HTTP-Servererweiterung zur dynamischen Anpassung der Übertragungsrate

Diplomarbeit
der Philosophisch naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von
Stattenberger-Bechter Silvia

2004

Leiter der Arbeit:
Prof. Dr. Torsten Braun

Forschungsgruppe Rechnernetze und verteilte Systeme (RVS)
Institut für angewandte Mathematik und Informatik

Zusammenfassung

Die Benutzeranzahl und das Angebot im Internet haben in den letzten Jahren massiv zugenommen. Immer mehr Benutzer wollen auch über drahtlose Links auf das Internet zugreifen. Die Bandbreite, die den Benutzern über solche drahtlose Links zur Verfügung steht, ist allerdings — wie oft auch der Zugang über ein normales Modem — beschränkt. Auf nicht drahtlose Verbindungen optimierte Mechanismen des Transportprotokolls TCP führen zu einer zusätzlichen, nicht notwendigen Belastung der ohnehin schon beschränkten Bandbreite. Anbieter und Benutzer wissen meist auch nicht, wieviel effektive Bandbreite einer Verbindung zur Verfügung steht. In dieser Arbeit wird versucht, Lösungen für diese Probleme zu finden. Dabei wurde ein Webserver entwickelt, der eine Benutzerverwaltung und ein Cache beinhaltet und als Proxy zwischen der Endleitung und dem Internet fungieren kann. Das Cache dient dazu, dass bereits einmal angeforderte Daten nicht erneut über das Internet angefordert werden müssen und ermöglicht es dem Benutzer, Vorteile eines Caches zu benutzen ohne dafür Speicher zur Verfügung zu stellen. Im Webserver integriert ist eine Funktion zum Messen der Bandbreite vom Webserver zum Benutzer. Dabei muss der Benutzer ausser einem normalen Webbrowser nichts zur Verfügung stellen. Gemessen werden auf Applikationsebene versendete Nutzdaten, das Resultat der Messung gibt also an, wieviel Bandbreite zur Übertragung von Nutzdaten zur Verfügung steht, nicht wie sonst die maximale Kapazitätsgrenze einer Leitung. Eine weitere im Webserver integrierte Funktion ist ein Bandbreitenbeschränker. Die Sendebandbreite kann aufgrund der vorher gemachten Messung oder aufgrund von Benutzereinstellungen (vom Anbieter oder vom Benutzer) beschränkt werden. Messungen haben ergeben, dass die Berechnungen des Bandbreitentesters sehr gut mit der Wirklichkeit übereinstimmen. Dabei wurde der gesamte Netzverkehr während des Tests beobachtet und mit den Messergebnissen verglichen. Auch die Beschränkung der Bandbreite funktioniert ab einer gewissen Sendedauer einwandfrei, die Abweichungen der erreichten Übertragungsrate von der tatsächlich eingestellten bewegen sich im unteren Promillbereich. Eine Beschränkung aufgrund der gemessenen Bandbreite liefert keine bemerkenswerte Vergrösserung der Bandbreite - Messungen haben jedoch ergeben, dass Übertragungen mit reduzierter Senderate stabiler und regelmässiger geschehen. Die Architektur von Webserver, Bandbreitenbestimmung und -reduzierung kann in Endpunkten zu drahtlosen Netzen (z.B. W-LAN Hubs oder Basisstationen von Mobilanbietern) eingesetzt werden. Dabei können auch nur Teile der Architektur benutzt werden, z.B. der Bandbreitenbeschränker, um eine Form von Quality of Service anzubieten.

Inhaltsverzeichnis

1 Einleitung

Das Internet erfreut sich in den letzten Jahren eines explosionsartigen Wachstums, sowohl was die Anzahl Benutzer betrifft (siehe Abbildung 1.1), als auch in der Menge der abrufbaren Informationen. Diese Informationen sind heute fast ausschliesslich in Form von Webseiten verfügbar. Während anfangs ausschliesslich textbasierte Webseiten existierten, schuf erst die Möglichkeit Bilder und andere (multimediale) Inhalte darzustellen die Grundlage für die Beliebtheit des Internets. Durch diese wachsende Beliebtheit des Internets wurde jedoch auch seine Tauglichkeit als Werbemedium erkannt, was dazu führte, dass die Hauptinformationen oft in einer Flut von Bildern, Werbebannern und anderen Bandbreite raubenden Dingen versteckt werden. Dies fällt besonders ins Gewicht, wenn der Endbenutzer ein drahtloses Endgerät benutzt, da die zur Verfügung stehende Bandbreite für solche Geräte meist beschränkt ist. Aber auch für andere Benutzer können die unerwünschten Beilagen zur angeforderten Information lästig werden und gegebenenfalls die Übertragungsdauer empfindlich verlängern.

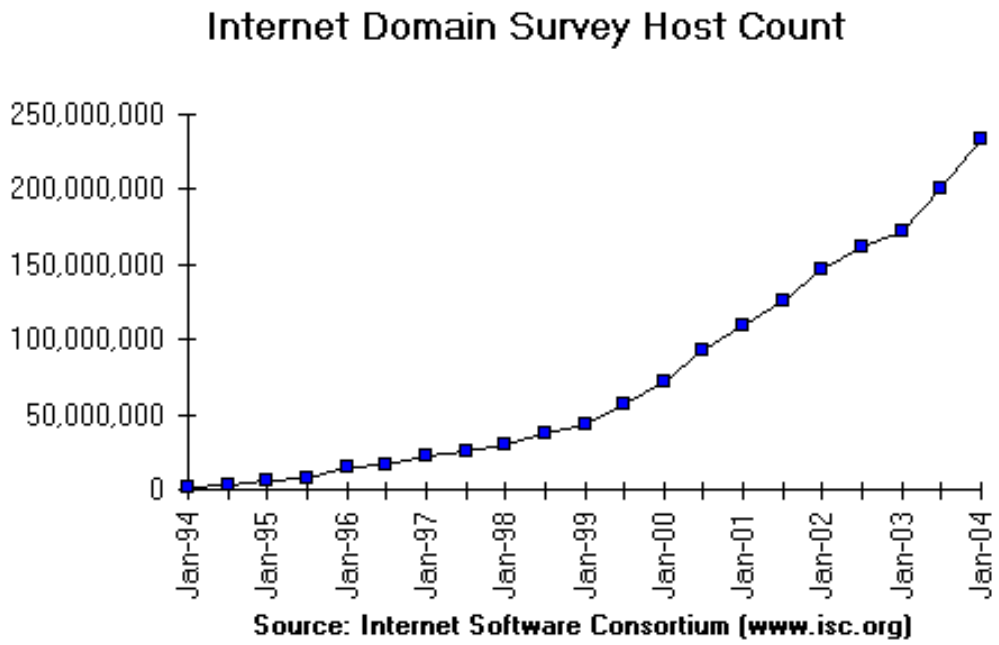


Abbildung 1.1: Wachstum der Internetbenutzer Weltweit

Moderne Desktop-Computer oder Laptops haben mit den verbreiteten Technologien normalerweise kein Problem damit, die anfallenden Daten zu verarbeiten oder darzustellen. Mit der wachsenden Zahl mobiler Computer wächst jedoch auch die Anzahl der Endgeräte, die nur eine beschränkte Bandbreite zur Verfügung haben. Beson-

ders die drahtlose Kommunikation wird immer beliebter. Wireless LAN (IEEE 802.11) bietet den Komfort der Mobilität und eine relativ hohe maximale Bandbreite (bis zu 11 Mbit/s mit 802.11b, bis zu 54 Mbit/s mit 802.11g) [24]. Die meisten neu gekauften mobilen Computer haben dementsprechend bereits eine Netzwerkkarte für die drahtlose Kommunikation nach dem Wireless LAN Standard integriert und die Verbreitung von privaten Wireless-Hubs sowie den öffentlich angebotenen Hot-Spots nimmt stark zu (siehe Abbildung 1.2).

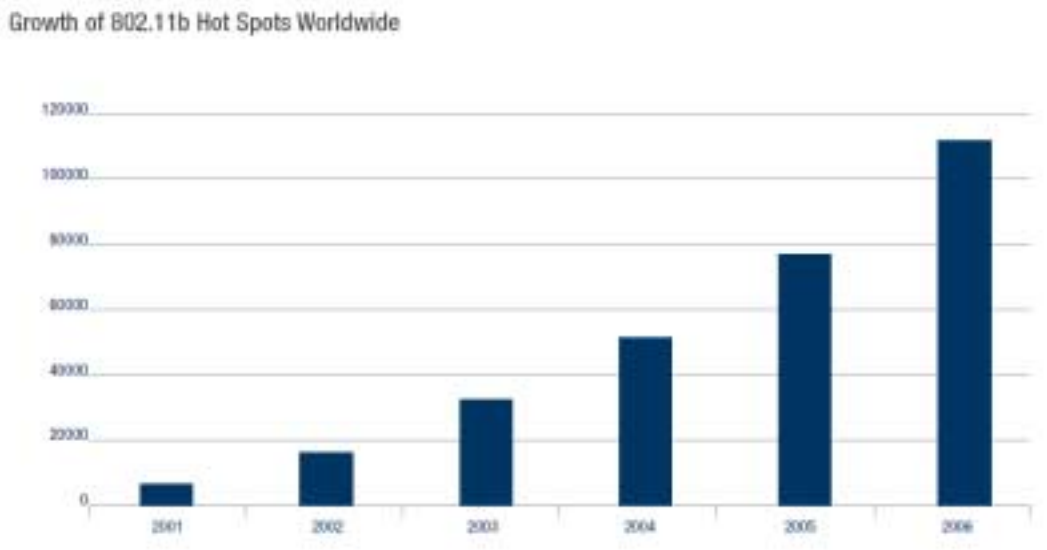


Abbildung 1.2: Voraussichtliches Wachstum der öffentlich zugänglichen Wireless LAN Hubs

Heute sind drahtlose Links zu einem wichtigen Teil des Internets geworden. In Zukunft wird erwartet, das die Anzahl der mobilen Endgeräte diejenige der fix installierten Geräte übersteigen wird [19]. Die Abbildung (siehe Abbildung 1.3) von TDK [22] zeigt die erwartete Entwicklung von Seiten der Betreiber von Hotspots. Dabei kann man beobachten, dass ein sehr grosses Wachstum im Bereich öffentlicher W-LAN Zugänge erwartet wird. Ähnlich optimistisch sind die Prognosen der Produzenten der Hardware (siehe Graphik in Abbildung 1.3), welche in den nächsten drei Jahren ein Wachstum von über 50% erwarten.

Auch der Internetanschluss über mobile Modems wird immer beliebter: Trotz der verhältnismässig sehr tiefen Übertragungsraten sind Mobiltelephonbenutzer, welche auf Daten zugreifen, stark im Ansteigen. Als Beispiel ist das Wachstum des Seitenangebotes eines japanischen Mobilunternehmens [40] aufgeführt (siehe Abbildung 1.4), dessen 40 Millionen Benutzer zunehmend auch Datenzugriff verlangen.

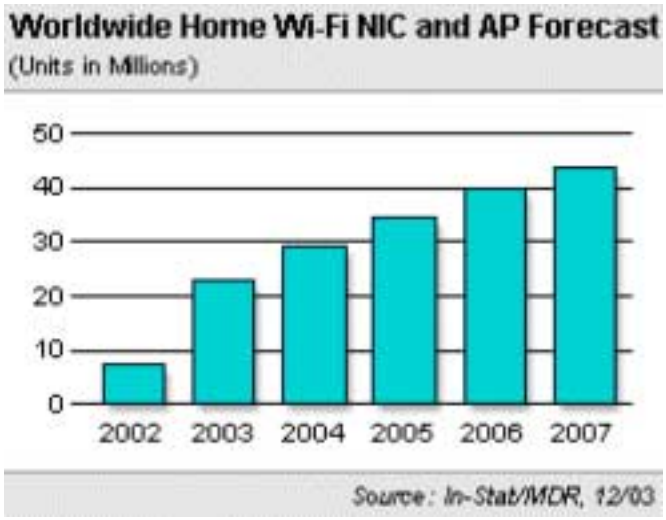


Abbildung 1.3: Voraussichtliches Wachstum von Hardware im drahtlosen Bereich

1.1 Problemstellung

Hauptproblem bei Verbindungen mit Endgeräten, die uneingeschränkt Zugang zum Internet wünschen, ist meist die limitierte Bandbreite. Auch bei einem guten Endgerät wie einem PC oder einem Laptop kann die Verbindung über eine Modemleitung ein grosses Ärgernis darstellen. Bei mobilen Verbindungen (z.B. über ein mobiles Telephon mit entsprechendem Modem) wird über das normale GSM oder GPRS Netz eine theoretische Höchstrate von 9,6 kbit/s erreicht. Dies kann schon das Laden einer einfachen Webseite mit ein paar kleinen Bildern mehrere Minuten dauern lassen. Hinzu kommt, dass die theoretisch maximale Übertragungsrate für Nutzdaten aus verschiedenen Gründen niemals erreicht wird. Abgesehen von den das Netz belastenden Kontrolldaten und Signalisierungspaketen kann es bei Knotenpunkten wie Routern oder bei Netzüberlastung im Backbone-Netzwerk zu einer Verminderung der Übertragungsrate kommen, die der Benutzer nicht voraussehen kann. Um eine optimale Übertragung zu erreichen muss somit entweder auf kostenpflichtige Dienstleistungen wie QoS (Quality of Service) zurückgegriffen werden oder die Daten müssen so versendet werden, dass beim ersten Sendeversuch so viele Pakete wie möglich beim Empfänger ankommen, ohne dass verlorene Pakete mehrmals gesendet werden müssen.

1.2 Ziel

Ziel dieser Arbeit ist es daher, die Anpassung der Übertragungsrate an die beim Benutzer verfügbare Bandbreite zu automatisieren, so dass Nachteile des benutzten Transportprotokolles TCP umgangen werden können. Der Benutzer hat die Möglichkeit, die ihm zur Verfügung stehende Bandbreite anzugeben, das Programm soll aber auch

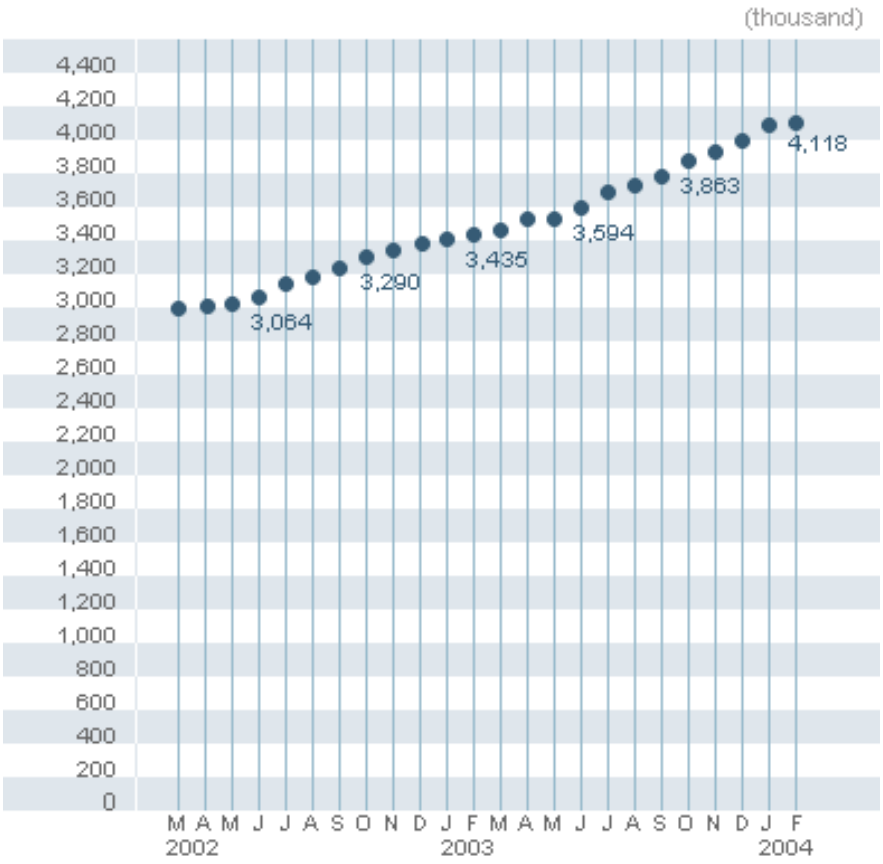


Abbildung 1.4: Seitenangebot für Mobiltelephone eines japanischen Mobilanbieters

in der Lage sein, die Bandbreite mit einer gewissen Sicherheit selbst zu bestimmen. Nach einer einmaligen Eingabe dieser Parameter soll ein Benutzerprofil angelegt werden, welches es dem Benutzer ermöglicht ohne die vorherige Angabe dieser Daten von der automatischen Anpassung zu profitieren.

