete Abweisung kein alleiniges Mittel zur Bekämpfung der Überlast sein.

Die Fairness der Warteschlange hängt von der verwendeten „Disziplin“ ab. Diese bestimmt, in welcher Reihenfolge die Anfragen aus der Schlange abgearbeitet werden. Das am weitesten verbreitete Verfahren ist Fist-In.-First-Out (FIFO) bzw. First-Come-First-Serve (FCFS). Hier werden die Anfragen entsprechend ihrer Ankunft in der Warteschlange abgearbeitet. Dieses Verfahren kann zumindest im Fall eines Dienstes in einem Rechnernetz nicht als fair betrachtet werden, da sich aggressive Klienten durch übermäßige Anfragen einen erheblichen Vorteil verschaffen können und somit eine der weiter oben ausgearbeiteten Anforderungen an ein faires System nicht erfüllt werden kann.

Als Alternative zu FIFO wurde das Fair Queuing entwickelt. Hier wird eine FIFO Warteschlange für jeden Klienten separat verwaltet. Diese Warteschlangen werden gleichberechtigt im Round-Robin Verfahren abgearbeitet. Durch dieses Verfahren soll eine gleichmäßige Verteilung der Ressourcen garantiert werden. Generell kann hier von einem theoretisch fairen System gesprochen werden, nur gibt es keinen Hinweis, wie die Anfragen in der Praxis einzelnen Klienten zugeordnet werden sollen. Und wie weiter oben beschrieben, ist eine einfache Zuordnung nach IP Adressen nicht akzeptabel.

4.4 Abweisung

Keine reale Warteschlange kann unendlich viele Anfragen aufnehmen. Aus diesem Grund wird bei anhaltender Überlast früher oder später der Punkt erreicht, an dem Anfragen abgewiesen werden müssen. Alternativ kann die Abweisung auch direkt ohne Warteschlange vor das Bediensystem geschaltet werden. Durch das Abweisen von Anfragen kann die Überlast sehr effektiv kontrolliert werden und somit ist dies aus der Sicht der reinen Überlastvermeidung sogar die beste denkbare Lösung, um eine Überlastsituation zu verhindern. In anderen Bereichen, wie z. B. den Stromnetzen, wird dieses Verfahren ebenfalls eingesetzt. Hier wird der Vorgang „Lastabwurf“ genannt, er entspricht aber logisch dem simplen Abweisen von Klienten bzw. Verbrauchern.

Die eigentliche Schwierigkeit besteht aber darin zu entscheiden, welche Anfragen abgewiesen werden und welche nicht. Per se ist eine Abweisung nicht unbedingt als unfair einzustufen, aber die Auswahl der abzuweisenden Anfragen muss nach fairen Kriterien erfolgen. Aus diesem Grund werden im Folgenden die gebräuchlichsten Entscheidungsverfahren untersucht. Dabei wird zwischen gedächtnislosen und zustandsbehafteten Verfahren unterschieden.

4.4.1 Gedächtnislose Verfahren

Die einfachste Art der Abweisung untersucht bei jeder Anfrage, ob im Moment des Eintreffens ein Platz im nachgelagerten Bediensystem bzw. in der Warteschlange frei ist. Steht ein Platz zur Verfügung, wird die Anfrage angenommen, ansonsten wird sie abgewiesen. Diese Vorgehensweise arbeitet schnell und effizient. Der Nachteil besteht darin, dass alle Anfragen gleich behandelt werden müssen. Das bedeutet, aggressiv anfragende Klienten können sich erhebliche Vorteile gegenüber anderen Klienten verschaffen. Entsprechend der Definition dieser Arbeit kann diese Form der Abweisung nicht als fair bewertet werden.

Als Erweiterung des Modells können den Klienten Prioritäten zugewiesen werden. Zu diesem Zweck verfügt das Abweisungssystem intern über einen kleinen Puffer für Anfragen. Dieser wird über eine Zeiteinheit gefüll, die Anfragen werden entsprechend ihrer Priorität gewichtet, die zur Verfügung stehenden Ressourcen werden vergeben und alle anderen Anfragen abgewiesen. Hierbei ist einerseits ein Modell mit absoluten Prioritäten denkbar, was bedeutet, dass immer erst alle Anfragen der höheren Priorität bearbeitet werden und nur wenn danach noch Ressourcen verfügbar sind, kommen die Anfragen der nächstniedrigeren Priorität zum Zug. Alternativ kann die Priorität als Gewichtungsfaktor

für die ansonsten probabilistische Auswahl der Anfragen dienen, was ein komplettes Blockieren des Dienstes durch Anfragen hoher Priorität verhindert. Dieses Verfahren liefert aber keine Lösung dafür, wie die Prioritäten fair vergeben werden sollen. Darüber hinaus können sich Klienten weiterhin durch aggressives Anfrageverhalten einen Vorteil verschaffen, der ihnen entsprechend dem Priorisierungssystem einen Vorteil gegenüber anderen Klienten mit gleicher Priorität bzw. gegenüber allen anderen Klienten verschafft. Im untersuchten Fall eines Dienstes im Internet kann die Abweisung somit auch durch die Einführung von Prioritäten nicht fair gestaltet werden.

4.4.2 Zustandsbehaftete Verfahren

Eine zustandsbehaftete Abweisung benutzt als Entscheidungsgrundlage das vorangegangene Anfrageverhalten des Klienten. So kann z. B. immer derjenige Klient bevorzugt werden, der bisher am wenigsten Ressourcen verbraucht hat, oder Klienten werden Quotas für ihre Zugriffe zugewiesen. Prinzipiell können an dieser Stelle die meisten Scheduling Systeme angewendet werden, die aus dem Bereich der Betriebssysteme bekannt sind. Allen zustandsbehafteten Systemen ist aber gemein, dass sie eine Art Historie der Zugriffe der einzelnen Klienten verwalten müssen. Weiter oben wurde jedoch gezeigt, dass diese Zuordnung von Anfragen zu Klienten kaum verlässlich möglich ist und deswegen sind all diese Verfahren im konkreten Szenario nicht einsetzbar bzw. als nicht fair einzustufen.

4.4.3 Zusammenfassung Abweisung

Abweisungsverfahren stellen eine sehr effiziente Methode dar, Überlast zu vermeiden. So lange Anfragen und Klienten aber nicht sicher zugeordnet werden können, ist kein Verfahren bekannt, das die geforderte Fairness umsetzt.

4.5 Reputationssysteme

Ein alternativer Weg, Benutzer zu fairem Verhalten zu veranlassen, sind Reputationssysteme. In einem solchen System sind die Benutzer bestrebt, sich vorbildlich zu verhalten, um eine gute Bewertung zu erhalten. Das eBay Bewertungssystem ist ein geeignetes Beispiel für ein solches Reputationssystem. In diesem Beispiel lässt sich die gute Bewertung langfristig auch in gute Geschäfte ummünzen. Andere Bewertungssysteme basieren nur auf dem Willen der Teilnehmer, sozial akzeptiert zu sein. Eine weitere Form von Reputationssystemen sind Blacklists, wie sie z. B. zur Spam-Bekämpfung benutzt werden.

So effizient ein solches System arbeiten kann, es ergeben sich gewisse Schwierigkeiten bei der Umsetzung. Im Fall der Blacklists erfolgt die Eintragung üblicherweise nach IP Adressen. Wie weiter oben gezeigt wurde, ist dies aber keine geeignete Methode, um Klienten zu identifizieren, so dass die Blacklists unter anderem aus diesem Grund sehr umstritten sind. Soll die Identifikation der Klienten auf anderem Weg erfolgen, wird eine Art Login notwendig, welches das Vorhandensein zumindest eines minimalen Benutzerkontos voraussetzt. Damit entsteht sowohl ein erheblicher Verwaltungsaufwand als auch ein für den Benutzer unbequemer Prozess inklusive eines weiteren Passworts, das vergessen werden kann.

Darüber hinaus muss das Reputationssystem eine gewisse Größe erreichen und es darf nicht zu viele parallele Systeme geben, da sonst die Bewertung durch die Vielzahl der Systeme an Wert verliert. Sollten sich mehrere Dienste zu einem Reputationssystem zusammenschließen, erhöht dies zwar die Chance eines Erfolges des Systems, es bringt aber neuen Verwaltungs- und Sicherheitsaufwand für die Übermittlung und Verwaltung des Systems mit sich.

Während sich normale Benutzer unter Umständen durch ein Reputationssystem zu einem verbesserten Verhalten bewegen lassen, bietet es viel Angriffsfläche für bösartige Angreifer, wie sie im Falle eines Denial-of-Service Angriffs zu erwarten sind. Trotz Absicherung durch Bilderrätsel, welche auch als *Completely Automated Public Turing Test to Tell Computers and Humans Apart* (CAPTCHA) [ABHL03] [ABL04] bekannt sind, existieren Möglichkeiten, dieses zu umgehen und so eine große Anzahl an Accounts anzulegen. Dieses Problem ließe sich nur mit massivem administrativem Aufwand und unter datenschutzrechtlichen Bedenken beheben. Darüber hinaus dürften sich diese Klienten vom Reputationssystem nicht von ihren Anfragen abhalten lassen, sondern getreu Willhelm Buschs Motto „Ist der Ruf erst ruiniert, lebts sich gänzlich ungeniert.“ vorgehen.

Wir fassen zusammen, dass Reputationssysteme ein Weg sein können, Fairness in Überlastsituationen herzustellen. Dies wird aber nur in Szenarien funktionieren, wo dieses System zumindest teilweise bereits aus einem anderen Grund vorhanden ist und die Benutzer sich sowieso authentifizieren. So könnte z. B. ein solches Verfahren in Social Networks wie Facebook, welche ausschließlich für angemeldete Benutzer zur Verfügung stehen, eingesetzt werden. Alle Benutzer hätten im Überlastfall die gleiche Möglichkeit, auf den überlasteten Dienst zuzugreifen, und müssten dafür mit einem Teil ihrer Reputation „bezahlen“. Auch vermehrte Zugriffe eines besonders an dem Dienst interessierten

Benutzers wären in diesem Modell möglich, indem dieser dann etwas mehr seiner Reputation aufgibt. Dieser Anwendungsfall beschränkt sich aber nur auf einen kleinen Teil der denkbaren Dienste, da im Standardfall kein funktionierendes Reputationssystem zur Verfügung steht. Wie im nächsten Abschnitt ersichtlich wird, sind Reputationssysteme mit den Bezahlverfahren verwandt, nur dass hier sozusagen mit dem guten Ruf bezahlt wird.

4.6 Bezahlverfahren

Die im Rahmen dieser Arbeit betrachteten Dienstanbieter erbringen ihre Leistungen kostenlos. Das bedeutet nicht, dass kein wirtschaftliches Interesse an einer Aufrechterhaltung des Dienstes besteht, da dieser Produkte und Leistungen bewerben kann bzw. zum Abschluss von Geschäften verwendet wird.

Als ein weiteres Modell wäre es nichtsdestotrotz möglich, zur Vermeidung von Überlastsituationen Anfragen mit einer Art Schutzgebühr zu belegen. Wie bereits kurz in der Einleitung erwähnt, ist die Grundaufgabe eines Marktes der Ausgleich zwischen Angebot und Nachfrage. Somit könnte eine Überlastsituation verhindert werden, indem über den Preis für eine Anfrage die Last beeinflusst wird. Ein solches Verfahren ist geeignet, die Überlast zu kontrollieren. Alle Klienten haben die gleichen Chancen, Anfragen zu kaufen. Aggressive Verhaltensweisen von einzelnen Klienten führen zu keinem Vorteil gegenüber anderen Klienten und einzelne Klienten mit einem besonderen, leicht gesteigerten Ressourcenbedarf können sich diese Anfragen durch einen kleinen Aufpreis erkaufen. Da es sich im Zweifel eher um Micropayments handeln dürfte, entsteht auch keine zu große Ungerechtigkeit zwischen Nutzern mit unterschiedlichen Vermögensverhältnissen, da es mehr um die psychologische Frage geht, einen sehr kleinen Betrag zu bezahlen, als um die Frage, sich die Summe leisten zu können [SZABO96].

Damit wäre ein System gefunden, welches die weiter oben geforderten Eigenschaften sowohl bezüglich der Überlastvermeidung als auch der Fairness erfüllen kann. Unglücklicherweise stehen aber diverse Argumente einem Verfahren, welches mit Micropayments in üblicher Währung arbeitet, entgegen. Hierauf wird im Folgenden eingegangen.

4.6.1 Kompatibilität und Verbreitung

Bisher existiert kein echtes Micropayment Verfahren, das eine ansatzweise zufriedenstellende Verbreitung erreicht hat. Es stehen zwar einige internettaugliche Bezahlverfahren zur Verfügung, aber keines ist wirklich für Micropayments im Bereich

von einzelnen Cents oder sogar darunter geeignet [BSI04]. Es wurden viele inkompatible Systeme entwickelt und die überwiegende Mehrheit der Nutzer nimmt an keinem dieser Systeme teil. Darüber hinaus ist sehr fraglich, ob sich dieser Zustand in absehbarer Zeit ändern wird. Das aktuell weltweit am weitesten verbreitete und ansatzweise für das Micropayment taugliche System dürfte die Zahlungsplattform Paypal sein. Aber selbst dieses System ist sehr weit davon entfernt, praktisch zur Überlastvermeidung einsetzbar zu sein.

4.6.2 Sicherheit

Sobald mit realen Zahlungsmitteln gearbeitet wird, verlangen Benutzer hohe Sicherheitsstandards. Weil die Beträge minimal sind, ist dies teilweise irrational. Ein Service, der einem Benutzer kostbare Arbeitszeit „stielt", richtet rein rechnerisch einen deutlich größeren Schaden an als eine fehlerhafte Micropayment Bezahlung. Die Benutzer reagieren in diesen Fällen aber sehr unterschiedlich und lassen sich z. B. ihre Zeit lieber „stehlen" als den Bruchteil eines Cent bei einem Miropayment. Daraus folgt, dass ein System zur Überlastvermeidung auf der Basis von Micropayments sehr stark abgesichert werden müsste. Dies erfordert einen umfangreichen technischen Aufwand bis zu Transport Layer Security (TLS) Verbindungen, die erhebliche Ressourcen verbrauchen [BAD01]. Damit wäre das Verfahren aber gerade im Überlastfall sehr kontraproduktiv.

4.6.3 Regulatorische Auswirkungen

Sobald reale Zahlungsmittel international transferiert werden, greift eine ganze Palette von juristischen Regularien bezüglich Steuern, Geldwäsche etc. Obwohl im Fall der Micropayments das Geld zur Überlastvermeidung nur als eine Art Schutzgebühr ohne Gewinnerzielungsabsicht erhoben wird, müssen diese Fragen sehr aufwändig geklärt werden.

4.6.4 Verhinderung des Double Spending

In einem Micropayment System muss verhindert werden, dass Tokens mehrfach ausgegeben werden. Dazu ist in der Regel eine Art zentrale Infrastruktur notwendig, die als vertrauenswürdige Referenzstelle bestätigt, dass ein Token nicht von einem Benutzer mehrfach benutzt wurde. Diese Infrastruktur kann sehr schnell einen neuen Flaschenhals darstellen und widerspricht dem Wunsch nach einer dezentral einsetzbaren Lösung.

4.6.5 Zusammenfassung Bezahlverfahren

Obwohl die Grundidee der Überlastvermeidung mittels Bezahlverfahren die grundlegenden Anforderungen erfüllt, zeigen die Ausführungen bezüglich der Probleme einer praktischen Implementierung große Schwierigkeiten auf. Diese dürften zumindest in naher Zukunft nicht überwunden werden, weshalb alternative Lösungsansätze gesucht werden müssen, um das Problem der Fairness in Überlastsituation zu lösen.

4.7 Zusammenfassung Überlastvermeidung

In diesem Kapitel wurde gezeigt, dass eine eindeutige Identifizierung von einzelnen Klienten im Internet praktisch kaum möglich ist. Daraufhin wurden Warteschlangen- und Abweisungsverfahren zur Überlastvermeidung behandelt, wobei sich herausstellte, dass hier keine Fairness im Sinn dieser Arbeit hergestellt werden kann. Reputationsplattformen könnten in bestimmten Anwendungsfällen, in denen eine solche Plattform bereits existiert, zur fairen Überlastvermeidung eingesetzt werden, aber einen generellen Lösungsansatz stellen sie ebenfalls nicht dar. Bezahlverfahren erscheinen theoretisch gut geeignet, es gibt aber zu viele Hindernisse, welche der praktischen Implementierung entgegenstehen. Aus diesem Grund werden im sich anschließenden Kapitel die Proof-Of-Work Funktionen eingeführt, welche eine Lösung des Problems jenseits der Schwächen existierender Ansätze versprechen.

5 Proof-Of-Work Funktionen

Der Grundgedanke einer Proof-Of-Work (POW) Funktion ist, dass ein Partner einem anderen auf einfache Weise belegen kann, dass er eine gewisse ressourcenintensive Aufgabe abgearbeitet hat. Das eigentliche Ergebnis dieser Aufgabe ist dabei primär nicht von Interesse, sondern es geht darum, die Ernsthaftigkeit eines Anliegens zu unterstreichen. Somit ist das Konzept eines POW mit einer Schutzgebühr oder sonstigen Maßnahmen vergleichbar, die dazu dienen, nicht ernsthafte Anfragen bzw. Interessenten abzuwehren.

Während außerhalb der Informatik Ideen wie die Schutzgebühr schon lange existierten, geht die Idee des POW auf die Arbeit *Pricing via Processing or Combatting Junk Mail* von Dwork und Naor aus dem Jahr 1992 zurück [DN92]. In dieser Arbeit wird vorgeschlagen, für den Versand jeder E-Mail ein solches POW zu verlangen, um dem schon seit 1978 [TEMP03a] existierenden Problem von Unsolicited Bulk E-Mail (UBE) Herr zu werden. Seit 1993 wird in dieser Beziehung auch der Begriff Spam [TEMP03b] verwendet.

Dieser Ansatz basiert auf der Grundbeobachtung, dass Spam-Versender sehr hohe Mengen an E-Mails versenden müssen, um gegen die sehr niedrigen Antwortraten anzukämpfen, während sonstige Benutzer nur relativ wenige E-Mails versenden. Nach Dwork und Naor soll die Abarbeitung des Proof-Of-Work die legitimen Benutzer nur marginal bremsen, während der Massenversand von Spam-Mails stark verzögert wird. Die Forschung von Laurie und Clayton [LC04] sowie die des Autors [GMW05] haben zwar gezeigt, dass ein ausreichend effektives Proof-Of-Work deutlich größer sein muss als ursprünglich angenommen, nichtsdestotrotz gibt es unterschiedliche Ansätze, POW Systeme gegen Spam E-Mail einzusetzen. Die entscheidende Verbesserung besteht darin, nur noch beim Erstkontakt ein POW zu fordern, welches dann auch entsprechend größer sein kann. Die Frage, wie sich Mailinglisten in ein solches POW basiertes Verfahren integrieren lassen, bleibt dagegen weitgehend ungeklärt.

Proof-Of-Work Funktionen werden in der Literatur auch „pricing functions“ [DN92], „delaying functions“ [GS98], „proofs of computational effort“ [DNW05], „moderately hard functions“ [ABMW03] oder „client puzzles“ [ANL00] genannt. Im Folgenden wird der Begriff Proof-Of-Work verwendet, da dieser in der Literatur am weitesten verbreitet ist.

5.1 Diskussion POW Funktionen

Bevor auf die weiteren Details eingegangen wird, soll eine Grundfrage bezüglich der POW Funktionen diskutiert werden. Ein häufiger Kritikpunkt besagt, dass POW Funktionen

wertvolle Ressourcen „verschwenden“. In der Tat tut eine POW Funktion auf den ersten Blick nichts anderes, als Rechenzeit zu verbrauchen. Primär ist dies auch richtig, aber genauer betrachtet ist das mittelbare Ergebnis dieser Berechnung, dass die Überlastsituation eines Servers bzw. Dienstes kontrolliert werden kann. Dabei wird der Service weiter aufrechterhalten und die Klienten können in fairerer Weise zugreifen. Darüber hinaus liegt ein großer Teil der aufgewendeten Ressourcen auf den Klienten typischerweise brach, so dass nur wenig Ressourcen tatsächlich zusätzlich verbraucht werden.

Unabhängig davon ist es in anderen Lebensbereichen üblich, einen erheblichen Einsatz von Ressourcen zu akzeptieren, um Überlastsituationen abzuwenden. Im Fall eines überlasteten Flughafens wird allgemein akzeptiert, dass Warterunden unumgänglich sind. Während eine einzige Wartrunde bis zu 1500 l Treibstoff verbraucht, kann mit der gleichen Menge Energie ein Rechner ca. siebzehn Jahre ununterbrochen POW Funktionen berechnen. Wird mit dem Einsatz eines kleinen Teils davon z. B. eine wichtige Onlineplattform effektiv vor Überlast geschützt, dürfte der volkswirtschaftliche Vorteil zumindest ähnlich groß sein wie im Fall des Flughafens, dessen Betrieb mittels der Warterunden aufrechterhalten wird.

Dieser Abschnitt hat gezeigt, dass POW Funktionen Ressourcen verbrauchen und dafür statt eines direkt nutzbaren Ergebnisses einen positiven Regelungseffekt im Überlastfall zur Folge haben, der den Einsatz der Ressourcen rechtfertigt.

5.2 Notwendige Eigenschaften von POW Funktionen

Nachdem geklärt wurde, was POW Funktionen sind und warum ihr Einsatz sinnvoll sein kann, werden in diesem Abschnitt die Eigenschaften von POW Funktionen aufgeführt. Im darauf folgenden Abschnitt werden die optionalen Merkmale untersucht, welche in einzelnen Anwendungsfällen wünschenswert sind.

5.2.1 Angemessener Berechnungsaufwand

Mittels eines Proof-Of-Work wird der Einsatz von Ressourcen dokumentiert. Somit muss eine POW Funktion einen entsprechenden Berechnungsaufwand besitzen. Obwohl keine präzise Grenze angegeben werden kann, darf von einem minimalen Aufwand im Bereich von einigen Zehntelsekunden auf typischer Hardware ausgegangen werden. Unterhalb dieser Grenze ist der fixe Aufwand zur Initialisierung der POW Software sowohl auf der Seite des Klienten als auch auf der des Servers nicht lohnend.

Die obere Grenze für den Berechnungsaufwand einer POW Funktion dürfte in den allermeisten Anwendungsfällen im Bereich von Minuten liegen. Dadurch unterscheiden sich POW Funktionen von echten kryptographischen Funktionen, denn diese sollen die Entschlüsselung so lange wie möglich verzögern. Eine Ausnahme bildet hier das Verfahren von Goldschlag und Stubblebine in *Publicly Verifiable Lotteries: Applications of Delaying Function* [GS98]. Um eine überprüfbare Lotterie umsetzen zu können, ist eine POW Funktion erforderlich, deren Berechnungsdauer sich eher im Bereich einer schwachen kryptographischen Funktion als einer typischen POW Funktion bewegt.

5.2.2 Sicherheit

Zu einer POW Funktion darf keine algorithmische Abkürzung bekannt sein, die es ermöglicht, den Berechnungsaufwand zu umgehen. Sollte eine solche Schwachstelle auftauchen, würde das POW seine Effektivität entweder ganz verlieren oder kann, falls die Abkürzung nur einer gewissen Anzahl Klienten bekannt ist, sogar kontraproduktiv arbeiten.

Entscheidend ist hier aber im Gegensatz zu kryptographischen Funktionen primär, ob eine solche Abkürzung bereits bekannt ist, und nicht, ob sie irgendwann entwickelt werden könnte. Während bezüglich der POW Funktion eine Abkürzung nur zeitnah von Interesse ist, werden im Fall der Krytographie alle vorhandenen und entsprechend verschlüsselten Daten schlagartig lesbar. Die POW Funktion kann hingegen zeitnah gewechselt werden.

5.2.3 Effiziente Überprüfung

Eine POW Funktion soll schwer zu berechnen, aber leicht zu überprüfen sein. An dieser Stelle ist insbesondere das Verhältnis zwischen Berechnungs- und Überprüfungsaufwand entscheidend. Sobald der Aufwand für die Überprüfung des POW Ergebnisses zu groß ist, werden durch den Server zu viele Ressourcen verbraucht, so dass das POW Verfahren selber in eine Überlastsituation geraten kann. Hier kann kein exakter Wert angegeben werden, aber eine POW Funktion sollte praktisch gesehen zumindest eine Ratio von circa eins zu zehntausend bieten.

5.2.4 Keine Synergieeffekte

Ein mit der Sicherheit der Funktion verwandter Aspekt ist die Forderung, dass bei der Berechnung mehrerer POW Funktionen des gleichen Typs keine algorithmischen Synergieeffekte auftreten dürfen. Eine minimale Verbesserung der realen Berechnungsdauer bei der wiederholten Berechnung dürfte unvermeidbar sein, da im Fall

des zweiten Durchlaufs das Programm bereits im Speicher liegt. Darüber hinausgehende Steigerungen dürfen aber nicht realisierbar sein. Wenn es bspw. möglich ist, eine Funktion mittels vorher berechneter Wertetabellen erheblich effizienter zu lösen, würde dies mehrfach zugreifenden Klienten einen Vorteil verschaffen, der nicht erwünscht ist.

5.2.5 Zusammenfassung: Notwendige Eigenschaften

In den vorangehenden Abschnitten wurde erläutert, welche Eigenschaften eine POW Funktion minimal erfüllen muss. Zusammenfassend kann gesagt werden, dass eine solche Funktion garantieren muss, dass sie ausreichend schwer zu berechnen und das Ergebnis zudem einfach zu überprüfen sein muss.

Dabei fällt auf, dass diese Anforderungen der bisher ungeklärten Frage zur Relation der Komplexitätsklassen *P* und *NP* entsprechen. Sollte also $P \neq NP$ sein, würde sich jedes Problem $\in NP$ und $\notin P$ theoretisch als Proof-Of-Work eignen. Im umgekehrten Fall $P = NP$ würde unter Umständen folgen, dass das Konzept des Proof-Of-Work theoretisch nicht umsetzbar ist, da zu allen denkbaren POW Funktionen durch Umformung eine Lösung in *P* gefunden werden kann. Momentan gehen die meisten Informatiker einerseits davon aus, dass wahrscheinlich $P \neq NP$ gilt und andererseits müsste selbst im gegenteiligen Fall nicht nur der entsprechende theoretische Beweis vorliegen, sondern vor allem eine effiziente praktische nutzbare Umformung in einen einfach zu lösenden Algorithmus in *P*. Erst damit könnten die POW Funktion real umgangen werden. Beides ist momentan nicht in Sicht, so dass diese Überlegungen eher theoretischer Natur sind und für die praktische Lösung des Problems der Fairness in Überlastsituationen momentan keine Rolle spielen.

5.3 Optionale Eigenschaften

Neben den notwendigen Eigenschaften für POW Funktionen existieren eine Reihe weiterer Eigenschaften, welche entweder allgemein wünschenswert sind oder für spezielle POW Anwendungsfälle benötigt werden.

5.3.1 Parametrisierbarkeit

Die Schwierigkeit einer POW Funktion muss für eine praktische Anwendung an die konkreten Gegebenheiten angepasst sein. Wie bereits weiter oben beschrieben, verlangt die Arbeit von Goldschlag und Stubblebine [GS98] ganz andere Berechnungsdauern als ein Verfahren zur Spam-Abwehr. Noch wichtiger ist die Parametrisierbarkeit, wenn mittels des POW bspw. die Last eines Dienstes fein ausgeregelt werden soll.

Daraus folgt, dass eine POW Funktion zumindest grob bezüglich ihrer Schwierigkeit anpassbar sein sollte. Wird eine Lastregelung oder ein vergleichbares Verfahren angewendet, ist darüber hinaus eine feingranulare Parametrisierbarkeit erforderlich. Theoretisch ist es möglich, aus vielen einzelnen kleinen POW Funktionen eine größere zusammenzusetzen. Dies hat aber den Nachteil, dass sich die Ratio zwischen Berechnungs- und Überprüfungsaufwand verschlechtert und die Lösung eine längere Repräsentation aufweist, welche übertragen werden muss. Somit ist dieses Verfahren keine sinnvolle Alternative zur Parametrisierung der eigentlichen POW Funktion.

5.3.2 Kompakte Repräsentation

Um ein POW System praktisch integrieren zu können, ist es wünschenswert, dass die Lösung des POW möglichst kompakt übertragen werden kann. Theoretisch beansprucht die Lösung eines POW mit einem Suchraum der Größe 2^n mindestens n Bits zur Repräsentation der Lösung. Das bedeutet, die Repräsentation der POW Lösungen kann sehr kompakt sein und es ist wünschenswert, diesem Optimum in der Praxis möglichst nah zu kommen. Daneben ist es ferner interessant – sollte das konkrete Verfahren einen vom Server initialisierten Token vorsehen –, dass auch dieser kompakt übertragen werden kann. Praktisch muss auch dieser die Größe einiger Bytes nicht überschreiten.

5.3.3 Aufstockbarkeit

Abhängig vom Einsatzgebiet eines POW Verfahrens, kann es wünschenswert sein, dass Lösungen nachträglich aufgewertet werden können. Das bedeutet, indem zusätzliche Ressourcen aufgewendet werden, entsteht ein entsprechend „wertvolleres“ POW, ohne dass die Ressourcen, welche in die erste Lösung investiert wurden, verloren sind. Ein praktisches Beispiel wäre ein Auktionsverfahren, bei dem Ressourcen mittels POW Lösungen ersteigert werden. Durch die Aufstockbarkeit hätten unterlegene Klienten die Möglichkeit, ihre POW Ergebnisse bis zur nächsten Runde zu verbessern. Entsprechend der Parametrisierung wäre es hier ebenfalls möglich, einzelne kleine POW zu kombinieren. Es würden aber die weiter oben erwähnten Nachteile bei der Übertragung sowie der Überprüfung hinzukommen, so dass eine echte Aufstockbarkeit der POW Funktion die bessere Lösung darstellt.

5.3.4 Individualisierung

Wird eine identische POW Funktion für unterschiedliche Dienste verwendet, besteht das Risiko, dass POW Lösungen mehrfach verwendet werden. Darüber hinaus ist es Klienten

möglich, eine größere Menge solcher Lösungen vorzugenerieren, um dann auf diesem Weg einen zeitlich konzentrierten Denial-Of-Service Angriff starten zu können. Beide Probleme verlangen nach einer Individualisierung der POW Funktion. Da nicht unendlich viele POW Funktionen entwickelt werden können und sollen, muss die generelle POW Funktion mit einer Art Token individualisierbar sein. Somit sind Lösungen nur im Kontext dieses Tokens gültig und in der Praxis kann jeder Dienst ein individuelles Token wählen und dieses auch mit der Zeit verändern, so dass Lösungen weder doppelt verwendet noch beliebig vorgeneriert werden können.

5.3.5 Hardwareangepasste Berechnungsdauer

Da es sich bei einer POW Funktion um eine algorithmische Aufgabe handelt, die mittels eines Rechners gelöst wird, hängt die Berechnungsdauer entscheidend von der Leistungsfähigkeit dieses Rechners ab. In der Literatur besteht deswegen der Wunsch nach einer POW Funktion, welche auf allen gängigen Hardwareplattformen möglichst vergleichbare Berechnungsdauern aufweist. Insbesondere die weiter unten im Detail beschriebenen speicherbasierten Verfahren sollen dieser Anforderung gerecht werden. Dabei bleibt aber festzuhalten, dass eine absolut konstante Berechnungsdauer nicht einmal besonders positiv wäre, da sonst Angreifer große Mengen an praktisch kostenloser Hardware aufkaufen könnten, um einen Angriff durchzuführen. Aus diesem Grund wird an dieser Stelle von einer POW Funktion nur gefordert, dass sich ihre Berechnungszeit möglichst proportional zum Wert der verwendeten Hardware verhält. Daraus folgt, dass eine POW Funktion nicht auf Verfahren aufbauen sollte, die auf bestimmten Hardwareplattformen, z. B. wegen eines 3D Beschleunigers, überproportional schneller ausgeführt werden können.

5.3.6 Konstante Berechnungsdauer

Bei einem Einsatz von POW Verfahren ist es wünschenswert, die Berechnungsdauer der POW Funktion vorhersagen zu können. Dadurch wird es einfacher, einzelnen Klienten jeweils ein vergleichbares POW zuzuteilen und vorherzusagen, wann die Klienten dieses etwa gelöst haben werden.

Ohne an dieser Stelle einen endgültigen Beweis vorlegen zu können, gehen wir aber davon aus, dass eine in dieser Hinsicht perfekte POW Funktion nicht existieren kann. Eine POW Funktion basiert wie weiter oben beschrieben auf der Asymmetrie zwischen Berechnungs- und Überprüfungsaufwand. Dieser wird erreicht, indem bei der Berechnung eine Lösung in

einem Lösungsraum gesucht wird, die dann bei der Überprüfung nur einmalig validiert werden muss. Soll nun die Berechnungsdauer genau bekannt sein, würde dies ja bedeuten, dass der Dienstanbieter das POW entweder selber gelöst hat oder die Konstruktion des POW eine Vorhersage dazu erlaubt, wo sich die Lösung im Lösungsraum befindet. In beiden Fällen wird das POW wirkungslos. Entweder hat der Dienstanbieter den vollen Berechnungsaufwand oder der Klient kann die Lösung mit dem Wissen, wo sich die Lösung befinden muss, abkürzen. Aus diesem Grund besteht wahrscheinlich die einzige Möglichkeit darin, die statistische Streuung der Berechnungdauern einzugrenzen, indem statt eines Ergebnisses in einem Lösungsraum mehrere Ergebnisse in einem oder mehreren Lösungsräumen gesucht werden. Dieser Ansatz wird weiter unten anhand der sich ergebenen Verteilungen genauer untersucht. Auch die Arbeit *An (Almost) Constant-Effort Solution-Verification Proof-Of-Work Protocol based on Merkle Trees* von Coelho [COEL07] verfolgt, wenn auch verklausuliert, einen entsprechenden Ansatz. Der Nachteil dieser Verfahren besteht allgemein darin, dass jede bekannte Teilung des POW in kleinere POW zu einer Verschlechterung der weiter oben geforderten kompakten Repräsentation führt. Im Fall der Arbeit von Coelho kann eine POW Lösung 11 kb einnehmen, während für ein nicht geteiltes POW wenige Bytes ausreichen.

5.3.7 Abgestufte Berechnungsdauer

In einigen Anwendungsfällen werden POW Funktionen gefordert, die eine Möglichkeit besitzen, die Berechnungsdauer in einem vorher bestimmten Maß abzukürzen. Hierdurch soll es bestimmten privilegierten Klienten möglich sein, schneller POW Funktionen zu berechnen als anderen. Diese Verfahren gehen davon aus, dass diese speziellen Klienten über ein Geheimnis verfügen, welches es ihnen erlaubt, die Berechnung abzukürzen. Ein solches Verfahren muss dahingehend sicher sein, dass sich andere Klienten nicht ebenfalls dieses Geheimnis beschaffen können. Abgesehen davon, dass diese nach Klienten abgestufte Berechnungsdauer nur selten verlangt wird, lässt sich jedes andere POW Verfahren in ein solches abgestuftes Verfahren umbauen, indem für jede Benutzerklasse ein eigenes POW mit entsprechender Schwierigkeit ausgegeben wird. Der jeweilige Klient einer privilegierten Benutzergruppe weist sich dann durch Public Key Krytographie [HDM77] als ein Mitglied dieser Gruppe aus. Statt ein Geheimnis zu besitzen, mit dem sich direkt die Berechnung der POW Funktion abkürzen lässt, verfügt er in diesem Fall über den privaten Schlüssel, was bezüglich der Implementierung zu einem vergleichbaren Ergebnis führt, ohne die eigentliche POW Funktion modifizieren zu müssen.

5.3.8 Nebeneffekte der POW Berechnung

Anfangs des Kapitels wurde bereits erläutert, warum es akzeptabel ist, Ressourcen für die Berechnung von POW Funktionen zu verwenden. Nichtsdestotrotz ist es ein erfreulicher Nebeneffekt, wenn die Berechnung einer POW Funktion zu sonstigen positiven Nebeneffekten führt. So wäre es z. B. denkbar, Funktionen zu verwenden, deren Ergebnis nicht nur den Ressourcenverbrauch belegen, sondern bspw. wissenschaftliche Projekte unterstützt, welche viel Rechenleistung benötigen. Mit BOINC [BOINC] existiert bereits ein System, welches die Infrastruktur für verteiltes Rechnen bereitstellt, wobei SETI@home [SETI] wohl das bekannteste darauf aufbauende Forschungsprojekt darstellt. Wünschenswert wäre es nun, dass die im Rahmen der POW Berechnung erbrachte Leistung solchen Projekten zugutekommen kann. Praktisch ergeben sich aber eine Reihe von Sicherheitsproblemen, da die wissenschaftliche Aufgabe den Sicherheitskriterien eines POW entsprechen muss. Im Fall von SETI@home könnte ein Angreifer behaupten, er habe den ihm zugewiesenen Suchraum untersucht und nichts gefunden. Etwas besser würden sich hier wissenschaftliche Ansätze eignen, die Verschlüsselungsverfahren brechen bzw. Hash-Kollisionen suchen. Im Normalfall reicht es als POW aus, wenn der Benutzer eine gewisse partielle Hash-Kollision findet (siehe HashCashLin), aber im seltenen Einzelfall kann so auch eine vollständige Kollision gefunden werden. Die TU Graz unterhält ein entsprechendes Projekt [TUGRAZ] auf der BOINC Plattform, um SHA-1 Kollisionen aufzuspüren.

5.4 POW Funktionstypen

In den vorangegangenen Abschnitten wurden die an POW Funktionen gestellten Anforderungen erläutert. An dieser Stelle werden die bisher vorgeschlagenen POW Funktionen vorgestellt. Dabei erfolgt die generelle Unterteilung in CPU basierte und speicherbasierte Funktionen.

5.4.1 CPU basierte Funktionen

Die am weitesten verbreitete Familie von POW Funktionen sind CPU basierte Funktionen. Dabei handelt es sich um Verfahren, welche praktisch am effizientesten durch den Einsatz von CPU Zyklen gelöst werden können. Generell bieten sich hier Verfahren aus der Krytographie an. Insbesondere Verfahren, die in dieser Form für einen krytographischen Einsatz zu schwach erscheinen, können sich als POW Funktionen eignen. Bereits in der

ersten Arbeit zum Thema POW von Dwork und Naor [DN92] werden drei CPU basierte Funktionen vorgeschlagen.

Das erste Verfahren verlangt vom Klienten, Quadratwurzeln in einem Restklassenring zu suchen. Hierzu müssen mögliche Lösungen mittels einer großen Anzahl an Multiplikationen durchprobiert werden, während die Überprüfung mittels einer einzigen Berechnung möglich ist. Das zweite Verfahren basiert im Grunde auf einem Angriff gegen eine schwache Variante des Fiat-Shamir Authentifizierungsverfahrens [FS87]. Um diesen Angriff durchzuführen, muss die verwendete Hashfunktion so oft ausgeführt werden, bis eine Lösung vorliegt. Zur Überprüfung der Lösung ist ebenfalls eine einfache Berechnung ausreichend. Der dritte Vorschlag von Dwork und Naor basiert auf dem von Pollard und Schnorr [PS87] vorgestellten Verfahren, das Ong-Schnorr-Shamir Signatur Verfahren zu brechen. Dazu muss eine Faktorisierung durchgeführt werden, was ebenfalls das Durchprobieren möglicher Faktoren nach sich zieht. Die Überprüfung ist hingegen durch eine Multiplikation der Faktoren deutlich leichter zu bewältigen.

Von diesen drei Verfahren hat sich in der Literatur in dieser Form keines durchgesetzt. Aktuell basiert das gebräuchlichste Verfahren auf der Arbeit von Adam Back [HC]. Seine HashCash Funktion basiert auf der Schwierigkeit, partielle Hash Kollisionen zu finden. Während im Fall der anderen Verfahren auf die jeweiligen Veröffentlichungen verwiesen wird, folgt eine detaillierte Beschreibung dieses Verfahrens, da es für den Rest der Arbeit von besonderem Interesse ist.

Im Fall der HashCash Funktion wird für jedes Proof-Of-Work ein individuelles Token erzeugt. Dieses dient dazu, das mehrmalige Verwenden des Ergebnisses zu verhindern, indem es außerhalb des Kontextes dieses Tokens wertlos wird. Die Aufgabe besteht nun darin, ein zweites Token zu finden, das kombiniert mit dem ersten als Eingabe einer Hashfunktion einen validen Digest erzeugt. Als Hashfunktion kommt in diesem Fall die weiter unten im Detail beschriebene SHA-1 Funktion zum Einsatz. Hierbei sind bei einem validen Digest die *n* ersten Bits gleich Null. Der Parameter *n* bestimmt die Schwere des POW, da mit jedem zusätzlich geforderten Nullbit die Anzahl der möglichen Lösungen halbiert wird und sich der Berechnungsaufwand statistisch gesehen verdoppelt.

Die Überprüfung einer Lösung hingegen ist mit minimalem Aufwand zu bewältigen, da nur die beiden Tokens kombiniert, die Hashfunktion einmalig ausgewertet und die Anzahl der führenden Nullbits überprüft werden müssen. Abbildung 4 verdeutlicht das Hashcash Verfahren.

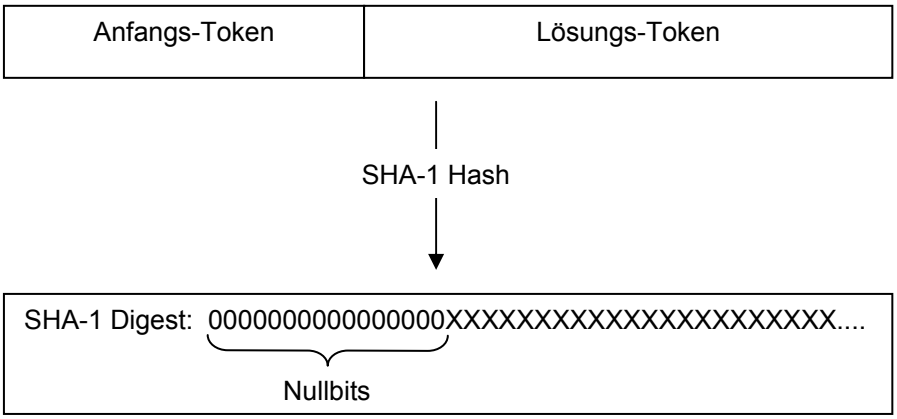


Abbildung 4: Das HashCash POW Verfahren

Auf dem gezeigten Verfahren des HashCash POW basiert die überwiegende Anzahl an Arbeiten zum Thema POW. Exemplarisch seien hier nur [DS01] und [ANL00] genannt, welche das HashCash Verfahren verwenden, selbst wenn sie Adam Back nicht zitieren. Tabelle 1 zeigt in einer kompakten Form, wie sich die HashCash Funktion bezüglich der formulierten Anforderungen verhält. Die optionalen Eigenschaften der Konstanten und abgestuften Berechnungsdauer sind in dieser Aufstellung nicht enthalten, da die Diskussion weiter oben gezeigt hat, dass diese im ersten Fall nicht zu erfüllen bzw. im zweiten Fall nicht notwendig sind.

Notwendige Eigenschaften	
Angemessener Berechnungsaufwand	Ein HashCash POW mit 19 Nullbits benötigt je nach Hardware etwa 5 Sekunden. Darüber hinaus kann die Berechnungsdauer über die Anzahl der Nullbits variiert werden, so dass die POW Funktion einen passenden Berechnungsaufwand aufweist.
Sicherheit	Die Sicherheit der Funktion basiert auf der Sicherheit der zu Grunde liegenden Hashfunktion. Das heißt, so lange eine sichere Hashfunktion existiert, kann auch HashCash als sicher angesehen werden.
Effiziente Überprüfung	Die HashCash Funktion ist mit einer einzigen Auswertung der Hashfunktion möglich. Die Überprüfung eines POW Ergebnisses ist somit sehr effizient.
Keine Synergieeffekte	So lange das Anfangs-Token variiert wird, beginnt die Suche nach einer Lösung immer unter identischen Voraussetzungen. Synergieeffekte existieren somit nicht.

Optionale Eigenschaften	
Parametrisierbarkeit	Hier liegt die Schwäche der HashCash Funktion. Sie ist zwar über die Anzahl der Nullbits parametrisierbar, aber daraus folgen exponentielle Schritte zwischen den verschiedenen Schwierigkeiten. Eine feinere Parametrisierung wäre wünschenswert.
Kompakte Repräsentation	Die Lösung des POW wird durch das Lösungs-Token übermittelt. Da dieses nur wenige Byte umfasst, ist die Repräsentation der Lösung sehr kompakt.
Aufstockbarkeit	Ein HashCash POW kann aufgestockt werden, indem die Suche mit identischen Anfangs-Token fortgesetzt wird, bis eine Lösung mit entsprechend mehr Nullbits gefunden wird.
Individualisierung	Durch das Anfangs-Token ist die HashCash Funktion per Definition individualisiert.
Hardwareangepasst Berechnungsdauer	Die Berechnung von Hashfunktionen verhält sich in etwa proportional zur generellen Leistung einer CPU. Somit verhält sich auch die Berechnungsdauer der HashCash Funktion proportional zur CPU Leistung.
Nebeneffekte der POW Berechnung	Ein HashCash POW kann dazu dienen, komplette Kollisionen der zu Grunde liegenden Hashfunktion zu finden. Weitere Nebeneffekte sind nicht bekannt.

Tabelle 1: Zusammengefasste Eigenschaften der HashCash Funktion

5.4.2 Speicherbasierte Funktionen

Die zweite Gruppe der POW Funktionen bilden die speicherbasierten Funktionen. Die Motivation zur Entwicklung dieser Proof-Of-Work Varianten rührt daher, dass sich die Leistungsfähigkeit unterschiedlicher CPUs sehr deutlich unterscheidet und der Berechnungsaufwand für ein CPU basiertes POW somit stark von der Hardware abhängt. Im Gegensatz zur Geschwindigkeit der CPUs steigt die Zugriffsgeschwindigkeit von Speichermodulen deutlich langsamer. Was Wulf und McKee in ihrem Artikel *Hitting the Memory Wall: Implications of the Obvious* [WM94] als Problem beschreiben, machen sich die Entwickler von speicherbasierten POW Funktionen zunutze. Die Funktionen sind so konstruiert, dass sie praktisch nur mittels intensiver Zugriffe auf eine Wertetabelle im

Hauptspeicher gelöst werden können. Hierbei muss die Tabelle so groß sein, dass diese nicht im Cache vorgehalten werden kann, da dieser deutlich schnellere Zugriffszeiten bieten würde.

Abdai et al. [ABMW03] schlagen ein interaktives POW Verfahren vor. Dabei wird die Tabelle im Speicher mittels einer Funktion *F* konstruiert, die rechtseindeutig, aber nicht linkseindeutig ist. *F* ist somit eine Einwegfunktion wie z. B. eine Hashfunktion. Mittels der Tabelle wird es möglich, jeweils die passenden Werte für die Umkehrfunktion *F'* zu finden, so dass gilt $F(F'(x)) = x$. Der Dienstanbieter berechnet nun *n* mal rekursiv die Funktion *F* und übermittelt das Ergebnis an den Klienten. Dieser muss mittels seiner Tabelle den Eingangswert der ersten Berechnung von *F* finden, indem er den sich durch *F'* aufspannenden Suchraum durchsucht. In dieser Suche entspricht jeder Aufruf von *F'* einem Speicherzugriff auf die Wertetabelle, so dass die Geschwindigkeit der POW Funktion von der Zugriffsgeschwindigkeit des Speichers abhängt.

Dwork et al. [DGN03] entwickeln diesen Ansatz weiter, indem sie weitere denkbare Angriffmöglichkeiten untersuchen und eine weiterentwickelte Funktion namens MBound vorstellen. Diese unterscheidet sich in erster Linie dadurch, dass sie für den Einsatz gegen Spam E-Mail gedacht ist und aus diesem Grund nicht interaktiv sein kann. Der somit vom Klienten selbst ermittelte Anfangspunkt der Suche hat dementsprechend kein sicher vorhandenes Ergebnis. Aus diesem Grund wird keine exakte Lösung erwartet, sondern nur eine, deren Hash Digest mit einer gewissen Anzahl Nullbits beginnt. Somit ist MBound gewissermaßen eine HashCash Variante, deren Zufalls-Token mittels der Wertetabelle aus dem Speicher bestimmt wird. Ein Nachteil dieses Verfahrens ist, dass die mehrere Megabyte große Wertetabelle übertragen werden muss, wofür Dwork et al. in einer weiteren Arbeit [DNW05] mit der Funktion Compact_MBound eine Lösung vorschlagen.

Coelho [COEL05] erweitert den ursprünglichen Ansatz, indem er in seinem Hokaido Protokoll auf jeder Ebene der Berechnung der Umkehrfunktion *F'* einen booleschen Zufallswert einfügt. So kann die Größe des zu durchsuchenden Baums, welcher bei Abadi et al. nur polynomiell war, exponentiell gestaltet werden, was das Verhältnis der Arbeit zwischen Dienstanbieter und Klienten erheblich verbessert. Darüber hinaus zeigt der Autor, dass die Random Access Bandbreite des Speichers der limitierende Faktor ist, da parallele Baumexploration durch die CPU parallelisiert werden kann, so dass die Bandbreite vollständig genutzt wird.

5.5 Hardwarebeschleunigung

Im Zusammenhang mit Proof-Of-Work Funktionen wird allgemein davon ausgegangen, dass zur Lösung nur typische Desktop Hardware zur Verfügung steht. Es soll aber nicht verschwiegen werden, dass sich mittels spezialisierter Hardware deutliche Geschwindigkeitsvorteile erreichen lassen. Insbesondere bezüglich der Berechnung von Hashfunktionen existieren viele Projekte, diese zu beschleunigen. Der am weitesten verbreitete Ansatz besteht dabei darin, Field Programmable Gate Arrays (FPGA) zu nutzen. Insbesondere die Hersteller Altera [ALTERA] und Xilinx [XILINX] bieten entsprechende Hardware zu moderaten Preisen an. Während sich die meisten Arbeiten, wie z. B. von Grembowski et al. [GLGN02] oder Ting et al. [TYLL02], damit befassen, wie die Leistungsfähigkeit der FPGA Implementierungen weiter verbessert werden kann, geben Chaves et al. [CKSV06a] [CKSA06b] auch einen Vergleich zu einer reinen Software-Implementierung der SHA-1 Funktion. Hierbei soll der Geschwindigkeitsvorteil in etwa bei einem Faktor hundert liegen. Selbst wenn anhand der im Rahmen dieser Arbeit vorgenommenen Messungen die Leistungsfähigkeit der CPU basierten SHA-1 Berechnung deutlich besser zu sein scheint als in der Arbeit von Chaves et al., verbleibt ein deutlicher Geschwindigkeitsnachteil gegenüber den FPGA-Implementierungen. Darüber hinaus sind weitere Geschwindigkeitsvorteile durch eine direkte Implementierung der Hashfunktion in klassischer Hardware denkbar.

Analog zu diesen Möglichkeiten könnte spezielle Hardware zur Lösung speicherbasierter Funktionen entwickelt werden. Konkret wäre hier z. B. ein außergewöhnlich großer Cache denkbar, denn sobald der Cache die komplette Wertetabelle von bspw. 32 MB bereithalten kann, reduziert sich die Berechnungsdauer der speicherbasierten POW Funktionen erheblich. Da aber der praktische Aufwand eines solchen Projektes deutlich größer erscheint als die hardwaretechnische Umsetzung von Hashfunktionen, ist nicht davon auszugehen, dass entsprechende Hardware speziell zur Lösung von POW Funktionen entwickelt und gebaut wird.

5.6 Zusammenfassung POW Funktionen

Es wurde in diesem Kapitel der Ansatz der so genannten Proof-Of-Work Funktionen eingeführt, der darauf basiert, einen nachweisbaren Einsatz von Ressourcen als eine Art fiktive Währung zu betrachten. Trotz der scheinbaren Verschwendung von Ressourcen wurden die grundsätzlichen Argumente für einen Einsatz dieser Technik vorgetragen. Es folgten sowohl die notwendigen als auch optionalen Eigenschaften von POW Funktionen

sowie eine Beschreibung der grundlegenden Funktionsvarianten. Abschließend wurde kurz erläutert, inwiefern POW Verfahren durch individuell angepasste Hardware beschleunigt werden könnten.

6 Übersicht POW Anwendungen

In diesem Abschnitt findet sich ein Überblick über die aktuellen und vorgeschlagenen Einsatzgebiete von POW Funktionen. Dieser Überblick ist hilfreich, da gezeigt wird, welche Vielzahl an Möglichkeiten die POW Technologie bietet. Daraus ergeben sich diverse Ansätze, welche sich für die POW basierte Überlastvermeidung adaptieren lassen.

6.1 POW als Spam E-Mail Gegenmaßnahme

Die Idee zu POW Funktionen entstand ursprünglich im Kontext der Bekämpfung von Spam E-Mail bzw. „Unsolicited Bulk Email“ (UBE) mit der Arbeit von Dword und Naor [DN92]. Seitdem hat es verschiedene Ansätze gegeben, dieses Verfahren real umzusetzen. Unter den prominentesten Versuchen findet sind das PennyBlack Projekt von Microsoft Research [PB], aus dem auch weiterführende Arbeiten zum Thema der speicherbasierten POW Funktionen von Abadi et al. [ABMW03] sowie Dword et al.[DGN03] hervorgegangen sind. Eine praktische Implementierung in Produkten der Firma Microsoft ist bisher aber nicht bekannt.

Ein weiterer bekannter Ansatz ist das HashCash [HC] Projekt von Adam Back. In diesem Rahmen wurde sowohl die gleichnamige auf Hash Kollisionen basierende POW Funktion entwickelt als auch ein komplettes Verfahren zur Spam-Bekämpfung. Es existieren hier auch praktische Implementierungen, wobei die allgemeine Verbreitung bisher als gering angesehen werden muss. Der interessanteste praktische Ansatz besteht darin, dass die weit verbreitete Spam Filter Software Spamassassin [SPAM] seit der Version 3.0 über spezielle Regeln für Hashcash verfügt und somit Nachrichten, welche ein POW aufweisen, Bonuspunkte im Spamassassin Scoring erhalten. Die Open Source Black- und Whitelisting Software TMDA (Tagged Message Delivery Agent) [TMDA] integriert ebenfalls eine HashCash Funktion. Das von Eric Johansson initiierte Projekt *Camram* kombiniert HashCash mit anderen Mechanismen wie Filtern etc., um ein praktisch anwendbares Werkzeug zur Spam-Bekämpfung zu erhalten [CAMRAM].

Insgesamt kann man zusammenfassen, dass es zum Thema Spam-Bekämpfung mittels POW eine Reihe von Arbeiten gibt, aber bisher kein Marktführer im Bereich E-Mail-Versand die Technologie umgesetzt hat. Das größte Hindernis bei einem klassischen POW Ansatz besteht darin, dass alle E-Mail-Klienten mehr oder weniger schlagartig umgerüstet werden müssten. So lange nur ein kleiner Teil der E-Mails mit POW versendet werden, ist es nicht praktikabel, E-Mails ohne valides POW abzulehnen bzw. zu löschen. Auf Grund der Vielzahl an existierenden E-Mail-Clients auf allen möglichen Plattformen bis hin zu diversen eingebetteten Systemen ist eine Umrüstung nur schwer denkbar.

Unter Umständen ist es deshalb realistischer, dass der POW Ansatz gar nicht im klassischen E-Mail-System eingesetzt wird, sondern in anderen einheitlicheren nachrichtenbasierten Systemen. So wäre es z. B. für einen Anbieter von Instant-Messaging Software deutlich einfacher, eine solche Funktion zu integrieren, da mit einem weitgehend einheitlichen Client gearbeitet wird. Die Anwendung von POW Funktionen in Instant Messaging Systemen wurde aber so weit bekannt noch nicht separat untersucht.

Unabhängig von diesen praktischen Einschränkungen existieren kritische Arbeiten wie die von Laurie und Clayton, welche den Ansatz, Spam per POW zu bekämpfen, generell in Frage stellen [LC04]. Hierbei sollte die Arbeit von Laurie und Clayton nicht überbewertet werden. Sie rechnen vor, dass ein POW, um effektiv zu wirken, so schwierig zu lösen sein muss, dass der reguläre E-Mail-Versand erheblich gestört wird. Einerseits kann der zugrunde liegende Rechenansatz auch anders gewählt werden [GMW05] und andererseits wird ein reales POW E-Mail-System nicht für jede E-Mail ein POW fordern. In einer Variante wird das POW nur von Versendern gefordert, welchen der Empfänger noch nicht geantwortet hat, was einer Art automatischem White Listing entspricht. Weitere Ansätze sehen vor, das POW mit einem Host basierten Reputationssystem zu verknüpfen [LC06] oder das POW entsprechend der Bewertung eines klassischen Spam Filters wie Spamassassin nur bei verdächtigen E-Mails zu verlangen [GS07].

Somit wartet der POW Ansatz darauf, von einem der Hersteller im Bereich E-Mail aufgegriffen zu werden. Die POW Überlastvermeidung in dieser Arbeit soll aus diesem Grund so konzipiert werden, dass sie in heutigen Systemen dezentral eingesetzt werden kann, ohne auf die Integration durch einen Marktführer angewiesen zu sein.

6.2 POW als Denial-of-Service Schutzmechnismus

Sowohl Juels und Brainard [JB99] als auch Feng und Reiter [WR03] schlagen vor, mittels POW Funktionen TCP Connection Depletion Attacks [FBGO05] [CERT96] zu bekämpfen. Diese Angriffe basieren auf einen Denial of Service Angriff gegen den TCP Stack, indem Verbindungen initiiert werden, aber der Verbindungsaufbau durch den Angreifer abgebrochen wird. Solange der entsprechende Timeout läuft, muss der Server Ressourcen für die Verbindung vorhalten, so dass bei einem massiven Angriff der TCP Stack schnell zusammenbrechen kann. Hier besteht der Ansatz darin, jeweils ein POW zu verlangen, bevor überhaupt Ressourcen reserviert werden, um die Angriffe so zu verhindern oder zumindest die Intensität stark zu reduzieren.

Juels und Brainard haben auf ihren Ansatz in 2007 ein US Patent erhalten [JB07]. Beide Arbeiten nutzen als POW Funktion Hash-Kollisionen und basieren somit indirekt auf dem HashCash Ansatz. Die entscheidende Weiterentwicklung der Arbeit von Feng und Reiter besteht darin, dass die mehr oder weniger festgeschriebene POW Schwierigkeit von Juels und Brainard durch eine Art Versteigerungsprotokoll ersetzt wird, was eine effektivere Parametrisierung der POW Funktion bewirken soll. In einer anderen Arbeit entwickeln Feng und Reiter [WR04] ihre Ideen weiter und führen den Begriff congestion puzzles (CP) ein. Die verwendete POW Funktion hat sich aber im Kern nicht geändert.

Laurens et al. [LEN06] geben in ihrer Arbeit einen Überblick über den Stand der Technik und schlagen vor, die Hashfunktion regelmäßig zu wechseln, um das Vorgenerieren von POW Tokens zu erschweren. Diese Variante erscheint aber angesichts der begrenzten Menge an Hashfunktionen und der alternativen Möglichkeiten durch wechselnde Initialiserungs-Tokens wenig Erfolg versprechend.

Bencsath et al. betrachten in ihrer Arbeit *Game Based Analysis of the Client Puzzle Approach to Defend Against DoS Attacks* [BVB03] POW basierte Denial-of-Service Schutzmechanismen mit spieltheoretischen Mitteln.

Feng schlägt in seiner Arbeit *The Case for TCP/IP Puzzles* [FENG03] vor, die Schwierigkeit der POW Funktion an das Verhaltensmuster der Klienten anzupassen, um besonders verdächtiges Verhalten zu erschweren. Darüber hinaus sollen leistungsschwache Klienten wie z. B. PDAs bevorzugt werden. Diese Mechanismen sind aber kritisch zu sehen, da in der Praxis wohl das Problem bestehen wird, diese verdächtigen Verhaltensweisen und schwachen Klienten sicher zu identifizieren, ohne einem Angreifer hier neue Angriffsmöglichkeiten durch Tarnung zu ermöglichen.

6.3 Sonstige POW Anwendungen

In diesem Abschnitt stellen wir weitere Ideen vor, wie POW Funktionen eingesetzt werden könnten. Die meisten dieser Ansätze sind über das Stadium einer Idee bisher nicht hinausgekommen, es finden sich aber trotzdem interessante Denkanstöße für weitere Entwicklungen.

Serjantov und Lewis stellen mit *Puzzles in P2P Systems* [SL03] ein Verfahren vor, das es ermöglichen soll, in einem Peer to Peer System mehr Fairness herzustellen. Klienten sollen durch POW Funktionen dazu gebracht werden, in dem Maße, wie sie Inhalte herunterladen,

auch selber Material zur Verfügung zu stellen. Klienten, die dies nicht tun bzw. neu in das Peer to Peer System einsteigen, müssen sich dies mittels POW Berechnungen erkaufen.

Aufbauend auf der Arbeit von Rivest et al. [RSW96] verwenden Goldschlag und Stubblebine in *Publicly Verifiable Lotteries: Applications of Delaying Functions* [GS98] POW Funktionen, um eine Lotterie zu entwickeln, deren Ausgang sich von jedem Teilnehmer selbstständig überprüfen lässt. Die gewinnende Losnummer lässt sich aus einem von Anfang an bekannten Datensatz per POW Funktion ermitteln. Die Dauer der Lotterie ist aber so gewählt, dass dieses Problem während der Laufzeit realistischerweise nicht gelöst werden kann. Im Nachhinein ist es aber jedem Teilnehmer möglich, die POW Funktion durchzurechnen und somit die Korrektheit des Ergebnisses zu überprüfen. Bei diesem Ansatz werden vergleichsweise schwierige Varianten der POW Funktionen benötigt, damit ihre Berechnungsdauer lang genug ausfällt.

Generell sind POW Funktionen ein Teilgebiet des Forschungsbereichs *Light Weight Securrity*. Dieser fasst Verfahren zusammen, welche nicht den Anspruch klassischer Sicherheits- bzw. Verschlüsselungsverfahren haben, möglichst nie entschlüsselt werden zu können. Das Entschlüsseln muss hier entweder nur eine gewisse minimale Schwierigkeit darstellen, so dass es sich im konkreten Fall nicht lohnt, oder das zeitlich versetzte Entschlüsseln ist sogar explizit gewollt wie z. B. im Fall der per POW gesicherten Lotterien.

In diesem Bereich existieren weitere Arbeiten bspw. zum Thema sicherer Benchmarks. Cui et al. beschäftigen sich in *Towards uncheatable benchmarks* [CLSY93] damit, wie klassische Benchmarks sicherer gemacht werden können. Oft wird durch Benchmarks nur eine Aufgabe vergeben, deren Ergebnis nicht überprüft wird oder welches sich durch ein alternatives Verfahren schneller ermitteln lässt. Ein geschickt optimiertes System könnte dies erkennen und die Lösung über die entsprechende Abkürzung ermitteln. Indem den untersuchten Systemen POW ähnliche Funktionen als Benchmark aufgegeben werden, können solche ungewünschten Optimierungen unterbunden werden.

In *Auditable Metering with Light Weight Security*[FM97] stellen Franklin und Malkhi einen Ansatz vor, mit dem mittels Timing Functions die Verweildauern von Klienten auf Webseiten sicher erfasst werden. Die timing functions sind eine Art POW Funktion mit einer kontinuierlich verbesserten Lösung, die im Ergebnis die auf der Webseite verbrachte Zeit widerspiegelt. Diese Lösung wird am Ende der Sitzung an den Server übergeben, so dass dieser eine von Dritten überprüfbare Statistik erstellen kann.

6.4 Zusammenfassung POW Anwendungen

In diesem Abschnitt haben wir eine Reihe von POW Anwendungen vorgestellt. Gemein ist den meisten Ansätzen, dass mittels der POW Funktion Vertrauen zwischen per se nicht vertrauenswürdigen Partnern bzw. Komponenten hergestellt wird. Dabei handelt es sich aber nicht zwingend um absolutes Vertrauen, sondern in vielen Anwendungen reicht eine Art relatives Vertrauen, welches ausreicht, um besonders unerwünschte Verhaltensweisen zu unterbinden.

Die wirklich praktischen Anwendungen der POW Funktionen sind bisher zugegebenermaßen rar, aber die Breite der Ansätze zeigt, welches Potenzial in dieser immer noch relativ neuen Idee bzw. Technologie steckt.

7 Einwegfunktionen

Sobald von den konkreten POW Umsetzungen ausreichend abstrahiert wird, stellt sich heraus, dass diese allgemein auf einem identischen Schema aufbauen. Es wird eine Einwegoperation mit wechselndem Eingangswert so oft berechnet, bis die Einwegoperation das gewünschte Ergebnis liefert. Diese Vorgehensweise impliziert insbesondere eine einfache Überprüfbarkeit des POW Ergebnisses, da hierfür die Einwegoperation allgemein nur ein einziges Mal ausgeführt werden muss. Andererseits handelt es sich im Kern immer um eine Suche und um keine geradlinige Berechnung, weshalb die Abarbeitungsdauer der POW Funktionen nur mit statistischen Mitteln angegeben werden kann. Bevor wir uns in den folgenden Kapiteln diesen Fragestellungen widmen, wird an dieser Stelle auf die Einwegfunktion als Kernbaustein fast jeden POW Verfahrens eingegangen.

7.1 Sicherheitsrelevante Eigenschaften

Um eine POW Funktion konstruieren zu können, muss die zu Grunde liegende Einwegfunktion eine zwingende Eigenschaft aufweisen:

Die Einwegfunktion darf sich nicht umkehren lassen (*preimage resistance*). Konkret wird gefordert, dass es keinen einfacheren Weg gibt zu einem gegebenen $F(x)$ den Wert x zu ermitteln, als F auf alle möglichen Werte von x anzuwenden, um so das passende x zu ermitteln.

Darüber hinaus ist eine *second preimage resistance* wünschenswert. Das bedeutet, bei einem gegebenen x darf es nicht möglich sein, anders als wiederum alle Möglichkeiten durchzuprobieren, ein y zu finden, so dass $F(x) = F(y)$ und $x \neq y$. Diese Eigenschaft ist zwar nicht zwingend für die Verwendung einer Einwegfunktion als POW, aber nur so ist es möglich, mehrere Lösungen zu einem identischen POW generieren zu lassen. Ohne *second preimage resistance* wäre es möglich, aus der ersten Lösung mit vermindertem Aufwand weitere Lösungen abzuleiten, was diese weiteren Lösungen als vollwertiges POW unbrauchbar macht.

Ein Blick auf die bekannten und verfügbaren Einwegfunktionen zeigt, dass diese Anforderungen eine Teilsumme der Anforderungen an krytographische Hashfunktionen darstellen und diese somit als Einwegfunktionen für die Anwendung in POW Anwendungen prädestiniert sind.

Für Hashfunktionen wird neben *preimage resistance* und *second preimage resistance* die *Collison resistance* gefordert. Diese besagt, dass es nicht möglich sein soll, zwei beliebige x

und *y* zu finden, so dass *F(x) = F(y)* und $x \neq y$. Während in einer krytographischen Anwendung diese Eigenschaft essenziell ist, um Angriffe zu unterbinden, ist im Fall der POW Funktion das Ergebnis *F(x)* als Anforderung vorgegeben, weshalb es für einen Angreifer nicht von Interesse ist, eine Kollision für ein beliebiges *F(x)* zu erhalten, da in diesem Fall keine validen Lösungen der POW Funktion entstehen.

7.2 Praktische Anforderungen

Neben den zwingenden Forderungen der *preimage* und *second preimage resistance* bestehen weitere praktische Anforderungen, damit eine Hashfunktion geeignet ist, um mit ihrer Hilfe ein POW Verfahren zu implementieren. Die Hashfunktion sollte dazu über folgende Eigenschaften verfügen:

- Eine weite Verbreitung, da so effiziente und getestete Implementierungen zur Verfügung stehen. Darüber hinaus ist bei einer weiten Verbreitung gesichert, dass die Qualität der Funktion einem dauerhaften Reviewprozess unterliegt und Schwachstellen schnell bekannt werden.

- Eine variable Länge des Hash Digest. Da im Fall der POW Funktion das Ergebnis der Funktion *F(x)* vorgegeben ist und es gilt, ein passendes x zu finden, stellt dies einen gewollten Brute Force Angriff gegen die *preimage resistance* dar. Um die POW Funktion parametrisieren zu können und überhaupt eine praktisch lösbare POW zu erhalten, muss die Länge des Hash Digest variabel sein, denn ein Brute Force Angriff gegen die volle Digestlänge gängiger Hashfunktionen ist praktisch nicht durchführbar.

7.3 Verbreitete Hashfunktionen

Nachdem die Anforderungen geklärt wurden, wird eine Auswahl der bekannten Hashfunktionen vorgestellt und auf ihre Verwendbarkeit hin untersucht.

7.3.1 HAVAL

Haval bietet als eine Besonderheit parametrisierbare Ausgabelängen von 128, 160, 192, 224 oder 256 Bits und drei unterschiedliche Sicherheitsstufen, die es erlauben, das gewünschte Sicherheitsniveau gegen die erforderliche Rechenzeit abzuwägen. Laut Wang et al. ist zumindest gegen HAVAL128 ein Angriff möglich, der es erlaubt, Kollisionen zu finden. Somit erfüllen nach aktuellem Stand alle Varianten bis auf HAVAL128 die formalen Anforderungen als POW Funktion. Eine Parametrisierbarkeit ist zwar gegeben, aber Digest

Längen von 160 Bits und mehr sind für den Fall der POW Berechnung als parametrisierbare Digestlänge viel zu groß und somit uninteressant.

7.3.2 Message Digest (MD) Funktionen

MD2 ist eine von Rivest et al. entwickelte und im RFC 1319 [RLK92] beschriebene Hashfunktion mit 128 Bits Ausgabewert. Frédéric Muller hat 2004 einen Angriff vorgestellt [MULLER04], so dass MD2 nicht mehr als sicher betrachtet werden kann.

MD4 ist ebenfalls von Rivest entwickelt und im RFC 1320 [RIV92a] beschrieben. Es hat wie MD2 einen Ausgabewert von 128 Bit. Nachdem Dobbertin [DOB96a] [DOB98] Kollisionen mit 2^{20} MD4 Berechnungen ermitteln konnte, behaupten Wang et al. [WFLY04] [WLFCY05] mit einem Aufwand von 2^8 Kollisionen ermitteln zu können, und Naito et al. [YSKO06] wollen sogar Kollisionen mit nur drei Berechnungen von MD4 ermitteln. Somit ist die Funktion praktisch nicht mehr einsetzbar.

MD5, wie M2 und MD4 auch von Rivest als Antwort auf Schwächen in MD4 entwickelt, ist im RFC 1321 [RIV92b] spezifiziert. Der Ausgabewert ist ebenfalls 128 Bits lang. Auch hier sind diverse Angriff bekannt geworden, insbesondere von Dobbertin [DOB96] sowie Wang et al. [WY05]. Diese untergraben die Sicherheit von MD5 im krytopgraphischen Sinn. Für POW Funktionen wäre MD5 eventuell noch nutzbar, da die gezeigten Angriffe nur den Aufwand von Brute Force Ansätzen auf ein gewisses Maß reduzieren. Es kann aber nicht ausgeschlossen werden, dass weitere Abkürzungen gefunden werden. Die Digestlänge ist bei den MD Funktionen allgemein nicht parametrisierbar.

7.3.3 RIPEMD

Die RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest) Funktion wurde von Hans Dobbertin et al. 1996 [DBP96] vorgestellt. Während für die ursprüngliche Variante RIPEMD, welche auf MD4 basierte, von Wang et al. [WLFCY05] Kollisionen gezeigt wurden, sind gegen die Varianten RIPEMD-128, RIPEMD-160, RIPEMD-256, RIPEMD-320 bisher keine erfolgreichen Angriffe bekannt. Die Funktionen haben einen ihrem Namen entsprechend langen Digest. Über die aufgezählten Varianten hinaus ist keine Parametrisierung der Digestlänge möglich.