Bei der Berechnung der Bandbreite soll grosser Wert darauf gelegt werden, dass einerseits der Benutzer möglichst entlastet wird (also auch, dass er auf dem Endgerät keinerlei zusätzliche Software installieren muss) und dass beim Ermitteln der maximalen Übertragungsrate so wenig Bandbreite wie möglich verschwendet wird. Weiter soll auch dafür gesorgt werden, dass zukünftig angeforderte Daten (Webseiten) mit einer möglichst optimalen Übertragungsrate versendet werden, d.h. dass die Bandbreite den angegebenen oder berechneten Parametern angepasst wird.

1.3 Mögliche Anwendungsgebiete

Überall, wo keine Klarheit darüber herrscht, wieviel Bandbreite einem Benutzer ef-

fektiv zur Verfügung steht, kann der Bandbreitentester der hier vorgestellten Arbeit eingesetzt werden. Auch bei eigentlich bekannten Übertragungsraten kann es sinnvoll sein, die Architektur einzubauen: Wenn ein öffentlicher *Hot-Spot* mehreren Benutzern zur Verfügung stehen muss, reduziert sich die bekannte Übertragungsrate von z.B. 11 Mbit/s - allerdings nicht linear zur Anzahl Benutzer, denn auch die Anzahl Kollisionen nimmt zu (siehe dazu [17]). Hier könnte eine Optimierung mit Hilfe der Bandbreitenerkennung und der entsprechenden Reduzierung das System verbessern. Das selbe gilt für Basisstationen von Mobilanbietern: Wenn Daten über GPRS oder UMTS versendet werden sollen, so kann die Übertragungsleistung auf diesem Wege verbessert werden. Weiter kann der Teil der Arbeit, der die Bandbreite anpasst, auf Applikationsebene so integriert werden, dass auf eine einfache Art und Weise die Bandbreite für Benutzer eingeschränkt werden kann, so zum Beispiel bei einem öffentlichen Netzzugang, wo jeder Benutzer nur einen Teil der Kapazität für sich nutzen darf.

1.4 Aufbau

Die Struktur dieser Arbeit ist wie folgt aufgebaut: Im zweiten Kapitel werden die grundlegenden Protokolle und Anwendungen erklärt. Im dritten Kapitel werden bereits vorhandene Arbeiten angeschaut, die Teilgebiete dieser Arbeit auf eine andere Art behandelt haben. Darauf folgt ein Kapitel, in welchem erst die Architektur erklärt wird, dann wird die Funktionalität mit den dabei aufgetretenen Problemen diskutiert. In Kapitel fünf wird die gewählte Implementation genauer anhand von Codebeispielen erklärt. Darauf folgt ein Kapitel mit Messergebnissen und Resultaten. Kapitel acht fasst die Ergebnisse zusammen schliesst die Arbeit ab.

2 Grundlegende Protokolle und Anwendungen

In diesem Kapitel sollen im Verlauf der Arbeit verwendete Konzepte, Protokolle und Begriffe etwas genauer erläutert werden. Es ist aber leider aus Platz- und Komplexitätsgründen nicht möglich, alle Protokolle und Konzepte im Detail zu erklären, es soll lediglich ein Überblick verschafft werden, der die nachfolgende Implementation verständlicher macht.

2.1 Das Transport Control Protocol

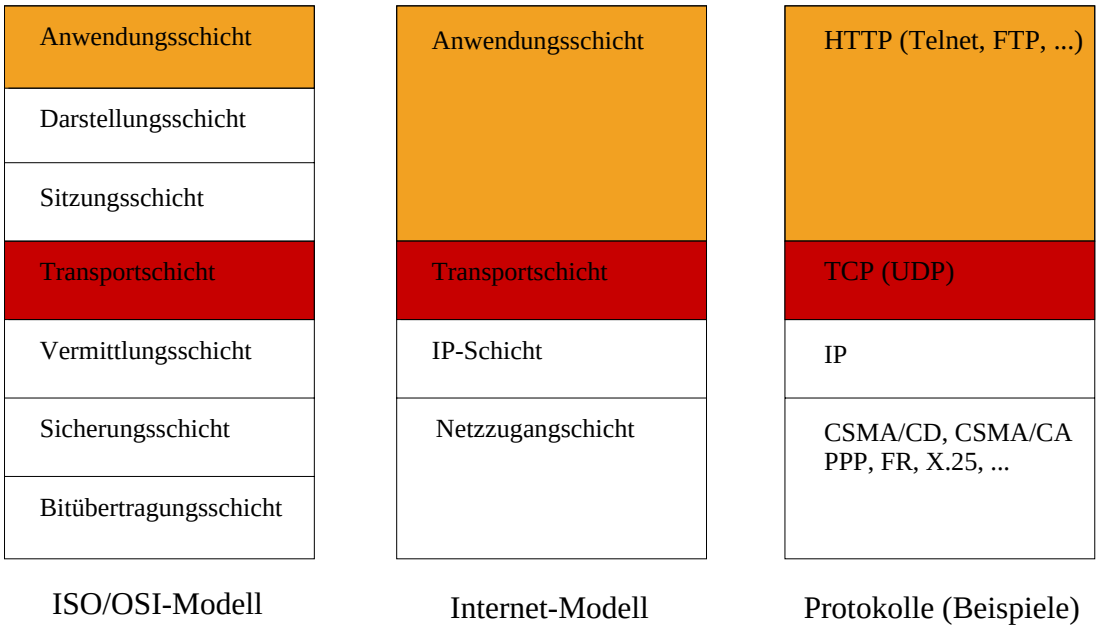


Abbildung 2.1: Referenzmodelle und Einteilung von benutzten Protokollen

Das Transport Control Protocol (TCP) [43, 48] findet sich im theoretischen Schichtenmodell von ISO/OSI auf Schicht vier (im eher angewandten Internet-Schichtenmodell auf Schicht drei) und wird vom HTTP-Protokoll (siehe 2.3) zum Transport benutzt (siehe Abbildung 2.1). Der Aufgabenbereich von TCP kann in zwei Hauptaufgaben unterteilt werden: Einerseits dem Vermitteln zwischen verschiedenen auf dem Rechner laufenden Anwendungen und dem Internet-Protokoll (IP) und andererseits der Fehlerkorrektur und Flusskontrolle.

2.1.1 Vermittlung zwischen Anwendungen und dem Internet-Protokoll

Auf einem Rechner laufen meist mehrere Anwendungen, die über das Internet kommunizieren, (quasi) gleichzeitig ab. Die ankommenden Daten müssen daher an die richtigen Anwendungen verteilt werden.

Dies geschieht mit einer sogenannten Portnummer, welche im Header des TCP Protokolls angegeben wird (siehe Abbildung 2.2). Die genaue Adresse, wohin die Daten gesendet werden, ist somit zusammengesetzt aus der IP-Adresse des Empfängers (adressiert den Rechner) und der Portnummer der Anwendung (adressiert die Anwendung). Für die Portadressierung stehen im TCP-Header 16 Bits zur Verfügung, somit stehen gesamthaft 65536 Ports zur Verfügung. Die ersten 2^{10} Ports sind als *wohlbekannte Portnummern* (well-known port numbers) für das System oder den Administrator (root) reserviert und definieren die Server-Portnummern für verschiedene Standarddienste wie FTP, HTTP, DNS, etc. Die Portnummern 1024-49151 sind sogenannte *registrierte Ports*. IANA (Internet Association for Names and Numbers [23]) verwaltet diese Nummern, welche von allen als Server-Port benutzt werden können. Die restlichen Ports (49152 - 65535) stehen für private Zwecke zur Verfügung und werden meist von TCP als Absenderports benutzt.

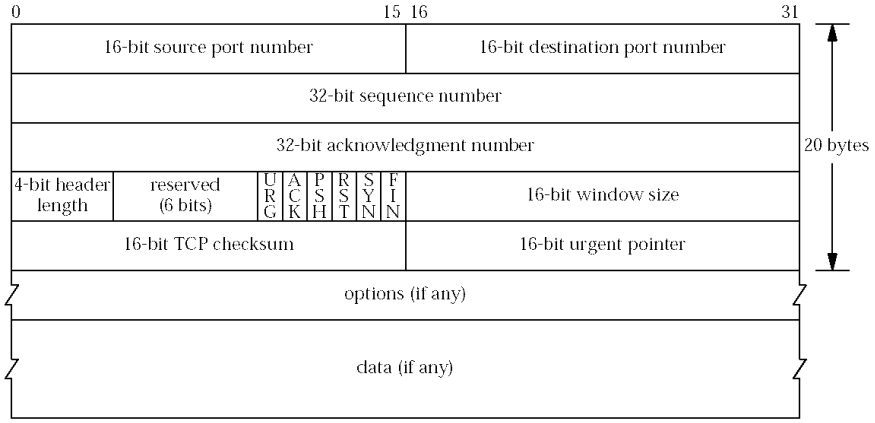


Abbildung 2.2: TCP-Header

Server, die Anwendungen bereitstellen, warten auf einer festgelegten Portnummer auf Anfragen von Clients. Da der Client die Initiative ergreift, muss er die Adresse des Servers (d.h. dessen IP-Adresse und Portnummer) kennen. Die Portnummer ergibt sich aus der Art des in Anspruch zu nehmenden Dienstes und ist entweder *well-known* (wie HTTP auf Port 80, SSH auf Port 21, DNS auf Port 53, etc.) oder dem Client auf eine andere Art bekannt (z.B. durch einen Link oder durch Eingabe vom Benutzer). Der Client dagegen benötigt keine wohlbekannte Portnummer. Der Server kann dem Client antworten, da in dessen Anfrage die Absenderadresse gespeichert ist (IP-Adresse und Portnummer). Der Client kann deshalb eine beliebige Portnummer wählen.

2.1.2 Fehlererkennung und -korrektur

Eine weitere Funktion von TCP ist die Fehlererkennung und -korrektur. TCP kontrolliert als erstes, ob die Daten korrekt übertragen wurde. Dies geschieht mit Hilfe von CRC (Cyclic Redundancy Check). Eine binäre Polynom-Division ermöglicht es, mit relativ grosser Wahrscheinlichkeit Übertragungsfehler zu erkennen, ohne dass zuviele redundante Bits übertragen werden müssen. Die 16 Bits im TCP-Header (siehe *16-Bit TCP checksum* in Abbildung 2.2) gewährleisten eine hohe Fehlererkennungsquote. CRC wird deshalb auch in sehr vielen Protokollen verwendet, um Übertragungsfehler in Header oder Daten zu erkennen.
Bei fehlerhafter Übertragung wird als Fehlerkorrektur das TCP-Datagramm (Paket) nochmals angefordert.

2.1.3 Paketverlust

Durch das Aufbauen der Verbindung vom Client zum Server werden auch Paketverluste und Vertauschungen in der TCP ist ein verbindungsorientiertes Protokoll. Dies bedeutet, dass vor dem Übertragen von Daten eine virtuelle Verbindung zwischen dem Client und dem Server aufgebaut wird. Paketreihenfolge erkannt und können korrigiert werden. Dies geschieht, indem die Bytes der gesendeten Nutzdaten numeriert werden. Ein Feld im Header (siehe *32-Bit sequence number* in Abbildung 2.2) enthält die Sequenznummer des ersten Bytes des Datenteils. Die weiteren Bytes in diesem TCP-Datagramm haben entsprechend jeweils um eins erhöhte Sequenznummern. Der Empfänger kann nun anhand der Sequenznummern und der jeweiligen Länge des Datenteils die gesendeten Daten wieder zusammensetzen und bemerkt so das Fehlen eines Paketes. Der Verlust von Daten wird umgangen, indem beim Senden eines TCP-Datagrammes eine Frist (Timeout) eingeräumt wird, innerhalb der der Empfang des TCP-Datagrammes quittiert sein muss. Läuft die Frist ab, ohne dass eine Quittung eingetroffen ist, dann wird das TCP-Datagramm erneut gesendet und die Frist wird angepasst (in diesem Fall vergrössert, dies geschieht mit einem Algorithmus, welcher mit Hilfe der alten Frist und der momentanen Verzögerung eine neue Frist berechnet). Dies wiederholt sich so lange, bis eine Quittung eintrifft oder die TCP-Verbindung abgebrochen wird, da offensichtlich der Kontakt zum Partner gänzlich verloren gegangen ist. Da mehrere TCP-Datagramme gesendet werden können, bevor eine Quittung eintreffen muss, muss erkennbar sein, welche Daten quittiert werden. Dies geschieht, indem in der Quittung die zuletzt erhaltene Sequenznummer angegeben wird (eigentlich wird das als nächstes erwartete Byte angegeben, siehe *32-Bit acknowledgment number* in Abbildung 2.2). Quittungen können auch kumulativ sein, d.h. dass eine Quittung mit der Quittungsnummer x alle Bytes mit kleinerer Nummer als x quittiert. Geht also z.B. eine Quittung verloren, aber die nächste kommt dafür an, so wird der Verlust einer Quittung durch die nächste kompensiert.

2.1.4 Schutz des Empfängers

Damit der Empfänger nicht mit Daten überflutet wird, die er nicht verarbeiten kann, wurde das Konzept des *Sliding-Window* eingeführt. Dies ist eine Erweiterung des *Stop-and-wait*-Verfahrens, wo der Absender nach jedem gesendetem Paket auf eine positive Bestätigung warten muss. Während dieser Zeit wird das Netzwerk nicht ausgelastet. Bei Sliding-Window wird nun eine bestimmte Zahl an Paketen festgelegt, die gesendet werden dürfen, noch bevor die Bestätigung eingetroffen ist. Dazu wird eine Art Fenster über die Sequenz der zu sendenden Pakete gelegt. Alle Pakete innerhalb dieses Fensters werden gesendet. Sobald eine Bestätigung für das erste Paket in diesem Fenster eingetroffen ist, verschiebt sich das Fenster in der Sequenz nach vorne. Soweit die normale Variante von Sliding-Windows. In TCP wird nun ein dritter Zeiger verwendet, der auf eine Position innerhalb des Fensters zeigt. Dieser trennt nun die Pakete, die bereits gesendet wurden und noch nicht bestätigt wurden von den Paketen, die noch nicht gesendet wurden, aber ohne Verzögerung gesendet werden dürfen. Dieser dritte Zeiger erlaubt es, das Fenster dynamisch zu verändern. Damit ist es TCP möglich eine Flusskontrolle zu realisieren. Der Empfänger kann mit den Bestätigungen sogenannte Window-Advertisements an den Absender schicken, um ihm mitzuteilen, das Fenster zu verkleinern oder zu vergrössern. Damit hat der Empfänger ein Mittel zu bestimmen, wie viele Daten ihm gesendet werden.

2.1.5 Überlastkontrolle

Eine Überlastung tritt ein, wenn ein Router zwischen zwei Endpunkten damit beginnt, Datagramme in Warteschlangen zu speichern oder Datagramme sogar zu verwerfen. Wie bereits beschrieben, reagiert TCP auf Verzögerungen mit Timeouts und darauf folgenden Wiederholungsübertragungen. Durch diese Vorgehensweise wird die Situation an einem überlasteten Router verschärft. Diese Situation wird als *Congestion Collapse* bezeichnet. TCP wendet nun verschiedene Verfahren an, um Überlastungen zu vermeiden oder notfalls diese schnell abzubauen. TCP verwendet dazu ein zweites Fenster, das sogenannte *Congestion-Window* (auch *Erlaubniszähler* genannt). Im Normalfall ist dieser Wert gleich dem Sliding-Window, das der Empfänger angefordert hat. Im Falle einer Überlastung wird das Congestion-Window um die Hälfte reduziert, minimal auf die Grösse von einem Segment. Das resultierende Fenster, das zur Übertragung angewandt wird, ist das Minimum vom Sliding-Window und vom Congestion-Window. Das Verfahren der exponentiellen Reduzierung des Congestion-Window wird als *Multiplicative Decrease Congestion Avoidance* bezeichnet. Gleichzeitig wird mit jeder Reduzierung des Congestion–Windows der Wiederholungstimer für die im zulässigen Fenster verbleibenden Segmente verdoppelt (Timer-Backoff-Strategie). Nach der Überlastung muss vermieden werden, dass sofort wieder eine Überlastung auftritt. Dies könnte z.B. geschehen, wenn TCP das Congestion-Window sofort wieder auf die Grösse des Sliding-Window setzt. Stattdessen wird das Slow-Start-Verfahren an-

gewandt. Dabei wird das Congestion-Window anfänglich auf die Grösse eines Segments gesetzt und bei jeder eintreffenden Bestätigung die Grösse des Congestion-Window um ein Segment erhöht. TCP kann zusätzlich ein zu schnelles Wachstum des Congestion-Window vermeiden, indem es, sobald das Congestion-Window wieder auf die Hälfte des Sliding-Window angestiegen ist, in eine sogenannte Phase der Überlastungsvermeidung eintritt. Dabei wird das Congestion-Window jeweils nur um ein Segment erhöht, wenn alle Segmente im aktuellen Fenster bestätigt worden sind.

2.1.6 Vermeiden von Paketverlusten

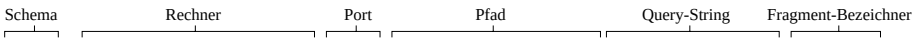
Bei Übertragung auf Medien mit niedriger Bandbreite ist es nicht nur wichtig, dass die Daten möglichst fehlerfrei ankommen (keine Paketverluste), sondern auch, dass die Übertragungsrate relativ konstant bleibt. TCP vergrössert die Übertragungsrate so lange, bis das Maximum überschritten ist und reduziert sie dann wieder. Dies führt allerdings dazu, dass beim Überschreiten Pakete verloren gehen (sie kommen nicht innerhalb der gesetzten Frist an) und das *Congestion–Window* halbiert wird, wodurch auch die Übertragungsrate halbiert wird. Aufgrund des *slow-starts* geht es relativ lange, bis die maximale Übertragungsrate wieder erreicht ist (siehe auch [28]). Entsprechend ist die Übertragungsrate immer am Schwanken. Die Überlegung geht nun dahin, dass Paketverluste durch die Überschreitung der maximalen Übertragungsrate vermieden werden können, indem eine Sende-Beschränkung gelegt wird, die knapp unterhalb der maximalen Bandbreite liegt. TCP erhöht die Übertragungsrate nun nur noch bis zu dieser Grenze. Somit kann vermieden werden, dass Pakete aufgrund von ungenügender Bandbreite verloren gehen und wiederholt gesendet werden müssen und es fällt das langsame Anpassen der Bandbreite beim daraus resultierende *slow-start* weg.

TCP ist ein sehr sicheres und verlässliches Protokoll. Die Aufwände um dies zu bewerkstelligen führen natürlich auch zu Nachteilen: Mehrere Forschungsprojekte und Messreihen [15, 27, 28] haben ergeben, dass die Massnahmen von TCP zur Sicherheit und zur Flusskontrolle den Durchsatz im Netz verringern. Besonders bei parallelen TCP-Strömen, wie dies bei der Übermittlung von Webseiten meist der Fall ist, können im Fall von Stau die einzelnen Ströme sich gegenseitig die Bandbreite streitig machen [20]. Durch das Limitieren der Bandbreite der einzelnen TCP-Ströme (welche vom Server aus nicht notwendigerweise nur zu einem Client führen müssen) können einerseits Staus vermindert werden, was auch zur Folge hat, dass weniger Pakete wiederholt gesendet werden müssen.

2.2 Uniform Resource Locators

Uniform Resource Locators (URL) sind die Standardmethode, um Ressourcen (Bilder, HTML-Seiten, CGI-Skripten, etc.) mit Hilfe des Hypertext Transfer Protokolls im Internet zu finden. URLs können aber auch Ressourcen anderer Protokolle bezeich-

nen. Manchmal wird auch der Begriff URI (Uniforme Recource Identifier [6]) benutzt. Eigentlich bezeichnet URI die Obermenge, welche neben URLs auch Uniform Resource Names (URN) beinhaltet, URNs sind aber im Netz nicht in Anwendung und nur vom Konzept her beschrieben, deshalb können die Begriffe URI und URL noch ausgetauscht werden.



http://www.iam.unibe.ch:4242/cgi/calender.cgi?monat=Januar#woche1

URLs bestehen aus einem Schema, einem Rechnernamen, einer Portnummer, Pfadinformationen, einem Query-String und einem Fragment-Bezeichner. Unter bestimmten Umständen kann jeder Teil des URL weggelassen werden.

- **Schema** Das Schema bezeichnet das zum Transport der Daten verwendete Protokoll. Die zur Zeit am häufigsten verwendeten Schemata sind: http:// (Hyper Text Transport Protocol), https:// (Secure HTTP) und ftp:// (File Transport Protocol). Oft wird ein Browser auch im offline-Modus verwendet, dann wird mit dem Schema file:// auf lokale Dateien zugegriffen. Weitere mögliche Schemata sind gopher:// (Dokumentübertragung und Navigation in Gopher-Systemen), mailto://, news:// und telnet://.

- **Rechner** Der Rechner-Teil der URL enthält den Namen des Computers, auf dem der Webserver läuft. Der Teil kann einen Domainname beinhalten (im obigen Beispiel iam.unibe.ch) oder aus einer IP-Adresse bestehen.

- **Portnummer** Die Portnummer muss in der Regel nicht angegeben werden, da den Schemata jeweils Standardports zugewiesen sind. Dies sind z.B. Portnummer 80 für http:// oder 443 für https://. Läuft ein Webserver wie in dieser Arbeit nicht auf dem Standardport, muss die entsprechende Portnummer nach dem Rechnernamen angegeben werden. Dabei sind Rechnername und Portnummer durch einen Doppelpunkt voneinander getrennt.

- **Pfadinformationen** Die Pfadinformation bezeichnet den Speicherort der angefragten Ressource. Dabei ist zu beachten, dass dieser Pfad nicht den absoluten Ort angibt sondern relativ zu einem konfigurierbaren Basisverzeichnis ist (z.B. /var/www/). Falls der Pfad mit einem ˜ Symbol beginnt, wird dieses Basisverzeichnis in der Regel auf das /public_html - Verzeichnis eines Benutzers gelegt. Die der Tilde unmittelbar folgende Zeichenkette bis zum Verzeichniskennzeichen wird dabei als Benutzername verwendet. An den Pfad können auch zusätzliche Informationen für auszuführende CGI-Programme übergeben werden.

- **Query-String** Der Query-String (manchmal auch als Index oder Such-String bezeichnet) kann Daten oder mehrere Paare von Namen und dazugehörigen Werten enthalten. Die im Query-String übermittelten Daten dienen meist als Parameter für CGI-Programme (siehe Abschnitt 2.4).

- **Fragment-Bezeichner** Fragment-Bezeichner bezeichnen eine bestimmte Stelle (Fragment) in der Ressource, werden aber nicht an den Server übermittelt und können deshalb nicht von CGI-Programmen verwendet werden. Sie dienen dem Browser dazu, bestimmte Stellen im Dokument aufzufinden. In HTML-Dokumenten beziehen sich die Fragment-Bezeichner auf die im Dokument befindlichen Anker-Tags.

2.3 Das Hypertext Transfer Protocol

Das Hypertext Transfer Protocol (HTTP) [5, 14, 6] ist die Sprache, in der Webbrowser und Webserver miteinander im World Wide Web (WWW) kommunizieren. Normalerweise wird man als Web-Nutzer die im HTTP festgelegten Abläufe nicht bemerken. Als Gestalter von Webseiten kann man allerdings mit Hilfe eines bestimmten Tags (<META>) zusätzliche HTTP-Mitteilungen festlegen, die für eine Seite an den Browser mitgesendet werden. Noch wichtiger sind Kenntnisse über HTTP für die Programmierung von CGI-Skripten, welche dynamische Seiten generieren.

HTTP legt das Format, den Inhalt von Mitteilungen zwischen Client und Server sowie deren Abfolge fest. Alle Mitteilungen werden zeilenweise als ASCII-Zeichenketten ausgetauscht, wodurch HTTP unabhängig von maschinenspezifischen Darstellungen von Zahlen wird.

2.3.1 Ablauf einer HTTP-Anfrage

Der Ablauf einer Kommunikation zwischen Client (Browser) und Server (Webserver) ist immer derselbe: Fordert der Browser eine Webseite an, so schickt er eine Nachricht, den Request, an den Webserver. Ein Request enthält einen Header und gelegentlich auch einen Nachrichten-Body. Der Server sendet daraufhin eine Antwortnachricht, die Response. Die Response enthält ebenfalls einen Header und in der Regel auch einen Nachrichtenbody. HTTP ist ein reines Request/Response-(Frage/Antwort)Protokoll. Jedem Response geht ein Request voraus. Die Struktur von Header/Body ist bei Request und Response gleich, auch wenn beide unterschiedliche Informationen enthalten. Der Header enthält sogenannte Metainformationen, d.h. Informationen über die Nachricht. Der Body enthält die Nachricht selbst. Die Anforderung der Beispielseite http://asterix.unibe.ch/~bechter/index.html ergibt somit folgende HTTP-Nachrichten:

Request:

```
GET http:/www.iam.unibe.ch/~bechter/index.html HTTP/1.1
Host: asterix.unibe.ch
Accept: image/gif, image/x-xbitmap, image/jepeg, image/pjepeg, image/xbm, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 4.6; Mac_PowerPC)
```

Response:

```
HTTP/1.1 200 OK
Date: Sat, 11. Dec 2003 16:34:31 GTM
Server: Apache/1.3.9 (Unix)
Last-Modified: Sat, 11. Dec 2003 16:32:19 GTM
ETag: "74916-554-3562efde"
Content-Length: 161
Content-Type: text/html

<HTML>
<HEAD>
<TITLE>Beispielseite</TITLE>
</HEAD>
<BODY BGCOLOR="FFFFFF" TEXT="#000000" LINK="#0000FF" VLINK="#00FF00"
ALINK="#FF0000" >
Beispielseite
</BODY>
</HTML>
```

Der Request hat in diesem Beispiel einen Header aber keinen Body, die Response hat sowohl Header wie einen Body im HTML-Format. Dabei werden Header und Body von einer Leerzeile getrennt.

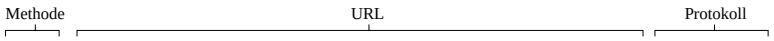
2.3.2 HTTP-Header

Der HTTP-Header baut auf dem MIME-Format [7] auf, hat sich aber anders weiterentwickelt. Wichtige Punkte beim HTML-Header sind:

- Die erste Zeile hat ein spezielles Format und wird entweder als Request-Zeile oder Status-Zeile bezeichnet.
- Die übrigen Header-Zeilen bestehen aus Name-Wert-Paaren, wobei der Name mit einem Doppelpunkt vom Wert getrennt ist. Diese Zeilen werden Header-Felder genannt.
- Bei Namen wird nicht auf Gross- oder Kleinschreibung geachtet.
- Die Header-Felder haben keine feste Reihenfolge
- Am Ende jeden Header-Feldes steht ein CRLF (Carriage-Return, Line Feed)
- Header und Body sind durch zwei CRLF getrennt.

2.3.3 Request-Zeile

Die Request-Zeile besteht aus mehreren Feldern, der Methode, der URL sowie Versionsangaben des verwendeten Protokolls:



GET http://www.iam.unibe.ch/~bechter/index.html HTTP/1.1

Request-Methoden müssen prinzipiell mit Grossbuchstaben angegeben werden. Nicht alle Webserver stellen alle Request-Methoden zur Verfügung. Nach der URL wird noch die Versionsnummer des verwendeten Protokolls angegeben. Die am meisten verwendeten Request-Methoden sind die folgenden:

- **GET** Der Server liefert die durch die URL identifizierte Seite zurück oder startet ein dadurch bezeichnetes CGI-Skript und liefert dessen Ausgabe als Ergebnis der Methode.
- **PUT** Der mit der Anforderungsmitteilung geschickte Inhalt soll unter der URL abgespeichert werden.
- **HEAD** Holt nur die HTTP-Header, aber keine Inhalte.
- **POST** Anweisung, auf dem Server abgelegte Informationen zu modifizieren (z.B. mit CGI).
- **CONNECT** Schaltet Proxies in einen Tunnelmodus um, z.B. um HTTP-Verbindungen via SSL zu ermöglichen.
- **TRACE** Weist den Server an, sämtliche übergebene Abfrage-Header unverändert an den Client zurückzusenden.

2.3.4 Status-Zeile

In der Status-Zeile (auch Response-Mitteilung genannt) werden wie bei der Request-Zeile Protokoll und Version angegeben, allerdings steht diese Information hier am Anfang und nicht am Ende. Nach einem Leerzeichen folgt ein dreistelliger Statuscode zusammen mit entsprechendem Text:

HTTP/1.1 200 OK

Webserver können diverse Statuscodes schicken. Es gibt fünf Gruppen von Statuscodes, wobei die Codes ab 200 angeben, dass die Methode erfolgreich ausgeführt wurde. Hier die wichtigsten Statuscodes:

- **1xx** Diese für HTTP 1.1 eingeführten Statuscodes werden auf einer unteren Ende des Protokolls benutzt.

- **2xx** Diese Statuscodes zeigen an, dass der Request erfolgreich ausgeführt wurde. *200 - OK* bedeutet, dass eine gültige Antwort sich im Rest der Response-Mitteilung befindet, *201 - Created* dass eine Seite erzeugt wurde (mit PUT oder POST).

- **3xx** Statuscodes, die mit einer 3 beginnen, weisen auf eine Umleitung hin - der Request ist gültig, die Ressource befindet sich jedoch an einem anderen Ort. Aufgrund dieser Codes sind weitere Aktivitäten des Clients nötig, um die Methode doch noch erfolgreich abzuschliessen.

- **4xx** Mit Nummer 4 beginnen Statuscodes, die angeben, dass die Anforderung eine falsche Syntax hatte oder dass die Methode nicht ausgeführt werden kann. Häufig treffen Surfer im Netz auf Fehlermeldungen wie *404 - Not Found*, wenn zum Beispiel ein Link nicht richtig gesetzt ist und entsprechend die angeforderte Seite nicht gefunden wurde, oder auf *403 - Forbidden*, wenn man keine Berechtigung hat, auf eine Seite zuzugreifen.

- **5xx** Wenn eine Anforderung zwar korrekt ist, aber der Server die Methode aus verschiedenen Gründen nicht ausführen kann, dann werden Statuscodes ab 500 zurückgemeldet. Oft gibt es die Fehlermeldung *500 - Internal Server Error*, die auf einen internen Fehler, beispielsweise auf einen Programmierfehler, hinweist.