7.3.4 Secure Hash Algorithm (SHA) Funktionen

Die SHA Familie von Hashfunktionen wurde von der National Security Agency (NSA) entwickelt. Es existieren die Varianten SHA FIPS PUB 180 (oft auch SHA-0 genannt),

SHA-1 FIPS PUB 180-1 sowie SHA-224, SHA-256, SHA-384 und SHA-512 (manchmal auch als SHA-2 bezeichnet). SHA-0 und SHA-1 erzeugen jeweils Digests mit 160 Bit. Die restlichen Funktionen haben ihren Namen entsprechende Digestlängen.

Es sind inzwischen einige Kollisions-Angriffe gegen SHA-0 und SHA-1 bekannt geworden. Im Fall von SHA-0 liegt der Aufwand für einen Geburtstagsangriff momentan nach Wang et al. [WLFCY05] bei 2^{39} Versuchen. Im Fall von SHA-1 liegt der zu erwartende Aufwand des bisher besten bekannten Angriffs von Wang et al. [WYY05] bei 2^{63}.

Auch im Fall der SHA Funktionen ist eine Parametrisierung der Digestlänge abgesehen von den vorgestellten Funktionsvarianten nicht vorgesehen.

7.4 Partielle Kollisionen

Die Übersicht der gängigen Hashfunktionen zeigt, dass durchaus Funktionen mit ausreichender Sicherheit zur Verfügung stehen. Die gewünschte Parametrisierbarkeit wird aber von keiner der in Frage kommenden Hashfunktionen geboten. Aus diesem Grund behelfen sich die Entwickler von POW Verfahren allgemein mit partiellen Kollisionen. Das bedeutet, dass kein echtes *preimage* im eigentlichen Sinn gefordert wird, sondern dass lediglich ein Teil des Digests mit der vorgegebenen Lösung übereinstimmen muss. Dadurch wird der Brute Force Angriff auf *die preimage resistance* kontrolliert vereinfacht, so dass die POW Berechnung praktisch durchführbar ist.

Im Fall einer perfekten Hashfunktion ist dieser Ansatz als sicher zu betrachten. So lange der Eingabewert und der Digest keinerlei nachvollziehbare Verbindung aufweisen, reduziert sich der Aufwand für einen Brute Force Angriff um die Hälfte für jedes Bit, welches von der Kollision ausgenommen wird. Praktisch muss aber an dieser Stelle zumindest angemerkt werden, dass momentan zum Thema partieller Hash-Kollisionen real kaum geforscht wird und somit die Wahrscheinlichkeit für krytographische Schwächen der Kollisionsresistenz bei partiellen Hash-Kollisionen nicht so weit abgesichert ist wie die totaler Hash-Kollisionen. Aktuell ist aber auch kein konkreter Angriff speziell auf die Resistenz bei Teilkollisionen für die gängigen Hashfunktionen bekannt, so dass diese Verfahren im Folgenden als genügend sicher betrachtet werden. Sollte sich in der Zukunft doch eine entsprechende Schwäche herausstellen, müsste analog zu anderen Verfahren die verwendete Hashfunktion gewechselt werden. Dies wäre aber eine rein technische Maßnahme, weil die darauf aufbauenden POW Verfahren allgemein nur eine möglichst leistungsstarke Hashfunktion benötigen, aber auf keine weiteren speziellen Eigenschaften der Hashfunktion angewiesen sind. So lange mit Teilkollisionen gearbeitet wird, ist sogar

die Länge des Digests bei der Auswahl der Funktion irrelevant, wenn die Funktion einen genügend langen Digest liefert, was bei allen bekannten Hashfunktionen gegeben ist.

7.5 Zusammenfassung Einwegfunktionen

Es wurde beschrieben, welche Anforderungen an Einwegfunktionen beim Einsatz im Rahmen der POW Funktionen gestellt werden, nämlich *preimage* und *second image resistance*. Hashfunktionen erfüllen diese Anforderungen per Definition. Daraufhin wurden zusätzlich die praktischen Anforderungen Verbreitung und Parametrisierbarkeit eingeführt. Eine Analyse der verfügbaren Hashfunktionen ergibt, dass zwar sichere und verbreitete Funktionen verfügbar sind, diese aber im Bereich der Digestlänge nicht annähernd genügend parametrisierbar sind. Als alternative Lösung wurden die Teilkollisionen diskutiert.

Zusammenfassend lässt sich so nachvollziehen, dass in der praktischen Umsetzung Teilkollisionen der SHA-1 Funktion die am weitesten verbreitete Variante der verwendeten Einwegfunktion darstellen. Auch die im Folgenden detailliert betrachtete und verbesserte HashCash Funktion bedient sich dieses Verfahrens.

8 HashCashLin

In diesem Kapitel wird die HashCashLin Funktion vorgestellt, welche die mangelhafte Parametrisierbarkeit der weiter oben beschriebenen HashCash Funktion entscheidend verbessert. Nach einer Erklärung der Motivation werden die Funktionsweise der Funktion und deren statistische Eigenschaften im Detail beleuchtet. Darüber hinaus werden die praktische Implementierung der Funktion sowie die zur Verifikation der theoretischen Ergebnisse durchgeführten Messungen beschrieben.

8.1 Die unzureichende Parametrisierbarkeit der HashCash Funktion

Die HashCash POW Funktion besticht durch ihre relativ einfache Implementierbarkeit, eine exzellente Ratio zwischen Berechnungs- und Überprüfungsaufwand und eine sehr kompakte Darstellung der zu übertragenden Lösung. Neben generellen Überlegungen zu CPU-basierten Funktionen besteht der größte Mangel der HashCash Funktion in der nur exponentiell möglichen Parametrisierbarkeit. Der folgende Absatz verdeutlicht diese Problematik.

Wie bereits kurz in Kapitel 5.4.1 beschrieben, basiert die HashCash Funktion auf teilweisen Hash-Kollisionen. Der Klient muss so lange unterschiedliche Tokens an einen Basistoken anhängen, bis der Digest der SHA-1 Funktion (160 Bits) mit einer definierten Anzahl an Nullbits beginnt. Der durchschnittliche Berechnungsaufwand, um einen Erwartungswert von 1 für einen Treffer zu erzielen, errechnet sich wie folgt:

$$\frac{2^{160}}{2^{160-k}} = 2^{k}$$

k: Natürliche Zahl, welche die Anzahl der Nullbits angibt

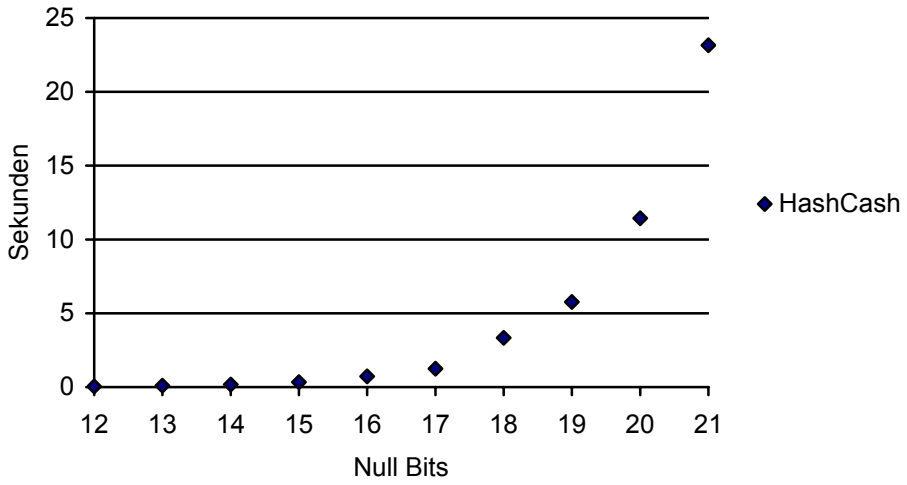


Abbildung 5: HashCash Berechnungsdauer

Da für *k* nur ganze positive Zahlen zulässig sind, ist die HashCash Funktion lediglich sehr grob parametrisierbar (siehe Abbildung 5). Das Ziel dieser Arbeit ist es aber, mit Hilfe von POW Funktionen adaptive Systeme zu konstruieren, so dass diese Parametrisierung nicht ausreichend ist. Systeme, welche die POW Schwierigkeit nur verdoppeln bzw. halbieren können, laufen ständig Gefahr, sich aufzuschaukeln, und arbeiten zwangsweise oft viel zu weit vom optimalen Betriebspunkt entfernt.

Eine mögliche Lösung wäre es, ein POW aus kleineren HashCash POW zusammenzusetzen, die kombiniert die entsprechende Schwierigkeit ergeben. Da dies aber bei großen zusammengesetzten POW die Überprüfung entsprechend der Anzahl der POW verlängert und sich auch die Repräsentation vergrößert, wurde mit der HashCashLin Funktion ein eleganteres Verfahren entwickelt, welches im folgenden Abschnitt beschrieben wird.

8.2 Beschreibung der HashCashLin Funktion

Wir nennen unsere Funktion HashCashLin, da sie im Gegenteil zur originalen HashCash Funktion, welche sich nur exponentiell parametrisieren lässt, linear parametrisiert werden kann. Dies erreichen wir, indem wir das Nullbitkriterium durch eine positive 32 Bit Ganzzahl ersetzen. Die ersten 32 Bits des Hash Digest müssen kleiner als diese Zahl sein, um dieses Kriterium zu erfüllen (Abbildung 6).

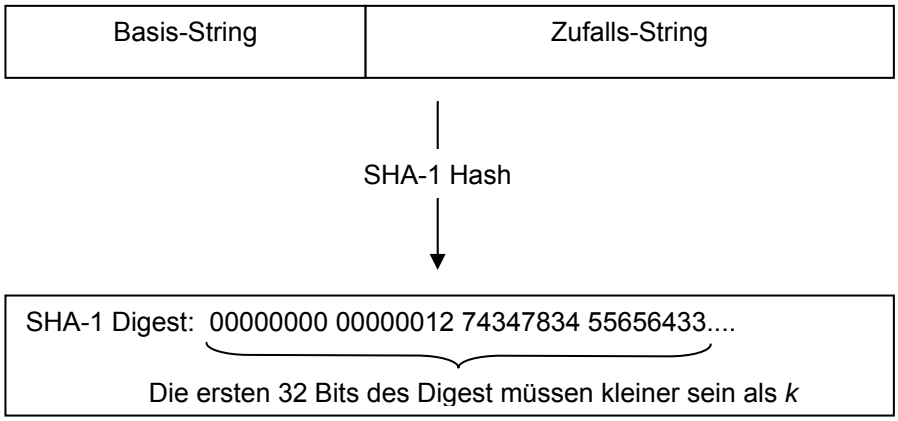


Abbildung 6: Schematische Darstellung des HashCashLin POW Verfahrens

Der Nachteil dieser Variante der Definition besteht darin, dass ein schweres POW durch eine kleine Zahl und ein leicht zu lösendes POW durch eine große Zahl dargestellt wird. Prinzipiell könnte man die Definition auch umgekehrt formulieren, nämlich dass der Digest größer als *k* sein muss. Das erschwert aber das Rechnen mit *k*, da man den Wert nun nicht mehr einfach multiplizieren kann, um z. B. die POW Schwierigkeit entsprechend zu verändern. Um Missverständnisse zu vermeiden, sprechen wir im Folgenden von leichten bzw. schweren POW, was respektive ein großes bzw. kleines *k* bedeutet. Die Bezeichnungen Größe, groß und klein sollten im POW Kontext vermieden werden.

Es folgt die formalisierte Bedingung, welche jede korrekte HashCashLin Lösung erfüllen muss:

$$k > SHA\text{-}1(b + z) \gg 128$$

b: Basis-String
z: Zufalls-String
k: Kriterium als Natürliche Zahl aus $[0 \ldots 2^{32}]$
$\gg$: Shift right Operator

Um einen Erwartungswert von eins zu erreichen, entsteht folgender Aufwand an Versuchen:

$$\frac{\frac{2^{160}}{2^{160-32}}}{k} = \frac{2^{32}}{k}$$

Das bedeutet, der Berechnungsaufwand des POW wächst linear mit *1/k*. Soll die Schwierigkeit des POW nun z. B. um *x* gesteigert werden, gilt *1/k' = x/k* bzw. *k' = k/x*. Das bedeutet, die Schwierigkeit einer HashCashLin Funktion mit Faktor *k* wird um den Faktor *x* gesteigert, indem *k* durch *x* geteilt wird.

Darüber hinaus ist die HashCashLin Funktion eine direkte Verallgemeinerung der HashCash Funktion. Somit sind alle parametrisierbaren Fälle des HashCash POW ein Spezialfall des HashCashLin POW. An dieser Stelle muss je nach Schwierigkeit des HashCash POW nur der jeweilige Bereich der betrachteten Bits im Fall der HashCashLin Funktion beachtet werden. Die Parametrisierung wird dabei wie folgt umgerechnet:

$$k = 2^{32-n}$$

k: HashCashLin Schwellwert als Natürliche Zahl aus $[0\ldots2^{32}]$
n: Nullbits der entsprechenden HashCash Funktion [0…32]

Abbildung 7 zeigt Beispielmessungen der HashCash und der HashCashLin Funktion, bei denen die HashCashLin Funktion entsprechend dieser Formel konfiguriert wurde.

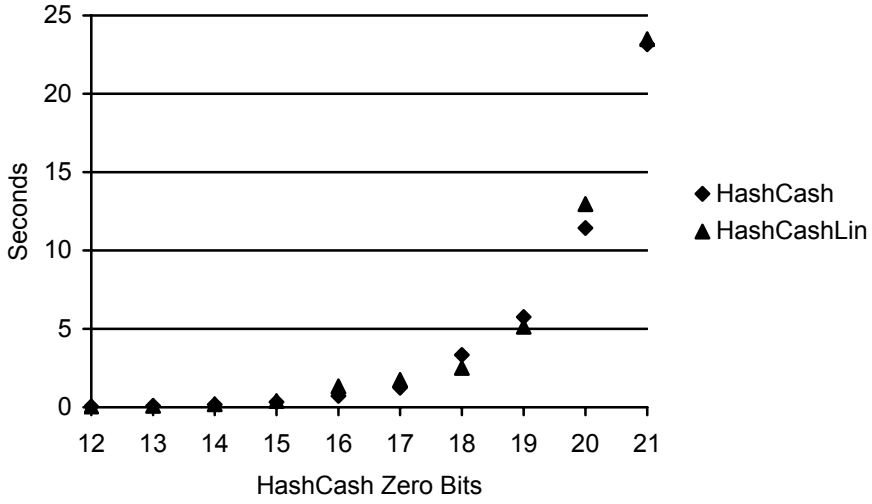


Abbildung 7: HashCash und HashCashLin mit equivalenter Berechnungsdauer

Während die HashCash Funktion aber für den besonders interessanten Bereich zwischen 15 und 21 Nullbits nur sieben Parametrisierungsmöglichkeiten bietet, verfügt die HashCashLin Funktion in diesem Bereich über $2^{17} - 2^{11} + 1 = 129.025$ Zwischenschritte. Abbildung 8 zeigt Beispielmessungen, bei denen zur Veranschaulichung jeweils drei POW Zwischenstufen eingefügt wurden.

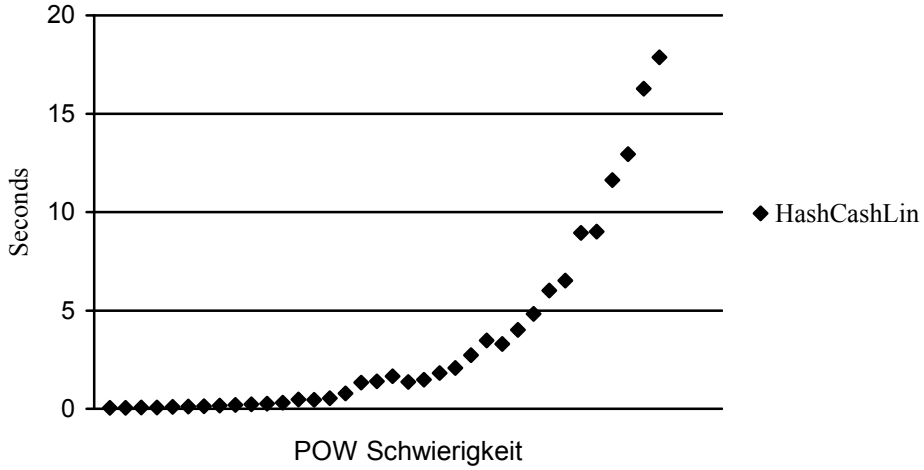


Abbildung 8: Messungen mit einer Auswahl an HashCashLin Zwischenschritten

Die Anzahl der Zwischenschritte ließe sich noch deutlich steigern, indem nicht nur die ersten 32 Bit, sondern 64, 128 oder 160 Bit des Digest betrachtet werden. Wir sind aber der Meinung, dass dies in der Praxis keinen Mehrwert bringt und die Werte für *k* nur unnötig unhandlich macht.

8.3 Implementierung

Die HashCashLin Funktion haben wir in Java implementiert. Hierbei konnte auf der HashCash Implementierung von Sebastian Gesemann [GESE03a] aufgebaut werden. Primär war dazu die Routine anzupassen, welche die Zulässigkeit der einzelnen Lösungen überprüft. Darüber hinaus wurde die Funktion so erweitert, dass während der Berechnung Fortschrittsmeldungen entnommen werden können. Dies ist bei einer praktischen Anwendung interessant, um z. B. einen Fortschittsbalken anzeigen zu können.

Diese Lösung basiert auf einem einzelnen Thread. Da am Markt aber vermehrt DualCore Prozessoren angeboten werden, erschien es sinnvoll, die CPU-intensive HashCashLin-Berechnung so durchzuführen, dass die volle Leistungsfähigkeit dieser Systeme ausgenutzt wird. Aus diesem Grund wurde zusätzlich eine Variante erstellt, welche mit mehreren Threads arbeitet. Hierbei wird der Suchraum aufgesplittet, so dass jeder Thread einen zugewiesenen Bereich abarbeitet. Dabei ist zu beachten, dass die zugewiesenen Bereiche deutlich getrennt sein sollten, um eine ineffiziente redundante Suche im Lösungsraum auszuschließen.

Um diese parallelisierte Implementierung zu testen, wurden jeweils 2048 Messungen mit gleicher POW Schwierigkeit von 2^{18} mit beiden Implementierungen durchgeführt. Dabei wurden einerseits die mittlere Anzahl an Durchläufen bis zum Finden einer Lösung und andererseits die reale Berechnungsdauer ermittelt. Ein detailliertes Messkapitel folgt, doch diese Messung verdeutlicht bereits, dass insbesondere die parallelisierte Implementierung der HashCashLin Funktion den erwarteten Leistungsanstieg bietet.

Zum Vergleich wurde die gleiche Messung auf einem System mit nur einem Kern durchgeführt und es ist ersichtlich, dass die Parallelisierung hier keine Vorteile bringt, sondern sogar minimale Nachteile durch den Verwaltungsoverhead entstehen. Es kann also geschlussfolgert werden, dass der Einsatz der jeweiligen Variante der HashCashLin Funktion auf die verwendete CPU abgestimmt werden muss. Eine Parallelisierung in mehr Threads, als die Hardware Prozessorkerne zur Verfügung stellt, bringt im Gegensatz zu anderen Anwendungen keinen Vorteil.

8.4 Statistische Eigenschaften der HashCashLin POW Funktion

Da es sich sowohl bei der HashCash als auch der HashCashLin Funktion um POW Funktionen handelt, deren Berechnungsaufwand nur probabilistisch vorhergesagt werden kann, wollen wir diesen Aspekt im Folgenden mathematisch konkretisieren.

Neben dem Erwartungswert für eine bestimmte Anzahl an POW Lösungen in einer bestimmten Berechnungszeit widmen wir uns der Verteilung der einzelnen Berechnungszeiten. Die Verteilung ist insbesondere für den Klienten interessant, da dieser nur einzelne POW berechnet und deswegen die faktische Berechnungsdauer entsprechend dieser Verteilung vom Erwartungswert abweichen wird. Hier ist es das Ziel, diese Streuung auf ein vertretbares Maß zu beschränken. Ein System mit korrektem Erwartungswert, aber zu weit streuender Verteilung könnte problematisch sein, da so die POW Berechnung auf Klientenseite zu sehr zu einem Glücksspiel statt zum Nachweis von Ressourceneinsatz würde.

8.4.1 Erwartungswert

Ein HashCashLin POW setzt sich aus vielen einzelnen Versuchen zusammen. Diese werden so oft wiederholt, bis eine korrekte Lösung entdeckt wurde. Daher berechnen wir zuerst die Wahrscheinlichkeit, bei jedem dieser Einzelexperimente eine Lösung zu finden. Diese Wahrscheinlichkeit ist bei jedem Einzelexperiment gleich, da der Suchraum keinerlei Struktur aufweist, mit der sich Lösungen eingrenzen ließen. Diese Eigenschaft hängt von

der Güte der eingesetzten Hashfunktion ab. Im Fall von SHA-1 kann nach aktuellem Forschungsstand eine ausreichende Güte für unsere Anwendung angenommen werden.

Der Erwartungswert für das Einzelexperiment:

$$p = \frac{v}{t}$$

p: Wahrscheinlichkeit, bei einem Versuch eine valide Lösung zu entdecken
v: Anzahl als Lösung valider Digests insgesamt
t: Anzahl möglicher Digests insgesamt

Bei mehreren Experimenten kann diese Einzelwahrscheinlichkeit mit der Anzahl der Versuche multipliziert werden, um den Erwartungswert eines Experiments mit entsprechend vielen Unterexperimenten zu erhalten. Daraus ergibt sich, dass genau der Kehrwert an Versuchen notwendig ist, um einen Erwartungswert von eins zu erhalten.

$$l = n \cdot p \rightarrow n = \frac{l}{p}$$

n: Anzahl im Mittel zu erwartender Versuche

8.4.2 Verteilung Einzelberechnung

Im Folgenden betrachten wir die Verteilung der Berechnungsdauern bei Berechnungen der HashCashLin Funktion. Diese ist von besonderem Interesse, da in der Praxis nur einzelne Instanzen der Funktion berechnet werden. Das bedeutet: Sollte die Funktion zu stark streuen, müssen Klienten mit stark unterschiedlichen Berechnungsdauern für ihr POW rechnen. Somit ist es wünschenswert, diese Streuung zu begrenzen.

Die Wahrscheinlichkeit *w,* nach *n* Versuchen eine Lösung gefunden zu haben, können wir wie folgt formulieren:

$$p = \frac{v}{t}$$

p: Wahrscheinlichkeit, bei einem Versuch eine valide Lösung zu entdecken
v: Anzahl als Lösung valider Digests insgesamt
t: Anzahl möglicher Digests insgesamt

$w = p \cdot (1\text{-p})^{n-1}$

w: Wahrscheinlichkeit, nach genau n Versuchen die erste Lösung zu erhalten

Die Definition von w entspricht der Definition der geometrischen Verteilung. Das bedeutet, die Berechnungsdauern der HashCashLin Funktion sind demnach geometrisch verteilt. Kumuliert man als w' die Werte von w für alle $n' \in [1\ldots n]$, erhält man die Wahrscheinlichkeit, nach n Versuchen mindestens eine Lösung zu erhalten.

$$w' = \sum_{i=1}^{n} p \cdot (1-p)^{i-1}$$

Da diese Berechnung insbesondere bei großen n, welche wir in unserem Fall erwarten, umständlich ist, bietet sich folgende Berechnung von w' an:

$w' = 1\text{-} (1\text{-}p)^n$

Hier wird w' über die Gegenwahrscheinlichkeit berechnet, n mal keine Lösung zu finden.

8.4.3 Verteilung mehrerer Berechnungen

Wie weiter oben beschrieben, ist eine geringe Streuung der Berechnungsdauern wünschenswert. Diese kann dadurch reduziert werden, dass die Anzahl der geforderten POW Lösungen heraufgesetzt wird. Nun möchten wir untersuchen, wie diese Maßnahme die Verteilung der Berechnungszeiten beeinflusst.

Hierbei bieten sich zwei Ansätze an. Einerseits kann die Berechnung von k identischen HashCashLin Funktionen als Verkettung dieser Berechnungen modelliert werden. Die Wahrscheinlichkeitsverteilung ergibt sich in diesem Fall aus der Faltung der Einzelverteilungen.

Zweitens kann die Berechnung von k HashCashLin Funktionen auch als einfache Berechnung einer Funktion, bei der k Lösungen gefunden werden müssen, angesehen werden, was eine einfachere Ermittlung der Verteilung ermöglicht. Im Folgenden werden beide Ansätze anwendet.

Der erste Ansatz, die einzelnen Verteilungsfunktionen zu falten, kann mathematisch wie folgt beschrieben werden:

$w_k' = w'(n) * w'(n) * w'(n) \ldots$k mal

Die praktische Berechnung dieser diskreten Faltung ist aber sehr aufwändig, da die Einzelwahrscheinlichkeiten ausmultipliziert und dann addiert werden müssen. Im Fall der HashCashLin Funktion arbeiten wir mit sehr großen *n*, so dass eine andere Variante der Berechnung wünschenswert ist.

An diesem Punkt bietet es sich an, die mehrfache Berechnung der Funktion als einfache Berechnung mit mehreren Lösungen zu modellieren. In diesem Fall kann die Wahrscheinlichkeit, nach *n* Versuchen genau *k* Lösungen gefunden zu haben, beschrieben werden:

$$w_k = \binom{n}{k} \cdot p^k (1-p)^{n-k} \qquad \text{(Definition der Binomialverteilung)}$$

w_k: Wahrscheinlichkeit, nach *n* Versuchen genau *k* Lösungen zu finden
p: Wahrscheinlichkeit des Einzelexperiments

Die Wahrscheinlichkeit, *k* Lösungen oder mehr zu finden, w_k, entspricht der Gegenwahrscheinlichkeit, 0 bis *k*-1 Lösungen zu finden. Daraus folgt:

$$w_k' = 1 - \sum_{i=0}^{k-1} \binom{n}{i} \cdot p^i (1-p)^{n-i}$$

w_k': Wahrscheinlichkeit, nach *n* Versuchen min. *k* Lösungen gefunden zu haben

Insbesondere für relativ kleine *k* und große *n* ist diese Variante deutlich einfacher zu berechnen als die weiter oben beschriebene Faltung.

8.5 Messungen zur Überprüfung der analytischen Ergebnisse

Um die in den vorangegangenen Abschnitten gefundenen Ergebnisse zu überprüfen, wurden Messreihen durchgeführt. Hierbei geht es darum, die gemessenen Werte den analytisch getroffenen Annahmen direkt gegenüberzustellen. Aus diesem Grund wird die Anzahl der Versuche gemessen, bis eine passende Lösung gefunden ist. Das bedeutet, die Ergebnisse sind unabhängig von der Leistungsfähigkeit der verwendeten Hardware und der konkreten Implementierung der HashCashLin Funktion. Wie sich diese Werte in konkreten Berechnungsdauern auf unterschiedlicher Hardware niederschlagen, untersuchen wir in einem weiteren Abschnitt.

Als Datenbasis wurden jeweils 512 Einzelmessungen der HashCashLin Funktion mit den Parametern 2^{32} bis 2^{20} durchgeführt.

Abbildung 9 vergleicht die gemessenen durchschnittlichen Berechnungsaufwände mit den analytischen Erwartungswerten. Es ist zu erkennen, dass die Kurven praktisch deckungsgleich verlaufen, so dass davon ausgegangen werden kann, dass unsere analytische Berechnung des Erwartungswertes dem realen Verhalten entspricht.

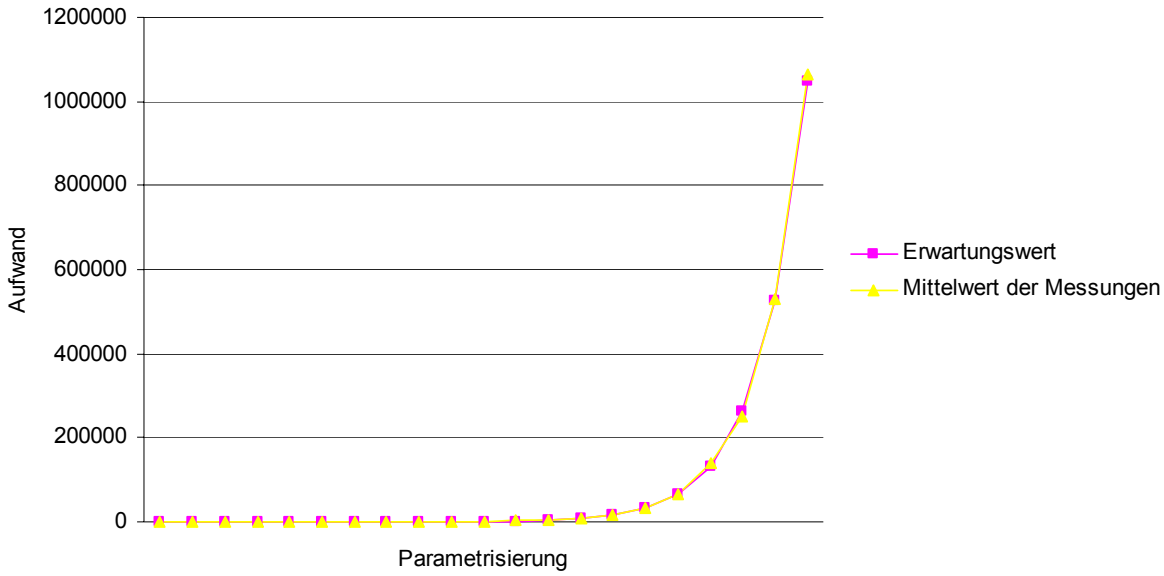


Abbildung 9: Vergleich Berechnungsaufwände zu Erwartungswerten

In Abbildung 10 sind die gemessene Dichtefunktion sowie die analytisch ermittelten Dichtefunktionen abgebildet. In diesem Fall wurden dafür 2048 Einzelmessungen mit dem HashCashLin Parameter 2^{18} herangezogen. Um eine sinnvolle Darstellung zu ermöglichen, wurden die Werte jeweils für einen Bereich von 100.000 Versuchen kumuliert.

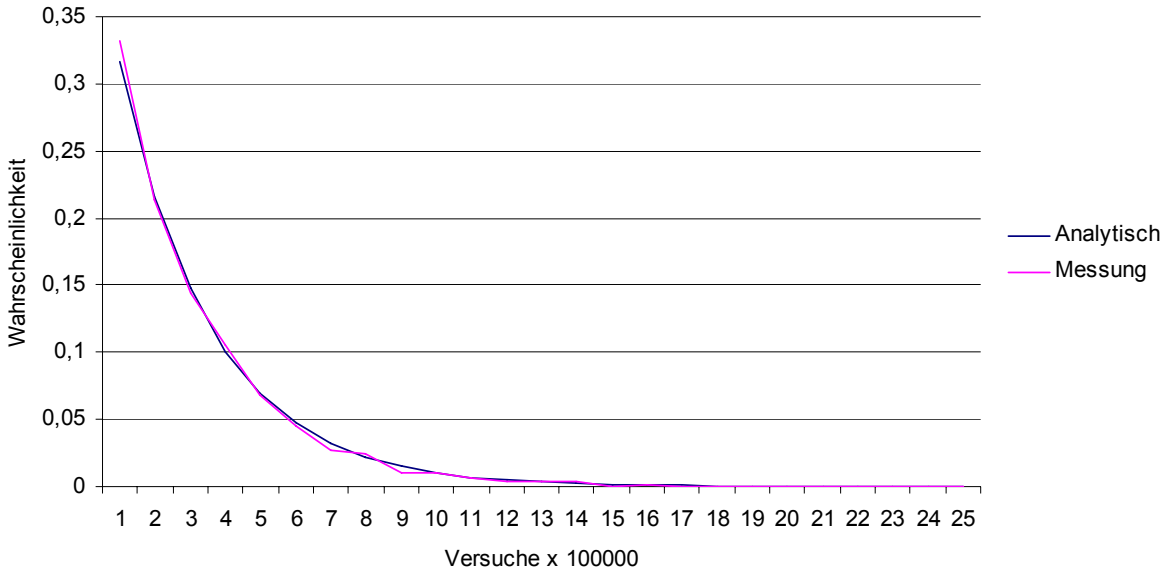


Abbildung 10: Vergleich gemessener und analytisch ermittelter Dichtefuntionen

Die Kurve verdeutlicht, dass die analytisch ermittelten Werte in diesem Fall mit dem realen Verhalten der Funktion übereinstimmen. Im Folgenden möchten wir auch unsere Messungen zu anderen HashCashLin Parametern als 2^{18} vergleichen. Da diese Parametrisierungen aber ganz unterschiedliche Anzahlen an Versuchen erfordern, können wir sie in herkömmlicher Weise schwer in einer Abbildung darstellen. Aus diesem Grund wird die X-Achse prozentual zum Erwartungswert der HashCashLin Parametrisierung skaliert

(

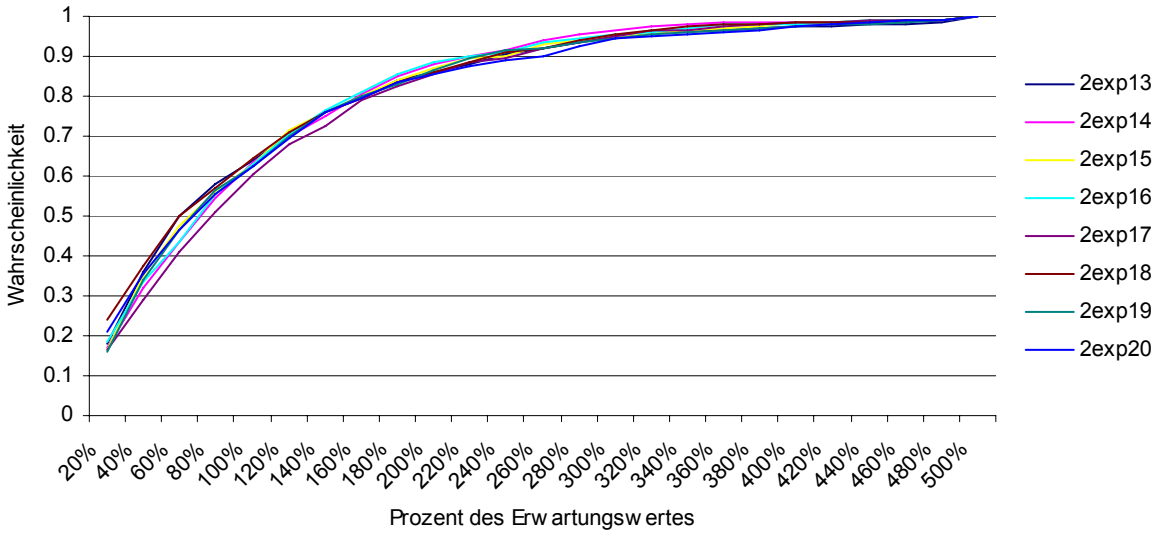


Abbildung 11). Darüber hinaus besteht in der Praxis das größte Interesse zu wissen, nach welcher Berechnungsdauer mit welcher Wahrscheinlichkeit mindestens eine Lösung gefunden wurde. Aus diesem Grund verwenden wir im Folgenden die Verteilungsfunktion anstatt der Dichte.