2.3.5 Inhaltsinformation

Bei Requests und Responses kann ein Inhalt die Mitteilung beschliessen. Ein Client könnte bei *PUT* eine HTML-Seite mitschicken und der Server antwortet bei *GET* mit einer Datei. Die Header, die in einer Antwort zur Beschreibung des Inhalts verwendet werden, sind wie folgt:

- **Content-Type:** ***Medienart*** Gibt an, von welcher Medienart der Inhalt der Antwort ist. Die Medientypen entsprechen dem MIME-Standard [8], so dass für eine normale HTML-Textseite *text/html* steht, für eine Postskriptseite *text/postscript*.

- **Content-Length:** ***Länge*** Der Wert gibt die Länge des Inhaltes in Bytes an.

- **Content-Encoding:** ***Kodierung*** Der Inhalt wird in einer Inhaltskodierung übertragen (z.B. 8bit, base64, gzip,...).

- **Content-Language:** ***Sprachkürzel*** Der Server kann hier dem Client die Sprache des Inhaltes mitteilen. Mit dem Sprachkürzel (und evtl. einem Ländercode) nach ISO-Norm können Sonderzeichen direkt eingegeben werden. *lang=DE-CH* im

HTML-Code bewirkt zum Beispiel, dass der Browser Umlaute wie *ä* direkt interpretieren kann.

- **Expires:** ***Datum*** Zum angegebenen Datum verliert der Inhalt seine Gültigkeit. Clients und Proxies können diese Information dazu nutzen, abgelaufene Inhalte aus dem Cache zu löschen.
- **Last-Modified:** ***Datum*** Die Information wurde am angegebenen Datum zuletzt modifiziert. Dadurch kann man veraltete Kopien einer Seite identifizieren.
- **Titel:** ***Titel*** Der Wert des Feldes ist der selbe wie in der HTML-Seite, der Client kann somit den Titel schon darstellen, bevor eine Seite komplett eingetroffen ist.
- **Location:** ***URL*** Falls eine Seite untere mehreren URLs (evtl. auch unter verschiedenen Versionen) verfügbar ist, kann der Server hier eine Liste solcher Adressen mitsenden und über Unterschiede Auskunft geben. Sollte das URL-Schema erweitert werden, wird dieses Feld durch **URI:** ***???*** ersetzt.
- **Allow:** ***Methoden*** Falls der Server angibt, dass bestimmte Methoden nicht auf einer Seite ausgeführt werden kann, schickt er diese zusätzliche Information, worin die erlaubten Methoden aufgeführt sind.

2.3.6 Neuladen von Dokumenten (Client-Pull und Server-Push)

Das Internet und das HTTP-Protokoll sind sehr statisch. Nach der Übertragung kann man Seiten nicht mehr dynamisch ändern um beispielsweise kleine Animationen einzubauen. Netscape hat nun zwei Mechanismen eingeführt, mit denen automatisches Neuladen von Dokumenten und kontinuierliche Änderungen an HTML-Seiten während ihrer Anzeige möglich sind: Client-Pull und Server-Push.

Client-Pull Die Idee beim Client-Pull ist, dass der Browser selbsttätig eine bestimmte Seite neu lädt. Dabei kann eine Zeit in Sekunden angegeben werden, nachdem der Browser mit dem Neuladen beginnt. Der Mechanismus basiert auf einem zusätzlichen Header in der Antwort des Servers: **Refresh:** ***Sekunden;URL***. Erhält der Browser eine Antwort-Mitteilung mit diesem Header, wartet er *Sekunden* und lädt dann automatisch die Seite *URL* nach, wobei bei jedem Nachladen eine neue TCP-Verbindung aufgebaut wird. Dieser Mechanismus wird in dieser Arbeit vor allem zum Weiterleiten des Benutzers bei den Konfigurationsseiten und bei der Bandbreitenbestimmung verwendet. **Server-Push** Im Gegensatz zum Client-Pull, wo sich der Browser die Daten vom Server holt, sendet der Server beim Server-Push-Mechanismus kontinuierlich Daten während einer Verbindung. Dazu benutzt er den speziellen MIME-Typ *multipart/x-mixed;boundary=Site*. Server-Push wird verwendet, wenn auf einer Seite Teile ständig erneuert werden sollen, wie zum Beispiel bei einer Seite, die ein kleines Bild einer Webkamera überträgt.

2.3.7 Secure Socket Layer

HTTP ist ein unsicheres Protokoll, es ist sehr einfach, die Kommunikation von zwei Computern über dieses Protokoll zu verfolgen. Dies ist bedenklich, wenn man über das Netz Authentifizierungsdaten, Kreditkartennummern oder andere sensible Daten senden will. Daher wurde von Netscape das Protokoll Secure Socket Layer (SSL) entwickelt, welches einen sicheren Kanal für die HTTP-Kommunikation zur Verfügung stellt. SSL ist inzwischen als IETF-Standard [25] anerkannt unter der offiziellen Bezeichnung TLS (Transport Layer Security). Fordert der Browser eine URL an, die mit https beginnt (Schema), so wird eine SSL/TSL-Verbindung zum Server aufgebaut, über die dann die HTTP-Transaktionen durchgeführt werden.

2.3.8 Mehrfachabfragen

HTTP übermittelt nur das angeforderte Dokument. Oft ist in diesem Dokument ein sogenannter Link integriert, welcher auf eine andere Textpassage, eine andere Datei, ein Bild oder eine andere Ressource zeigt. HTTP kümmert sich nicht um den Inhalt der übermittelten Daten, dies ist die Aufgabe des Client-Browsers. Dieser analysiert die enthaltene Datei und stellt sie auf dem Bildschirm dar. Ist nun eine Referenz in der Datei enthalten, so muss der Browser diese anfordern. Auch wenn sie auf dem selben Server abgelegt ist wie die bereits erhaltene Datei, muss der Browser doch eine neue TCP-Verbindung zum Webserver aufbauen und die Datei über HTTP anfordern.

2.4 Common Gateway Interface

Für viele Anwendungen, die heute im Web alltäglich sind (wie z.B. das Auswerten von Benutzereingaben, Warenkörbe in E–Commerce Anwendungen, etc.) ist es unabdingbar, dass der Webserver in der Lage ist, eigenständig Programme auszuführen. Dies geschieht mit Hilfe von Skript-Programmen, die als CGI-Skripte oder kurz CGIs bezeichnet werden. Das Common Gateway Interface (CGI), kann als eine Art Schnittstelle zwischen dem Webserver und den dort abgelegten Programmen angesehen werden. Die Kommunikation zwischen CGI Programm und Webserver geschieht über eine Reihe festgelegter Umgebungsvariablen. Erhält ein Webserver eine Anfrage nach einer statischen Webseite, sucht er das entsprechende HTML-Dokument in seinem Dateisystem und sendet es an den Browser zurück. Übergibt man dem Webserver eine Anfrage nach einem CGI-Skript, führt er das Skript als eigenen Prozess (z.B. ein externes Programm) aus. Der Server übergibt diesem Prozess einige Parameter und sammelt dessen Ausgabe wieder ein. Die Parameterübergabe erfolgt in Abhängigkeit von der HTTP Methode, welche beim Aufruf des CGI Programms gewählt wurde. Diese Daten werden dann an den Client zurückgegeben, als wären die Daten aus einer statischen Datei gekommen (siehe Abbildung 2.3).

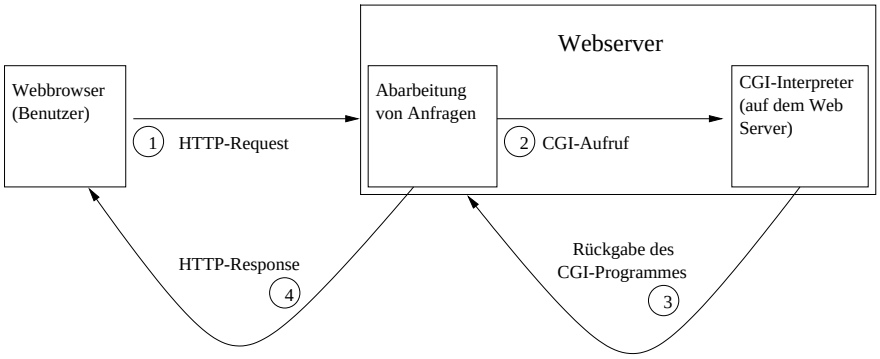


Abbildung 2.3: Ausführung eines CGI-Programmes

CGI baut auf HTTP auf. Es ist unter anderem deshalb so mächtig, weil es das Manipulieren der Metadaten ermöglicht, die zwischen Browser und Server ausgetauscht werden. Damit werden unter anderem folgende Aktionen möglich:

- Es können an die Bedürfnisse des Clients angepasste Inhaltstypen, Sprachen oder andere Codierungen zur Verfügung stellen
- Man kann die URL der letztbesuchten Seite des Benutzersherausfinden
- Die Ausgaben können dem verwendeten Browsertypen angepasst werden
- Es kann festgelegt werden, wie lange ein Dokument auf dem Rechner des Benutzers zwischengespeichert werden soll, bevor eine neue Version dieses Dokumentes geladen werden muss

Beispiel: Ein Client (z.B. der Browser) fordert eine eine Seite an, deren URL auf ein vom Webserver auszuführendes Programm verweist. Dieses befindet sich in einem dem Server bekannten Verzeichnis. Daraufhin führt der Server das Skript oder Programm aus. Das Skript erzeugt daraufhin einen Antwort-Header, eine Leerzeile und den Inhalt der Antwort. Die Ausgabe des Programms erfolgt auf der Bildschirmausgabe, die vom Client umgeleitet wird.

2.4.1 Umgebungsvariablen

CGI-Skripte können vordefinierte Umgebungsvariablen benutzen, in denen Informationen über den Webserver und die Clients zu finden sind. Viele dieser Informationen stammen aus den HTTP-Headern der Requests. Von allen möglichen Variablen ist je nach Request nur ein Teil belegt, manche werden nur bei bestimmten Arten von Requests gesetzt. Wichtige Umgebungsvariablen sind:

- **QUERY_STRING**: Die dem Skript übergebene Abfrage (wird mit einem Fragezeichen an die URL angehängt).
- **REMOTE_ADDR**: Die IP-Adresse des Clients, von dem der Request kommt. Dies kann unter Umständen auch die IP-Adresse eines Proxies sein, welches sich zwischen Client und Server befindet.
- **REQUEST_METHOD**: Die Art des HTTP-Requests.
- **CONTENT_LENGTH**: Die Menge der Daten (in Bytes), die über STDIN an das CGI-Programm übergeben werden.
- **CONTENT_TYPE**: Der Medientyp des Nachrichten Body, z.B. *application/x-www-form-urlencoded*.
- **PATH_INFO**: Zusätzlich an das CGI-Programm übergebene Pfadinformationen

Zusätzlich können CGI auch HTTP-Header übergeben werden, die vom Webserver nicht als solche erkannt, aber trotzdem weitergeleitet werden. Ihnen wird das Präfix *HTTP_* vorangestellt. Eine kleine Auswahl von solchen Headern, welche in dieser Arbeit benutzt werden:

- **HTTP_ACCEPT**: Eine Liste der vom Client akzeptierten Medientypen.
- **HTTP_COOKIE**: Name-Wert-Paar, welches zuvor vom Server gesetzt wurde.
- **HTTP_REFERER**: Die URL des Dokumentes, auf welches der Benutzer zugegriffen hat, bevor er das CGI-Programm aufgerufen hat (z.B. ein Hyperlink oder ein Formular). Diese Mitteilung kann statistisch oder anders von grossem Nutzen sein, ist aber nicht zuverlässig, da sie relativ leicht fälsch- oder unterdrückbar ist.
- **HTTP_USER_AGENT**: Name und Versionsnummer des Clients, von dem der Request ausging.

Es ist auch möglich, Webservern in der Konfigurationsdatei zusätzliche Umgebungsvariablen anzugeben. Dies kann nützlich sein, wenn viele CGI-Skripte mit der gleichen Konfiguration arbeiten sollen, wie etwa mit dem Namen eines Datenbankservers.

2.4.2 Ausgabe von Dokumenten

Der Datenfluss vom CGI-Programm zum Server erfolgt stets über die Standardausgabe des Programms. Meist geben CGI-Skripte HTML-formatierte Dokumente zurück. Bevor ein Skript aber Inhalte zurückgibt, muss es dem Server in einer Headerzeile mitteilen, welchen Medientyp die Inhalte haben sollen. Bei einer normalen HTML-Seite

muss deshalb erst die Zeile *Content-type: text/html* gesendet werden, die dem Server den Inhaltstypen mitteilt. Danach folgen zwei Leerzeilen, die den Header abschliessen und den Nachrichtenteil einleiten.

2.4.3 Parameter und Parameterübergabe

CGI definiert Parameter, die die Anfrage eines Clients beschreiben. Neben den Umgebungsvariablen können dies Eingaben von Formularen sein, welche an das CGI-Programm weitergegeben werden. Bevor der Browser die Formulardaten an den Server senden kann, müssen sie vom Browser codiert werden. Fast auschliesslich wird dafür der Medientyp *application/x-www-form-urlencoded* benutzt. Jedes HTML-Formularelement besitzt Attribute für Namen und Werte. Als erstes sammelt der Browser die Namen und Werte für jedes Formularelement. Danach codiert er jede dieser Zeichenketten, indem alle Zeichen, die in HTTP eine Sonderbedeutung haben oder nicht im ASCII-Zeichensatz enthalten sind (z.B. ä ö ü ; / ? : & = + $) durch ein Prozentzeichen und zwei hexadezimale ASCII-Nummern ersetzt werden. Leerzeichen werden durch ein + ersetzt. Die Zeile *Danke für Alles!* sähe codiert so aus: *Danke+f%FCr+Alles%21*. Als nächstes verbindet der Browser jedes Name-Wert-Paar mit einem Gleichheitszeichen. Gibt der Benutzer beispielsweise als Namen den Wert *bechter* an, so wäre das Ergebnis *Name=bechter*. Die einzelnen Name-Wert-Paare werden mit dem &-Zeichen miteinander verbunden
(Bsp. *name=bechter&email=bechter@iam.unibe.ch*).