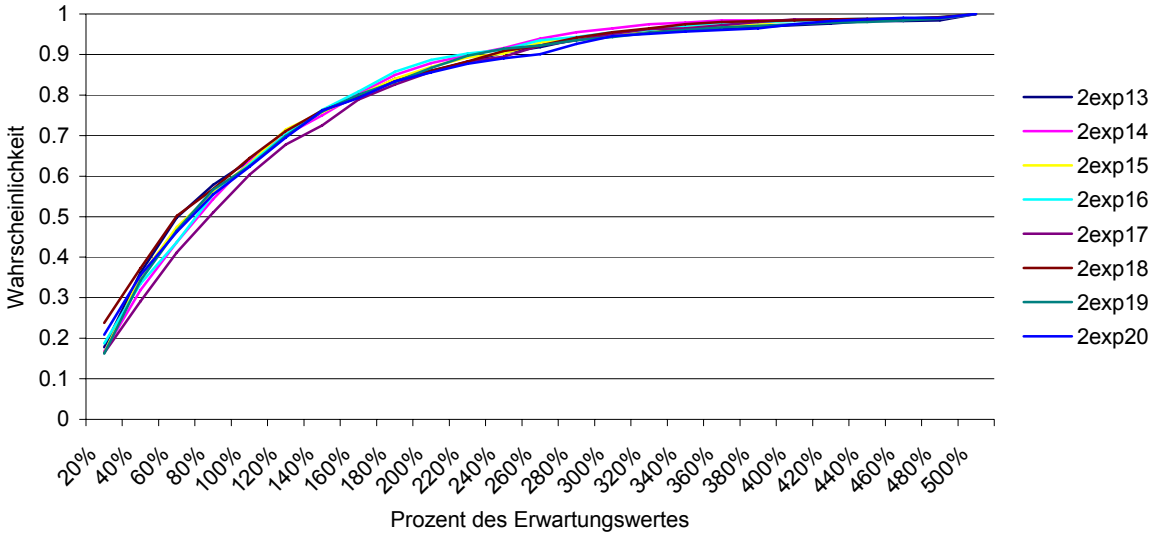


Abbildung 11: HashCashLin Verteilungsfunktionen

Anhand dieser Kurven lässt sich relativ gut abschätzen, welcher Berechnungsaufwand im Verhältnis zum Erwartungswert notwendig ist, um mit einer bestimmten Wahrscheinlichkeit mindestens eine Lösung gefunden zu haben. Bei 100 % des Erwartungswertes wird bspw. statistisch gesehen nur in ca. 63 % der Fälle eine Lösung gefunden.

Wie im theoretischen Teil vorweggenommen, streut diese Verteilung für die praktische Anwendung doch relativ stark. Deswegen untersuchen wir im Folgenden das Verhalten zusammengesetzter HashCashLin POW.

Abbildung 12 zeigt die Dichtefunktion der zusammengesetzten Varianten des 2^{18} HashCashLin POW. Die Kurven wurden mittels des weiter oben beschriebenen Faltungsverfahrens generiert.

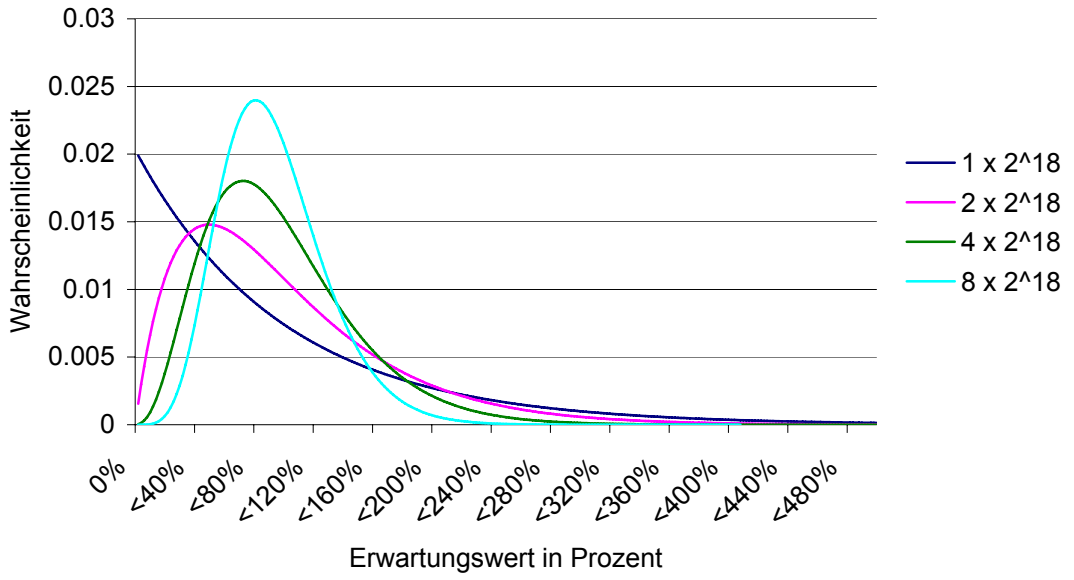


Abbildung 12: Dichtefunktionen verschiedener Faltungen

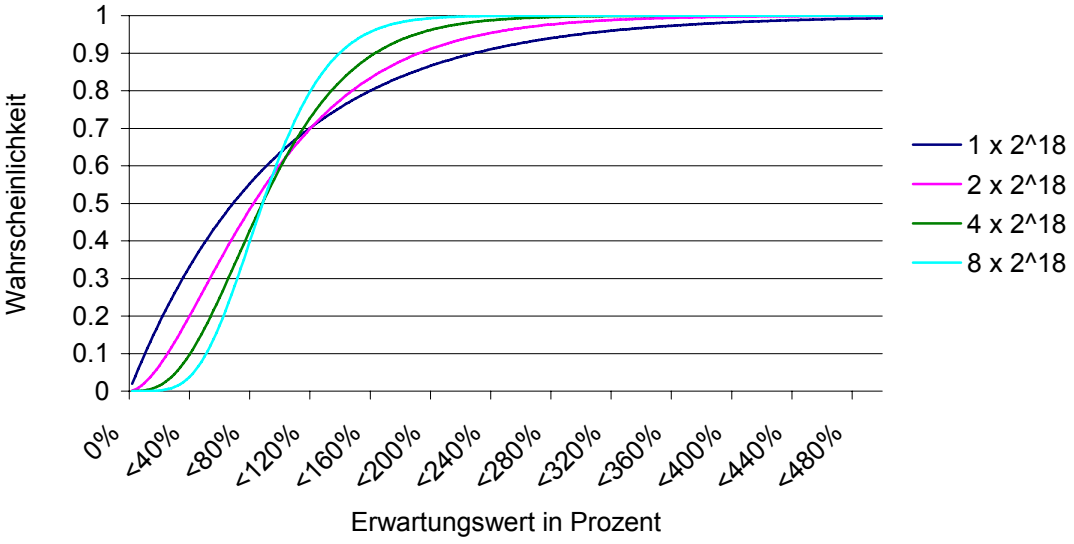


Abbildung 13: Verteilungsfunktion verschiedener Faltungen

Es ist zu erkennen, dass sich der Maximalwert der Dichtefunktion weiter auf den Erwartungswert hinbewegt und zu beiden Seiten steiler wird. Dies bedeutet, dass die zusammengesetzten Funktionen wie erwünscht weniger stark streuen als die Einzelfunktionen. Abbildung 13 zeigt die dazugehörige Verteilungsfunktion, die dieses Ergebnis bestätigt.

Abschließend vergleichen wir die analytischen Ergebnisse zur Faltung mit unseren Messungen. Hierzu nehmen wir die weiter oben beschriebenen 2048 Messungen der HashCashLin 2^{18} Variante und werten sie aus, indem wir jeweils 2, 4, bzw. 8 Messungen zusammenfassen. Abbildung 14 zeigt die daraus resultierenden Dichtefunktionen und Abbildung 15 die Verteilungsfunktionen.

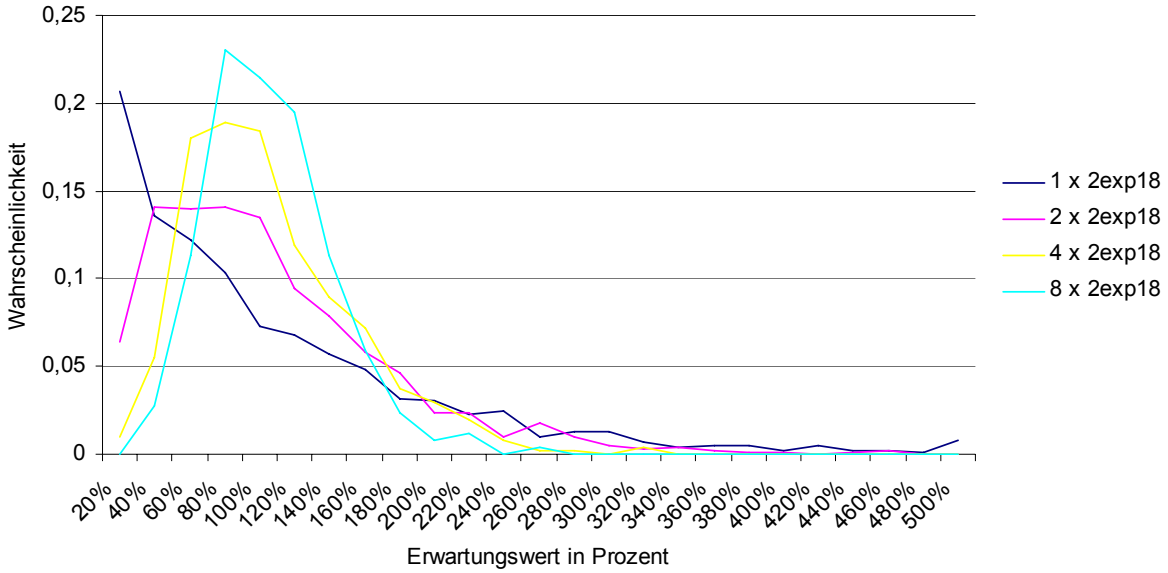


Abbildung 14: Dichtefunktionen für gefaltete Messwerte

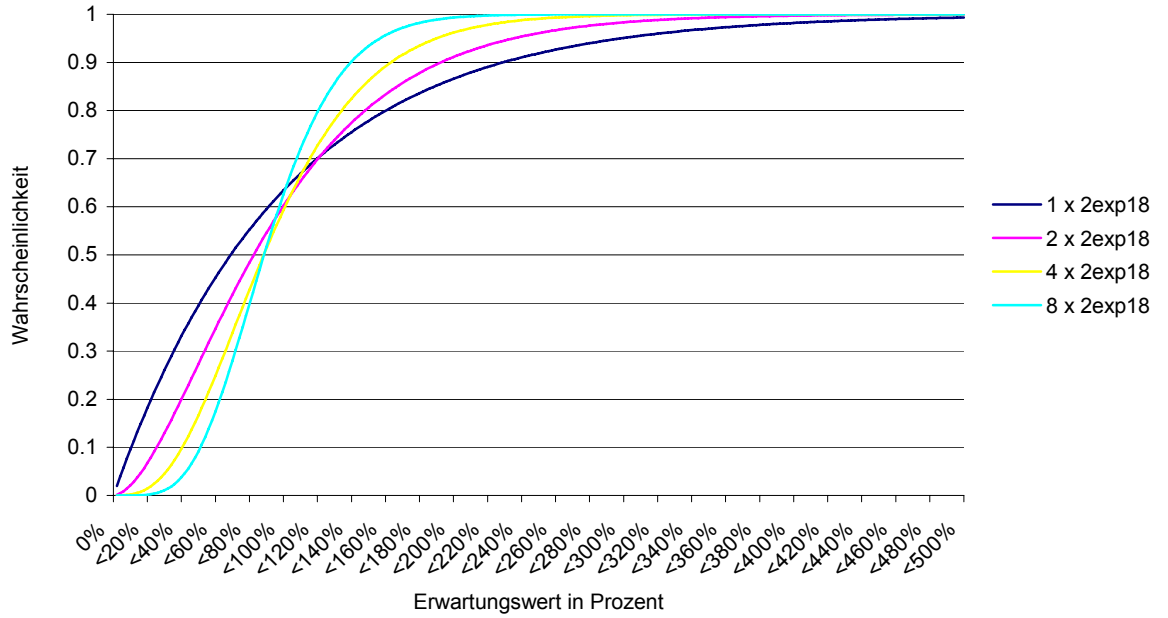


Abbildung 15: Verteilungsfunktionen für gefaltete Messwerte

Für die praktische Anwendung erscheint nach diesen Betrachtungen insbesondere die Variante, eine HashCashLin Funktion vierfach berechnen zu lassen, sehr interessant. Der Overhead durch die Vergrößerung der Lösung ist in diesem Fall noch kein größeres Problem und die Verteilung konzentriert sich bereits deutlich in Richtung des Erwartungswertes.

8.6 Messungen auf realer Hardware

Bisher wurde im Rahmen der Messreihen zur HashCashLin Funktion die Anzahl der benötigten SHA-1 Berechnungen erfasst. Dieses Vorgehen hat den Vorteil, dass die Messergebnisse von der konkret verwendeten Hardware- und Softwareumgebung unabhängig sind. Zum Abschluss des Kapitels erscheint es dennoch angebracht, die Berechnungsdauern auf konkreten Plattformen zu untersuchen, um die anderen Ergebnisse einordnen zu können. Aus diesem Grund wurden Messungen auf den folgenden zwei Plattformen durchgeführt:

- Intel Pentium 1,6 GHz 768 MB RAM, Windows XP, Java 1.6.0
- Intel Core 2 Duo 2,2 GHz 2048 MB RAM, Windows XP, Java 1.6.0

Theoretisch sind auf einem herkömmlichen Prozessor bei einer reinen CPU-Aufgabe keine Geschwindigkeitsvorteile durch Parallelisierung zu erwarten. In letzter Zeit werden aber selbst in handelsüblichen Rechnern immer mehr Zwei- und Vierkernprozessoren verwendet. Dem wurde Rechnung getragen, indem jede der Messungen als einzelner Thread und ein zweites Mal in einer Version mit zwei Threads durchgeführt wurde.

8.7 SHA-1 Berechnung

Bevor die eigentliche Berechnung der HashCashLin Funktion gemessen wird, erfolgt als Vorstufe eine Messung der reinen SHA-1 Leistung der einzelnen Systeme. Hierzu wurden zehn Millionen Ausführungen der SHA-1 Funktion [GESE03b] gemessen und im Anschluss die Anzahl der Durchläufe pro Sekunde errechnet. Zum Vergleich wurde auf den jeweiligen Rechnern ebenfalls der CPU-Teil des Cinebench R10 Benchmarks [MAXON] ausgeführt. Der Benchmark lässt sich nur auf Systemen mit Zweikernprozessoren in der Version mit zwei Prozessen ausführen, so dass dieser Wert lediglich für den Intel Core Duo erhoben werden konnte. Tabelle 2 zeigt die Ergebnisse der beschriebenen Messungen.

Hardware	Ein Prozess	Zwei Prozesse	Cinebench R10 (1 CPU)	Cinebench R10 (2 CPU)
Intel Core Duo 2,2 GHz	842.105 SHA-1/s	1.438.228 SHA-1/s	2165 CB-CPU	4110 CB-CPU
Intel Pentium 1,6 GHz	471.275 SHA-1/s	463.113 SHA-1/s	1383 CB-CPU	---

Tabelle 2: Messergebnisse auf unterschiedlicher Hardware

Ein Vergleich der SHA-1 Werte mit dem jeweiligen Cinebench Ergebnis zeigt, dass sich die SHA-1 Berechnungsleistung in etwa proportional zur sonstigen Leistungsfähigkeit der CPU verhält. Darüber hinaus ist ersichtlich, dass die reine Taktfrequenz des Prozessors keinen direkt Rückschluss auf die SHA-1 Leistung zulässt. Wie zu erwarten war, kann die Leistung auf dem Zweikernprozessor durch einen zweiten Thread beinahe verdoppelt werden, während auf den anderen Prozessoren keine Leistungsgewinne erzielt werden.

8.8 HashCashLin Berechnung

Zur Ermittlung der eigentlichen HashCashLin Leistung wurden auf jedem Rechner 2048 Funktionen mit einem Parameter von 2^{16} berechnet. In einem ersten Durchlauf wurde nur ein Thread und im darauf folgenden wurden zwei Threads verwendet. Auf Grund der probabilistischen Natur der Funktion sind die Aufgaben nicht fest auf die einzelnen Threads verteilt, sondern jeder Thread arbeitet so lange, bis insgesamt 2048 Lösungen vorliegen. Tabelle 3 zeigt die Ergebnisse der Messungen.

Hardware	Ein Prozess	Zwei Prozesse
Intel Core Duo 2,2 GHz	172 s	89 s
Intel Pentium 1,6 GHz	297 s	305 s

Tabelle 3: Messergebnisse zur HashCashLin Funktion

Der Erwartungswert für 2048 Lösungen der HashCashLin Funktionen mit Parameter 2^{16} ergibt sich aus folgender Berechnung:

$$\frac{2048 \cdot 2^{32}}{2^{16}} = 2^{27} \text{ SHA-1 Berechnungen}$$

Wenn man nun den Erwartungswert von rund 134 Millionen SHA-1 Berechnungen durch die ermittelte SHA-1 Leistung der jeweiligen Systeme teilt, erhält man die in Tabelle 4 dargestellten Werte.

Hardware	Ein Prozess	Zwei Prozesse
Intel Core Duo 2,2 GHz	159 s	93 s
Intel Pentium 1,6 GHz	284 s	289 s

Tabelle 4: Mittels der SHA-1 Leistung vorausberechne Berechnungsdauern

Es fällt auf, dass sich diese vorausgesagten Berechnungsdauern relativ nah an den real gemessenen Werten befinden. Es kann zusammengefasst werden, dass die Berechnungsdauer der HashCashLin Funktion direkt von der SHA-1 Leistung der jeweiligen CPU abhängt und den Vorhersagen entspricht.

8.9 Zusammenfassung HashCashLin

Mit der HashCashLin Funktion wurde eine POW Funktion entwickelt, welche die Grundidee der HashCash Funktion, partielle Hash-Kollisionen als CPU-basiertes POW zu verwenden, weiter verfeinert. Einerseits ist es nun möglich, das POW feingranular einzustellen, so dass die POW Funktion für geregelte POW Verfahren erst wirklich nutzbar wird. Andererseits wurde eine parallelisierte Implementierung für Multikern-Prozessoren entwickelt und die Möglichkeit der Streuung der Berechnungsdauern durch ein zusammengesetztes POW untersucht. Darüber hinaus erfolgte eine eingehende Untersuchung zu den statistischen Eigenschaften der HashCashLin Funktion, sowohl theoretisch als auch durch Messungen. Als Ergebnis steht eine POW Funktion zur Verfügung, mit deren Hilfe sich adaptive POW Mechanismen umsetzen lassen.

9 Parametrisierung des Proof-Of-Work

Es wurde mit der HashCashLin Funktion eine präzise parametrisierbare Funktion entwickelt. Nun stellt sich die Frage, wie die eigentliche Parametrisierung im Fall der Überlastvermeidung erfolgen soll. Hierfür müssen zwei grundlegende Fragen geklärt werden: einerseits wie multiple Anfragen eines Klienten behandelt werden sollen und mittels welchen Mechanismus die Schwierigkeit des POW festgelegt wird.

9.1 POW für multiple Anfragen

In einer praktischen Anwendung werden viele Klienten mehrere Anfragen gleichzeitig oder kurz nacheinander an den Dienstanbieter senden. Nun muss geklärt werden, wie mit diesen Anfragen verfahren werden soll. Im Folgenden werden wir die unterschiedlichen Möglichkeiten beleuchten, wie mit dieser Situation umgegangen werden kann.

9.1.1 Kumulierte POW

Bei mehreren Anfragen eines Klienten stellt sich die Grundfrage, ob ein POW für jede einzelne Anfrage oder aber ein gesamtes POW pro Klient erhoben werden soll. Der Vorteil einer einzigen großen POW Aufgabe pro Klient liegt darin, dass der Overhead für die Anforderung und die Übermittlung der POW Lösung reduziert wird. Technisch wäre es im Fall einer gut parametrisierbaren POW Funktion wie HashCashLin möglich, ein solches entsprechend schwierigeres POW zu verlangen. In diesem Fall muss die Schwierigkeit so angepasst werden, dass der Erwartungswert bis zum Finden der Lösung des kumulierten POW der Summe der Erwartungswerte der einzelnen POW entspricht.

Nachteilig ist zu bewerten, dass Anfragen in gewisse Slots aufgeteilt werden müssen, um sie dann unter einem POW zusammenzufassen. Dadurch entstehen eventuell zusätzliche Verzögerungen, da ein anfragender Prozess z. B. warten muss, bis der nächste Slot an Anfragen dieses Rechners abgearbeitet wird.

Ein zu starkes Bündeln der einzelnen POW führt zu mehr statistischer Streuung. Wie wir im vorangegangenen Kapitel gesehen haben, trägt die Verwendung multipler POW dazu bei, die Streuung zu reduzieren. Werden nun Anfragen zu sehr gebündelt, wird dem ungewollt entgegengewirkt.

Darüber hinaus können nicht alle Anfragen eines Rechners gebündelt betrachtet werden. Unter Umständen fragen ganz unterschiedliche lokale Prozesse den Dienst an. Diese verfügen eventuell auf dem Klienten über unterschiedliche Prioritäten. So lange jeder

Prozess das ihm auferlegte POW selbst abarbeitet, wird diese Priorisierung auch im Fall der POW Berechnung beibehalten. Wird aber die POW Berechnung zusammengefasst, ist es nur schwer möglich, nicht alle lokalen Prozesse und ihre Anfragen gleich zu behandeln.

9.1.2 Progressive POW Schwierigkeit

Neben der Schwierigkeit, ob multiple Anfragen im Rahmen des POW zusammengefasst werden sollen, stellt sich die Frage, ob das Anfrageverhalten des jeweiligen Klienten berücksichtigt werden soll oder ob alle Anfragen unabhängig von diesen Randbedingungen gleich behandelt werden sollen.

Es wäre denkbar, eine Progression in die Bestimmung der POW Schwierigkeit einfließen zu lassen. Das würde bedeuten, ein Klient muss für seine zweite Anfrage innerhalb einer Zeiteinheit mehr POW leisten als für die erste. Dies hat den verlockenden Vorteil, dass äußerlich besonders bösartiges Verhalten überproportional bestraft wird. Faktisch entspräche dies einer Art weichen Quota. Statt den Zugriff auf eine feste Anzahl Zugriffe zu limitieren, würden je nach Ausgestaltung der Progression nur zusätzliche Anfragen besonders erschwert werden.

Hier greifen aber wieder die Bedenken, welche bereits in Kapitel 4 zu Quotas als allgemeine Lösungen gegen Überlast vorgebracht wurden. Es ist im aktuellen Internet praktisch kaum möglich, den Klienten als feste Einheit von Seiten des Servers auszumachen. Manche Rechner erscheinen durch IP Masquerading als bösartige Klienten, obwohl sie nur die legitimen Anfragen multipler Klienten weiterleiten. Auf der anderen Seite tarnen sich Angreifer hinter mehreren IP Adressen und Dial-UP Leitungen mit dynamischen IP Adressen.

Unabhängig von der Schwierigkeit, den Klienten als Einheit überhaupt zu identifizieren, besteht bei progressiven POW das Problem, das verwendete Verfahren zu parametrisieren. Es wird ungleich schwieriger, die Last im gewünschten Bereich zu halten, denn es muss die Schwierigkeit pro Klient verwaltet werden und zudem können die Auswirkungen von Änderungen nur bestimmt werden, wenn nicht nur die Anfragelast in etwa konstant bleibt, sondern auch die Verteilung dieser Anfragen auf die einzelnen Klienten.

9.2 Zusammenfassung POW für multiple Anfragen

Wir haben die Möglichkeit diskutiert, pro Klient kumulierte POW zu verlangen. Obwohl dieser Ansatz helfen kann gewisse Verwaltungsaufwendungen zu sparen, überwiegen

unserer Meinung nach die Nachteile dieser Lösung. Ein POW System zur Überlastvermeidung müsste so weit verkompliziert werden, dass dies die Vorteile nicht rechtfertigt. Aus diesem Grund werden wir im Rest dieser Arbeit nur mit Modellen arbeiten, welche jede Anfrage des Klienten mit einem separaten POW belegen.

Ähnlich verhält es sich mit der Möglichkeit, progressive POW Schwierigkeiten einzusetzen. An sich erscheint die Möglichkeit, besonders aggressive Klienten besonders zu bestrafen, sehr reizvoll. Eine Betrachtung der Detailprobleme zeigt aber, dass solch ein Verhalten kaum zuverlässig erkannt werden kann (siehe Masquerading etc.). Darüber hinaus erschwert es die Konstruktion fast aller Verfahren zur dynamischen Anpassung der POW Schwierigkeit. Da es aber das Ziel dieser Arbeit ist, die Überlast situationsabhängig zu kontrollieren, setzen wir im Folgenden die Möglichkeit der progressiven Schwierigkeitsbestimmung nicht um.

Das Ergebnis dieses Kapitels ist, dass sich trotz der offensichtlichen Vorteile differenzierter POW Verfahren der direkte Ansatz, jede Anfrage getrennt und unabhängig vom anfragenden Klienten zu betrachten, bei Berücksichtigung aller Randbedingungen als praktikabelste Lösung herausstellt.

9.3 Parametrisierung des Proof-Of-Work

Es wurde bereits geklärt, dass alle Anfragen eines Klienten separat behandelt werden sollen. Nun stellt sich die Frage, wie die eigentliche Parametrisierung im Fall der Überlastvermeidung erfolgen soll. Hierfür werden drei Verfahren betrachtet: das Festlegen einer festen Parametrisierung, die Ermittlung per Auktion und die Anpassung mittels Regelung.

9.3.1 Feste Parametrisierung

Der einfachste Ansatz besteht darin, einen festen Wert für die Schwierigkeit des POW zu ermitteln. Dieses Verfahren hat den Vorteil, dass praktisch kein Verwaltungsaufwand anfällt, da die Parametrisierung der POW Funktion bekannt ist und nicht übertragen werden muss. Dies ist insbesondere in sehr großen verteilten Systemen vorteilhaft, da auch Anfragen von Klienten zwischen Servern weitergereicht werden können, ohne dass ein neues POW erforderlich wird.

Die Nachteile hinsichtlich der Überlastvermeidung bestehen insbesondere in der mangelnden Adaptivität dieser Lösung. Einerseits wird Klienten ein unnötiges POW

auferlegt, während der Server sich noch gar nicht im Überlastbereich befindet, und andererseits kann bei genügend Anfragen trotz POW eine Überlastsituation auftreten.

Das bedeutet, fest parametrisierte POW Funktionen sind zur Überlastvermeidung, wie sie in dieser Arbeit angestrebt wird, nicht geeignet. Eine feste Parametrisierung stellt eher eine Art Schutzgebühr dar. In anderen Anwendungsfällen, wie z. B. der Bekämpfung von Spam, ist dieses Verfahren vielversprechender. Hier geht es nicht darum, die Last bzw. das E-Mail-Aufkommen auf einen bestimmten Wert zu bringen, sondern gewisse unerwünschte E-Mails sollen nicht mehr versendet werden. Hier kann ein fest eingestelltes POW sinnvoll sein.

Allen Verfahren mit fest eingestellter POW Schwierigkeit ist gemein, dass zumindest langfristig eine Änderung notwendig ist, um der steigenden technischen Leistungsfähigkeit der Klienten gerecht zu werden. Diese Anpassung kann entweder erfolgen, indem eine unabhängige Stelle in gewissen Abständen einen neuen Wert festlegt, oder die Steigerung wird gleich mit in das Verfahren für einen gewissen Zeitraum eingebaut, indem die Entwicklung z. B. entsprechend Moores Law [MOORE65] extrapoliert wird.

Dieser kurze Exkurs in die Spam-Problematik zeigt, dass dieses Verfahren durchaus seine Anwendungsgebiete hat, aber im Fall der fairen Überlastvermeidung nicht sinnvoll einsetzbar ist. Aus diesem Grund werden im Folgenden alternative Verfahren vorgestellt. Detaillierte Informationen und Überlegungen zur festen Parametrisierung von POW Funktionen finden sich unter anderem in der Arbeit *How to Configure Proof-Of-Work Functions to Stop Spam* [GMW05].

9.3.2 POW Auktionen

Der zweite Ansatz, die Schwierigkeit des POW im Überlastfall zu bestimmen, geht davon aus, diese nicht im Vorhinein analytisch zu ermitteln und festzulegen, sondern einen marktähnlichen Mechanismus zu benutzen. Auf einem genügend liquiden Markt gleichen sich Angebot und Nachfrage aus. Wenn man den Dienstanbieter als Verkäufer betrachtet, den Klienten als Käufer sowie das POW als Währung ansieht, folgt, dass ein POW Markt eine Überlastsituation kontrollieren kann. Die Anfragen der Klienten werden mit dem Angebot des Servers in Einklang gebracht. Dies geschieht, indem über den Markt der Preis, in diesem Fall speziell die Schwierigkeit des POW, angepasst wird, bis ein Gleichgewicht herrscht. Somit ist ein Proof-Of-Work-Markt eine Variante, die Parametrisierung der POW Funktion zu bestimmen.

Am Rand sei hier erwähnt, dass in letzter Zeit auch in anderen Bereichen Marktmechanismen dazu verwendet wurden, Parameter in ein Gleichgewicht zu bringen, welche keinen wirklichen Zusammenhang zur eigentlichen Volks- bzw. Betriebswirtschaft zeigen. So haben Berlemann und Schmidt [BS01] die Vorhersagequalität von Political Stock Markets (PSM) untersucht und haben Spann und Skiera [SPSK04] die Anwendbarkeit entsprechender Märkte für die Marktforschung allgemein beschrieben.

In der Wirtschaftswissenschaft kennt man unterschiedlich aufgebaute Märkte. Im konkreten Fall der POW Überlastvermeidung bietet sich ein Markt in Form einer Auktion an, da hierbei sehr kurzfristig ein Ausgleich zwischen Angebot und Nachfrage entsteht. Wang und Reiter [WR03] haben bereits ein POW basiertes Auktionsverfahren zur Bekämpfung von Denial-of-Service Angriffen entwickelt.

Der große Vorteil der Auktion gegenüber anderen Formen der POW Parametrisierung besteht darin, dass im Überlastfall Angebot und Nachfrage immer ausgeglichen bleiben. Da genau die vorhandenen Ressourcen versteigert werden, läuft das System mit einer definierten Last. Bei anderen Verfahren mit fester oder geregelter POW Schwierigkeit kann nie ausgeschlossen werden, dass nicht doch eine gewisse Überlast entsteht, wenn die POW Parametrisierung falsch ist bzw. auf Grund einer Lastspitze zu langsam nachgeregelt wird. Daraus folgt, dass sich das Auktionsverfahren insbesondere für Dienste eignet, welche mit unteilbaren Ressourcen arbeiten. Stellt der Dienst z. B. den Zugriff auf eine nicht parallelisierbare Hardware, wie ein angeschlossenes Teleskop, einen Roboter oder Ähnliches zur Verfügung, wäre eine Überbuchung sehr unerwünscht und somit eignet sich in diesen Fällen ein Auktionsverfahren.

Die Nachteile eines Auktionsverfahrens liegen in dem relativ hohen Verwaltungs- und Kommunikationsaufwand. Gebote müssen abgegeben, gespeichert, ausgewertet und Zuschläge erteilt werden. Neben diesen Reibungsverlusten entsteht auch eine zusätzliche Verzögerung beim Zugriff auf den Dienst, da neben der POW Berechnung die Auktion abgewartet werden muss. Darüber hinaus ist dieser Ansatz für den Klienten leider im Zweifel recht unübersichtlich. Er kann nicht im Voraus sagen, was er tun muss, um einen Zuschlag zu erhalten, sondern kann lediglich historische Daten als Referenz für das zu erwartende Ergebnis heranziehen. Das zeigt, dass ein Auktionsverfahren nur dann sinnvoll ist, wenn die zu schützenden Ressourcen relativ groß sind und sich der zusätzliche Verwaltungsaufwand somit lohnt. Alternativ besteht die Gefahr, dass die Auktion

serverseitig selber mehr Ressourcen verschlingt, als die direkte Abarbeitung der eigentlichen Anfragen benötigen würde.

Es kann zusammengefasst werden, dass sich auktionsbasierte POW Verfahren gut den Gegebenheiten anpassen und nicht teilbare Ressourcen elegant verwalten können. Gleichzeitig haben sie den Nachteil des hohen Verwaltungsaufwandes sowie der Verzögerung durch die Auktion, so dass eine Anwendung für ein allgemeines faires Überlastmanagement im Rahmen dieser Arbeit nicht umgesetzt wird.

9.3.3 POW Regelung

Neben der festen Parametrisierung und einem Auktionsverfahren stellt die dynamische Festlegung durch den Server eine weitere Möglichkeit dar, die POW Schwierigkeit einzustellen. Hierbei ist es dem Server möglich, das POW nach Belieben anzupassen. Zeitlich verzögert erhält er eine Rückmeldung über die Effekte der aktuellen Parametrisierung, indem die Anzahl der Anfragen beobachtet wird. Ziel der serverseitigen Anpassung ist es, die Last möglichst nah am optimalen Betriebspunkt zu halten, so dass weder Überlast entsteht noch Kapazitäten ungenutzt verfallen. Es wird deutlich, dass es sich somit um einen Regelkreis handelt. Der Regler bestimmt mit der Schwierigkeit des POW die Stellgröße, der Dienst stellt die Regelstrecke dar und die Last ist die Regelgröße, welche an die Führungsgröße in Form des optimalen Betriebspunktes angenähert werden soll. Abbildung 16 verdeutlicht diese Modellierung als Regelkreis.

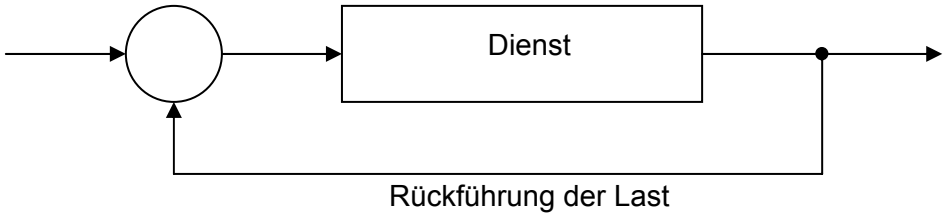


Abbildung 16: Blockschaltbild des Regelkreises

Die eigentliche Aufgabe besteht nun darin, einen möglichst effizienten Regler zu finden. Bevor wir uns diesem Thema im folgenden Kapitel im Detail widmen, müssen die Ziele der Regelung formuliert werden. Diese entsprechen den klassischen Vorgaben der Regelungstechnik, die im Folgenden kurz dargestellt werden.

- Keine bleibende Regeldifferenz

Nach einer Ausregelungsphase soll das System die angestrebte Führungsgröße erreichen und nicht dauerhaft von dieser abweichen. Konkret bedeutet dies, dass die Last des Servers beim optimalen Betriebspunkt liegen soll. Die einzige Ausnahme besteht im Betriebszustand mit einer minimalen POW Schwierigkeit bzw. ohne POW. Sollten trotzdem alle Anfragen abgearbeitet werden können und entsteht somit keine Überlast, ist dies ebenfalls ein erwünschter Betriebszustand.

- Schwingungsverhalten

Die Regelstrecke soll sich so wenig wie möglich aufschwingen. Sollte trotzdem eine Schwingung entstehen, soll diese so schnell wie möglich stabilisiert werden. Eine Schwingung der Last bedeutet, dass der Dienst nicht an seinem optimalen Betriebspunkt arbeitet und entweder in eine Überlastsituation gerät oder bei entgegengesetzter Schwingrichtung Kapazitäten brachliegen.

- Schnelle Ausregelung

Die Antwort der Regelstrecke auf eine Sprungfunktion der Regelgröße soll in einer möglichst schnellen Ausregelung bestehen, ohne wiederum das Schwingungsverhalten negativ zu beeinflussen.

Die Vorteile, eine POW Anpassung über ein Regelungsverfahren zu realisieren, bestehen darin, dass im Gegensatz zu einer festen Parametrisierung der POW Funktion diese an die aktuelle Last angepasst werden kann. Diesen Vorteil bieten zwar ebenfalls die Auktionsverfahren, welche aber einen erheblichen Verwaltungsaufwand sowie Verzögerungen mit sich bringen. Der einzige Nachteil der Regelung besteht darin, dass je nach Schwingungs- und Ausregelungsverhaltens des verwendeten Reglers kurzfristige Über- und Unterlastsituationen nicht gänzlich ausgeschlossen werden können. Trotz dieses Nachteils verfolgen wir den Ansatz der POW Regelung weiter, da dieser bei Verwendung eines geeigneten Reglers am aussichtsreichsten ist. In den folgenden Kapiteln wird ein solcher Regler entworfen, implementiert und getestet.