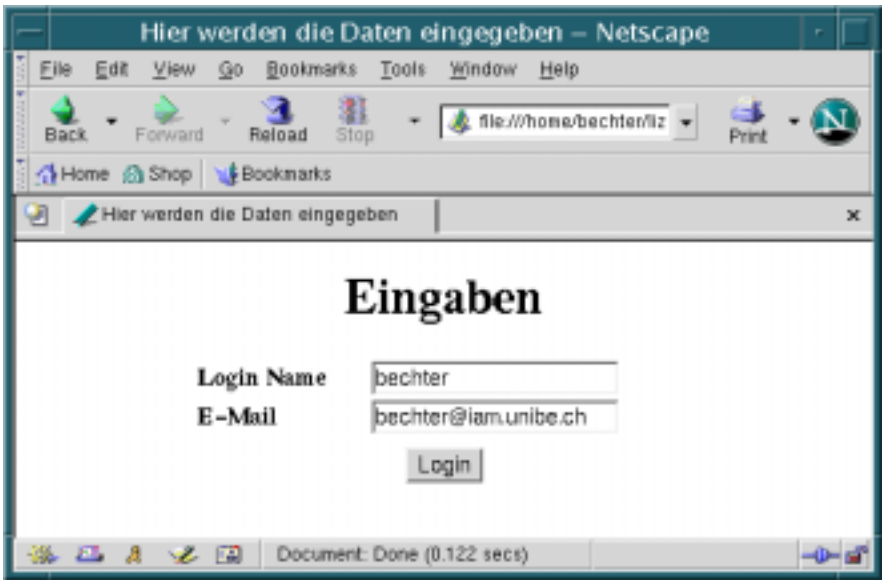


Abbildung 2.4: Beispiel eines Eingabefensters

Wird das Formular (siehe Abbildung 2.4) abgeschickt, codiert der Browser die Angaben folgendermassen: *name=bechter&email=bechter%40iam.unibe.ch*

- **POST** Mit der POST-Methode wird die erzeugte Zeichenkette als Nachrichten-Body des HTTP-Requests übergeben. Der komplette HTTP-Request sähe dann so aus:

```
POST /cgi-bin/namenscgi.cgi HTTP/1.1
Host: localhost
Content-Length: 43
Content-Type: application/x-www-form-urlencoded

name=bechter&email=bechter\%40iam.unibe.ch
```

- **GET** Hätte man statt dessen die GET-Methode benutzt, sähe der Request folgendermassen aus:

```
GET /cgi-bin/namenscgi.cgi?name=bechter&email=bechter\%40iam.unibe.ch
HTTP/1.1
Host: localhost
```

2.5 Hyper Text Markup Language

Hyper Text Markup Language (HTML [11]) ist die Beschreibungssprache für Dokumente im World Wide Web und bildet zusammen mit HTTP die Grundlage des Internets. Es ist eine *Auszeichnungssprache*, mit der man lediglich markiert, wie bestimmte Textpassagen aussehen sollen. Die Auszeichnungsmarkierungen werden erst formatiert, wenn der Browser des Clients die Seite darstellt. Da jeder Benutzer seinen Browser nach den eigenen Wünschen konfigurieren kann, bestimmen verschiedene Zeichensätze und Fenstergrössen das tatsächliche Aussehen der Seite. Eine HTML-Seite besteht aus fortlaufendem ASCII-Text, in dem der eigentliche Inhalt der Seite und die Auszeichnung des Inhalts gemischt ist. Die Markierung von Textpassagen mit Formatierungskommandos geschieht durch sogenannte ***Tags*** (*Marken*). Alle HTML-Tags werden von den spitzen Klammern $<$ und $>$ eingeschlossen. Auch Graphiken oder Applikationen werden mit Hilfe von Tags in die Seite eingebettet. Ein wichtiges Tag ist auch der *Link*, der dazu dient, andere Dokumente oder Dokumentteile zu referenzieren: Wenn man im Browser den Link anwählt, wird das entsprechende Dokument angefordert.

2.6 Web Server

Ein Webserver stellt den Benutzern des Internets Daten und Dienste zur Verfügung. Dabei muss er in erster Linie in der Lage sein, das HTTP-Protokoll zu verstehen und

einzuhalten, nur so ist eine weltweite Einheit zu erreichen. Welche Dienste ein Webserver sonst noch zur Verfügung stellen sollte, ist nicht klar festgelegt: Weder im RFC für HTTP noch in dem für HTML werden irgendwelche Anforderungen an den Server gestellt. Es ist somit dem Betreiber des Servers freigestellt, welche Dienste er zur Verfügung stellen will. Der Apache-Server wird weltweit am meisten benutzt, Umfragen haben eine Verbreitung von über 65% ergeben. Er erfüllt sämtliche Anforderungen, die in den RFCs (und darauf folgenden Errata) von HTTP [5, 14, 6], HTML [11] und Authentisierung [16]. Weiter werden auch die Sprachencodes nach ISO 639-2, die Länderkürzel nach ISO 3166-1, die Best Current Practice [3] und die Standard Tracks [4] unterstützt .

2.6.1 Proxy

Als Web-Proxy versteht man ein dazwischengeschaltetes Programm, welches sowohl als Server als auch als Client arbeitet, um Requests an Stelle von Clients durchzuführen. Requests können dabei direkt verarbeitet oder an andere Server weitergeleitet werden. Im letzteren Fall können die Requests eventuell abgeändert werden. Ein Web-Proxy muss laut RFC 1945 [5] die Bedürfnisse von Server als auch Client erfüllen. Als transparente Proxies gelten solche, welche die Requests nicht verändern und vom Benutzer unbemerkt arbeiten. Nicht transparente Proxies können Requests verändern um zusätzliche Dienste wie Filterprogramme oder Medientyp-Veränderungen anzubieten.

2.7 Prozesse und Interprozesskommunikation

Ein Programm während der Ausführung wird Prozess genannt. Ein Prozess ist unter anderem durch die benutzten Ressourcen definiert. Ein Prozess kann (im UNIX Betriebssystem mit dem `fork` Befehl) weitere Prozesse starten, sogenannte Kind-Prozesse (child process). Ein solcher Kind-Prozess ist anfangs fast eine identische Kopie des Elternprozesses. Sie unterscheiden sich nur durch ihre Prozessidentifikationsnummer und ihre Ressourcenbenützung (siehe dazu auch die `fork` Dokumentation [32]). Sämtliche weiteren Daten, die dem Elternprozess zur Verfügung standen, stehen somit auch dem Kind-Prozess zur Verfügung. Da der Kind-Prozess allerdings auf einer unabhängigen Kopie der Daten arbeitet, können die Änderungen vom Elternprozess nicht eingesehen werden.

Der so angelegte Kind-Prozess kann nun als unabhängige Instanz des Elternprozesses dessen Aufgaben wahrnehmen: bei einem Server (z.B. Webserver, Datenbankserver), der parallel mehrere Anfragen gleichzeitig verarbeiten muss, ist es beispielsweise üblich, pro Anfrage einen Kind-Prozess zu starten, der dann die Anfrage autonom bearbeitet. Andererseits kann der Kind-Prozess auch dazu dienen, ein völlig neues Programm zu starten. Dazu ersetzt man sämtliche Information des Kind-Prozesses (d.h. Code- und Datensegment, Stack, ...) mit denen eines neuen Prozesses (siehe dazu

die `exec` Dokumentation [31]). Dies ist beispielsweise der Fall, wenn aus einem Programm eine Shell aufgerufen wird, in der dann weitere Kommandos ausgeführt werden können.

Wird ein Kind-Prozess beendet, so wird dies dem Elternprozess mittels eines Signals SIGCHLD mitgeteilt. Es ist dann die Aufgabe des Elternprozesses, die vom Kind-Prozess benutzten Ressourcen wieder freizugeben (siehe dazu die `wait` Dokumentation [36]). Ein Kind-Prozess, der nicht auf diese Weise beendet wurde, bleibt als sogenannter *Zombie* in der Prozesstabelle bestehen, und die Ressourcen können nicht freigegeben werden. Dies geht so lange gut, bis die Prozesstabelle gefüllt ist und das System zusammenbricht.

Es gibt aber auch Situationen, wo es sinnvoll ist, dass Ressourcen geteilt werden können. Leichtgewichtige Prozesse (auch Threads genannt) teilen Programmcode, Ressourcen und globale Daten. Für sämtliche Anfragen, die ein Webserver erhält, wird ein leichtgewichtiger Kind-Prozess (Child-Thread) gestartet. In der Regel können diese Prozesse ausgeführt werden ohne sich gegenseitig zu beeinflussen, teilen jedoch den Programmcode, Ressourcen und globale Variablen.

2.7.1 Shared Memory

In der Regel erhält jeder Prozess Speicher für seine Ausführung zugeteilt. Auf die in diesem Speicher abgelegten Daten haben die anderen Prozesse keinen Zugriff. Manchmal ist es jedoch nötig, dass gemeinsamer Speicherplatz (Shared Memory) angelegt wird. Dies ist in dieser Arbeit zum Beispiel der Fall bei der Berechnung der Bandbreite: Das Hauptprogramm (Parent Process) muss Zugriff auf die Daten des Kind-Prozesses haben. Da der Rückgabewert des Kind-Prozesses nicht genügt, muss das Hauptprogramm schon während der Ausführung des Kind-Prozesses Zugriff auf dessen Daten haben. Dazu wird ein gemeinsamer geteilter Speicher angelegt. Darauf können sämtliche Kind-Prozesse und auch der Haupt-Prozess zugreifen. Ein Problem mit dem gemeinsamen Speicher ist die Zugriffskontrolle. Bei dieser Implementation ist es so geregelt, dass immer nur ein Prozess schreibend auf den gemeinsamen Speicher zugreift. Dies ist mit einer Variablen *busy* geregelt.

2.7.2 Semaphoren

Es kann vorkommen, dass Prozesse gleichzeitig auf gewisse Ressourcen zugreifen wollen, die nur von einem Prozess gleichzeitig benutzt werden können. Diese Situation kommt in dieser Arbeit an der Stelle vor, wo mehrere TCP-Verbindungen gleichzeitig Daten vom Server verlangen und senden wollen. Dieses Problem wird mit der Hilfe von Semaphoren gelöst. Semaphore sind Integer-Variablen auf denen zwei Operationen *up* und *down* definiert sind. *up* und *down* sind Verallgemeinerungen von *sleep* und *wakeup*. Wird *down* auf eine Variable mit grösser als 0 angewandt, so wird der

Wert um 1 erniedrigt. Wird *down* auf 0 angewandt so legt sich der Prozess schlafen. Die Operation *up* angewandt auf einen Semaphor der den Wert 0 enthält, weckt einen (zufällig ausgewählten) Prozess auf, der sich bezüglich des Semaphors schlafen gelegt hat, und weckt ihn auf. Der Wert bleibt 0. Ist kein Prozess bezüglich des Semaphors blockiert, wird der Semaphor um 1 erhöht.

2.8 Möglichkeiten zur Bestimmung der Bandbreite

Die Bestimmung der Bandbreite einer Verbindung zwischen einem Webserver und einem Client ist aus mehreren Gründen keine einfache Aufgabe: zum einen ist durch das IP Protokoll nicht sichergestellt, das Pakete vom Client zum Server genau denselben Weg nehmen wie Pakete vom Server zum Client. Man muss also schon aus diesem Grund zwischen der Bandbreite zum Server (Uplink) und der Bandbreite vom Server (Downlink) unterscheiden. Dieser Unterschied wird umso wichtiger, falls zusätzlich eine asymmetrische Verbindungstechnik (z.B. ADSL) verwendet wird. Diese Art der Verbindung hat in letzter Zeit mehr und mehr an Verbreitung zugenommen. Der Unterschied zwischen Up- und Downlink kann dabei beträchtlich sein (z.B. Uplinkbandbreite 100 kbit/s, Downlinkbandbreite 600 kbit/s).

Für unsere Anwendung ist jedoch nur der Downlink von Interesse, da der Client im Normalfall nur wenig Daten an den Server senden wird (z.B. Anfragen von Webseiten oder Einträge in Formulare). Die Geschwindigkeit der Downlinkverbindung kann auf verschiedene Weise gemessen werden. Dabei muss man zwischen Verfahren unterscheiden, welche die Kooperation des Clients voraussetzen, und solchen, die ohne diese Hilfe auskommen. Im Allgemeinen sind erstere Verfahren genauer, aber weniger benutzerfreundlich, da der Benutzer vorab Software auf seinem Client installieren und konfigurieren muss. Des weiteren könnte es zu Konflikten mit Sicherheitsregelungen kommen. Letztere Verfahren sind benutzerfreundlich, da man sie ohne weitere Softwareinstallation aufrufen kann, sie haben allerdings den Nachteil, dass die Genauigkeit leidet, da beide Verbindungen (Up- und Downlink) in die Messung mit einbezogen werden. Diesen Fehler kann man dadurch minimieren, dass man das Verhältnis der übertragenen Datenmenge stark zugunsten des Downlinks verschiebt.

3 Related Work

3.1 Analyse und Modellierung von Verkehrsdaten

Mehrere Arbeiten beschäftigen sich mit der Analyse und Modellierung von Weitverkehrsdaten [50, 41]. Zunächst lässt sich feststellen, dass TCP den überwältigenden Anteil des heutigen Netzwerkverkehrs ausmacht (ca. 90 - 95 % [50]). Dies gilt sowohl für die Grössen Bytes/Sekunde, als auch für Pakete/Sekunde und Übertragungen/Sekunde. Ein ähnliches Verhalten kann man für Webdaten beobachten, welche einen Anteil von 60 – 65 % bei allen drei Messgrössen besitzen. Dass dieser homogene Anteil keine Selbstverständlichkeit ist, zeigt das Beispiel von DNS: Dessen Anteil am Bytevolumen ist verschwindend gering (nur 1 – 3 %) wohingegen es einen erheblichen Anteil an den Verbindungen stellt (ca. 30 %). Dies lässt sich dadurch erklären, dass für fast jede Verbindung eine DNS-Anfrage notwendig ist, deren Grösse jedoch nur wenige Bytes beträgt.

- **Wide-Area Traffic: The Failure of Poisson Modeling [41]** In dieser Arbeit wurde gezeigt, dass das Verhalten des Datenverkehrs praktisch aller Internetapplikationen (Telnet, FTP, HTTP) nicht einer einfachen Poisson-Verteilung genügt, sondern dass kompliziertere Verteilungen beigezogen werden müssen. Am besten kann das Verhalten von Internetverkehr durch Verteilungen mit einem “heavy tail” beschrieben werden. Das bedeutet, dass der Anteil an grossen Paket- und Datenvolumina grösser ist, als es die Poisson-Verteilung nahelegt. Diese Tatsache untermauert auch die für diese Arbeit wichtige Feststellung, dass Webdaten kein glattes Verhalten zeigen, sondern dass häufig sogenannte “Bursts”, also Perioden mit hoher Netzbelastung auftreten. Diese Bursts führen in allen TCP-Applikationen, also auch im Webbrowser zu einer Reduktion der Bandbreite und damit zu unerwünschten Qualitätseinbussen.

- **Wide-Area Internet Traffic Patterns and Characteristics (Extended Version) [50]** In dieser Arbeit wird ebenfalls beschrieben, dass die durchschnittliche Grösse einer Webverbindung nur bei 15 - 17 Paketen von insgesamt ca 10 kByte ist. Die Dauer der Verbindung beträgt dabei durchschnittlich ca. 12 Sekunden. Dabei werden zu Spitzenzeiten 200000 Verbindungen über einen einzigen Link übertragen. Es ist daher kaum machbar, eine individuelle Optimierung des Bandbreitenverhaltens einer Webverbindung im Inneren des Netzes durchzuführen. Die einzelnen Router wären durch die schiere Menge der gleichzeitig zu verarbeitenden Daten schlicht überlastet. Sämtliche Veränderungen am Verhalten müssen daher an den Endpunkten der Kommunikation (d.h. Webserver und Webclient) geschehen.

3.2 Unzulänglichkeiten der Flusskontrolle von TCP

Das TCP-Protokoll verdankt seine grosse Verbreitung über die letzten Jahre unter anderem seinen Fähigkeiten, mehreren Benutzern gleichzeitig einen gerechten Anteil der zur Verfügung stehenden Bandbreite zuzuteilen und dabei das Netzwerk nicht zu überlasten. Die dazu verwendeten Algorithmen gehen jedoch von Voraussetzungen aus, die in modernen Kommunikationsnetzen, insbesondere Hochgeschwindigkeits-Weitverkehrsnetzen, und drahtlosen Zugangsnetzen nicht mehr zutreffen. Dazu zählt insbesondere die Fehlerrate, die in beiden Fällen nicht so gering ist wie vom TCP Algorithmus angenommen (im Falle von Hochgeschwindigkeitsnetzen liegt dies am grossen Bandbreite x RoundTripTime (RTT) Produkt). TCP interpretiert ein fehlerhaftes Paket stets als Hinweis einer Netzwerküberlastung und halbiert die Fenstergrösse (und damit die Übertragungsrate) um die Hälfte. Dies führt im Mittel zu einer inakzeptabel schlechten Auslastung der zur Verfügung stehenden Bandbreite.