10 Konstruktion eines POW Reglers

In diesem Kapitel wird ein Regler beschrieben, welcher es ermöglicht, die Schwierigkeit des POW im Sinne der Überlastvermeidung zu parametrisieren. Anfangs werden die zu beachtenden Rahmenbedingungen behandelt und im Anschluss wird auf die konkrete Konstruktion des Reglers eingegangen.

10.1 Rahmenbedingungen

Entgegen gewöhnlicher Regelstrecken hat die POW basierte Überlastvermeidung einige eher unübliche Eigenschaften, welche bei der Konstruktion des Reglers beachtet werden müssen.

10.1.1 Arbeitsbereich

Die HashCashLin POW Funktion verfügt über sehr umfangreiche Parametrisierungsmöglichkeiten. Es können einerseits POW eingestellt werden, die sich in unter einer Millisekunde berechnen lassen, aber andererseits auch POW, welche mehrere Stunden oder Tage benötigen (mit einer trivialen Anpassung der Bitlänge des Kriteriums sind praktisch beliebige Berechnungsdauern möglich). Der Regler muss somit in der Lage sein, möglichst schnell in den konkret effektiven Arbeitsbereich der Funktion zu gelangen. Beim Übersteuern wird das POW wiederum viel zu schwer und dadurch würden die Anfragen komplett heruntergeregelt. Dieser Zustand ist besonders ungünstig, da er sich praktisch kaum korrigieren lässt. Die Klienten berechnen in diesem Fall unrealistisch schwere POW Funktionen und melden sich erst am Ende der Berechnung wieder beim Server. Ist das POW andererseits zu leicht, zeigt es über einen großen Parameterraum praktisch keine Wirkung.

10.1.2 Schwingungsverhalten

Die Regelstrecke unterliegt mehreren probabilistischen Faktoren, welche dem Server nicht exakt bekannt sind. Diese sind die auf den Klienten jeweils zur POW Berechnung zur Verfügung stehende Leistung, das Eintreffverhalten der Anfragen und insbesondere der jeweilige Berechnungsaufwand eines konkreten POW. Diese Faktoren können die Regelung leicht zum Schwingen bringen, was durch den zu konstruierenden Regler unterbunden bzw. minimiert werden muss. Dabei soll der Regler trotzdem ohne bleibende Regeldifferenz ausregeln.

10.1.3 Totzeit

Systembedingt beansprucht die Berechnung von POW Funktionen einen gewissen Zeitraum. Dieser erwünschte Effekt führt dazu, dass die Regelstrecke eine erhebliche Totzeit aufweist. Wird die POW Schwierigkeit geändert, dauert es relativ lange, bis alle Klienten ihr POW nach den alten Parametern abgearbeitet haben und die neue Parametrisierung greift. Darüber hinaus variiert mit jeder Veränderung der POW Schwierigkeit auch die Totzeit. Systeme mit solch einer extrem variablen Totzeit stellen in der Regelungstechnik eher einen Ausnahmefall dar. In diesem konkreten Fall muss diese beachtet werden, da sonst starke Schwingungen auftreten können.

10.2 Regelstrategie

Als Antwort auf die beschriebenen Rahmenbedingungen wird eine gestaffelte Regelstrategie angewendet, welche aus drei Schichten besteht. Abbildung 17 verdeutlicht den Aufbau. Dabei ist zu beachten, dass die Schichten aufeinander aufbauen. Tritt ein großer Sprung in der Regelgröße auf, fällt das Verfahren nach Erreichen eines entsprechenden Kriteriums jeweils auf das darüberliegende, gröbere Verfahren zurück.

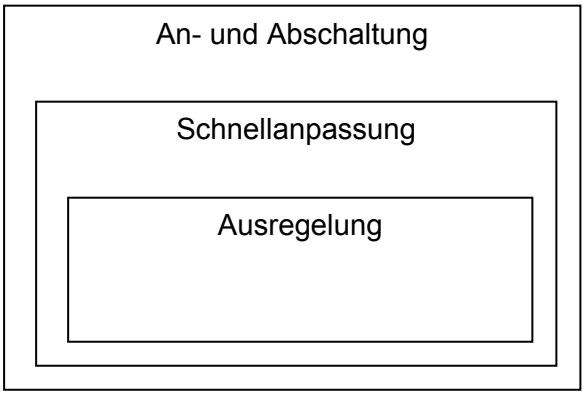


Abbildung 17: Schematische Darstellung des dreischichtigen Reglers

Im Anschluss werden die drei Schichten des Reglers zuerst kurz beschrieben und dann folgt eine detaillierte Erklärung der jeweiligen Schicht sowie deren Parametrisierung.

10.2.1 An- und Abschaltung

Die An- und Abschaltung entscheidet, ob die POW Lastregelung überhaupt aktiviert wird. Sollte die POW Schwierigkeit zu niedrig werden, lohnt sich der Einsatz nicht mehr, da mehr Verwaltungsaufwand entsteht, als Regelungseffekte eintreten. Durch die Möglichkeit, die Regelung im Teillastbereich vollkommen abzuschalten, verursacht sie hier nur einen

minimalen zusätzlichen Verwaltungsaufwand, der sich auf die Beobachtung der Last beschränkt.

10.2.2 Schnellanpassung

Der große Parametrisierungsbereich der HashCashLin POW Funktion kann zeitnah kaum durch ein reines Ausregelungsverfahren beherrscht werden. Aus diesem Grund wird eine Schnellanpassung vorgeschaltet, welche es ermöglicht, den Arbeitsbereich der ausregelnden Schicht zeitnah zu erreichen.

10.2.3 Ausregelung

Als Ergebnis der Regelung soll keine dauerhafte Regeldifferenz verbleiben. Die Schnellanpassung kann das System aber nur sehr grob stabilisieren. Im Anschluss ist ein präzises Ausregeln erforderlich, um die Last möglichst nah am Sollwert zu halten. Diese Aufgabe wird von dieser dritten Schicht erfüllt. Bei der Auswahl der hier eingesetzten Regelverfahren steht somit die Vermeidung einer dauerhaften Regeldifferenz im Vordergrund. Darüber hinaus muss aber auch auf kleinere Schwankungen in der Regelgröße reagiert werden, wenn diese zu klein sind, um die Schnellanpassung zu aktivieren.

10.3 Auswahl der Regler und deren Parametrisierung

Es wurde eine Regelstrategie mit drei aufeinender aufbauenden Schichten entworfen. Die weitere Aufgabe besteht darin, bezüglich jeder Schicht den zu verwendenden Algorithmus sowie dessen Parametrisierung zu bestimmen. In diesem Kapitel werden die drei Schichten von außen nach innen abgearbeitet. Zuvor wird aber die Frage des jeweiligen Anpassungsintervalls für alle drei Schichten gemeinsam geklärt.

10.3.1 Anpassungsintervall und vorgeschaltete Filter

Bevor auf die einzelnen Regler eingegangen wird, ist die generelle Frage zu klären, mit welchem Intervall die rundenbasierten Regler arbeiten. Dieses Intervall kann prinzipiell für jede Schicht einzeln bestimmt werden. Selbst bezüglich der An- und Abschaltung bzw. der oberen und unteren Schnellanpassung wäre eine Differenzierung möglich. Wie sich im Folgenden zeigen wird, ist aber bereits ohne diese Differenzierung die Anzahl der einstellbaren Parameter so groß, dass es ratsam erscheint, das Anpassungsintervall nach einem einheitlichen Verfahren für alle Regelmechanismen zu bestimmen.

Generell kann resümiert werden, dass beim Anpassungsintervall zwischen einer schnellen Anpassung durch ein kurzes Intervall und der Schwingungsdämpfung durch ein längeres Intervall abgewogen werden muss.

So lange keine Filtermechanismen eingesetzt werden, kann als untere Schranke für das Intervall die zu erwartende Berechnungsdauer des POW auf normaler Hardware angesetzt werden. Jedes noch kürzere Intervall muss fast zwangsläufig zu Schwingungen führen, da dauerhaft übersteuert wird. Dieser Umstand erklärt sich darüber, dass die Klienten zumindest diese Zeit benötigen, um ein POW zu errechnen, und somit der Regelungseffekt erst mit entsprechender Verzögerung eintritt. Die Berechnungsdauer des POW variiert je nach Parametrisierung sehr stark, was so auch gewünscht ist. Daraus ergibt sich, dass ein zeitlich fixiertes Anpassungsintervall den Anforderungen nicht gerecht werden kann. Abbildung 18 verdeutlicht diese Feststellung durch die Lastkurve eines Experiments. Dabei wird ein identisches System einmal mittels eines fest eingestellten Anpassungsintervalls ausgeregelt und einmal mittels eines dynamischen Intervalls. Im konkreten Fall ist das feste Intervall zu klein gewählt und man erkennt, wie die Last in eine starke Schwingung gerät. Im zweiten Beispiel wird mittels eines dynamischen Intervalls, welches das Dreifache der erwarteten mittleren POW Berechnungsdauer beträgt, gearbeitet und das System regelt sauber aus.

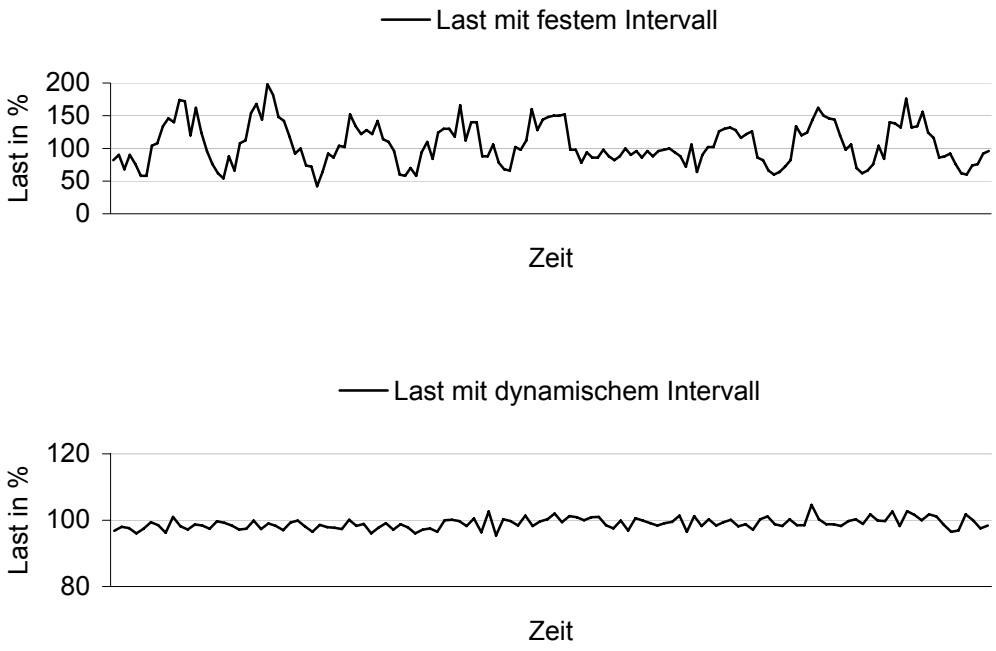


Abbildung 18: Schwingungen bei Verwendung eines festen Anpassungsintervalls

Im Folgenden stellt sich die Frage, ob die Schwingungen bei kurzen Anpassungsintervallen durch Filterverfahren kontrolliert werden können. Auf diesem Weg wäre es möglich, den Vorteil der schnelleren Anpassung zu nutzen, ohne in das unerwünschte Schwingverhalten zu geraten. Klassische Filterverfahren wie z. B. Kalman Filter [KAL60] erscheinen hier wenig geeignet, da diese davon ausgehen, dass ein realer Zustandswert existiert, welcher durch eine Störung wie z. B. Rauschen überlagert ist. Diese Störung soll nun durch den Filter herausgerechnet werden. Im Fall der POW Lastregelung stellt sich die Situation aber anders dar. Nach einer Änderung des POW finden auf Klientenseite Vorgänge statt, deren Ergebnis erst mit einer Zeitverzögerung für den Server sichtbar wird. So beginnen einige Klienten mit einer unbekannten Menge an Ressourceneinsatz POW Lösungen zu berechnen, während andere Klienten davon Abstand nehmen, den Service unter den aktuellen Bedingungen zu nutzen. Eine definitive Aussage über diese Faktoren ist erst dann möglich, wenn genügend Klienten die Chance hatten, zumindest ihr erstes POW entsprechend den neuen Konditionen zu berechnen. Vor diesem Zeitpunkt ist diese entscheidende Information in den gemessenen Lastwerten nicht enthalten und somit sind Filterverfahren kein geeigneter Weg, diese Information zu extrahieren. Im Rahmen einer weiteren Forschung wäre es aber durchaus denkbar, eine schnellere Anpassung zu erreichen. Dabei müsste man nicht wie in dieser Arbeit die absolute Entwicklung der Last betrachten, sondern diese mit einer auf der Verteilungsfunktion des POW beruhenden Annahme über die Lastentwicklung nach der Anpassung vergleichen. In Szenarien mit einer großen Anzahl an Klienten sollte es so möglich sein, bevor die mittlere Berechnungsdauer des neuen POW erreicht ist, eine Aussage darüber zu treffen, welche Last zu erwarten ist, wenn das aktuelle POW voll greift.

Um auf den Filteransatz zurückzukommen, wäre es möglich, sobald sich im Rahmen der längerfristigen Ausregelung eine Schwingung mit konstanter Frequenz etabliert hat, diese Schwingung rechnerisch auszugleichen. Diese Frequenz hängt von den jeweiligen Gegebenheiten der aktuellen Zusammenstellung der Klienten ab, deshalb kann sie aber kaum vorhergesagt werden. Es müsste somit jeweils abgewartet werden, bis die Schwingung entsteht, um sie dann bekämpfen zu können. Hier scheint eine schlichte Vergrößerung des Anpassungsintervalls auf einen etwas größeren Wert den einfacheren und effizienteren Weg darzustellen, von Anfang an ein stabileres System zu erhalten.

Aus diesen Überlegungen folgt die Parametrisierung des Anpassungsintervalls, das über die zu erwartende mittlere Berechnungsdauer des aktuellen POW bestimmt wird. Darüber hinaus sollte das Anpassungsintervall selbst bei sehr kleinen POW nicht zu klein werden, damit sonstige Effekte, wie z. B. das Scheduling des Servers etc., ausgeglichen werden. Aus diesem Grund wird neben dem Multiplikator für die aktuelle POW Berechnungsdauer ein absoluter Minimalwert für das Anpassungsintervall konfiguriert.

Somit ergibt sich folgende Parametrisierung:

$$adapt\_interv = \max(pow\_dur \cdot pow\_factor, interv\_min)$$

adapt_interv: das Intervall der Anpassung (in ms)
pow_dur: zu erwartende Berechnungsdauer des POW (in ms)
pow_factor: Faktor zur Bestimmung der Dämpfung
interv_min: minimales Intervall (in ms)

Randbedingungen:

$pow\_factor > 1$
$interv\_min > 0$

Das Anpassungsintervall kann entsprechend dieser Definition über die beiden Parameter pow_factor und pow_dur eingestellt werden und gilt dann für alle drei Schichten des Reglers.

10.3.2 Intervallabbruch

Das reguläre Regelintervall hat eine Schwäche bezüglich akut auftretender Lastspitzen. Wird gleichzeitig eine größere Anzahl Klienten gestartet, muss zumindest das Ende des Regelintervalls abgewartet werden, bevor gegengesteuert werden kann. Dies kann unerwünscht lange dauern und eventuell dazu führen, dass bereits eine bedrohliche Überlastsituation entstanden ist, bevor die Regelung greift. Aus diesem Grund ist ein Abbruch des Intervalls vorgesehen. Dieser erfolgt, sobald im aktuellen Intervall ein definiertes Vielfaches der Anfragen, welche für das gesamte Intervall zulässig wären, eingegangen ist. Durch dieses Verfahren kann die Regelung zeitnah auf plötzliche Lastspitzen reagieren.

Der Intervallabbruch wird wie folgt parametrisiert:

interv_stop: Schwellwert für den Intervallabbruch als Faktor der Führungsgröße

Randbedingungen:

interv_stop > 1

10.3.3 An- und Abschaltung

Die An- und Abschaltung bestimmt, wann die POW Überlastregelung in Betrieb ist. Das heißt, es muss einerseits eine Bedingung festgelegt werden, wann die Regelung aktiv wird, und andererseits, wann sie wieder abgeschaltet wird. Die Bedingung für das Anschalten der Regelung wird durch einen Schwellwert für die Serverlast gebildet. Sobald dieser in einem Anpassungsintervall überschritten wird, startet die Regelung. In diesem Moment wird die POW Schwierigkeit auf einen konfigurierten Initialwert gesetzt, welcher dann in den folgenden Intervallen mittels Schnellanpassung und Ausregelung angepasst wird. Der Initialwert sollte so gewählt werden, dass mit einem sehr schnell zu berechnenden POW begonnen wird. Wie bereits weiter oben beschrieben, ist es deutlich besser, die Schwierigkeit in den nächsten Intervallen zu erhöhen, als mit einem zu schweren POW zu starten und die Anfragen damit praktisch zum Erliegen zu bringen.

Das Abschalten der POW Regelung erfolgt ebenfalls durch einen Schwellwert. Damit eine kurzfristige Schwankung der Last nach unten nicht zu einer Abschaltung der Regelung mit gleich darauf wieder folgender Anschaltung führt, ist das Unterschreiten des Schwellwertes als Kriterium nicht ausreichend. Aus diesem Grund wird zusätzlich verlangt, dass das aktuell geforderte POW sehr leicht ist. Hier bietet sich der bei der Anschaltung verwendete Initialwert als Kriterium an. Mittels beider Bedingungen ist sichergestellt, dass die Last wirklich auf Grund der weniger gewordenen Anfragen der Klienten gesunken ist und nicht weil beim POW übersteuert wurde und somit die Last kurzfristig gesunken ist.

Daraus folgt die Parametrisierung:

on_limit: Schwellwert für Aktivierung (in % Last)
off_limit: Schwellwert für Deaktivierung (in % Last)
on_pow: initiales POW nach Aktivierung sowie Schwellwert für Deaktivierung

Randbedingungen:

$on\_limit \geq off\_limit$

10.3.4 Schnellanpassung

Die Schnellanpassung ermöglicht es, die Stellgröße POW Schwierigkeit möglichst schnell in eine Position zu bringen, von der aus das Ausregeln mit klassischen Verfahren möglich wird. Dazu wird eine exponentielle Anpassung verwendet. Das bedeutet, so lange die Schnellanpassung aktiv ist, wird die POW Schwierigkeit, je nachdem, ob es sich um die obere oder untere Schnellanpassung handelt, mit einem passenden Parameter multipliziert. Daraus folgt eine Parametrisierung wie bei der An- und Abschaltung über obere und untere Schwellwerte sowie den jeweiligen Anpassungsfaktor. Dabei muss die Parametrisierung so gewählt werden, dass die Schnellanpassung nicht direkt vom unteren Anpassungsbereich in den oberen oder umgekehrt gelangt, da sonst keine Ausregelung einsetzen kann.

Darüber hinaus muss die Schnellanpassung noch einen weiteren Fall abdecken. Es kann passieren, dass Klienten, welche über weitere Leistungsreserven verfügen, zugreifen und eine Last erzeugen, welche unterhalb der Schwelle für die obere Anpassung liegt. In diesem Fall tut sich die folgende Ausregelung eventuell sehr schwer, bis der aktuelle Arbeitsbereich der POW Funktion gefunden wird. So lange die Klienten über Leistungsreserven verfügen, hat die Erhöhung der POW Schwierigkeit keinen Einfluss auf die Last des Servers. Um diesem Umstand Rechnung zu tragen, wird erstens bei einmal aktivierter oberer Schnellanpassung die Last immer so weit reduziert, dass die Führungsgröße zumindest erreicht oder unterschritten wird. Zweitens wird die obere Schnellanpassung nicht nur regulär über den Schwellwert aktiviert, sondern auch, wenn die Regelgröße Last in einer gewissen Anzahl von aufeinanderfolgenden Intervallen über der Führungsgröße lag.

Abbildung 19 verdeutlicht die Vorteile der Schnellanpassung. Es wurde ein identischer Lastfall einmal mit Schnellanpassung und ein zweites Mal ohne simuliert. Es ist zu

erkennen, dass die Schnellanpassung das System deutlich schneller stabilisiert als die reine Ausregelung mittels eines Integral-Reglers in der zweiten Variante.

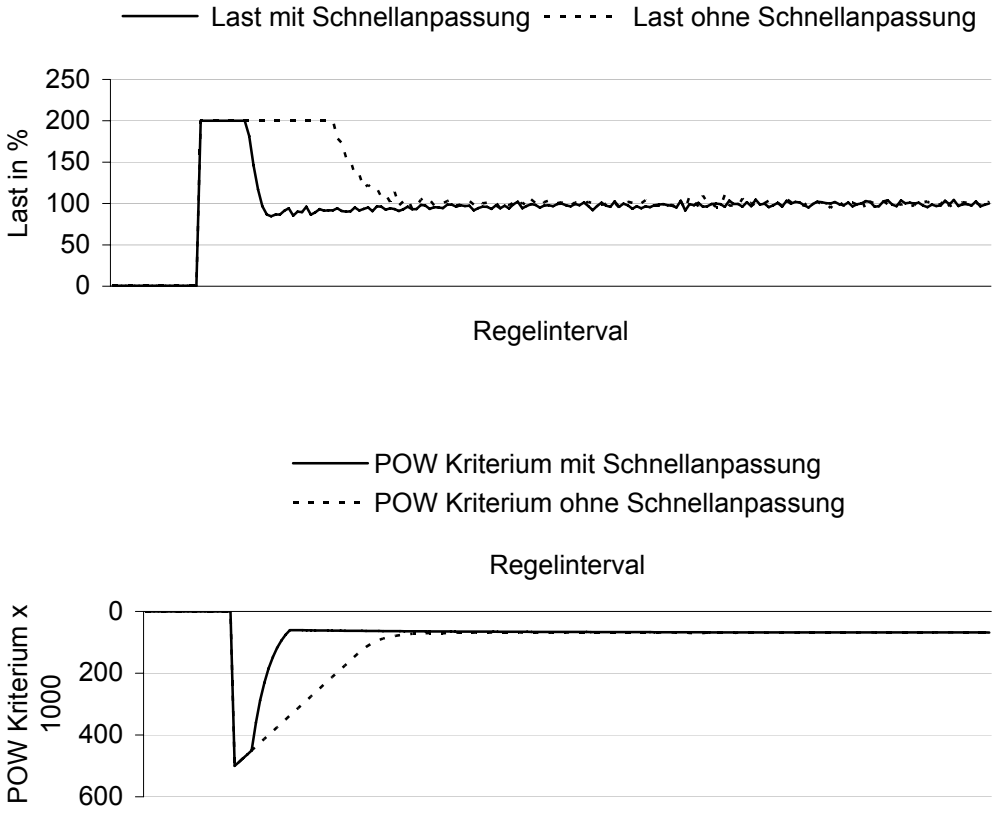


Abbildung 19: Vergleich eines Systems mit und ohne Schnellanpassung

Durch das Verfahren der Schnellanpassung ist sichergestellt, dass die Ausregelung ein POW vorfindet, welches ein schnelles Ausregeln ermöglicht. Die sich daraus ergebenden Parameter sind:

upper_limit: Schwellwert für Deaktivierung (in % Last)
upper_count: Anzahl Intervalle über Sollwert für alternative Aktivierung
upper_factor: Anpassungsfaktor
lower_limit: Schwellwert für Aktivierung (in % Last)
lower_factor: Anpassungsfaktor

Randbedingungen:

upper_factor < 1 *
upper_count > 1
lower_factor > 1 *

upper_limit · upper_factor > lower_limit

lower_limit · lower_factor < upper_limit

* Im Fall der HashCashLin Funktion gilt, je kleiner ein Parameter ist, desto größer ist die Schwierigkeit, das POW zu lösen bzw. umgekehrt, je größer ein Parameter ist, desto geringer ist die Schwierigkeit, das POW zu lösen.

10.3.5 Ausregelung

Durch die beiden vorgeschalteten Verfahren beginnt die Ausregelung in einen Zustand, bei dem sich, je nach Güte der Schnellanpassung, der Stellwert bereits relativ nah am Sollwert befindet. An dieser Stelle können wir die in der Regelungstechnik bewährten linearen Regler einsetzen. Dazu untersuchen wir den Proportional- (P), Integral- (I) und Differential- (D) Regler bezüglich ihrer Eignung für die POW Überlastregelung. Abbildung 20 zeigt die charakteristische Sprungantwort dieser drei Regler:

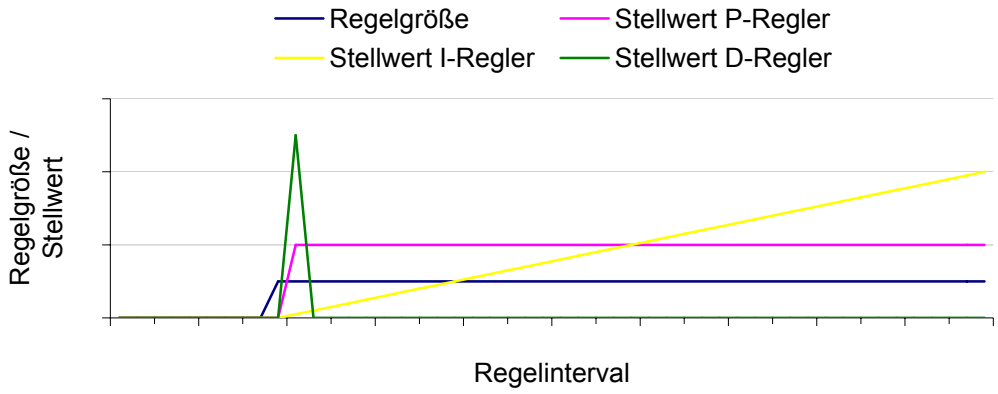


Abbildung 20: Sprungantwort des P-, I- und D-Reglers

- P-Regler

Der Proportionalregler stellt einen linearen Zusammenhang zwischen der Regelgröße und dem Stellwert her. Die Vorteile dieses Reglers sind seine einfache Implementierbarkeit und seine direkte Reaktion auf Veränderungen der Regelgröße.

Dem steht der Nachteil entgegen, dass er keine dämpfende Wirkung hat und somit zum Schwingen neigt. Dies ist im Fall der POW Regelung zu beachten, da die Gefahr von Schwingungen durch das probabilistische Verhalten und die Hysterese der POW Funktionen hoch ist.

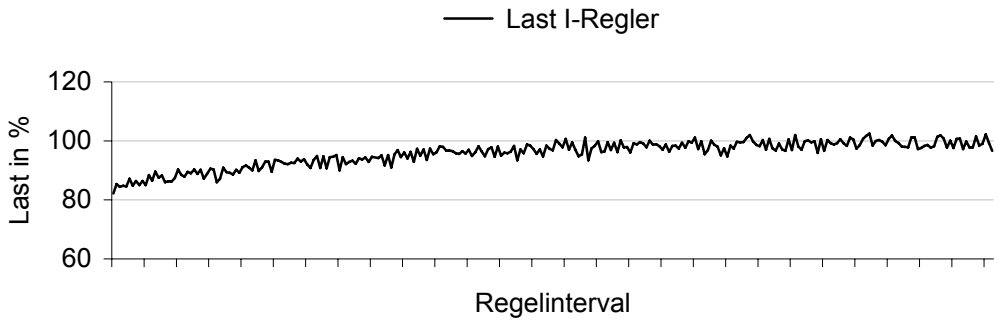
- I-Regler

Der Integralregler bildet aus dem Integral zwischen Regelgröße und Führungsgröße den Stellwert. Dadurch regelt der I-Regler das System dauerhaft ohne bleibende Regeldifferenz aus und neigt wenig zum Schwingen. Dies wird mit dem Nachteil erkauft, dass der I-Regler jeweils eine gewisse Zeit benötigt, bis seine Regelung greift. Insgesamt ist der I-Regler von seinen allgemeinen Eigenschaften her sehr gut für den Einsatz der POW Ausregelung geeignet.

- D-Regler

Der Differentialregler (D-Regler) setzt die Ableitung der Regelgröße in einen Stellwert um. Dies hat den Vorteil, dass der D-Regler besonders schnell auf Sprünge der Regelgröße reagiert. Es bedingt aber auch, dass der D-Regler keinerlei dauerhafte Änderung des Stellwerts bewirkt und somit alleine kaum einsetzbar ist. Darüber hinaus übersteuert der D-Regler bei starken Sprüngen leicht, was schnell zu unerwünschten Schwingungen führen kann. Da durch die probabilistische Natur der POW Funktionen dauerhaft kleinere Sprünge in der Regelgröße auftreten, ist der D-Regler in der Ausregelungsphase ungeeignet, da er in diesen Fällen das System stark in Unruhe versetzen würde.

Die Betrachtung der einzelnen Regelglieder hat ergeben, dass im Fall der POW Lastregelung ein I-Regler unerlässlich ist, da nur dieser die Regelgröße dauerhaft ausregelt. Darüber hinaus ist eine Kombination mit einem P-Regler möglich. Der Vorteil des PI-Reglers gegenüber dem reinen I-Regler liegt in der schnelleren Annäherung an den Sollwert, was durch eine größere Unruhe der Regelgröße erkauft wird. Abbildung 21 vergleicht die Ausregelung einer identischen Situation mittels des I-Reglers und des PI-Reglers.



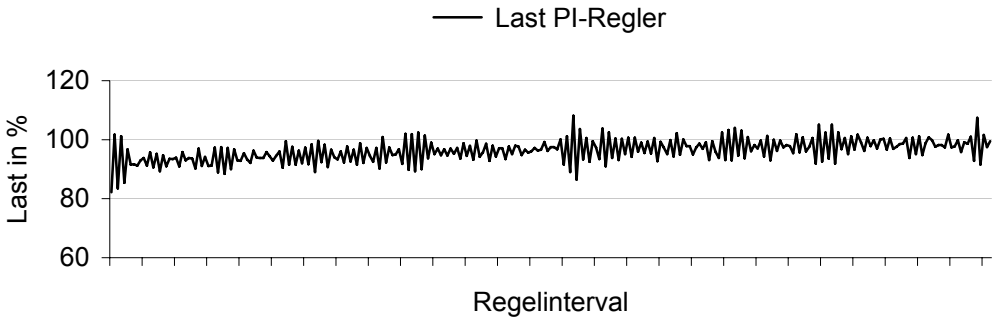


Abbildung 21: Vergleich der Ausregelung mittels I-Regler und PI-Regler

Es ist zu erkennen, dass der PI-Regler erhebliche Schwingungen des Sollwertes bewirkt, weshalb im Rahmen dieser Arbeit der reine I-Regler verwendet wird. Es wäre aber unter Umständen möglich, durch ein zusätzliches Verfahren den P-Anteil der Ausregelung sozusagen ausklingen zu lassen und so die Schwingungen des PI-Reglers zu begrenzen. Der zu erwartende Vorteil gegenüber einem gut eingestellten I-Regler rechtfertigt aber in diesem Fall nicht die zusätzliche Komplexität des Systems. Abbildung 22 zeigt, wie mittels eines vergrößerten Parameters eine schnellere Ausregelung mittels der I-Regelung möglich wird. Der Parameter darf nur nicht zu groß gewählt werden, da sonst wieder Schwingungen in das System geraten können. Abbildung 23 zeigt ein Beispiel mit deutlich zu großem Parameter, so dass die Regelung in starke Schwingungen gerät.

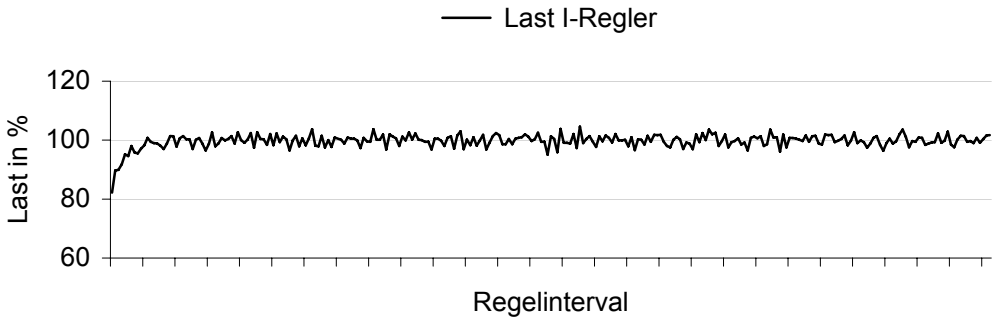


Abbildung 22: Ausregelung mittels I-Regler und vergrößertem Parameter

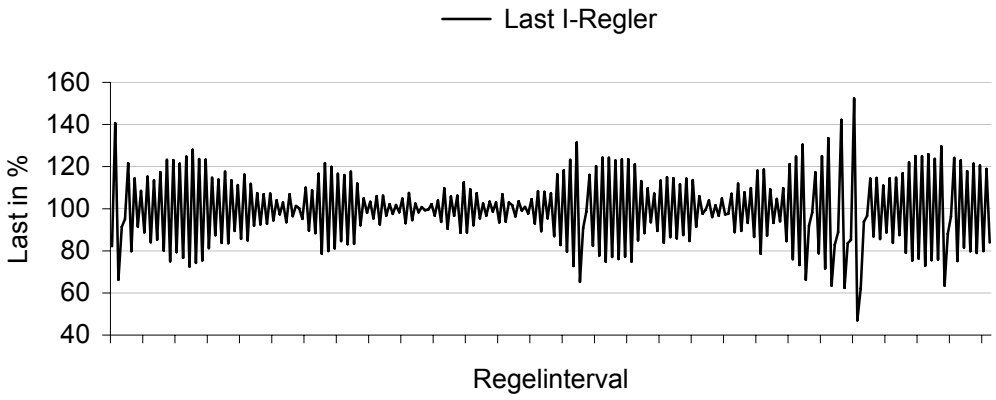


Abbildung 23: Ausregelung mittels I-Regler und zu großem Parameter

In der konkreten Konstellation mit vorgeschalteter Schnellanpassung muss der I-Regler jeweils bei Beendigung der Schnellanpassung neu initialisiert werden. Das Integral darf entsprechend erst ab diesem Punkt berechnet werden. Darüber hinaus muss die Parametrisierung des I-Reglers relativ zu dieser Initialisierung erfolgen. Ein fester Parameter wäre wegen der ganz unterschiedlich großen Werte ungeeignet.

Die Parametrisierung erfolgt dementsprechend wie folgt:

i_factor: Faktor, mit dem das Integral der Lastabweichung multipliziert wird als Anteil des POW Anfangswertes der Ausregelung

Randbedingungen:

$0 < i\_factor < 1$

10.4 Zusammenfassung Regelung

Mit der dreischichtigen Regelung aus An- und Abschaltung, Schnellanpassung und Ausregelung wurde ein System entworfen, welches an die spezifischen Anforderungen der POW Regelung angepasst ist. Dabei bleibt der Entwurf so einfach wie möglich und baut auf bekannten und erprobten Verfahren wie den Schwellwerten und dem I-Regler auf. Die konkrete Parametrisierung wurde noch nicht vorgenommen, es konnte aber bereits aufgezeigt werden, welche Probleme in der Praxis umschifft werden müssen. An erster Stelle steht hier die Vermeidung der systembedingt drohenden Schwingungen.

Ohne weiteres wäre es möglich, das Verfahren nach Belieben zu komplizieren, indem z. B. Kennfelder zur Abbildung des typischen Lastverhaltens einer Anwendung oder künstliche Intelligenz zur Vorhersage der Last eingesetzt wird. Unserer Ansicht nach liefern diese

Verfahren bei geeigneter Parametrisierung aber nur geringe Voreile, sind dafür jedoch deutlich schlechter einzustellen. Damit steht zu befürchten, dass solche Verfahren falsch konfiguriert im praktischen Einsatz schlechtere Ergebnisse liefern als der robuste dreischichtige Ansatz.

In den folgenden Abschnitten wird das entworfene Verfahren zuerst mittels einer Simulation erprobt. Im Anschluss wird ein Prototyp beschrieben, welcher die POW Überlastvermeidung in einen HTTP basierten Dienst integriert.

11 Simulation

Die POW Lastregelung soll insbesondere in großen Szenarien ein faires Überlastverhalten herstellen. Praktisch ist es aber nicht möglich, das Verfahren in einem solchen Fall zu erproben bzw. einen solchen als Experiment real nachzustellen. Um dennoch die Leistungsfähigkeit des Verfahrens testen zu können, wird in diesem Kapitel eine Simulation entwickelt, mittels derer das reale Verhalten der Regelung zumindest approximiert werden kann.

Auf Grund der begrenzten Hardwareressourcen können im Fall der Simulation keine POW Funktionen real gelöst werden. Da die Eigenschaften der HashCashLin Funktion aber in den vorangegangenen Kapiteln hinreichend untersucht wurden, kann mittels der bekannten Verteilungen und eines Zufallsgenerators eine fiktive Berechnungsdauer für jede einzelne POW Funktion ermittelt werden.

11.1 Generierung realistischer Berechnungsdauern

Der große Vorteil der Simulation besteht in der künstlichen Generierung der Proof-Of-Work-Berechnungsdauern. Folglich ist es unerlässlich, dass die Verteilung dieser Zufallszahlen mit der realen Verteilung der Berechnungsdauern übereinstimmt. Im Folgenden wird zunächst das Verfahren zur Generierung dieser Zufallszahlenreihe hergeleitet und im Anschluss durch Messungen überprüft. Dabei muss die Verteilung der generierten Berechnungsdauern mit der Verteilung der real ermittelten Berechnungsdauern übereinstimmen. Es wird an dieser Stelle ausführlich auf die korrekte Verteilung der Berechnungsdauern eingegangen, weil diese einen entscheidenden Einfluss auf das Schwingverhalten der POW Regelung besitzen. Eine Simulation mit falschen Berechnungsdauern würde unweigerlich falsche Ergebnisse hervorbringen.

Ausgangspunkt für die konkrete Generierung der geometrisch verteilten Zufallszahlen sind die in Java zur Verfügung stehenden, gleichverteilten Zufallszahlen auf dem Intervall [0, 1]. Um die Gleichverteilung zu untersuchen, wurde eine Messung mit einer Million nacheinander generierten Zufallszahlen durchgeführt. Abbildung 24 zeigt die Dichte der ermittelten Werte, wobei immer Intervalle der Größe 0.01 zusammengefasst wurden. Die Güte der Gleichverteilung erscheint für die Zwecke der Simulation ausreichend.

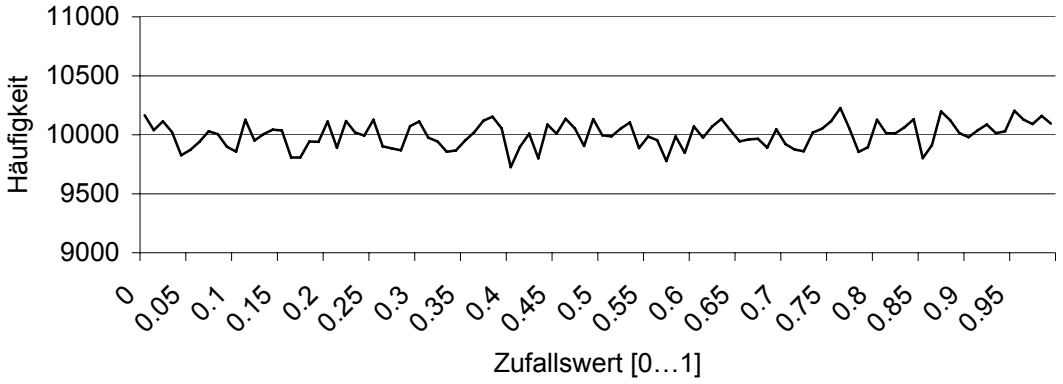


Abbildung 24: Dichte von einer Million Zufallszahlen in 100 Intervallen

Im folgenden Schritt müssen diese gleichverteilten Zufallszahlen in eine geometrische Verteilung umgeformt werden. Dazu wird die Inversionsmethode verwendet:

Die Wahrscheinlichkeit, nach *n* Versuchen keine Lösung erhalten zu haben, ist:

$$(1\text{-}p)^n$$

p: Wahrscheinlichkeit, eine Lösung zu erhalten

Nach *n* Versuchen mindestens eine Lösung erhalten zu haben, ist entsprechend:

$$1\text{-}(1\text{-}p)^n$$

Nun muss die Umkehrfunktion X dieser Wahrscheinlichkeitsverteilung ermittelt werden:

$$X = 1\text{-}(1\text{-}p)^n$$

$$1\text{-}X = (1\text{-}p)^n$$

$$\log_{1-p}(1-X) = n$$

Zur praktischen Berechenbarkeit umgeformt in:

$$\frac{\ln(1-X)}{\ln(1-p)} = n$$

Zur Verifikation dieses Verfahren wurde die Verteilung auf drei unterschiedliche Arten geplottet (siehe Abbildung 25).

- Analytisch: Plot einer geometrischen Verteilung mit dem Parameter p = 1/10000

- Zufallszahlen: Generierung von einer Million Zufallszahlen mittels der Inversionsmethode
- Experimentell: 100.000 Berechnungen der HashCashLin Funktion

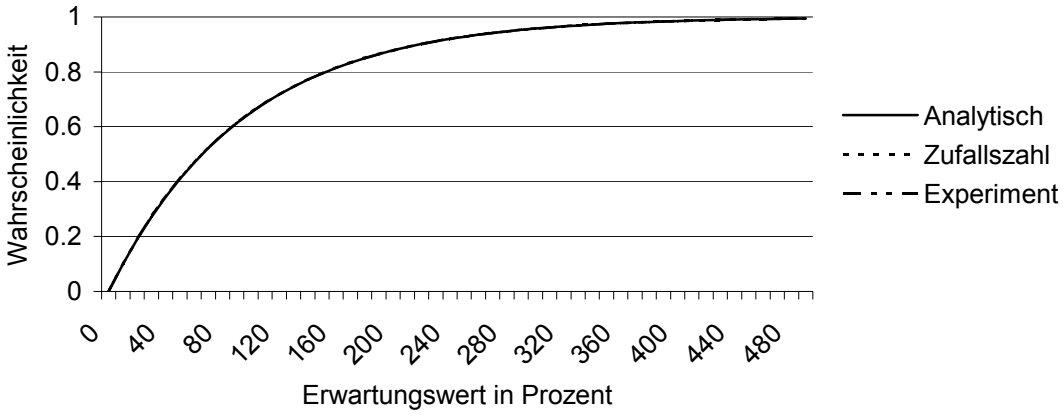


Abbildung 25: Plot der Wahrscheinlichkeitsverteilungen

Die abgebildeten Verteilungsfunktionen zeigen, dass die Werte des praktischen Experiments mit den Werten der geplotteten geometrischen Verteilung sowie den Zufallszahlen, welche mittels der Inversionsmethode umgeformt wurden, einen identischen Verlauf zeigen. Somit wird davon ausgegangen, dass die generierten Zufallszahlen das reale Verhalten der HashCashLin Funktion angemessen abbilden.

11.2 Testszenario

Entscheidend für die Aussagekraft der Simulationsergebnisse ist die Konzeption eines geeigneten und möglichst realitätsnahen Testszenarios. Aus diesem Grund werden im Abschnitt zum Klientenverhalten typische Nutzungsraster isoliert, welche im Anschluss die Grundlage des verwendeten Messzyklus bilden.

11.2.1 Klientenverhalten

In einem realen Überlastszenario existiert eine große Menge an unterschiedlichen Klienten- bzw. Nutzerverhalten. Um dennoch eine möglichst aussagekräftige Simulation durchführen zu können, werden die Nutzer in zwei bzw. drei Gruppen unterteilt. Im Folgenden werden die drei Typen genauer charakterisiert.

Standard Klient

Der „Standard Klient“ entspricht einer Verallgemeinerung eines durchschnittlichen intensiven Nutzungsverhaltens. Der Klient sendet so weit möglich eine gewisse Anzahl von Anfragen pro Zeiteinheit. Konkret ist dies für die Simulation eine Anfrage pro zwei Sekunden. Da hier ein realer Nutzer vorausgesetzt wird, steigert er die Anzahl seiner Anfrageversuche, wenn Anfragen abgelehnt werden sollten, wobei maximal ein Anfrageversuch pro Sekunde gesendet wird, da davon ausgegangen wird, dass diese manuell ausgelöst werden müssen.

Referenz-Klient

Der „Referenz-Klient“ entspricht von seinem Zugriffsverhalten her einem Standard-Klienten. Der einzige Unterschied besteht darin, dass die Zugriffe dieses Klienten in der Simulation für die Bewertung der Regelung individuell erfasst werden. Die für diesen Klienten gemessenen Werte spiegeln somit die individuelle Perspektive eines einzelnen legitimen Nutzers auf das System wider.

Bösartiger Klient

Der „bösartige Klient“ entspricht einem Rechner, welcher einen Denial-of-Service-Angriff durchführt. Der Klient startet so viele Angriffe, wie es ihm technisch möglich ist. Da der Angriff vollautomatisch abläuft, entstehen hierbei auch nur minimale Verzögerungen, so dass der Klient bis zu mehrere hundert Anfragen pro Sekunde absetzen kann.

11.2.2 Messzyklus

Die Grundlage der Simulation ist ein Messzyklus. Dieser definiert die jeweilig simulierte Lastsituation, mit der sich die POW Regelung konfrontiert sieht. Die Gestaltung des Messzyklus hat ganz erhebliche Auswirkungen auf die Ergebnisse. Insbesondere darf die Regelung nicht speziell an die Charakteristik des Zyklus angepasst werden, da sonst zwar in der Simulation exzellente Ergebnisse erreicht werden können, aber die Regelung in sonstigen Fällen nur deutlich schlechtere Ergebnisse liefert. Ein besonders negatives Beispiel stellen hier die Testzyklen zum Abgasverhalten von PKW dar [BOSCH07], weil die Regelung oft genau auf den Zyklus optimiert wird und dafür in realen Anwendungsfällen deutlich schlechter abschneidet.

Aufbauend auf den im vorangegangenen Abschnitt definierten drei Typen von Klienten wird der in Abbildung 26 abgebildete Messzyklus definiert.

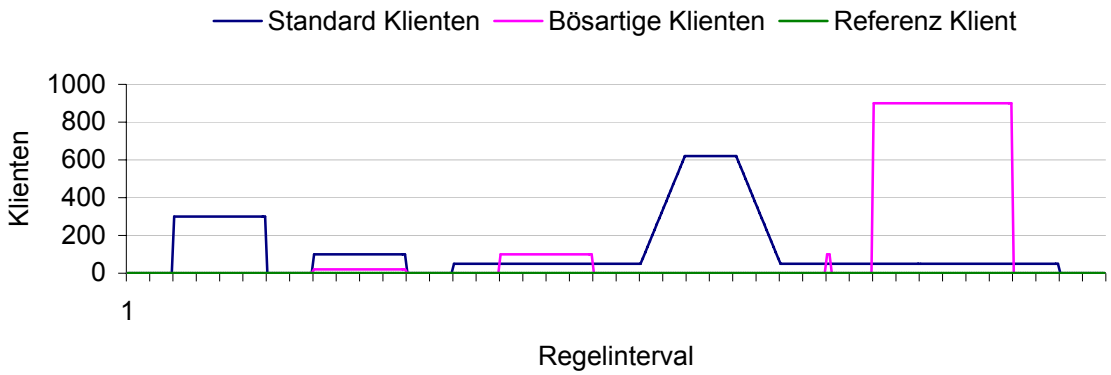


Abbildung 26: Messzyklus für die Simulation

Der Zyklus besteht aus einer Reihe von einzelnen Tests, die sehr unterschiedliche Anforderungen an die Regelung stellen. Während der gesamten Simulation probiert der Referenz-Klient, gleichmäßig zuzugreifen, so dass entsprechende Messwerte erhoben werden können.

11.3 Vergleichsverfahren

Um die Ergebnisse der Simulation bewerten zu können, wird neben der POW Regelung ein Vergleichsverfahren simuliert. Hier wird ein typisches Überlastvermeidungssystem angenommen, welches nach Erreichen seiner Maximallast Anfragen probabilistisch ablehnt und so eine Überlast effektiv verhindert.

Die Simulation erfolgt dabei mit dem gleichen Messzyklus. In der Praxis dürfte der Vergleich etwas komplizierter ausfallen, da insbesondere nach einer Überlastsituation Klienten, welche zuvor nur eine Fehlermeldung erhalten haben, mit einem exponentialen Backoff wieder auf den Dienst zugreifen dürften. Die Simulation der eigentlichen Überlastsituation beeinflussen diese Effekte aber nicht. Nur das Abklingen der Überlast dürfte in der Praxis dadurch etwas langsamer erfolgen.

11.4 Bewertungsmaßstäbe

Das Ziel dieser Arbeit ist es, ein System zu entwerfen, welches eine faire Überlastvermeidung umsetzt. Um die Leistung der POW Regelung bewerten zu können, müssen aus diesem Ziel quantifizierbare Werte abgeleitet werden. Dabei werden die beiden Anforderungen, einerseits die Überlast zu vermeiden und andererseits dabei fair vorzugehen, mittels der beiden Werte Lastabweichung und Zugriffsratio getrennt erfasst.

11.4.1 Lastabweichung

Mit der Lastabweichung wird die Abweichung der realen Last vom Sollwert ermittelt. Diese Differenz zwischen Regelgröße und Führungsgröße wird über den gesamten Zeitraum kumuliert, indem die vorzeichenbereinigten Differenzen addiert werden. Es wäre möglich, die Abweichung noch über verschiedene Faktoren zu bewerten, um z. B. eine Überlast stärker zu gewichten als eine Unterlast. Auf diese Möglichkeit wird hier aber bewusst verzichtet, um die Bewertung nicht weiter zu verkomplizieren. Eine Ausnahme stellt die Unterschreitung der Last bei deaktiverter POW Regelung dar. Diese wird nicht mitkumuliert, da dieser Zustand keinem Mangel der Regelung entspricht, sondern bei geringer Last explizit erwünscht ist. Abbildung 27 verdeutlicht die Ermittlung der Lastabweichung.

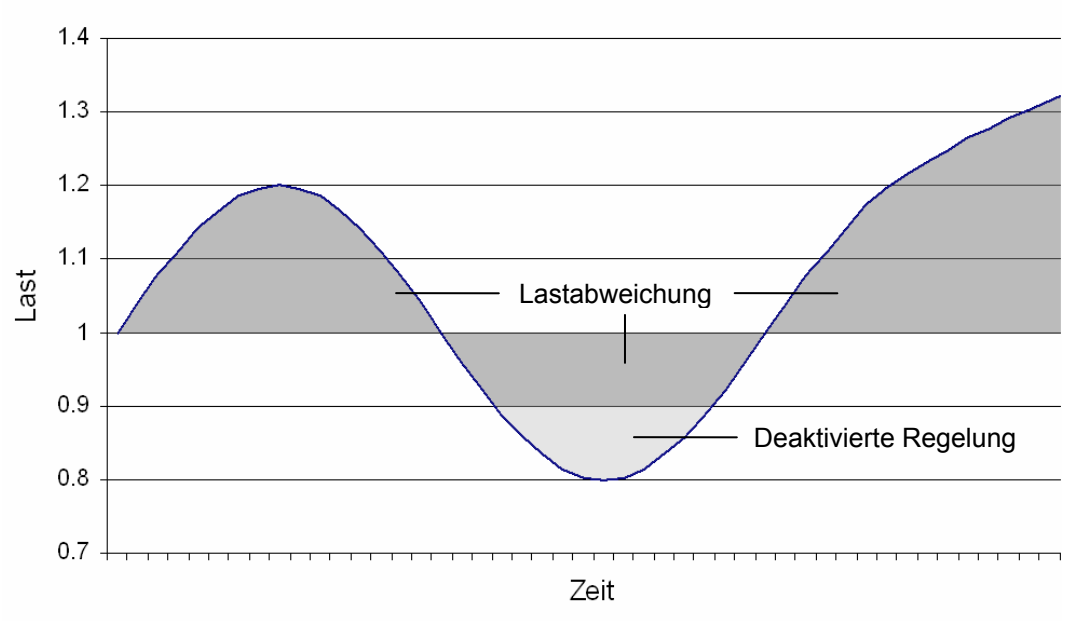


Abbildung 27: Ermittlung der Lastabweichung

11.4.2 Zugriffsratio

Unter Fairness wird verstanden, dass der legitime Klient eine maximale Möglichkeit hat, seine Anfragen beantwortet zu bekommen. Dies wird mit der Zugriffsratio in einen Messwert übersetzt, indem ermittelt wird, wie das Verhältnis erfolgreicher Zugriffe zu beabsichtigten Zugriffen ist. Das bedeutet, im optimalen Fall liegt das Verhältnis bei eins zu eins und der legitime Klient kann jeden seiner beabsichtigten Zugriffe durchführen. Die durch den Klienten verbrauchte Rechenzeit zur Lösung eventueller POW Funktionen wird hier nicht betrachtet, da davon ausgegangen wird, dass diese ansonsten brachliegen würde.

Bei diesem Messwert ist zu beachten, dass der absolute Wert nicht besonders viel aussagt. Durch die unterschiedliche Dimensionierung der Kapazitäten des Dienstanbieters und die Anzahl der Klienten kann er ganz unterschiedlich ausfallen. Ihre Aussagekraft bekommt die Zugriffsratio des legitimen Klienten in vergleichenden Messungen unterschiedlicher Überlastvermeidungstechniken.

11.5 Simulation

Nachdem alle Randbedingungen der Simulation geklärt sind, folgt die Simulation selber. Hierzu wurden sowohl die POW Regelung als auch das Vergleichsverfahren mit dem Messzyklus konfrontiert. Die POW Regelung wurde dabei mit den in Tabelle 5 wiedergegebenen Werten parametrisiert. Zur Bedeutung der einzelnen Werte wird auf Kapitel 10 verwiesen.

Parameter	Wert	Parameter	Wert	Parameter	Wert
interv_min	2000ms	off_limit	0,6	lower_limit	0,7
interv_factor	3	upper_limit	2	lower_factor	1,5
interv_stop	2	upper_count	5	i_factor	0,025
on_limit	0,8	upper_factor_big	0,4	Sollwert Last	100 Zugriffe/s
on_crit	500.000	upper_factor_small	0,8		

Tabelle 5: Parametrisierung der POW Regelung

Während der Simulation wurden die folgenden vier Werte für jedes Intervall erfasst:

- Serverlast (Abbildung 28): Die Anzahl an Anfragen pro Sekunde, welche der Server bearbeiten muss. So lange die POW Regelung inaktiv ist, werden hierfür alle Anfragen gezählt. Sobald die Regelung hingegen aktiv ist, zählen nur Anfragen mit gültigem POW.

- Zugriffe Referenz-Klient (Abbildung 29): Der Referenz-Klient versucht entsprechend einer intensiven Nutzung des Dienstes alle zwei Sekunden eine Anfrage zu senden. Dies entspricht einer Anfragelast von 0,5 Anfragen/s. Wie viele Anfragen davon erfolgreich abgearbeitet werden, gibt dieser Messwert an.

- Anzahl an Klienten (Abbildung 30): Die Anzahl der jeweils zugreifenden Klienten wird erfasst, so dass nachvollzogen werden kann, welchem Stadium des Messzyklus die jeweiligen Werte entsprechen.

- POW Kriterium (Abbildung 31): Dieser Wert entspricht dem aktuellen Kriterium des POW. Anhand dieses Wertes kann direkt beobachtet werden, wie die POW Regelung

das POW anpasst. Der Wert kann naturgemäß nur für die POW Regelung erfasst werden und nicht für das Vergleichsverfahren.

Die Simulation wurde mittels Java implementiert und ein Durchlauf beansprucht auf einem Rechner mit Java1.6.0 und Intel Core 2 Duo 2,2 GHz ca. 45 Sekunden. Im folgenden Abschnitt werden die Ergebnisse der Simulation analysiert und bewertet.

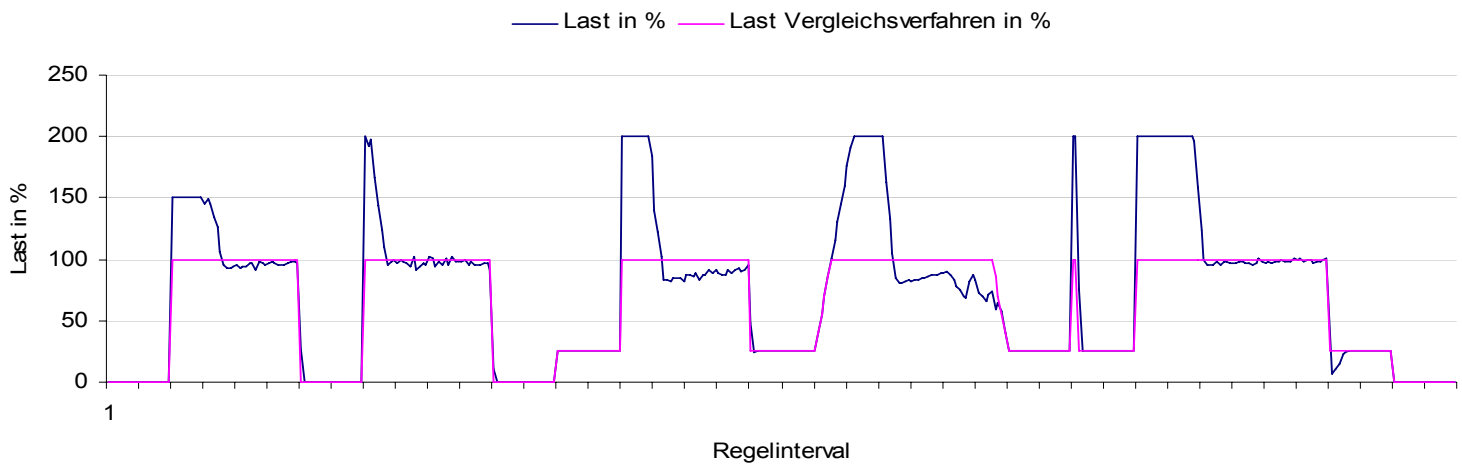


Abbildung 28: Serverlast

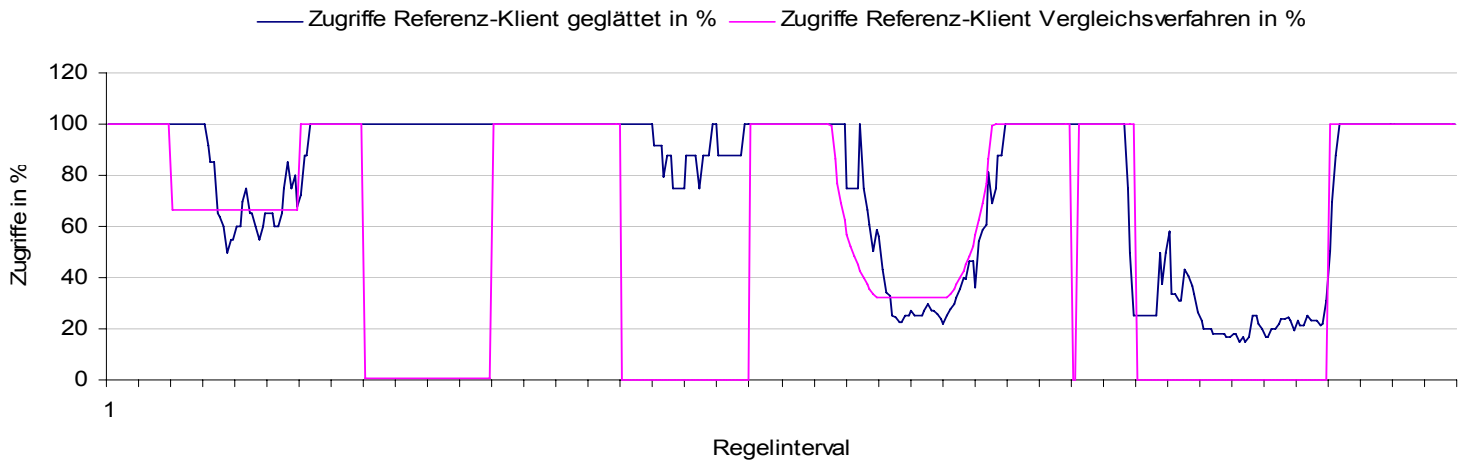


Abbildung 29: Zugriffe Referenz-Klient

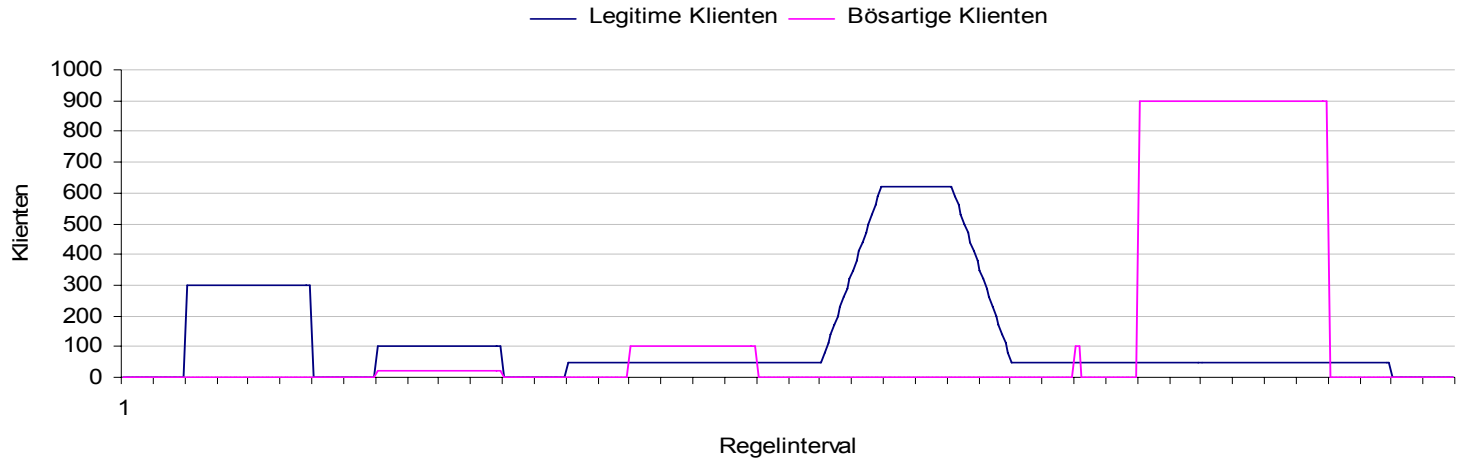


Abbildung 30: Anzahl Klienten

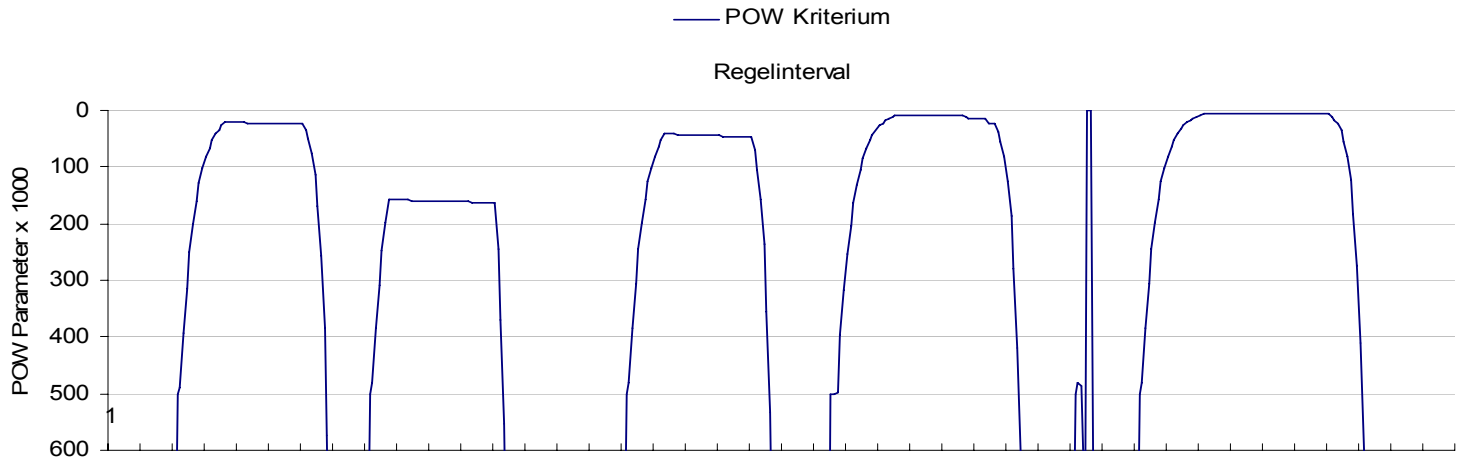


Abbildung 31: Proof-Of-Work Kriterium

11.6 Bewertung der Messergebnisse

Es wird in einem ersten Schritt der Verlauf der Serverlast betrachtet. Dieser stellt gewissermaßen die Pflicht dar, während die Zugriffe des Referenz-Klienten die Kür bilden. Verfahren, welche die Serverlast effektiv begrenzen, sind bekannt und sie arbeiten meistens sogar noch effektiver als die POW Regelung. Die Idee der fairen Überlastvermeidung besteht aber darin, in einem solchen Fall für den typischen Klienten trotzdem eine gewisse Servicequalität aufrechtzuerhalten, was durch die Zugriffsratio des Referenz-Klienten gemessen wird.

11.6.1 Bewertung der Serverlast

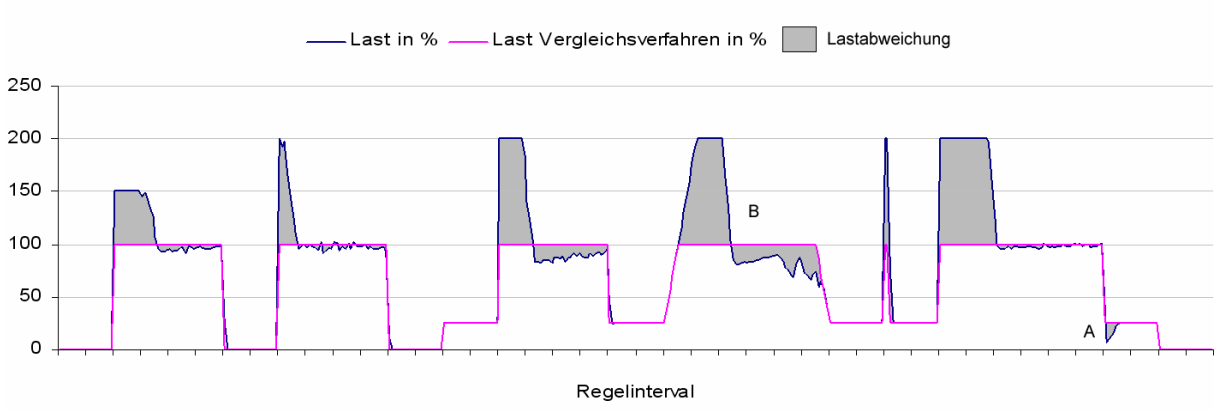


Abbildung 32: Serverlast mit Markierung der Lastabweichung

Bei der Analyse der gemessenen Serverlast ist ersichtlich, dass die An- und Abschaltung der POW Regelung gut funktioniert. Sofort nachdem die Last anliegt, wird die POW Regelung aktiviert. Lediglich die Abschaltung nach besonderen Lastspitzen erfolgt mit minimaler Verzögerung, was ein leichtes Übersteuern nach unten bewirkt. Dies ist insbesondere an Punkt A in Abbildung 32 sichtbar. Da dieser Fall einer in minimaler Zeit in sich zusammenfallender Zugriffsintensität in der Praxis eher selten anzutreffen sein dürfte und darüber hinaus ein ständiges An- und Abschalten der Regelung vermieden werden soll, kann dieses leichte Übersteuern toleriert werden.

Die Schnellanpassung bringt die Ausregelung in kleineren bis mittleren Überlastsituationen schnell in ihren eigentlichen Arbeitsbereich. Gewisse Schwächen sind bei starker Überlast zu erkennen. Hier könnte die Schnellanpassung noch kurzfristiger arbeiten. Ein einfaches Erhöhen des entsprechenden Parameters ist aber nur begrenzt hilfreich, da sonst die Anpassung in kleineren Überlastsituationen leidet. Eventuell könnte man hier mittels einer Schnellanpassung durch größenabhängige Parameter weitere Verbesserungen erzielen. Das

plötzliche Zuschalten sehr vieler Klienten mit minimaler Anlaufzeit dürfte in der Praxis aber nicht ganz so abrupt erfolgen, so dass die Schnellanpassung hier etwas mehr Zeit haben dürfte, um zu reagieren.

Die Ausregelung funktioniert bei gleich bleibenden Anfrageverhältnissen sehr gut. Die Last erreicht ihren Sollwert und dieser wird mit minimalen Abweichungen gehalten. Dabei spielt das absolute Lastniveau keine Rolle. Wie schnell der Sollwert erreicht wird, hängt dabei vor allen Dingen davon ab, wie nah die Last am Sollwert liegt, wenn die Übergabe der Regelung durch die Schnellanpassung erfolgt. Eine gewisse Schwäche offenbart die Ausregelung im Umgang mit dem hoch dynamischen Teil des Erprobungszyklus (Punkt B). Hier gelingt es der Ausregelung nicht, die Last optimal nachzuführen. Die Last entfernt sich teilweise vom Sollwert und die Schnellanpassung greift ein.

Das Vergleichsverfahren hingegen hält die Last dauerhaft auf dem Sollwert und stellt aus der Sicht der reinen Überlastvermeidung quasi den Optimalfall dar. Numerisch bedeutet dies für den Messzyklus eine Lastabweichung der Serverlast von ca. 17,4 Prozentpunkten für die POW Lastregelung und nahezu null Prozentpunkten für das Vergleichsverfahren.

11.6.2 Bewertung der Zugriffe des Referenz-Klienten

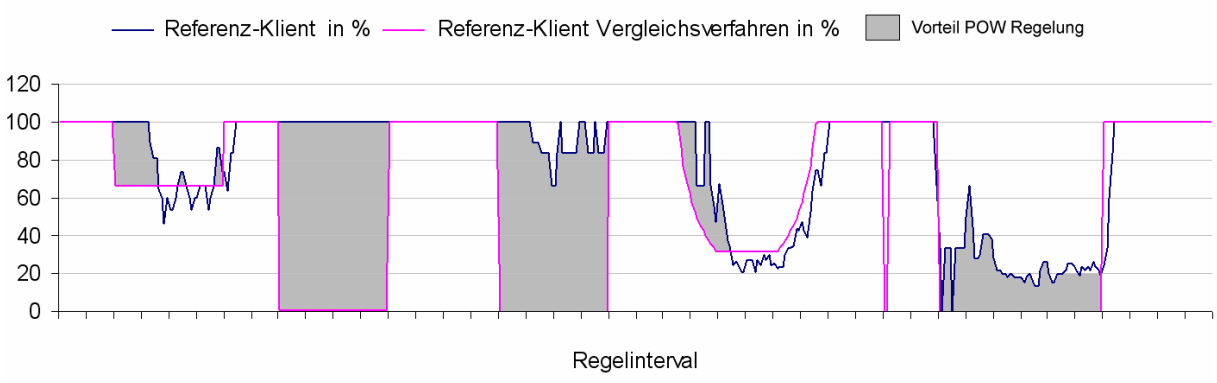


Abbildung 33: Zugriffe des Referenz-Klienten mit Markierung des POW Vorteils

Die Zugriffe des Referenz-Klienten stellen die eigentliche Messlatte dar, an der sich die POW Überlastregelung messen lassen muss. Das Ziel ist es, den Dienst trotz der unterschiedlichen Überlastsituationen für den normalen Klienten, welcher in der Messung durch den Referenz-Klienten repräsentiert wird, verfügbar zu halten. Der optimale Fall sieht so aus, dass dessen Zugriffe uneingeschränkt weiter möglich sind. Dieses Ziel wird im Messzyklus nicht erreicht. Die POW Regelung arbeitet zeitweise mit einem POW, welches es dem Klienten nicht ermöglicht, die volle Anzahl an Anfragen abzusetzen. Es wäre

selbstverständlich möglich, diesen Zustand durch veränderte Parameter der Anfragstruktur herbeizuführen, aber dies ist nicht das Ziel der Messung. Es sollen sowohl Leistungsfähigkeit des Systems als auch dessen Grenzen aufgezeigt werden. Obwohl der Referenz-Klient Einschränkungen unterworfen ist, kann festgestellt werden, dass der Service für den Klienten trotzdem zu jeder Zeit verfügbar bleibt. Im Normalfall wird von einer Anfrage alle zwei Sekunden ausgegangen und selbst in der ungünstigsten Situation kann immerhin noch eine Anfrage etwa alle zehn Sekunden abgesetzt werden. In der Praxis dürfte dies ausreichen, um den Dienst des Servers, z. B. ein Online-Buchungssystem, noch nutzen zu können.