Sehr viele Ansätze untersuchen die Möglichkeit, durch Änderung im TCP-Algorithmus den Durchsatz im Netz zu verbessern. Beispielhaft zu nennen wären:

- **Scalable TCP: Improving Performance in Highspeed Wide Area Networks [29]** In Hochgeschwindigkeits-Weitverkehrsnetzen besteht das Problem, dass TCP einen schlechten Durchsatz erreicht auf Grund des Slow-Starts: Bei einer 10 Gigabit- Verbindung und einer Round-Trip-Time von 0.2s dauert es nach einem Paketverlust 4 Stunden und 43 Minuten, bis wieder die volle Sendeleistung erreicht wird. Durch eine Änderung in der Anpassung des Congestion-Windows nach einem Paketverlust werden bis zu 100% höhere Durchsatzraten erzielt.

- **Improving Throughput and Maintaining Fairness using Parallel TCP [20]** Die Arbeit beschreibt einen Ansatz, der es ermöglichen soll, mit Hilfe von Round-Trip-Time Fairness zwischen TCP-Strömen zu erreichen, wenn das Netz überlastet ist. Wenn das Netz unterbelastet ist, soll der Ansatz die unbenutzte Bandbreite effektiver nutzen. Dies wird getan, indem verhindert wird, dass TCP das *congestion window* schnell erhöhen kann. Weiter wird die Frist mit einer künstlichen Variable hoch gehalten, so dass virtuell eine längere Antwortzeit erreicht wird. Bei Überlast wird nun nicht ein höherer Durchsatz erreicht, die parallelen TCP-Verbindungen konkurrieren jedoch weniger um die vorhandenen Ressourcen.

- **Achieving High Throughput in Low Multiplexed, High Bandwidth, High Delay Environments [26]** In dieser Arbeit wird wie bei [20] Bandbreitenverlust aufgrund von *Slow-Start* in Netzen mit hoher Bandbreite behandelt. Der Ansatz beruht auf der Benutzung des *ECN-Bits* (siehe [45]). Mit der Hilfe des ECN-Bits wird unterschieden *milder* Überlast (Pakete können noch gepuffert werden) und normaler Überlast (Paketverlust). Bei milder Überlast wird das *Congestion-Window* nicht verkleinert, die Grösse wird *eingefroren*. Dabei wird beim Eintreffen eines ECN-Bits das Vergrössern der Fenstergrösse angehalten, was zur

Vermeidung von Überlast führen kann. Der Ansatz von TCP/E führt zu einer guten Auslastung der Links.

- **Performance Evaluation of TCP over WLAN 802.11 with the Snoop Performance Enhancing Proxy [39]** In drahtlosen Netzen eignet sich TCP wenig zum Transport, da es bei Paketverlust auf Überlast schliesst und entsprechend die Bandbreite reduziert. Diese Reaktion ist für drahtlose Netze wenig geeignet, weil der Paketverlust hier andere Ursachen haben kann. Diese Arbeit schlägt nun deshalb eine Verbesserung mit Hilfe eines *Performance Enhancing Proxy* (PEP) vor, welches die Aufgabe von TCP auf dem drahtlosen Link übernimmt, ohne das Paketformat zu verändern. Dabei werden die ankommenden TCP-Pakete im Proxy gespeichert und mit dem *Snoop-Protocol* über den drahtlosen Link gesendet. Im Unterschied zum TCP-Mechanismus wird bei einem Paketverlust das *Congestion-Window* nicht halbiert und auch die Frist wird nicht angepasst. Weiter wird der Paketverlust nicht dem Sender mitgeteilt, die Pakete werden nur lokal erneut übertragen. Simulationen haben ergeben, dass je höher der Paketverlust ist, desto besser ist die Verbesserung in der Übertragungsrate: bei einem Paketverlust von 30% erreicht das *Snoop-Protocol* eine 68-fache Verbesserung durch das Caching und das Setzen einer lokalen Frist. Die Resultate dieser Arbeit sind sehr vielversprechend könnten mit den Resultaten meiner Arbeit kombiniert werden, sofern diese in drahtlosen Netzen zum Einsatz kommt.

- **A Split Stack Approach to Moblility-Providing Performace-Enhancing Proxies [12]** In dieser Arbeit wird mit dem *split-stack* Ansatz ein Proxy auf der Anwendungsschicht vorgeschlagen, um den TCP-Durchsatz auf dem letzten Abschnitt zu erhöhen. Dabei laufen die Applikationen auf dem Client, alle Netzwerkaufrufe geschehen durch das Proxy. Aufgrund ähnlicher Implementation in diesem Bereich bringt dieser Ansatz ähnliche Vorteile bei der TCP-Fehlerbehandlung wie beim *Snoop Protocol* (siehe [39]). Weiter kann durch das Einsetzen des Proxies eine Mobilität wie bei Mobile IP erreicht werden. Der Ansatz ist interessant, weil er neben der Durchsatzvergrösserung auch andere Möglichkeiten bietet.

- **Enhancements and Performance Evaluation of Wireless Local Area Networks [46]** In dieser Arbeit werden verschiedene Ansätze untersucht, die den Paketverlust in drahtlosen Netzwerken verbessern sollen. Dies wird mit dem Verändern von Parametern des CSMA/CA-Protokolls gemacht (*Slot Time*, *SIFS* und *Minimum Contention Window*). Die Resultate sind gut, allerdings lassen sie sich nicht übernehmen, da unsere Arbeit nicht nur das CSMA/CA Protokoll bei W-LAN betrifft. Eine Kombination der beiden Ansätze ist jedoch möglich, allerdings wohl eher unwahrscheinlich, da die eine Arbeit auf Applikationsebene, die andere auf der Sicherungsschicht aufsetzt.

Alle vorgestellten Ansätze konzentrieren sich jedoch darauf, die Performance von TCP

im jeweils untersuchten Anwendungsgebiet zu verbessern. Die Tauglichkeit der Vorschläge im breiten Spektrum der TCP-Anwendungen bleibt noch zu zeigen. In dieser Arbeit wurde daher der Ansatz verfolgt, durch eine Beschränkung der Sendebandbreite auf Applikationsebene die Fehlerwahrscheinlichkeit (insbesondere in Zugangsnetzten mit geringen Bandbreiten) gering zu halten.

3.3 Architekturen zum Filtern von Webinhalten

- **MobiWeb: Enabling Adaptive Continuous Media Applications over 3G Wireless Links [37]** In dieser Arbeit wird das Problem von Real-Time Applikationen über drahtlose Links behandelt. Es wird dazu eine Proxy basierte Netzwerkarchitektur zur Performance-Erhöhung von RealTime Flüssen über drahtlose Links vorgestellt. Es beinhaltet:

  – Ein Priority Schema
  – einen Timer, welche Links erkunden. Wenn Ressourcen vorhanden sind, werden diese genutzt, wenn wenig vorhandensind werden verstärkte Flüsse zurückgestuft.
  – Admission Control für RealTime Traffic: Es werden zwei Stufen unterschieden, *base level of Quality* und *Transparent zu Best Effort Traffic*

  Wireless Links leiden unter limitierten Ressourcen und unter sich ständig verändernden Variablen, wie Handoffs und physikalische Umwelteinflüssen, was folgende Nachteile mit sich bringt: weniger Bandbreite, höheres Delay, höherer Paketverlust. Die Autoren ziehen daraus den Schluss, dass Wireless Links für RealTime Anwendungen nicht geeignet sind.

  Mobiweb hat nun zum Ziel, Support für Anpassungen anzubieten: Eine angepasste Applikation ist aufgeteilt in mehrere Qualitätsstufen, was mit Proxies an beiden Seiten des drahtlosen Links realisiert wird. Weiter sollen Zwischenstrominterferenzen eliminiert werden.

  Vorteil des Designs:

  – So werden die Datenflüsse den Kapazitäten des Links wie auch der mobilen Station angepasst. So können Änderungen extrem schnell gemacht werden (lokal, statt bei den Endstationen).
  – Die Lösung mit dem Proxy ist ideal für Multicast, da die Quelle sich nicht dem langsamsten Empfänger anpassen muss.
  – Zwischenstrominterferenzen: Ist ein Link überlastet, werden alle Ströme zurückgeschraubt, nicht einzelne selektiv. Die Lösung dazu von Mobiweb: Nur ein Fluss wird jeweils auf ein Mal angepasst bis keine weiteren Anpassungen mehr nötig sind. Als Resultat erhält man eine Mischung zwischen Admission Control und einem Priority-Schema.

– Schutz der RealTime Applikationen vor kurzzeitigen Netz- bzw. Linkschwankungen, um nicht unnötig kleine Anpassungen machen zu müssen.

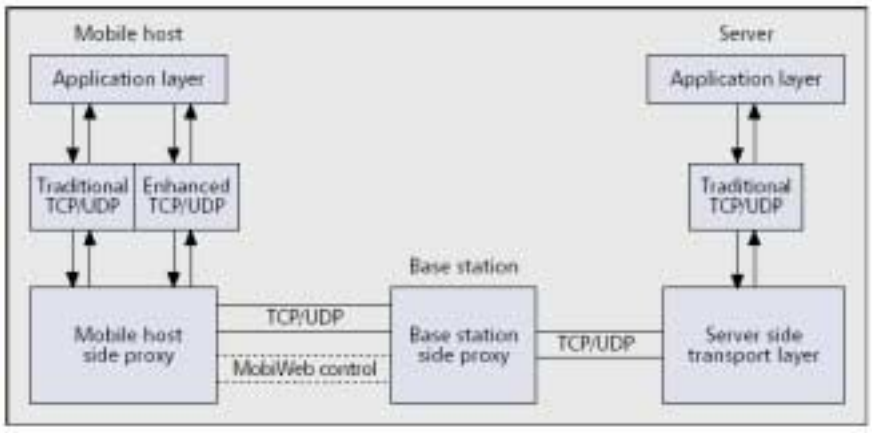


Abbildung 3.1: Die Proxy basierte Architektur von MobiWeb [37]

Als Architektur wurde ein Modell mit zwei Proxies gewählt, eines beim mobilen Endgerät und eines bei der Basisstation des drahtlosen Links (siehe Abbildung 3.1. Anstatt die Daten über TCP/IP zu übertragen, wird der Transport auf diesen Layers von MobiWeb übernommen. Da in dieser Arbeit der Client möglichst keine Software installieren sollte (also auch kein Proxy), kommt diese Architektur nicht in Frage.

- **WebExpress: A client/intercept based system for optimizing Web browsing in a wireless environment [21]**

  WebExpress behandelt vornehmlich den Zugang zum Internet über drahtlose Weitverkehrsnetze. Auch hier werden die Problemthemen von drahtloser Übertragung behandelt:

  – Reduzierte Bandbreite
  – Hohe Verzögerung
  – Hohe Kosten
  – grosse Fehlerrate

  WebExpress ist auf professionelle Anwender wie Versicherungsagenten, Vertreter oder mobile Service-Arbeiter ausgerichtet, die drahtlose Kommunikation für Einsätze in der Berufswelt benötigen. Zur Effizienzsteigerung wird ein *Client-Intercept* Modell vorgestellt (siehe Abbildung 3.2).

  Die Idee des Intercept-Modells setzt bei der Reduktion des Overheads von HTTP an: Anstatt für jeden Request eine neue TCP-Verbindung zum Client aufzubauen

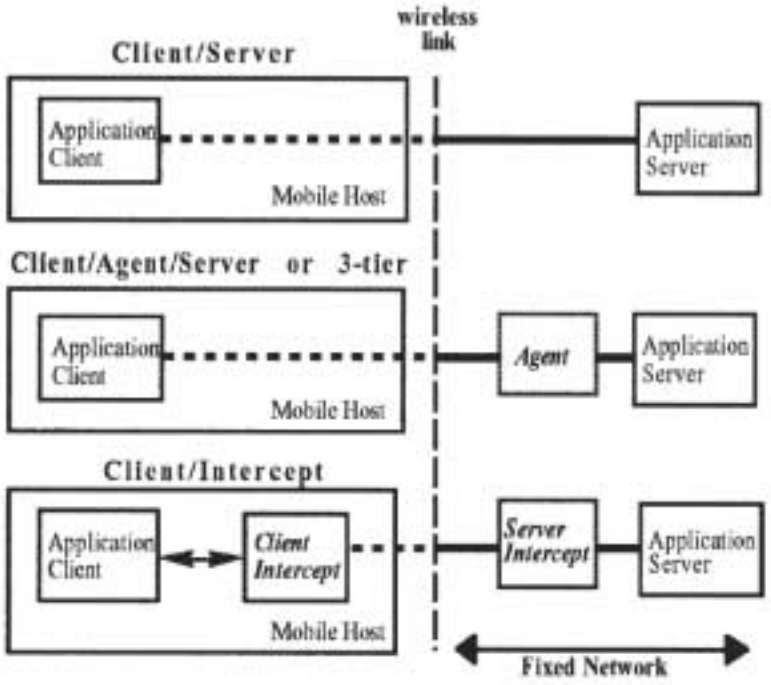


Abbildung 3.2: Client-Intercept Modell zur Regelung des Internet-Zuganges zu drahtlosen WANs [21]

werden alle Anfragen vom Intercept zusammengefasst und über nur eine Verbindung gesendet. Auch redundante HTTP-Headerinformationen werden so weggefiltert. Vorteil dieser Methode ist, dass sie transparent für Web-Browser und -Server ist. Durch ein Cache beim Client wird der Datenverkehr zusätzlich reduziert. Auch CGI-Abfragen werden so bearbeitet, dass sie das Cache nicht unnötig belasten. Als Resultat kann WebExpress im besten Fall eine Datenreduktion von bis über 90% vorweisen.

Der Ansatz von WebExpress ist sehr interessant, die Ergebnisse exzellent. Die Idee lässt sich als Ganzes nicht übernehmen, da auch hier vom Client erwartet wird, dass die entsprechende Software (hier: Client Intercept und Cache) installiert wird. Trotzdem ist der Ansatz sicher auch für lokale Netzwerke und private Anwender interessant.

4 Aufbau und Funktionalität

Das Design des in dieser Arbeit entwickelten Webservers orientiert sich an der klassischen Server-Proxy-Architektur: Der Client wird soweit als möglich von Konfigurations- und Installationsarbeiten entlastet und sämtliche notwendigen Funktionen sind im Server implementiert. Der Client muss zur Benutzung deshalb nur diesen Server als Proxy-Server zum Internet angeben. Der Server kann somit als eigenständiger simpler Webserver benutzt werden oder aber als Proxy-Server mit den erweiterten Funktionen zum Bandbreite testen und beschränken.

Abbildung 4.1: Zugangs- und Registrierungsfenster

Dabei muss festgehalten werden, dass ein reiner HTTP-Server, wie er zum Bereitstellen statischer Webseiten verwendet wird, für diese Arbeit nicht ausreicht. Zur Verarbeitung der Benutzereingaben muss auch CGI-Funktionalität verfügbar sein. Doch sogar mit einem CGI-Interpreter, durch welchen sich sehr viele verschiedene Funktionen in einem Webserver implementieren lassen, ist es nicht möglich, die Bandbreite zu einem Clientrechner zu bestimmen. Dies liegt insbesondere daran, dass der CGI-Interpreter und der Webserver unterschiedliche, voneinander unabhängige Prozesse sind (siehe Kapitel 2.7). Es ist deshalb ein weiterer Baustein im Webserver erforderlich, der eine Verbindung zwischen diesen Prozessen herstellt und so den Bandbreitentest ermöglicht. Zur Beschränkung der Bandbreite ist ein weiterer Baustein nötig. Dieser Bandbreitenbegrenzer muss auch die Koordination der Sendeprozesse übernehmen. Als fünfter Bestandteil muss noch eine Proxy-Funktionalität hinzugefügt werden,

da erst diese es dem Benutzer ermöglicht, auch externe Seiten aufzurufen und gleichzeitig die Bandbreitenbeschränkung zu benutzen. Insgesamt lässt sich also das Webserverprogramm wie in Abbildung 4.2 gezeigt darstellen, wobei oberen drei Bausteine in praktisch allen klassischen Webservern zu finden sind, während der Bandbreitentester und der Bandbreitenbegrenzer die eigentliche Neuerung darstellen.