Numerisch bedeutet dies, dass der Referenz-Klient während des Messzyklus mit der POW Regelung 79,1 % seiner Anfragen beantwortet bekommt, während es im Vergleichsverfahren lediglich 56,4 % sind.

Darüber hinaus wird sichtbar, dass die POW Regelung aus der Sicht des Referenz-Klienten besonders gut funktioniert, wenn bösartige Klienten den Dienst beanspruchen. Diese gehen deutlich aggressiver vor und verschaffen sich so einen Vorteil. Dieser Vorteil wird durch die POW Regelung zunichtegemacht, da nur die Abarbeitungsgeschwindigkeit der POW Berechnung über die Zugriffe entscheidet und nicht die Aggressivität, mit der Anfragen gestartet werden. Konkret bedeutet dies, dass zeitweise alle Anfragen im Fall der POW Regelung abgearbeitet werden, während es im Fall des Vergleichsverfahrens beinahe gar keine mehr sind. In einem Überlastszenario, an dem lediglich andere legitime Klienten beteiligt sind, fällt der Vorteil der POW Regelung hingegen geringer aus.

11.7 Zusammenfassung Simulation

Es wurde ein Verfahren entwickelt, das die Berechnungsdauern einzelner HashCashLin Funktionen unter Einhaltung der realen Verteilung per Zufallsgenerator erzeugt. Das Verhalten der Klienten wurde bestimmt und ein darauf aufbauender Messzyklus entworfen. Um die Ergebnisse der Simulation bewerten zu können, wurde das Vergleichsverfahren beschrieben und die Bewertungsmaßstäbe wurden festgelegt. Daraufhin erfolgte die eigentliche Simulation, bei der die abgebildeten Messwerte ermittelt wurden. Die Analyse ergab, dass die Serverlast durch die POW Regelung ausreichend kontrolliert wurde, aber das Vergleichsverfahren erwartungsgemäß noch bessere Ergebnisse liefert. Bei der Zugriffsratio des Referenz-Klienten hingegen schnitt das POW Verfahren deutlich besser ab als das Vergleichsverfahren.

Dieses Kapitel hat gezeigt, dass die POW Lastregelung auf ein großes Überlastszenario anwendbar ist und dabei eine deutlich größere Fairness aufweist, als das auf Abweisungen von Klienten basierende Vergleichsverfahren.

12 Prototyp

Mittels der vorangegangenen Simulation wurde gezeigt, dass die POW Regelung auch in größeren Szenarien Überlast fair begrenzen kann. In diesem Kapitel soll darüber hinaus mittels eines Prototyps demonstriert werden, wie die Implementierung eines solchen Systems ganz konkret in einer realen Serverarchitektur umgesetzt werden kann.

Die Aufgabe des Prototyps ist es, die Last auf einem HTTP basierten Dienst, wie z. B. einem Online-Buchungssystem, auf eine definierte Anzahl an Zugriffen pro Zeiteinheit zu begrenzen. Dabei soll die Lastregelung als unabhängige Komponente vor die eigentliche Anwendung geschaltet werden, damit sie ohne Anpassungen für weitere Anwendungen eingesetzt werden kann.

Im Anschluss wird einerseits beschrieben, wie die serverseitige POW Regelung implementiert und integriert wird, und andererseits, wie die passende Software auf der Seite des Klienten aussieht, welche das Proof-Of-Work berechnet. Im Anschluss werden der für die Erprobung des Prototyps verwendete Versuchsaufbau sowie die dabei gewonnenen Messwerte behandelt.

12.1 Server

Als HTTP-Server wird der Java Servlet Container Tomcat verwendet. Um eine möglichst einfache Integration in bestehende Systeme zu ermöglichen, wird die eigentliche POW Regelung als so genannter Servlet-Filter [FILTER] realisiert. Das bedeutet, jede Anfrage wird unsichtbar für die eigentliche Anwendung durch diesen Filter geleitet, welcher die Möglichkeit hat, die Anfrage weiterzuleiten, zu verändern oder gänzlich umzuleiten. Diese Funktionen macht sich die POW Regelung zunutze, indem beim Weiterleiten der Anfragen die Messwerte zur Last erhoben werden, die später als Regelgröße dienen. Sobald die POW Regelung aktiviert ist, werden Anfragen ohne gültiges POW auf eine entsprechende Seite weitergeleitet, welche die Berechnung auslöst. Anfragen mit gültigem POW werden dagegen direkt zur eigentlichen Anwendung weitergeleitet. Abbildung 34 verdeutlicht den beschriebenen Aufbau.

Die Weiterleitung zur POW Berechnung ist als gewöhnliche HTTP-Weiterleitung implementiert und durch die HTML-Seite, welche die POW Berechnung auslöst, werden dem Klienten sowohl das aktuelle Token als auch der aktuelle Parameter für die HashCashLin Berechnung mitgeteilt.

Um das HashCashLin POW zu regeln, verfügt der POW Filter über den weiter oben beschriebenen dreischichtigen Regler. Die Parameter der Regelung können über eine HTML basierte Administrationsoberfläche eingestellt werden. Darüber hinaus wurde in den Filter eine Logging Funktionalität integriert, so dass Messwerte zum jeweiligen Betriebszustand der Regelung erfasst werden können.

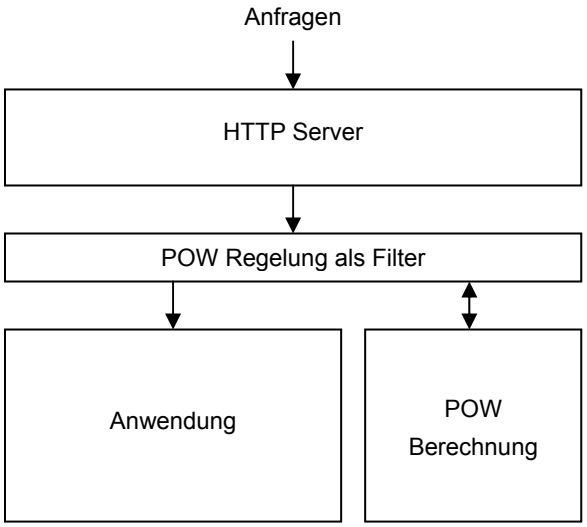


Abbildung 34: Serverseitiger Aufbau des Prototyps

12.2 POW Klient

Die eigentliche Berechnung bzw. die Suche der passenden POW Lösung muss auf der Hardware des Klienten erfolgen. Optimal wäre hier ein passendes Browser Plug-In. Eine solche komplett native Lösung bietet die beste Performance und Integration in den Browser. Leider steht momentan keine passende Lösung zur Verfügung. Auf Grund der vielen praktisch benutzen Betriebssysteme und Browser wäre die Entwicklung einer entsprechenden POW Lösung mit sehr viel Aufwand verbunden. Darüber hinaus müsste das Plug-In von jedem teilnehmenden Benutzer separat installiert werden.

Somit müssen andere Lösungen gefunden werden, wie die POW Regelung mit aktuellen Browsern umgesetzt werden kann. An dieser Stelle bieten sich drei Technologien an: Java Skript, Java Applets und Flash Aktion Skript. Zur Auswahl der passenden Technologie sind insbesondere deren praktische Verbreitung, die Leistungsfähigkeit der SHA-1 Berechnung sowie die jeweilige Lizenzsituation entscheidend.

12.2.1 JavaSkript

Ursprünglich als Erweiterung für HTML-Webseiten gedacht, hat sich JavaSkript [ISO16262] zu einer mächtigen Softwareplattform entwickelt. Nachdem anfangs nur einzelne Beispielprogramme ausgetauscht wurden, existieren inzwischen umfangreiche Bibliotheken wie z. B. die Google Maps API [MAPS], mit deren Hilfe interaktive Anwendungen im GIS-Bereich entwickelt werden können. Ein großer Vorteil von JavaScript ist die sehr weite Verbreitung. Eine in jeder Hinsicht verlässliche Statistik zur Verbreitung ist nicht verfügbar, aber es kann davon ausgegangen werden, dass weit über 90 % der Klienten mit JavaScript ausgestattet sind und diese Funktion auch aktiviert haben [JSa] [JSb]. Für die Verwendung von JavaScript sind darüber hinaus keine kostenpflichtigen Werkzeuge erforderlich.

Paul Johnston stellt eine Implementierung der SHA-1 Funktion in JavaScript als Open Source Projekt zur Verfügung [JOHN02], welche für die HashCashLin Berechnung genutzt werden kann.

Abschließend wurden Messungen zur Leistungsfähigkeit der SHA-1 Implementierung durchgeführt. Die Ergebnisse sind in Tabelle 6 zu sehen.

Hardware	JS Interpreter	SHA-1 Implementierung	Leistung
Intel Pentium 1,6GHz	WinXp, Firefox 2.0	Paul Johnston	402 SHA-1/s
Intel Pentium 1,6GHz	WinXp, IE6 6.0	Paul Johnston	390 SHA-1/s
Intel Core Duo 2,2 GHz	WinXp, Firefox 2.0	Paul Johnston	678 SHA-1/s
Intel Core Duo 2,2 GHz	WinXp, IE 6.0	Paul Johnston	544 SHA-1/s

Tabelle 6: Messergebnisse zur JavaScript SHA-1 Leistung

Leider sind die Messergebnisse mehr als enttäuschend. Verglichen mit den Messungen in Kapitel 8.7 sind mit der Java Implementierung z. B. auf dem Core Duo 2,2GHz 1.438.228 SHA-1 Berechnungen pro Sekunde möglich. In der JavaScript Variante werden mit dem gleichen Rechner nur 678 SHA-1/s erreicht, das bedeutet, die JS Implementierung ist um einen Faktor von ca. 2.100 langsamer. Eine deutliche Beschleunigung der Implementierung von Paul Johnston erscheint mit den von JavaScript gebotenen Sprachmitteln nicht möglich.

Dabei muss beachtet werden, dass die JavaScript Messung lediglich in einem Prozess ablief, aber selbst im Vergleich zur Implementierung mit nur einem Prozess fällt die JavaScript Variante so deutlich ab, dass sich der Einsatz einer JavaScript POW Klienten-

Lösung verbietet. Jeder Angreifer könnte das POW System leicht aushebeln, indem er eine leistungsfähigere POW Implementierung einsetzt und sich so einen mehr als tausendfachen Vorteil verschafft.

12.2.2 Flash ActionScript

Die zweite Möglichkeit, das Proof-Of-Work auf der Seite des Klienten berechnen zu lassen, stellt die Flash-Plattform [FLASH] mit ihrer Programmiersprache ActionScript dar. Klassischerweise wird Flash nicht mit der Anwendungsprogrammierung in Verbindung gebracht, sondern auf Grund der historischen Herkunft auf eine Autorenlösung für interaktive Grafik reduziert. Nichtsdestotrotz hat sich die integrierte Sprache ActionScript von einer grafischen Möglichkeit, Makros zu erstellen, zu einer praktisch vollwertigen Programmiersprache entwickelt. Seit der Version 9 verfügt der Flash Player auch über einen Just In Time (JIT) Compiler, welcher eine deutlich verbesserte Performance erwarten lässt.

Die Verbreitung des Flash Plug-Ins liegt laut dem Hersteller Adobe [ADOBE] bei nahezu hundert Prozent. Selbst wenn man der Statistik des Herstellers naturgemäß skeptisch gegenübertreten sollte, erscheint ein Wert deutlich über 90 % realistisch, da immer mehr beliebte Anwendungen, wie z. B. die aktuell stark wachsenden Online-Videoplattformen, auf den Flash-Player setzen.

Aufbauend auf dem JavaScript Ansatz von Paul Johnston existierte eine freie SHA-1 Implementierung in ActionScript Version 3 von Henri Torgemane [TORG07]. Diese wurde verwendet, um die Leistungsfähigkeit der Flash Plattform zu testen. Tabelle 7 zeigt das Ergebnis dieser Messungen.

Hardware	JS Interpreter	SHA-1 Implementierung	Leistung
Intel Pentium 1,6GHz	WinXp, Flash 9.0	Henri Torgemane	5.944 SHA-1/s
Intel Core Duo 2,2 GHz	WinXp, Flash 9.0	Henri Torgemane	8.808 SHA-1/s

Tabelle 7: Messergebnisse zur ActionScript SHA-1 Leistung

Die Ergebnisse zeigen gegenüber JavaScript um etwas mehr als eine Größenordnung bessere Ergebnisse. Die Flash Laufzeitumgebung lief im Test wie der JavaScript Interpreter nur auf einem Prozessorkern. Trotz der gesteigerten Leistung liegt die Flash Implementierung der SHA-1 Funktion immer noch knapp zwei Größenordnungen hinter den Messwerten der Java Variante mit einem Thread. Darüber hinaus verlangt der

Hersteller Adobe für gewisse Teile der Entwicklungsumgebung Lizenzgebühren, so dass insgesamt der Einsatz von Flash als POW Klient nicht sinnvoll erscheint.

12.2.3 Java Applet

Die dritte Variante, das POW auf der Klientenseite berechnen zu lassen, stellt ein Java Applet dar. Die Java Plattform weist verglichen mit den beiden anderen Alternativen eine deutlich geringe Verbreitung auf. Generell kann mit nur 50 % bis 60 % gerechnet werden. Dafür überzeigt die Leistungsfähigkeit der SHA-1 Berechnung, die zumindest bei der Verwendung des Java Browser Plug-In [PLUG] so gut ausfällt, wie die weiter oben im Rahmen der Erprobung der HashCashLin Funktion gemessenen Werte. Darüber hinaus fallen sowohl für das Plug-In als auch für die Java Entwicklungsumgebung keinerlei Lizenzgebühren an. Aus diesen Gründen wird im Folgenden der Kliententeil des Prototyps in Form eines Java Applets umgesetzt.

12.2.4 Implementierung des Klienten

Weil die Java Plattform die Anforderungen am besten erfüllt, wird der Klient als Java Applet implementiert. Dieses Applet berechnet die geforderten Tokens und zeigt dem Benutzer den Verlauf über einen Fortschrittsbalken an. Sobald durch das Applet die vom Server geforderte Lösung gefunden wurde, wird der Browser wieder auf die ursprüngliche URL weitergeleitet, wobei die Lösung als GET Parameter angefügt wird. Dies kann in sehr seltenen Fällen dazu führen, dass eine bestehende Anwendung nicht ohne Änderungen mit diesem Verfahren kompatibel ist, wenn die Anwendung mit zusätzlichen GET Parametern nicht umgehen kann. Alternativ wäre es auch denkbar, die Lösung als separate HTTP Header-Zeile zu übertragen. Diese elegantere Variante würde aber einen Zugriff des Applets auf den Header bedingen, welcher nicht ohne weiteres gegeben ist.

Die konkrete Implementierung wurde entsprechend dem ursprünglichen Java1.0 Sprachstandard durchgeführt, da keine neueren Funktionen zwingend benötigt wurden und so eine maximale Kompatibilität auch mit älteren Browsern gegeben ist. Eine erneute Implementierung der HashCashLin Funktion war nicht notwendig, da die Java Implementierung, welche bereits zu Messzwecken erstellt wurde, genutzt werden konnte. Abbildung 35 zeigt einen Screenshot der Applet-Oberfläche.

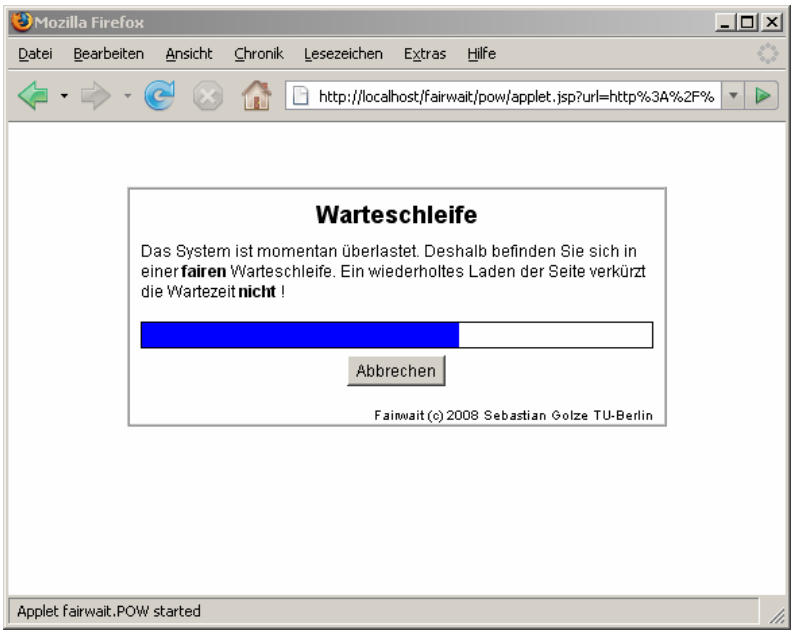


Abbildung 35: Screenshot der Applet-Oberfläche

12.2.5 POW Forschrittsanzeige

Ein auf den ersten Blick trivialer, aber doch sehr wichtiger Teil der POW Berechnung auf der Klientenseite ist die Fortschrittsanzeige. Im Fall der POW Berechnung entsteht für den Benutzer schnell der Eindruck, sein Rechner bzw. Browser sei abgestürzt. Deshalb muss dem Benutzer möglichst intuitiv kommuniziert werden, dass dem nicht so ist. Die einfachste Lösung wäre hier z. B. ein sich bewegendes Symbol wie bspw. die bekannte Sanduhr. Mit einer solchen Variante zeigt man dem Benutzer, dass der Rechner weiterhin arbeitet. Er erhält aber keine Information, wie lange der Prozess noch dauern wird. Die elegantere Lösung ist hier ein Fortschrittsbalken. Dieser signalisiert einerseits, dass das Programm arbeitet, und gibt andererseits einen Ausblick über die noch zu erwartende Laufzeit. Unglücklicherweise findet man viele Fortschrittsbalken, die alles andere als optimal arbeiten. Ein guter Fortschrittsbalken sollte kontinuierlich, entsprechend dem Fortgang der Anwendung durchlaufen, ohne länger stehen zu bleiben oder neu initialisiert zu werden.

Im Fall der POW Berechnung ist ein solcher Balken nur schwer umzusetzen, da die exakte Berechnungsdauer systembedingt bei der Berechnung von POW Funktionen nicht im Voraus bekannt ist. Der einfachste Ansatz, den Balken nur bis zum Erwartungswert für die Berechnungsdauer laufen zu lassen, bedeutet, dass der Balken praktisch oft bei 100 % steht, während die POW Suche noch nicht beendet ist. Aus diesem Grund muss eine bessere Lösung gefunden werden. Sinnvoller ist es, über dem Balken die Wahrscheinlichkeit

anzuzeigen, zum aktuellen Zeitpunkt eine Lösung gefunden zu haben. Das bedeutet zwar, dass die Suche schon oft beendet ist, bevor der Balken nahe bei hundert Prozent steht, aber dieses Verhalten wird durch den Benutzer akzeptiert und auf der anderen Seite ist so sichergestellt, dass sich der Balken bis zum Ende der Suche dauerhaft bewegt, ohne vorzeitig das Ende zu erreichen.

In der Praxis ist es ausreichend, den Fortschrittsbalken nur in gewissen Zeitintervallen zu bewegen. Ein Update nach jeder Überprüfung einer möglichen POW Lösung wäre sehr ineffizient und man würde weit mehr Leistung für den Fortschrittsbalken aufwenden als für die Berechnung selber. Nichtsdestotrotz besteht das Problem, bei jedem Update des Balkens die Wahrscheinlichkeit ermitteln zu müssen, mit welcher bisher eine Lösung gefunden wurde. Dies ergibt sich aus der weiter oben gezeigten Formel:

$$w' = 1 - ((1-p)^n)$$

w': die Wahrscheinlichkeit, nach n Versuchen mindestens eine Lösung zu erhalten
p: Wahrscheinlichkeit, bei einem Versuch eine valide Lösung zu entdecken
n: Anzahl der Versuche

Da n praktisch sehr große Werte annimmt, ist die Berechnung von w' mit einem nicht unerheblichen Aufwand verbunden. Ziel ist es aber, den Fortschrittsbalken mit einem minimalen zusätzlichen Aufwand umzusetzen. Als Alternative kann auf eine Eigenschaft der Verteilungsfunktionen zur POW Berechnungsdauer zurückgegriffen werden. Im Kapitel zu den statistischen Eigenschaften der HashCashLin Funktion wurden die Verteilungsfunktionen unterschiedlicher Berechnungsdauern dargestellt, indem die X-Achse relativ zum Erwartungswert der Funktion skaliert wurde. Es fällt dabei auf, dass sich die Kurven bereits bei sehr einfachen Berechnungen stark annähern. Abbildung 36 verdeutlicht diesen Effekt.

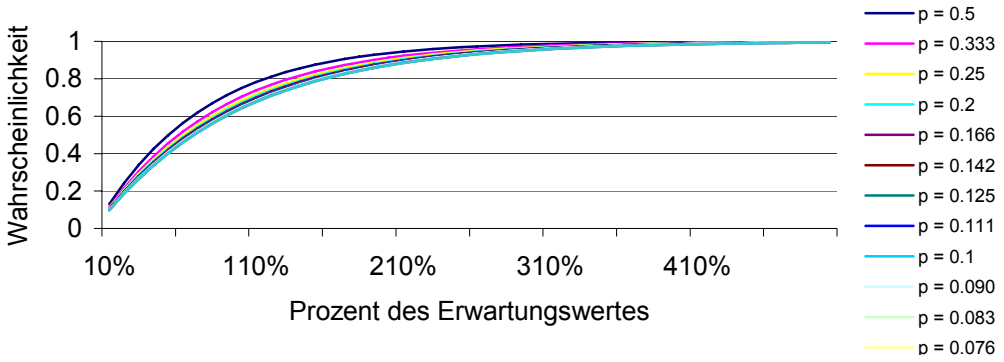


Abbildung 36: Verteilungsfunktionen HashCashLin

Der Abbildung kann entnommen werden, dass es für praktisch alle HashCashLin POW Berechnungen mit realistischen Schwierigkeiten legitim ist, die Wahrscheinlichkeit, mindestens eine Lösung gefunden zu haben, in eine direkte Relation zum Erwartungswert zu setzen. Das bedeutet für die praktische Implementierung des Fortschrittsbalkens, dass mit einer im Voraus berechneten Wertetabelle gearbeitet werden kann. Die Tabelle enthält dabei die Wahrscheinlichkeiten in Abhängigkeit vom Erwartungswert. Durch diesen Mechanismus kann der Fortschrittsbalken sehr effizient implementiert werden und zeigt dem Benutzer einen aussagekräftigen Wert bezüglich der laufenden POW Berechnung an.

12.3 Erprobung

Die Funktion des Prototyps wurde mittels eines Versuchsaufbaus überprüft. Das Ziel dieser Erprobung ist es nicht, die Parametrisierung der Regelung oder das Klientenverhalten im Detail zu untersuchen. Dazu eignet sich die weiter oben beschriebene Simulation deutlich besser. Es geht vielmehr darum zu zeigen, dass der entwickelte Prototyp mit der gegebenen Parametrisierung voll funktionsfähig ist und wie ein praktischer Betrieb der fairen POW Lastregelung aussehen kann.

12.3.1 Versuchsaufbau

Für die Erprobung wurde ein Netz mit den im Folgenden beschriebenen Randbedingungen verwendet. Ein Rechner fungiert als Server. Dieser verfügt einerseits über eine Implementierung der POW Regelung sowie zu Vergleichszwecken über eine Implementierung des weiter oben beschriebenen Vergleichsverfahrens, das auf der Abweisung von Anfragen basiert. Alle Messungen werden jeweils mit beiden Varianten durchgeführt, um zu untersuchen, wie sich die POW Regelung im Vergleich zu diesem Standardansatz verhält.

Neben dem Server werden acht hardwaretechnisch identische Klienten (AMD Athlon 64 X2 4200x, 2 GB RAM) eingesetzt. Alle Rechner sind per 1 GBit Ethernet mit dem Server verbunden. Die Klienten können entsprechend dem Messzyklus in ihrem Anfrageverhalten konfiguriert werden. Die Leistungsfähigkeit des Servers beträgt angenommene zwei Anfragen pro Sekunde. Dieser Wert liegt unter dem realen Leistungslimit des verwendeten Servers. Die Einschränkung auf diesen Wert soll dafür sorgen, dass mit acht Klienten das Leistungslimit deutlich überschritten werden kann und nicht andere Faktoren wie z. B. die Bandbreite des Netzwerkes einen Flaschenhals darstellen. Darüber hinaus ist so die Dokumentation der Ergebnisse serverseitig möglich, ohne dass der Messeingriff die Ergebnisse signifikant verfälscht.

12.3.2 Messzyklus

Bezogen auf den Messzyklus wird auf den weiter oben angestellten Überlegungen bezüglich der Simulation aufgebaut. Erste Probeläufe haben aber gezeigt, dass die Komplexität des in der Simulation genutzten Messzyklus mit acht Klienten nicht sinnvoll nachgestellt werden kann. Aus diesem Grund wird für die Erprobung des Prototyps ein vereinfachter Messzyklus verwendet. Abbildung 37 zeigt dessen Verlauf.

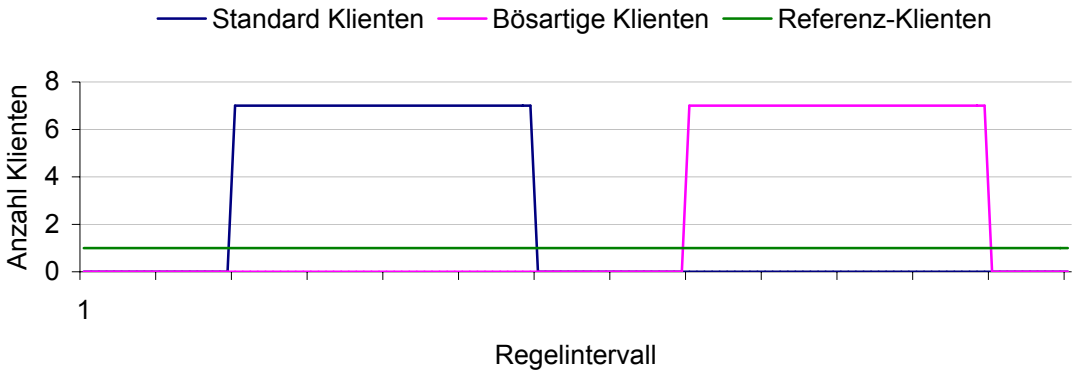


Abbildung 37: Messzyklus zur Erprobung des Prototypen

Die Gestaltung des Messzyklus zielt darauf ab, zu erproben, wie sich das System in einer Überlastsituation verhält, die einerseits durch legitime Klienten hervorgerufen wird und andererseits durch die Zugriffe von bösartigen Klienten entsteht. Die Leistungsfähigkeit des Servers ist mit zwei Zugriffen pro Sekunde definiert. Legitime Klienten greifen alle zwei Sekunden zu, während bösartige Klienten so oft zugreifen, wie es ihnen möglich ist. Dabei bleibt zu beachten, dass es in der Praxis möglich wäre, bösartige Klienten zu konstruieren, welche noch etwas intensiver zugreifen als die hier verwendeten.

12.3.3 Vergleichsverfahren und Bewertungsmaßstäbe

Bezüglich des verwendeten Vergleichsverfahrens wird analog zur Simulation vorgegangen. Auch in diesem Fall erfolgt der Vergleich mit einem System, welches die Überlast durch Abweisen der Klienten kontrolliert. Die Bewertungsmaßstäbe werden ebenfalls von der Simulation übernommen, so dass auch in diesem Fall Lastabweichung und Zugriffsratio untersucht werden. Weitere Details zu diesen Aspekten finden sich in Kapitel 11.

12.3.4 Messungen

Sowohl die POW Regelung als auch das Vergleichsverfahren wurden mittels des Messzyklus getestet. Dabei wurden analog zu den Messungen in Kapitel 11 die vier Werte Serverlast, Zugriffe Referenz-Klient, Anzahl Klienten sowie das POW Kriterium für jedes Intervall erfasst. Abbildung 38 bis Abbildung 41 zeigen die Ergebnisse dieser Messungen. Weitere Details zur Bedeutung der Messwerte finden sich in Kapitel 11.5.

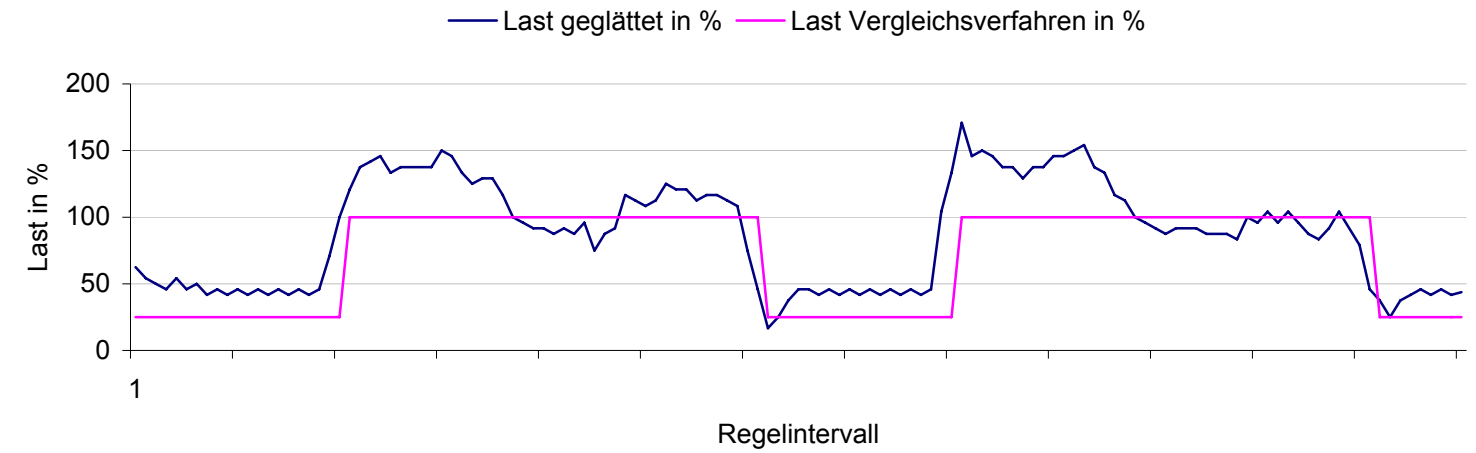


Abbildung 38: Serverlast

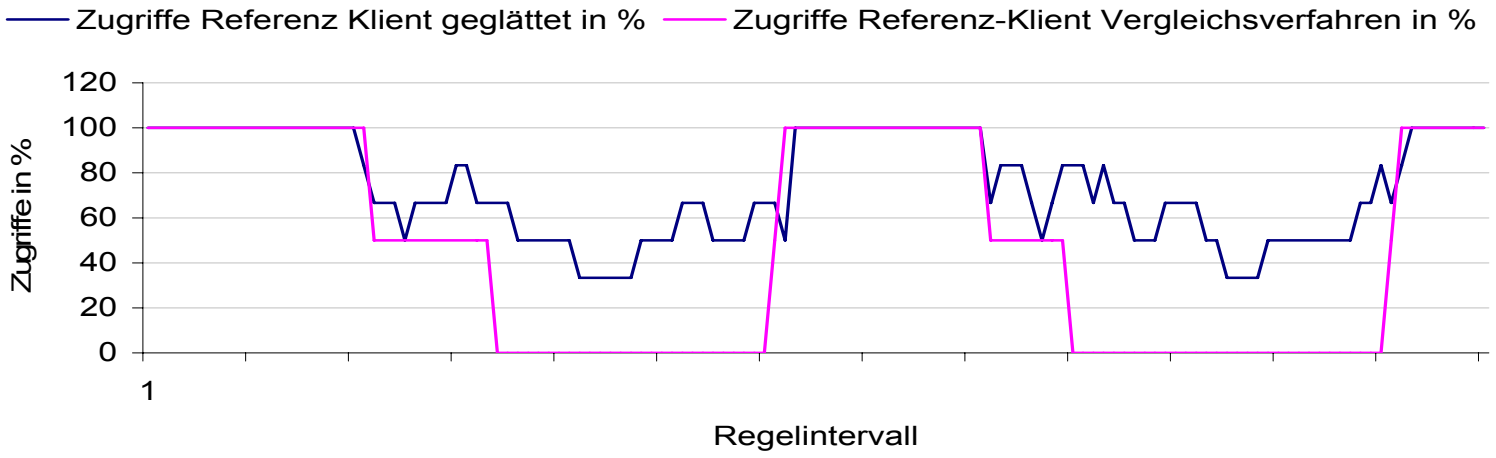


Abbildung 39: Zugriffe Referenz-Klient

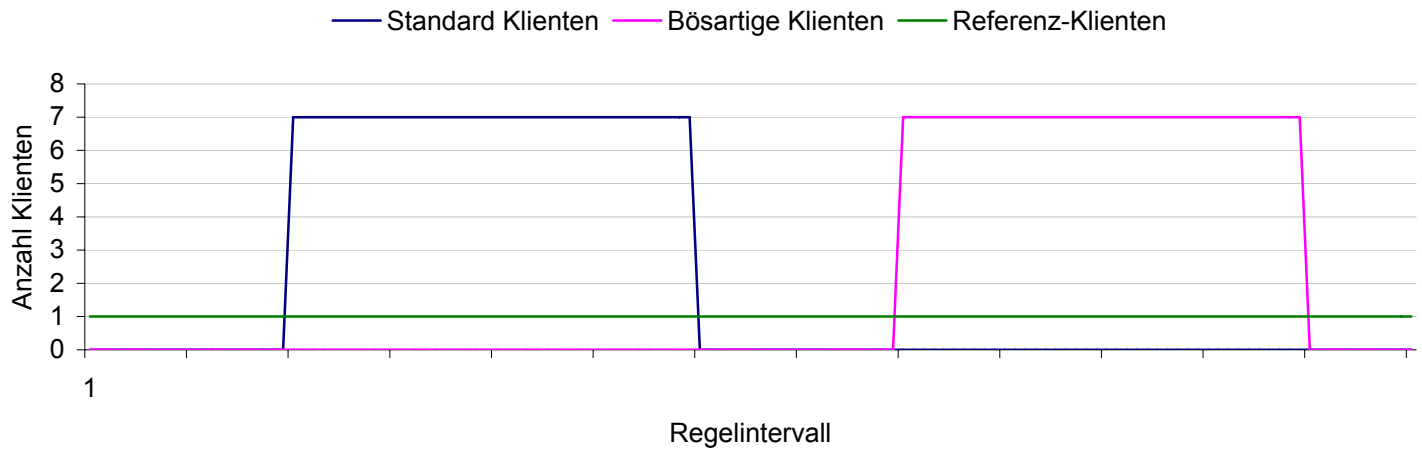


Abbildung 40: Anzahl Klienten

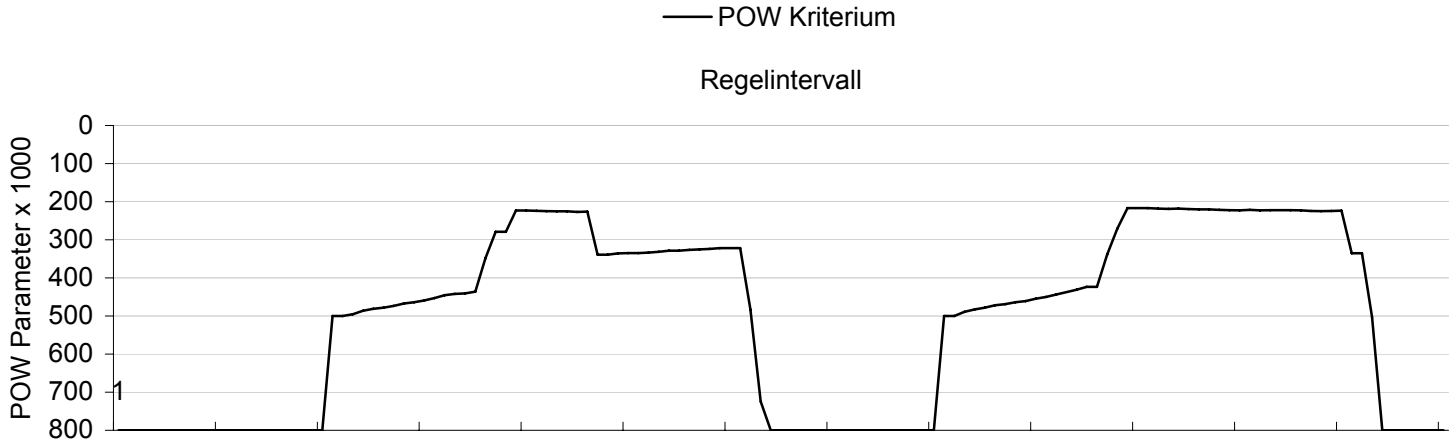


Abbildung 41: Proof-Of-Work Kriterium

12.3.5 Bewertung der Messergebnisse

Bei der Bewertung der Ergebnisse werden die Serverlast sowie die Zugriffsratio des Referenz-Klienten betrachtet.