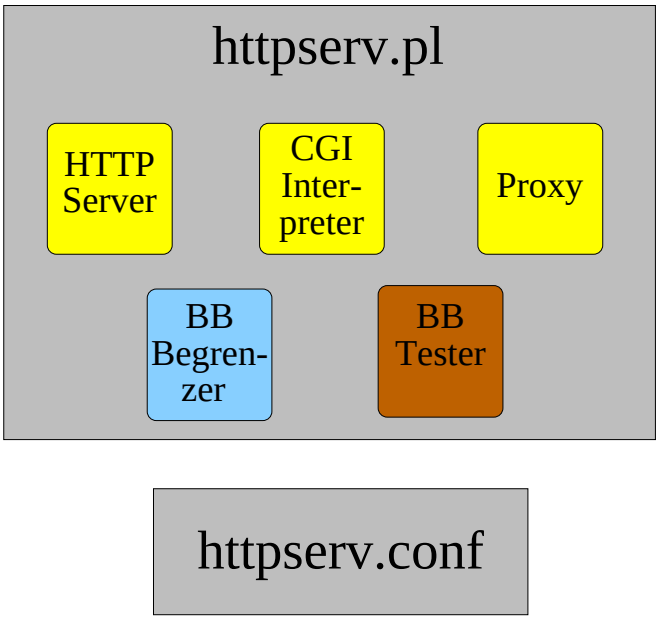


Abbildung 4.2: Architektur des Webservers

Die ebenfalls in Abbildung 4.2 dargestellte Konfigurationsdatei *httpserv.conf* beinhaltet dabei verschiedene Einstellungsmöglichkeiten, die vom Administrator des Servers an die jeweilige Situation und an die Bedürfnisse von Benutzer oder Anbieter angepasst werden können. Dazu zählen sowohl die IP-Adresse und Portnummer des Servers als auch Angaben zur Datenbankverbindung, Suchpfade für Webseiten und CGI-Skripte, Zeitschranken für den Bandbreitentest sowie Einstellungen zum Cache wie zum Beispiel die Lebensdauer gespeicherter Seiten (falls in diesen nicht ein früheres Erneuerungsdatum angegeben ist).

Die volle Funktionalität des vorgestellten Webservers wird erst im Zusammenspiel mit einer Datenbank erreicht. In dieser Datenbank werden sowohl die Angaben zu den Benutzern (Benutzername, Passwort, IP-Adresse) gespeichert als auch die Angaben zur Bandbreitenbeschränkung (sei es durch den Bandbreitentester bestimmt oder explizit angegeben). In einer weiteren Datenbank werden von den Benutzern angefragte Seiten gespeichert, um sie bei einer erneuten Anfrage schneller verfügbar zu haben. Die Umgebung in der sich der Webserver befindet, lässt sich also wie folgt darstellen:

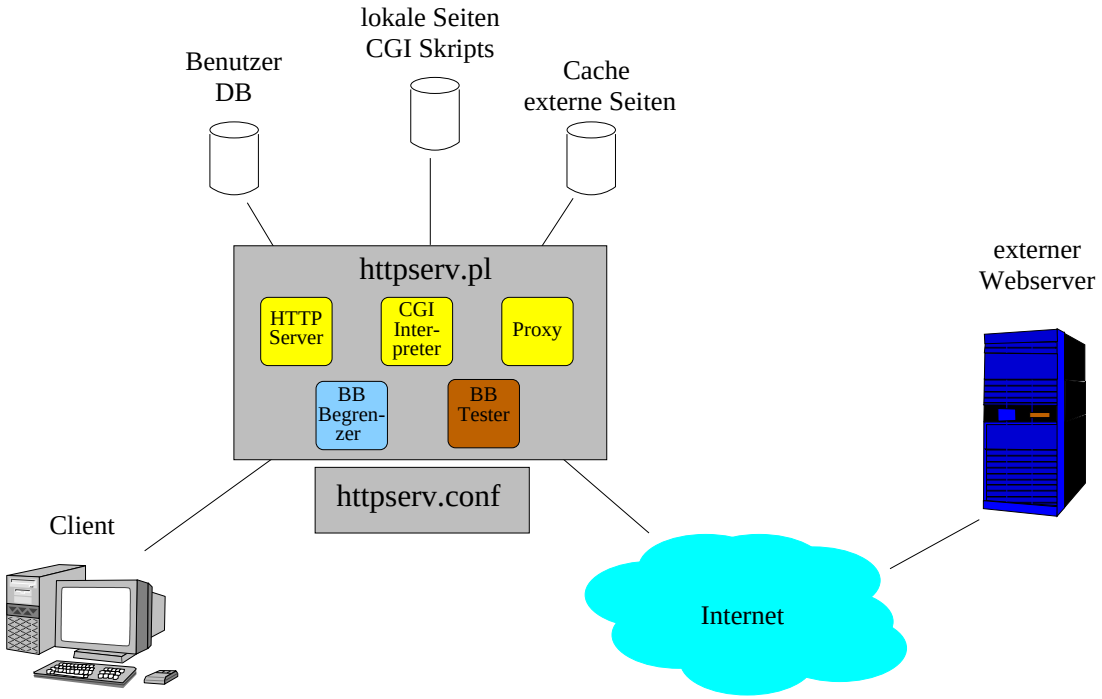


Abbildung 4.3: Webserver und Umgebung

4.1 Funktionalität der Bestandteile

4.1.1 Webserver

Das Kernstück der Architektur liegt im Webserver, der alle Benutzereingaben speichert, verwaltet und die angeforderten Seiten je nachdem aus dem Cache oder direkt vom zuständigen Server holt und an den Benutzer weiterleitet. Ein heute gebräuchlicher Webserver (z.B. Apache) kann zwar einiges dieser geforderten Funktionalität bereitstellen (z.B. das Anfordern von HTML-Seiten), in Zusammenarbeit mit anderen Programmen (z.B. `squid`) auch Cache-Funktionalität übernehmen, aber trotzdem bleiben einige Schwierigkeiten, insbesondere bei der Bestimmung der Bandbreite zum Client, die für die ganze Architektur den Einsatz derartiger Software verhindern. Es wurde daher ein einfacher Webserver (`httpserv.pl`) implementiert, der — in der Programmiersprache Perl entwickelt — danach entsprechend den Anforderungen modifiziert und erweitert wurde.

Der Webserver ist ein einfacher HTTP-Server, der nur zwei Arten von Requests versteht: GET und PUT. Damit deckt er jedoch bereits einen sehr grossen Teil des üblichen Benutzerverhaltens ab, welches überwiegend aus dem Abrufen von Webseiten (GET-Request) und zu einem etwas geringeren Teil aus dem Ausfüllen von Formularen (z.B. die Eingabe bei Suchmaschinen) besteht.

4.1.2 CGI-Interpreter

Der CGI-Interpreter ist ein Perl Interpreter, der vom Webserver aufgerufen wird. Dabei werden einige der wichtigsten CGI-Umgebungsvariablen gesetzt (z.B. REQUEST_METHOD), allerdings bei weitem nicht alle. Man kann den CGI-Interpreter daher nicht als voll funktionsfähig bezeichnen; für die hier benötigten Zwecke ist er jedoch ausreichend: Er muss nur in der Lage sein, die speziell zur Bandbreitenbestimmung und Benutzerverwaltung entwickelten Hilfsprogramme (`auswerten.cgi`, `bildtest.cgi` und `register.cgi`) mit dem Hauptprogramm `httpserv.pl` zu verbinden.

4.1.3 Proxy

Das Proxy ist derjenige Bestandteil des Webserver Programmes, welches aufgerufen wird, falls der Benutzer eine externe Seite anfordert. Falls die Seite bereits im Cache vorliegt, wird natürlich keine Anfrage an den externen Server geschickt. Der Benutzer kann jedoch auch bestimmen, dass eine Seite immer aktuell vom Server geholt wird, auch wenn sie im Cache vorliegt. Dies geschieht mit der normalen *reload* Funktion des Browsers. Des weiteren werden alle Seiten, die länger als eine in der Konfiguration angegebenen Zeitdauer im Cache liegen, aktualisiert.

4.1.4 Bandbreitentester

Der Bandbreitentester misst durch das Verschicken von Dateien bekannter Grösse unter gleichzeitiger Bestimmung der Sendedauer die Bandbreite, die zum Zeitpunkt der Messung auf der Verbindung vom Server zum Client verfügbar ist. Um eine ausreichende Genauigkeit zu erreichen, dabei aber die Verbindung – insbesondere für geringe Bandbreiten – nicht übermässig zu belasten, wird die Messung einige Male mit jeweils steigender Dateigrösse durchgeführt. Sobald die Sendedauer für eine Datei eine in der Konfigurationsdatei angegebene Schranke erreicht (oder wenn die maximale Dateigrösse erreicht ist), wird das Ergebnis als hinreichend genau betrachtet.

4.1.5 Bandbreitenbegrenzer

Der Bandbreitenbegrenzer dient dazu, die TCP-Verbindung zu stabilisieren und durch Überlast entstandene Paketverluste und die dadurch von TCP ausgelöste Halbierung der Übertragungsrate zu vermeiden (siehe dazu auch Kapitel 2.1). Weiter kann sie von Providern dazu verwendet werden, einem Benutzer jeweils nur einen Teil der zur Verfügung stehenden Bandbreite zuzuteilen. Vor jeder Übertragung von Daten wird aus der Datenbank die maximale Bandbreite des betreffenden Benutzers ausgelesen.

Die Daten werden danach mit der entsprechend eingeschränkten Bandbreite übertragen. Damit nicht mehrere parallele Verbindungen mit der maximalen Bandbreite aufgebaut werden, wird der Datenbankzugriff koordiniert und eingeschränkt.

4.1.6 Datenbank und Benutzerschnittstelle

Sowohl zur Benutzerverwaltung als auch zum Speichern, Updaten und Versenden von geladenen HTML-Seiten wird eine Datenbank benötigt. Für beide der oben genannten Anwendungsgebiete ist dabei jeweils nur eine Tabelle in der Datenbank notwendig, was den Aufwand einer SQL-Datenbank zunächst etwas gross erscheinen lässt und die Frage aufwirft, ob nicht einfache Tabellen im Programm (eventuell verbunden mit einer Hash-Suche) dieselben Dienste leisten könnten. Andererseits ist eine SQL-Datenbank hinsichtlich Verwaltung, Programmieraufwand und Lesbarkeit des Codes bei weitem fortgeschrittener. Plant man nun in Zukunft die beiden getrennten Bereiche zu verbinden, etwa die Einträge im Cache zu personalisieren, so macht sich der anfängliche Mehraufwand der SQL-Installation schnell bezahlt. Auch andere Erweiterungen wie Zugangserlaubnis mit Passwort lassen sich mit der Datenbank sicher und schnell verwirklichen. Weiter muss man sich mit dem Benutzen der Datenbank nicht um die Sicherheit von synchron ablaufenden Zugriffen auf Einträge kümmern.

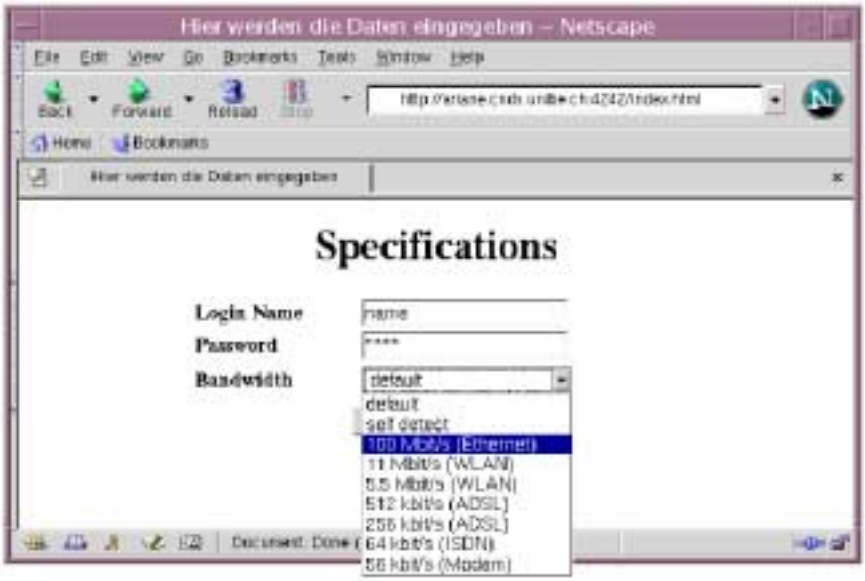


Abbildung 4.4: Zugangs- und Registrierungsfenster mit Optionen

Die Benutzerdatenbank wird durch ein CGI-Skript, welches die Benutzereingaben auswertet, gefüllt. Falls sich ein Benutzer neu anmeldet (`Register` Knopf, siehe Abbildung 4.1), wird ein neuer Eintrag in der Datenbank erstellt. Dabei muss der Benutzer-

namen eindeutig sein, kann also nicht von mehreren Benutzern gewählt werden. Wird ein bereits verwendeter Benutzername eingegeben, wird der Benutzer darauf aufmerksam gemacht (siehe Abbildung 4.5). Meldet sich ein bereits existierender Benutzer an (`Login`-Knopf), werden seine Eingaben mit den Einträgen in der Datenbank verglichen und die Datenbank wird gegebenenfalls aktualisiert. So kann ein Benutzer beispielsweise eine neue Bestimmung der Bandbreite anfordern oder die Bandbreite selber neu einstellen. Die Benutzerschnittstelle mit möglichen Optionen ist in Abbildung 4.4 dargestellt.

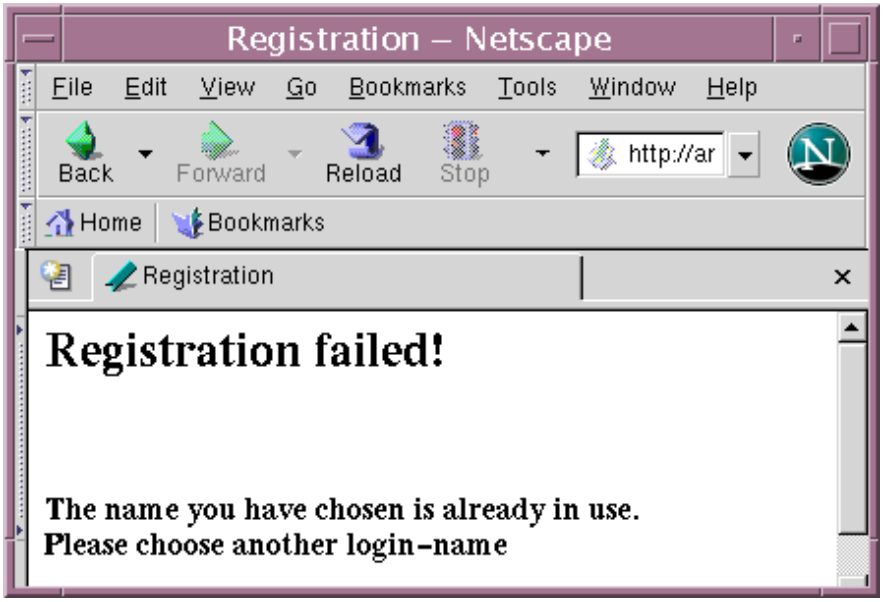


Abbildung 4.5: Ausgegebene Seite bei erfolgloser Registrierung

Zum Zwecke der Benutzerverwaltung ist für die Identifizierung der Benutzer ein frei wählbarer login-Namen von maximal 8 Zeichen Länge vorgesehen. Die Daten des Benutzers werden dabei durch ein Passwort von ebenfalls maximal 8 Zeichen Länge vor Missbrauch geschützt. Zur Sicherheit ist jedoch dabei zu beachten, dass ein Webbrowser standardmässig keine Verschlüsselung der eingegebenen Daten vornimmt, d.h. sowohl Benutzername als auch Passwort werden im Klartext übertragen. Hier schafft das https Protokoll Abhilfe, welches sämtliche Benutzereingaben verschlüsselt, und dabei einen sehr guten Sicherheitsstandard garantieren kann. Aufgrund der komplexen Implementierung dieses Protokolls wurde jedoch für den einfachen `httpserv` Webserver auf diese Sicherheit verzichtet.