Bezüglich der Serverlast ist bereits in den Abbildungen zu erkennen, dass die POW Lastregelung die Überlast effektiv begrenzt. Der Verlauf weicht aber in einem gewissen Rahmen vom idealen Lastverlauf des Vergleichsverfahrens ab. Numerisch bedeutet dies, dass über den gesamten Messzyklus eine Abweichung von 13,5 Prozentpunkten für die POW Lastregelung beobachtet wird, während das Vergleichsverfahren hier eine Abweichung von praktisch null Prozentpunkten erreicht.

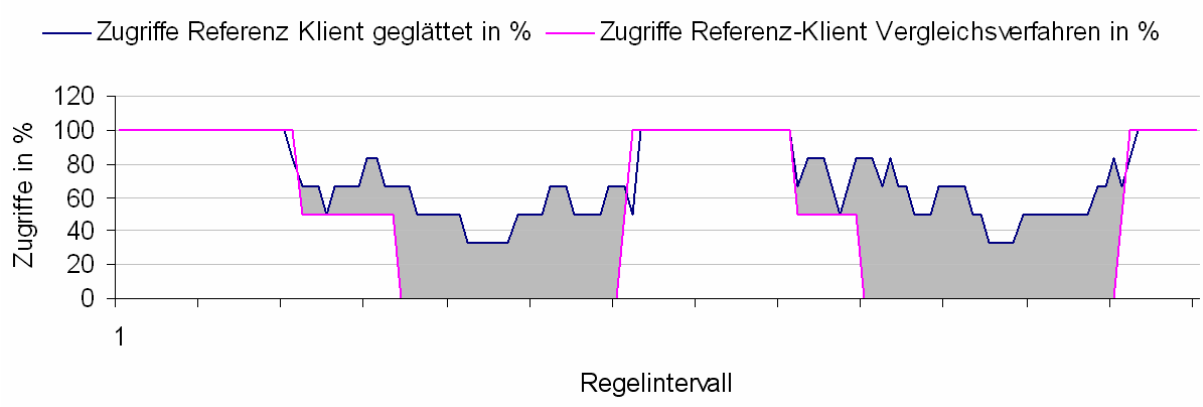


Abbildung 42: Zugriffe des Referenz-Klienten mit Markierung des POW Vorteils

Bezüglich der Zugriffe des Referenz-Klienten verdeutlicht Abbildung 42 die Ergebnisse. In den beiden Überlastphasen wird der Service mit gewissen Einschränkungen für den Referenz-Klienten stabil aufrechterhalten. Bei Anwendung des Vergleichsverfahrens gehen die Zugriffe des Referenz-Klienten hingegen gegen null und der Service ist somit praktisch nicht mehr nutzbar. In Zahlen bedeutet dies, dass im Fall der POW Regelung 73,9 % der Anfragen des Referenz-Klienten beantwortet werden, während es im Fall des Vergleichsverfahrens nur 47,3 % sind. Selbst dieser geringere Wert kommt nur durch die Phasen zwischen den eigentlichen Überlastmessungen zu Stande. Ein Messzyklus mit einer dauerhaften Überlast würde diesen Wert nahe null Prozent sinken lassen.

12.4 Zusammenfassung Prototyp

Es wurde ein komplett funktionsfähiger Prototyp geplant und implementiert, welcher zeigt, dass die POW Lastregelung real in Webapplikationen integriert werden kann. Die serverseitige Einbindung wurde über einen Servletfilter gelöst. Auf der Seite des Klienten

wurde nach einer Evaluierung verschiedener Techniken eine Java Applet Lösung ausgewählt, weil sie neben anderen Faktoren die größte Abarbeitungsgeschwindigkeit zeigte. Um dem Benutzer das Proof-Of-Work Verfahren zu vermitteln, wurde ein Konzept für einen aussagekräftigen Fortschrittsbalken entwickelt und umgesetzt.

Mittels eines Messzyklus in einem Szenario mit acht Klienten wurde die Leistungsfähigkeit des Prototyps real erprobt. Die Analyse der Ergebnisse hat gezeigt, dass sich der Prototyp sehr ähnlich zu den weiter oben mittels Simulation gewonnenen Messwerten verhält. Die Nachführung der Serverlast ist etwas schlechter als beim Vergleichsverfahren, aber durchaus ausreichend. Die POW Lastregelung erhält dafür im Überlastfall den Service für den Referenz-Klienten aufrecht, was mittels des Vergleichsverfahrens nicht möglich ist. Somit verfügt die POW Lastregelung über eine deutlich bessere Fairness als das Vergleichsverfahren.

13 Zusammenfassung

In der vorliegenden Arbeit wird gezeigt, dass die Aufgabe der Überlastvermeidung auch trotz zukünftiger Hardwareentwicklungen mit großer Wahrscheinlichkeit fortbestehen wird. Während genügend Lösungen existieren, um die Überlast effektiv zu begrenzen, garantiert keine in heterogenen Netzen allgemein anwendbare Lösung eine ausreichende Fairness. Fairness wird in diesem Kontext als Chancengleichheit zwischen den Klienten verstanden.

Die besondere Schwierigkeit, eine faire Überlastvermeidung umzusetzen, besteht darin, dass in der heutigen Internetstruktur Klienten nicht mehr über ihre IP Adresse zuverlässig identifiziert werden können. Damit laufen die bekannten Verfahren wie z. B. Quotas ins Leere. Als Lösung werden Proof-Of-Work Funktionen vorgeschlagen. Dabei wird dem Klienten ein Rechenaufgabe pro gewünschten Zugriff auf den Dienst bzw. Server auferlegt. Auf diese Weise wird insbesondere das Anfrageverhalten von aggressiven Klienten gedrosselt und es kann eine deutlich fairere Situation hergestellt werden.

Die Anforderungen an Proof-Of-Work Funktionen werden im Detail beleuchtet sowie die in der Literatur bekannten Verfahren erläutert. Dabei stellt sich heraus, dass die am meisten verwendete HashCash Funktion eine Schwäche in ihrer Parametrisierbarkeit aufweist. Mittels der im Rahmen dieser Arbeit entwickelten HashCashLin Funktion wird diese Schwäche behoben. Eine detaillierte Untersuchung der statistischen Eigenschaften der Funktion bildet die Basis der folgenden Anwendung der HashCashLin Funktion für die Überlastvermeidung.

Mit dem Ziel, eine POW basierte Überlastvermeidung zu konstruieren, werden unterschiedliche Verfahren betrachtet, die Schwierigkeit des POW zu parametrisieren. Dabei fällt die Wahl auf einen regelungstechnischen Ansatz, der im Folgenden detailliert beschrieben wird. Die Regelung erfolgt über ein dreischichtiges Verfahren, das sowohl eine schnelle Anpassung des POW als auch eine präzise Ausregelung sicherstellt. Dabei wird auf die Besonderheiten der POW Regelung bezüglich der variablen Totzeit und der starken Neigung zu Schwingungen eingegangen.

Mittels einer Simulation wird das Verhalten der entwickelten POW Regelung für ein größeres Überlastszenario mit einem anspruchsvollen Messzyklus erprobt. Als Vergleichsverfahren wird ein klassisches Abweisungsverfahren herangezogen. Dabei zeigt sich, dass die POW Regelung erwartungsgemäß die Last etwas ungenauer führt als das Abweichungsverfahren, dafür aber eine deutlich bessere Fairness im Überlastfall ermöglicht. Insbesondere bei Angriffen aggressiver Klienten kann die Fairness stark

gesteigert werden. Konkret bedeutet dies, dass der Dienst für Klienten mit Einschränkungen voll verfügbar bleibt, was beim Vergleichsverfahren nicht möglich ist.

Nachdem mittels der Simulation die generelle Eignung des Verfahrens zur Steigerung der Fairness in Überlastsituationen gezeigt wurde, wird ein Prototyp erstellt, welcher demonstriert, wie die POW Regelung sowohl auf der Seite des Servers als auch der des Klienten ganz konkret implementiert werden kann. Die Funktion des Prototyps wird abschließend durch einen an die Simulation angelehnten Messzyklus überprüft. Es wird deutlich, dass sich die Ergebnisse der Simulation in der Praxis wiederfinden und das Verfahren auch praktisch Fairness in Überlastsituationen herstellen kann.

Der wissenschaftliche Beitrag dieser Arbeit besteht im Kern in den folgenden vier Punkten. Erstens wurde das Thema Proof-Of-Work Funktionen in der Literatur bisher nie in diesem Umfang und dieser Tiefe beleuchtet. Zweitens wurde mittels der HashCashLin Funktion eine neue, besonders feingranular paramterisierbare POW Funktion entwickelt. Drittens entstand die dreischichtige POW Regelung inklusive einer praktisch einsetzbaren Parametrisierung der Regelung. Viertens wurden erstmals sowohl eine ausführliche Simulation als auch ein voll funktionsfähiger Prototyp entwickelt und intensiv messtechnisch untersucht.

Im sich anschließenden Kapitel wird abschließend ein Ausblick auf zukünftige Fragestellungen und weitere Forschungsthemen gegeben.

14 Ausblick

Die Anwendung von POW Funktionen ist bisher kaum intensiv untersucht worden. Das bedeutet, dass es hier besonders viele noch offene Fragestellungen gibt. Die erste und wichtigste Fragestellung ist, wie sich die zunehmende Verwendung von Netzwerken, die aus gekaperten PCs bestehen (Bot Netze), auf die POW Technologie auswirkt. Auf den ersten Blick kann sich ein Angreifer so eine große Menge Rechenleistung beschaffen, um ein POW System angreifen zu können. Hier muss untersucht werden, ob die POW Überlastvermeidung auch in einem solchen Szenario funktionieren kann und welche Auswirkungen dies auf das Bot-Netzwerk hat. So wäre es z. B. denkbar, dass der massive Verbrauch an Rechenleistung durch den gekaperten Klienten entweder manuell oder maschinell deutlich besser erkannt werden kann als nur die minimale Aktivität eines herkömmlichen Backdoor Programms.

Eine weitere Aufgabe ist die Entwicklung einer POW Funktion mit einer möglichst geringen statistischen Streuung der Berechnungsdauern. Ideal wäre eine POW Funktion mit deterministischer Berechnungsdauer. Dies würde z. B. die Regelung der POW Schwierigkeit erheblich vereinfachen, da das System deutlich gleichmäßiger liefe. Bei der Entwicklung dieser Funktion dürfen aber die restlichen Parameter, welche z. B. die HashCashLin Funktion bietet, nicht leiden, so dass die Entwicklung einer solchen Funktion eine große Herausforderung darstellt. Dabei muss beachtet werden, dass dieses Vorhaben eventuell aus theoretischer Sicht unmöglich sein könnte. In diesem Fall wäre zumindest ein entsprechender Beweis wünschenswert.

Ein weiteres Forschungsfeld sind die in dieser Arbeit gezeigten Regelungsmechanismen. Neben der weiteren Abstimmung der Parameter könnten intelligentere Verfahren entwickelt werden. So wäre es z. B. möglich, ein proaktives Regelungssystem zu entwerfen, welches entweder aus historischen Daten oder auf Grund von Benutzereingaben ein bestimmtes Klientenverhalten vorausahnen kann. So kann das POW schon entsprechend konfiguriert werden, bevor z. B. die eigentliche Lastspitze auftritt. Auf diesem Weg könnte eventuell eine noch effektivere Überlastvermeidung entwickelt werden.

Auf eher technischer Ebene ergeben sich ebenfalls weitere Herausforderungen. Um eine praktisch wirklich breite Anwendbarkeit zu erhalten, erscheint es sinnvoll, neben dem POW Servlet Filter eine Implementierung als Apache Modul umzusetzen. Auf diesem Weg könnte jede Webseite, die aktuell mittels des Apache HTTP Servers zur Verfügung gestellt wird, einfach um eine POW Lastregelung erweitert werden. Analog dazu wäre es in der

Praxis sinnvoll, die POW Berechnung auf der Klientenseite mit einem entsprechenden Browser Plug-In durchzuführen. Selbst wenn die eigentliche Leistungsfähigkeit der Java Implementierung gar nicht so deutlich unter der nativen Implementierung liegt, ist die Integration in den Browser besser und es entsteht ein Zeitvorteil, da weder der Code für das Applet geladen, die Java Virtual Machine gestartet noch der Bytecode vom Just-In-Time Compiler vorbereitet werden muss.

Neben der weiteren wissenschaftlichen Entwicklung der POW Überlastvermeidung sollte die Implementierung in einem realen Anwendungsfall erprobt werden, um zu ermitteln, wie sich das Verfahren im praktischen Einsatz bewährt.

Auf Grund der Weiterentwicklung von Angriffen gegen die SHA-1 Hashfunktion muss diese Entwicklung beobachtet und gegebenenfalls ein Umstieg z. B. auf RIPEMD erfolgen. Dazu muss die Leistungsfähigkeit entsprechender RIPEMD Implementierungen überprüft und die effizienteste in die HashCashLin Funktion integriert werden.

Als weitere Vertiefung der Arbeit kann die POW Lastregelung dahingehend erweitert werden, dass Anfragen entsprechend ihrem Abarbeitungsaufwand unterschiedlich priorisiert werden.

15 Publikationen

Im Rahmen dieser Arbeit sind die folgenden Veröffentlichungen entstanden:

- GOLZE Sebastian, MÜHL Gero: *Fair Overload Handling using Proof-Of-Work Functions*. In: IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2006), Seiten 14-21, Phoenix, Arizona, USA, Januar 2006. IEEE Computer Society.

- GOLZE Sebastian, MÜHL Gero, JAEGER Michael C.: *Fighting Link Farms using Proof-Of-Work*. In: IADIS WWW/Internet 2005 Conference, Band 1, Seiten 85-92, Lissabon, Portugal, Oktober 2005. IADIS Press.

- GOLZE Sebastian, MÜHL Gero, WEIS Torben: *How to Configure Proof-Of-Work Functions to Stop Spam*. In: Hannes Federrath, Herausg., Sicherheit 2005, Band P-62 GI Edition: Lecture Notes in Informatics (LNI), Seiten 165-174, Regensburg, April 2005. Köllen Verlag.

- GOLZE Sebastian, MÜHL Gero, WEIS Torben, JAEGER Michael C.: *Lastregelung von Web Services mittels Proof-Of-Work Funktionen*. In: Jens B. Schmitt, Paul Müller, and Reinhard Gotzhein, editors, Kommunikation in verteilten Systemen (KiVS 2005), Band P-61 *GI Edition: Lecture Notes in Informatics (LNI)*, Seiten 57-64, Kaiserslautern, Februar 2005. Köllen Verlag.

16 Bibliographie

[ABHL03] AHN Lous von, BLUM Manuel, HOPPER Nicholas J., LANGFORD John: CAPTCHA: Using Hard AI Problems For Security. In: Advances in Cryptology, Eurocrypt 2003.

[ABL04] AHN Luis von, BLUM Manuel, LANGFORD John: Telling Humans and Computers Apart (Automatically). In: Communications of the ACM, Volume 47, Issue 2, Pages 56 – 60, 2004.

[ABMW03] ABADI Martin, BURROWS Mike, MANASSE Mark, WOBBER Ted: Moderately Hard, Memory-bound Functions. In: Proceedings of the 10th Annual Network and Distributed System Security Symposium, 2003.

[ABRAM70] ABRAMSON Norman: The Aloha System – Another Alternative for Computer Communications, Proceedings of the 1970 Fall Joint Computer Conference, 1970.

[ADOBE] Adobe Systems Incorporated: Adobe Flash Player Version Penetration, http://www.adobe.com/products/player_census/flashplayer/version_penetration.html (Zugriff 11.11.2008).

[ALTERA] Altera Corporation, http://www.altera.com (Zugriff 11.11.2008).

[ANL00] AURA Tuomas, NIKANDER Pekka, LEIWO Jussipekka: DoS-resistant authentication with client puzzles. In: 8th International Workshop on Security Protocols, Seiten 170–181, Springer-Verlag, 2000.

[BAD01] BADIA Logan: Real World SSL Benchmarking, Rainbow Technologies, Inc., 2001.

[BOINC] University of California: BOINC – Berkeley Open Infrastructure for Network Computing, http://boinc.berkeley.edu (Zugriff 11.11.2008).

[BOSCH07] ROBERT BOSCH GMBH: Kraftfahrtechnisches Taschenbuch, 26. Auflage, Vieweg Verlag, 2007.

[BROWN06] BROWN Stuart: Ip Assignment, Per Capita – The IP rich list – as decided by ARIN, RIPE, et al., 2006,

http://www.modernlifeisrubbish.co.uk/article/ips-assigned-per-capita (Zugriff 11.11.2008)

[BS01] BERLEMANN Michael, SCHMIDT Carsten: Predictive Accuracy of Political Stock Markets Empirical Evidence from a European Perspective, Sonderforschungsbereich 373, Humboldt-Universität zu Berlin, 2001.

[BSI04] BREITSCHAFT Markus, KRABICHLER Thomas, STAHL Ernst, WITTMANN Georg: Sichere Zahlungsverfahren für E-Government, Bundesamt für Sicherheit in der Informationstechnik (BSI), E-Government-Handbuch, Bundesanzeiger Verlag, 2004.

[BVB03] BENCSATH B., VAJDA I., BUTTYAN L.: A Game Based Analysis of the Client Puzzle Approach to Defend Against DoS Attacks. In: IEEE Conference on Software, Telecommunications and Computer Networks (SoftCom2003), 2003.

[CAMRAM] Eric Johansson: Camram, "Campaign for Real Mail", http://sourceforge.net/projects/camram (Zugriff 11.11.2008).

[CERT96] CERT® Advisory CA-1996-21: TCP SYN Flooding and IP Spoofing Attacks, http://www.cert.org/advisories/CA-1996-21.html (Zugriff 11.11.2008).

[CKSA06b] CHAVES Ricardo, KUZMANOV Georgi, SOUSA Leonel, VASSILIADIS: Improving SHA-2 Hardware Implementations. In: Lecture Notes in Computer Science, Band 4249, Seiten 298 - 310, Springer, 2006.

[CKSV06a] CHAVES Ricardo, KUZMANOV Georgi, SOUSA Leonel, VASSILIADIS: Rescheduling for optimized SHA-1 calculation. In: SAMOS Workshop on Computer Systems Architectures Modelling and Simulation, Seiten 425 - 434, 2006.

[CLSY93] CAI J., LIPTON R.J., SEDGEWICK R., YAO A.: Towards uncheatable benchmarks. In: Proceedings of the Eighth Annual Structure in Complexity Theory Conference, Seiten 2 – 11, IEEE, 1993.

[COEL05] COELHO Fabien: Exponential Memory-Bound Functions for Proof of Work Protocols, Technical Report A/370/CRI, CRI, Ecole des mines de Paris, 2005.

[COEL07] COELHO: Fabien: An (Almost) Constant-Effort Solution-Verification Proof-Of-Work Protocol based on Merkle Trees, Technical Report A/390/CRI, Ecole des mines de Paris, 2007.

[DBP96] DOBBERTIN Hans, BOSSELARS Antoon, PRENEEL Bart: RIPEMD-160: A Strengthened Version of RIPEMD. In: Fast Software Encryption, Lecture Notes In Computer Science, Band 1039, Seiten 71 – 82, Springer, 1996.

[DGN03] C. Dwork, A. Goldberg, and M. Naor, "On Memory-Bound Functions for Fighting Spam", Proceedings of the 23rd Annual International Cryptology Conference (CRYPTO 2003), August 2003.

[DN92] DWORK Cynthia, NAOR Moni: Pricing via Processing or Combatting Junk Mail. In: Lecture Notes in Computer Science 740 (Proceedings of CRYPTO'92), 1993, Seiten 137-147.

[DNW05] DWORK Cynthia, NAOR Moni, WEE Hoeteck: Pebbling and Proofs of Work. In: CRYPTO 2005: 25th Annual International Cryptology Conference, Seiten 37 - 54, Lecture Notes in Computer Science, Springer, 2005.

[DOB96] DOBBERTIN Hans: Cryptanalysis of MD5 Compress, 1996.

[DOB96a] DOBBERTIN Hans: Cryptanalysis of MD4, Lecture Notes in Computer Science, Band 1039, Seiten 53 - 69, Springer, 1996.

[DOB98] DOBBERTIN Hans: Cryptanalysis of MD4, Journal of Cryptology, Band 11, Seiten 253 - 271, Springer, 1998.

[DS01] DEAN Drew, STUBBLEFIELD Adam: Using client puzzles to protect TLS. In: 10th USENIX Security Symposium, Seiten 1–8, 2001.

[EURO] Europarat: Europäische Menschenrechtskonvention, Artikel 6, Recht auf ein faires Verfahren, 1950.

[FBGO05] FARRAPOSO Silvia, BOUDAOUD Karima, GALLON Laurent, OWEZARSKI Philippe: Some Issues raised by DoS Attacks and the TCP/IP Suite. In: 4ème Conférence sur la Sécurité et les Architectures Réseau (SAR'2005), 2005.

[FENG03] FENG, Wu-chang: The Case for TCP/IP Puzzles. In: ACM SIGCOMM 2003 Workshops, Seiten 322 - 327.

[FILTER] Sun Microsystems Inc., Sun Developer Network: The Essentials of Filters, http://java.sun.com/products/servlet/Filters.html (Zugriff 11.11.2008).

[FLASH] Adobe Systems Incorporated: Adobe Flash Player, http://www.adobe.com/de/products/flashplayer (Zugriff 11.11.2008).

[FM97] FRANKLIN Matthew K., MALKHI Dahlia: Auditable Metering with Lightweight Security. In: Lecture Notes In Computer Science; Vol. 1318, Proceedings of the First International Conference on Financial Cryptography, Seiten 151 – 160, 1997.

[FRAN86] FRANCEZ Nissim: Fairness (Texts and Monographs in computer science), Springer, New York, 1986.

[FS87] FIAT Amos, SHAMIR Adi: How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Proceedings on Advances in Cryptology - CRYPTO 86. Seiten 186–194, Springer-Verlag, 1987.

[GESE03a] GESEMANN Sebastian: YAHCT – Yet Another Hash Cash Tool - 1.01, http://www.hashcash.org/libs/java/yahct (Zugriff 11.11.2008).

[GESE03b] GESEMANN Sebastian: SHA1 – Secure Hash Algorithm 1, 2003.

[GKR07] GOLA Peter, KLUG Christoph, REIF Yvette: Datenschutz- und presserechtliche Bewertung der „Vorratsdatenspeicherung",Gesellschaft für Datenschutz und Datensicherung e.V. (GDD), Bonn, 2007.

[GLGN02] GREMBOWSKI Tim, LIEN Roar, GAJ Kris, NGUYEN Nghi, BELLOWS Peter, FLIDR Jaroslav, LEHMAN Tom, SCHOTT Brian: Comparative Analysis of the Hardware Implementations of Hash Functions SHA-1 and

SHA-512. In: Lecture Notes in Computer Science, Band 2433, Springer, 2002.

[GLO07] GLOGAU Dirk: Zurück in die Zukunft, Network Computing, Ausgabe 01/2007, CMP-WEKA Verlag GmbH.

[GMW05] GOLZE Sebastian, MÜHL Gero, WEIS Torben: How to Configure Proof-Of-Work Functions to Stop Spam. In: Hannes Federrath, Herausg., Sicherheit 2005, Band P-62 GI Edition: Lecture Notes in Informatics (LNI), Seiten 165-174, Regensburg, April 2005. Köllen Verlag.

[GRUND] Grundgesetz für die Bundesrepublik Deutschland, Artikel 3, Gleichheit vor dem Gesetz, 1949.

[GS07] GARDNER-STEPHEN Paul: Escalating The War On SPAM Through Practical PoW Exchange, 2007.

[GS98] GOLDSCHLAG David M., STUBBLEBINE Stuart G.: Publicly Verifiable Lotteries: Applications of Delaying Functions. In: Lecture Notes in Computer Science, Band 1465, Seiten 1305 - 1338, Springer 1998.

[HC] BACK Adam: HashCash – A Denial-of-Service Counter-Measure, http://www.hashcash.org (Zugriff 11.11.2008).

[HDM77] HELLMAN Martin E., DIFFIE Bailey W., MERKLE Ralph C.: Cryptographic apparatus and method, U.S. patent no. 05,830,754, Ausgestellt 6. September 1977.

[HEIß87] HEIß, Hans-Ulrich: Überlast in Rechensystemen. Modellierung und Verhinderung, Springer-Verlag Berlin, 1987.

[IH04] IHLENFELD Jens: Vodafone will UMTS beschleunigen – Schnelleres Surfen dank Datenkompression, golem.de, 2004.

[ISO16262] ISO/IEC 16262: Information technology – ECMAScript language specification, 2002.

[JB07] JUELS A., BRAINARD J.: Cryptographic countermeasures against connectiondepletion attacks, U.S. patent no. 7,197,639, Ausgestellt 27. März 2007.

[JB99] JUELS Ari, BRAINARD John: Client Puzzles: A Cryptographic Defense Against Connection Depletion Attacks. In: Proceedings of NDSS '99 (Networks and Distributed Security Systems), Seiten 151-165, 1999.

[JOHN02] Paul Johnston: A JavaScript implementation of the Secure Hash Algorithm, SHA-1, as defined in FIPS PUB 180-1, 2002.

[JSa] Webhits internet design GmbH: Web-Barometer: http://www.webhits.de/deutsch/index.shtml?webstats.html (Zugriff 11.11.2008).

[JSb] Browser Statistics, http://www.w3schools.com/browsers/browsers_stats.asp (Zugriff 11.11.2008).

[KAL60] KALMAN Rudolf E.: A new Approach to Linear Filtering and Prediction Problems. In: Transactions of the ASME – Journal of Basic Engineering, 1960.

[KANT] KANT Immanuel: Kritik der praktischen Vernunft (1787), Herausgeber: Heiner Klemme, Horst D. Brandt, Meiner Verlag, 2003.

[LC04] LAURIE Ben, CLAYTON Richard: "Proof-Of-Work" Proves Not to Work, In: Third Annual Workshop on Economics of Information Security (WEIS04), University of Minnesota, 2004.

[LC06] LIU Debin, CAMP L Jean: Proof of Work can Work. In: The Fifth Workshop on the Economics of Information Security (WEIS 2006), University of Cambridge, 2006.

[LEN06] LAURENS Vicky, EL-SADDIK Abdulmotaleb, NAYAK Amiya: Requirements for Client Puzzles to Defeat the Denial of Service and the Distributed Denial of Service Attacks. In: The International Arab Journal of Information Technology, Volume 3, 2006.

[MAEH91] MAEHLER Martin: Job Scheduling under Fairness Aspects. In: Messung, Modellierung und Bewertung von Rechensystemen (6. GI/ITG-Fachtagung) Informatik-Fachberichte 286, Seiten 46 - 60, Springer, 1991.

[MAPS] Google Inc.: Google Maps API, http://code.google.com/apis/maps/index.html (Zugriff 11.11.2008).

[MAXON] MAXON Computer GmbH: MAXON CINEBENCH R10, http://www.maxon.net/pages/download/cinebench_d.html (Zugriff 11.11.2008).

[MOORE65] MOORE Gordon: Cramming More Components Onto Integrated Circuits. In: Electronics 1965.

[MSRS04] SCHRECKENBERG M., SELTEN R.: Human Behaviour and Traffic Networks, Springer-Verlag Berlin, 2004.

[MULLER04] MULLER Frédéric: The MD2 Hash Function Is Not One-Way, ASIACRYPT 2004, Lecture Notes in Computer Science, Band 3329, Seiten 214 - 229, Springer, 2004.

[MWD07] MUNIR Kashif, WELZL Michael, DAMJANOVIC Dragana: Linux beats Windows! – or the Worrying Evolution of TCP in Common Operating Systems, Universität Insbruck, 2007.

[NAGLE85] NAGLE John: On Packet Switches With Infinite Storage, RFC 970, 1985.

[PARK57] PARKINSON C. N.: Parkinson's Law, and Other Studies in Administration, Houghton Mifflin, 1957.

[PB] Microsoft Research: The Penny Black Project, http://research.microsoft.com/research/sv/PennyBlack (Zugriff 11.11.2008).

[PH07] PU Jian, HAMDI Mounir: Applying Router-Assisted Congestion Control to Wireless Networks: Challenges and Solutions. In: Workshop on High Performance Switching and Routing (HPSR'07), 2007.

[PLUG] Sun Microsystems Inc.: Desktop Java, Java Plug-in Technology, http://java.sun.com/products/plugin (Zugriff 11.11.2008).

[POSTEL81] POSTEL Jon: Internet Protocol, DARPA Internet Program, Protocol Specification, RFC 791, 198.

[PS87] POLLARD John M., SCHNORR Claus P.: An Efficient Solution of the Congruence x2 + ky2 = m(mod n). In: IEEE Transactions on Information Theory, 1987.

[RE96] REKHTER Y. et al.: Address Allocation for Private Internets, RFC 1918, 1996.

[RFC2460] The Internet Society: Internet Protocol, Version 6 (IPv6), Specification, RFC 2460, 1998.

[RFC2581] ALLMAN M., PAXSON V., STEVENS W.: TCP Congestion Control, RFC 2460, 1999.

[RIV92a] RIVEST Ron: The MD4 Message-Digest Algorithm, RFC 1320, 1992.

[RIV92b] RIVEST Ron: The MD5 Message-Digest Algorithm, RFC 1321, 1992.

[RLK92] RIVEST Ron, LINN John, KALISKI Burt: The MD2 Message-Digest Algorithm, RFC 1319, 1992.

[RSW96] RIVEST Ronald, SHAMIR Adi, WAGNER David: Time-lock puzzles and timed-release Crypto, Bericht MIT/LCS/TR-684, MIT, 1996.

[SE01] SRISURESH P., EGEVANG K.: Traditional IP Network Address Translator (Traditional NAT), RFC 3022, 2001.

[SETI] University of California: SETI@home Search for Extraterrestrial Intelligence, http://setiathome.berkeley.edu (Zugriff 11.11.2008).

[SL03] SERJANTOV Andrei, LEWIS Stephen: Puzzles in P2P Systems, 8th Cabernet Radicals Workshop, Corsica, 2003.

[SP] SPIEGEL ONLINE: Herbst-Spezial der Bahn – Ein fast unmögliches Angebot, www.spiegel.de, 2005.

[SPAM] The Apache SpamAssassin Project: SpamAssassin, http://spamassassin.apache.org (Zugriff 11.11.2008).

[SPSK04] SPANN Martin, SKIERA Bernd: Einsatzmöglichkeiten virtueller Börsen in der Marktforschung. In: Zeitschrift für Betriebswirtschaft, Ausgabe 74, Seiten 25 - 48, 2004.

[SZABO96] SZABO Nick: The Mental Accounting Barrier to Micropayments, 1996, http://szabo.best.vwh.net/micropayments.html (Zugriff 11.11.2008).

[TEMP03a] TEMPLETON Brad: Reflections on the 25th Anniversary of Spam, 2003, http://www.templetons.com/brad/spam/spam25.html (Zugriff 11.11.2008).

[TEMP03b] TEMPLETON Brad: Origin of the term "spam" to mean net abuse, http://www.templetons.com/brad/spamterm.html (Zugriff 11.11.2008).

[TMDA] MASTALER Jason R.: Tagged Message Delivery Agent (TMDA), http://tmda.net (Zugriff 11.11.2008).

[TORG07] TORGEMANE Henri: As3 Crypto Framework 1.3, 2007, http://crypto.hurlant.com (Zugriff 11.11.2008).

[TUGRAZ] TU Graz: SHA-1 Collision Search Graz, http://boinc.iaik.tugraz.at/sha1_coll_search (Zugriff 11.11.2008).

[TYLL02] TING Kurt K., YUEN Steve C.L., LEE K.H., LEONG Philip H.W.: An FPGA Based SHA-256 Processor. In: Lecture Notes in Computer Science, Band 2438, Seiten 449 - 471, Springer, 2002.

[WFLY04] WANG Xiaoyun, FENG Dengguo, LAI Xuejia, YU Hongbo: Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD, Rump Session, Crypto 2004.

[WIRTH95] WIRTH Niklaus: A Plea for Lean Software. In: Computer, Band 28, Ausgabe 2, Seiten 64 – 68, 1995, IEEE Computer Society Press.

[WLFCY05] WANG Xiaoyun, LAI Xuejia, FENG Dengguo, CHEN Hui, YU Xiuyuan: Cryptanalysis of the Hash Functions MD4 and RIPEMD. Eurocrypt 2005, Lecture Notes in Computer Science, Band 3494, Seiten 1 - 18, Springer, 2005.

[WM94] WULF Wm. A., MCKEE Sally A. : Hitting the Memory Wall: Implications of the Obvious, Computer Architecture News, 23(1): Seiten 20 - 24, March 1995. Also UVa Technical Report CS-94-48, 1994.

[WR03] WANG, Xiao Feng, REITER, Michael K.: Defending Against Denial-of-Service Attacks with Puzzle Auctions. In: Proceedings of the 2003 IEEE Symposium on Security and Privacy (SP'93).

[WR04] WANG XiaoFeng, REITER Michael: Mitigating bandwidth-exhaustion attacks using congestion puzzles. In: Proceedings of the 11th ACM conference on Computer and communications security, ACM, 2004.

[WY05] WANY Xiaoyun, YU Hongbo: How to Break MD5 and Other Hash Functions, Advances in Cryptology EUROCRYPT 2005, Lecture Notes in Computer Science, Band 3496, Seiten 19 - 35, Springer, 2005.

[WYY05] WANG Xiaoyun, YIN Yiqun Lisa, YU Hongbo: Finding Collisions in the Full SHA-1. In: Lecture Notes in Computer Science, CRYPTO 2005, Band 3621, Seiten 17 - 36, Springer, 2005.

[XILINX] Xilinx Inc. http://www.xilinx.com (Zugriff 11.11.2008).

[YSKO06] YUSUKE Naitp, SASAKI Yu, KUNIHIRO Noboro, OHTA Kazuo: Improved Collision Attack on MD4 with Probability Almost 1, Lecture Notes in Computer Science, Band 3935, Seiten 129 - 145, Springer, 2006.