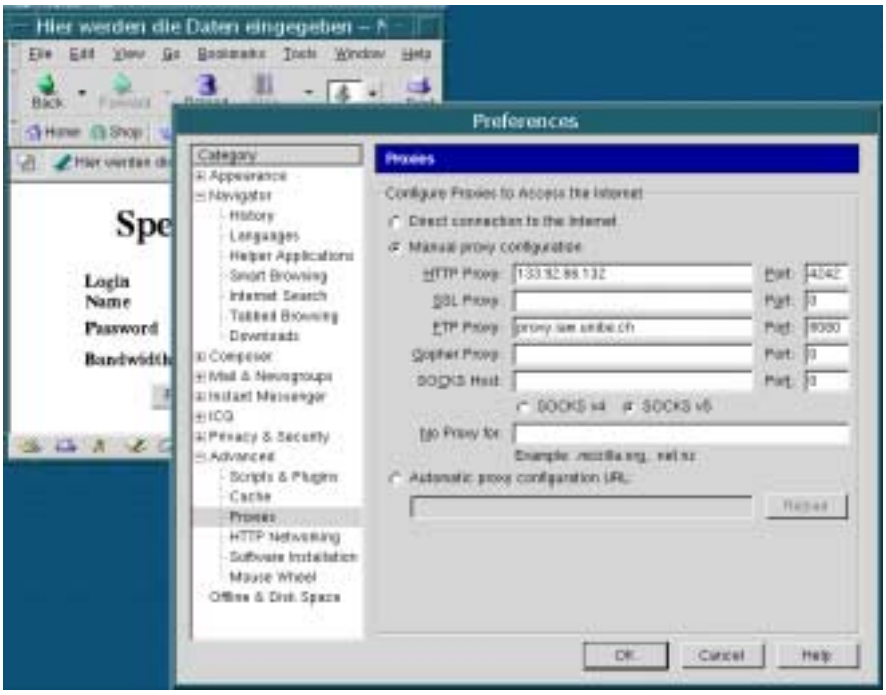


Abbildung 4.6: Nötige Einstellung des HTTP-Proxys

4.2 Ablauf von Anfragen nach internen und externen Seiten

Damit der Benutzer die Bandbreitenbeschränkung für alle Seiten des WWW und nicht nur für Seiten, die auf diesem Webserver liegen, zur Verfügung steht, müssen *alle* Anfragen des Benutzers über diesen Webserver laufen. Das lässt sich dadurch erreichen, dass die Adresse und Portnummer des Webservers in den Proxyeinstellungen des Browsers eingetragen werden (siehe Abbildung 4.6), der Webserver als als Proxy-Server benutzt wird. Die Anfragen, die an den Server gelangen, kann man in zwei Kategorien unterteilen: Externe Requests (nach Seiten, die sich auf fremden Servern befinden), die sicherlich den grössten Anteil stellen und interne Requests.

4.2.1 Anfragen nach externen Seiten (siehe Abbildung 4.7)

Schickt ein Benutzer eine Anfrage nach einer externen Seite (1), wird diese Anfrage vom Webserver direkt an das Proxy weitergeleitet (2). Das Proxy überprüft zunächst, ob die Seite bereits im Cache vorhanden ist (3). Falls die Seite gefunden wird und aktuell ist, kann sie direkt an den Benutzer geschickt werden. Andernfalls wird die Seite vom externen Server angefordert (4), im Cache gespeichert (5) und an den Benutzer geschickt (6 – 8). Der Benutzer hat ausserdem die Möglichkeit explizit eine aktuelle Version einer Seite anzufordern (über die Refresh-Funktion beim Browser). Bei jedem

Sendevorgang an den Benutzer wird aus der Datenbank die eingetragene Bandbreite gelesen (7) und die Sendegeschwindigkeit vom Server zum Client wird entsprechend reduziert (8).

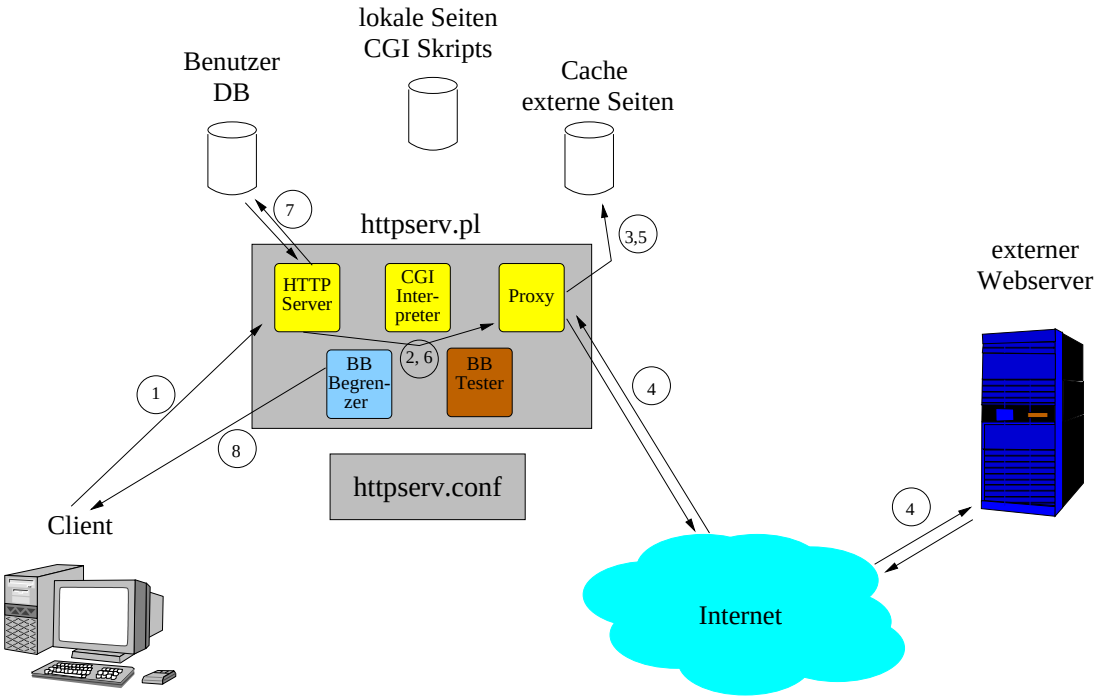


Abbildung 4.7: Ablauf einer Anfrage nach einer nicht im Cache gespeicherten Seite

4.2.2 Interne Anfragen

Benutzeranfragen nach lokalen Seiten lassen sich in Anfragen nach HTML-Seiten und CGI-Skripte unterteilen. HTML-Seiten können direkt von der Festplatte gelesen und an den Benutzer geschickt werden. Für CGI-Skripte wird der Interpreter gestartet, der das Skript ausführt und das Ergebnis an den Benutzer sendet.

4.3 Bandbreitentest

Beim Eintreffen einer Bandbreitentestanfrage wird zunächst überprüft, ob der Bandbreitentester verfügbar ist. Dadurch wird verhindert, dass sich durch zwei parallele Messungen die Ergebnisse verfälschen. Anschliessend wird der Benutzer auf ein CGI-Skript weitergeleitet, das den Bandbreitentest durchführt. Daraufhin wird der Bandbreitentest durchgeführt, bis eine befriedigende Genauigkeit erreicht wurde. Das Ergebnis der Messung wird in der Datenbank gespeichert und dem Benutzer auf einer

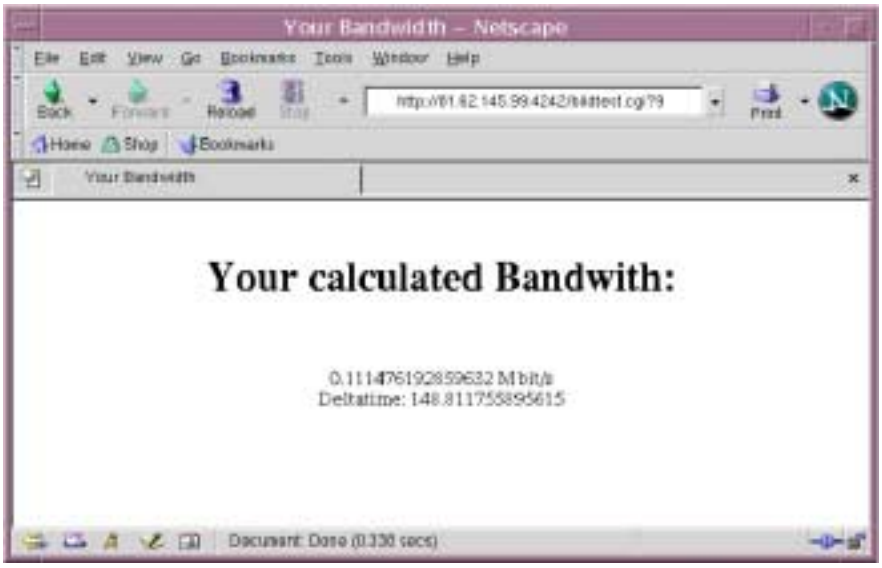


Abbildung 4.8: Ergebnis des Bandbreitentests auf ADSL

Webseite bekanntgegeben (siehe Abbildung 4.8). Der Benutzer wird nach kurzer Anzeige des Messresultates weitergeleitet, so dass er sämtliche seiner Einträge sieht und gegebenenfalls anpassen kann (siehe Abbildung 4.12). Folgend wird der Aufbau und Ablauf des Bandbreitentests etwas detailierter beschrieben.

4.3.1 Problematik

Bei der Bandbreitenmessung hat man mit zwei konträren Ansprüchen zu tun: zum einen ist es aus Sicht des Benutzers erstrebenswert, wenn die Messung der Bandbreite so schnell wie möglich geschieht und möglichst wenig Ressourcen verbraucht. Allerdings ist offensichtlich, dass eine kurze Messung nur eine ungenügende Genauigkeit liefern kann. Man hat daher zu entscheiden, wie eine befriedigende Genauigkeit erreicht werden kann, ohne dass übermässig viel Zeit für die Messung aufgebracht werden muss.

Der übliche Weg zur Erhöhung der Genauigkeit besteht sicherlich darin, die selbe Messung mehrmals hintereinander durchzuführen. Dabei ist zu beachten, dass auch die Dateigrösse eine wesentliche Rolle spielt: eine zu kleine Datei wird auf einer schnellen Leitung in einer Zeit übertragen, die nahe an der Auflösungsgrenze des Kernels ist, eine grosse Datei führt bei einer langsamen Leitung zu einer übermässig langen Wartezeit und einer unnötigen Belastung der ohnehin begrenzten Bandbreite. Der gewählte Lösungsansatz besteht daher darin, für jede Messung die Dateigrösse schrittweise zu erhöhen. Erreicht man damit, das die Sendedauer über einem gewissen Grenzwert liegt, kann man annehmen, dass die Messung hinreichend genau ist. Um dies

zu bewerkstelligen wurden zufällige Bilder verschiedener Grösse erzeugt, angefangen bei 8’753 Bytes wurde die Grösse jeweils inetwa verdoppelt, das grösste Bild ist 2’073’621 Bytes gross.

4.3.2 Kommunikation zwischen Webserver und CGI Prozessen

Die automatische Messung der Bandbreite und das möglicherweise mehrfache Senden verschieden grosser Dateien erfordert eine über das übliche Mass hinausgehende Kommunikation zwischen Webserver und CGI-Skript. Da sich die normale Kommunikation vom CGI-Skript zum Webserver auf den Rückgabewert (32 Bit Integer) beschränkt, musste ein anderer Weg gefunden werden, Daten zwischen diesen beiden Prozessen auszutauschen, ohne die Messung durch aufwendige Synchronisationsmassnahmen zu stören.

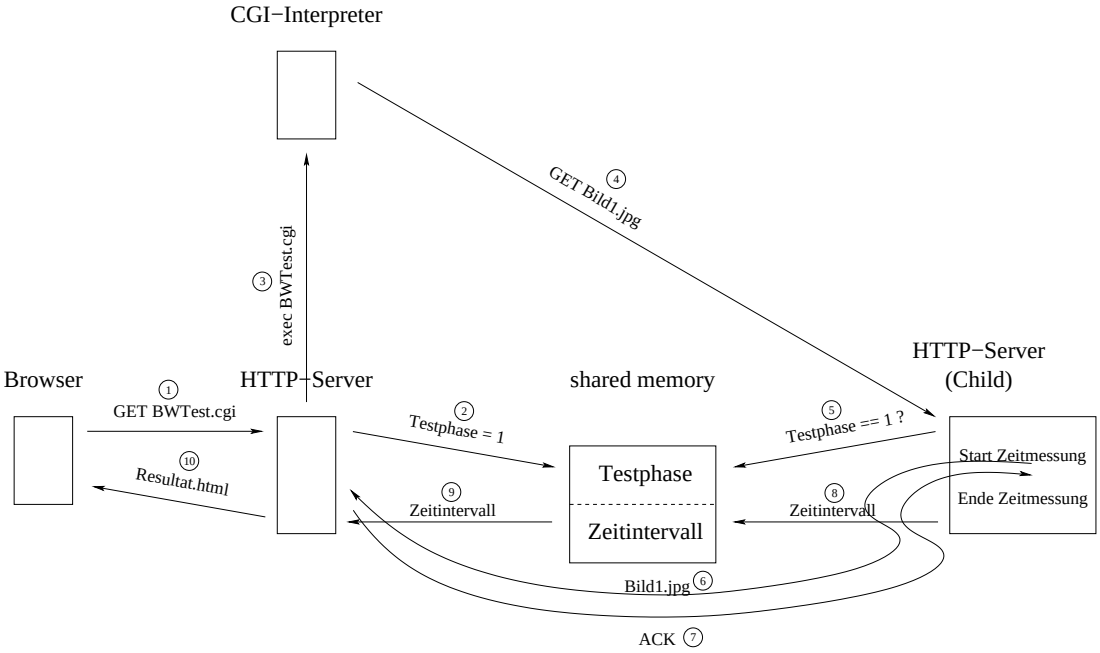


Abbildung 4.9: Ablauf des Bandbreitentests mit Zugriff auf den gemeinsamen Speicher

Aufgrund der Menge der auszutauschenden Daten wurde ein gemeinsamer Speicherbereich (*shared memory*) eingerichtet, auf den beide Prozesse sowohl lesend als auch schreibend zugreifen können (siehe Abbildung 4.9). In diesem Speicherbereich wird als erstes eine Variable *testphase* gesetzt, falls eine Messung in Gang ist. Danach werden die zur Messung benötigten Variablen in diesen Speicherbereich geschrieben (Anfang und Ende jeder Zeitmessung, Grösse des zuletzt übertragenen Bildes), wo von allen Kindprozessen, die am Messvorgang beteiligt sind, auf sie zugegriffen werden kann. Die Variable *testphase* dient dazu sicherzustellen, dass jeweils nur ein

CGI-Skript, das eine Bandbreitenmessung durchführt, gestartet werden kann. Ist bei Messbeginn bereits eine andere Messung im Gange, wird der Benutzer darauf hingewiesen und der Vorgang wird wiederholt (siehe Abbildung 4.10). Da keine andere Prozesse auf den gemeinsamen Speicherbereich zugreifen können, kann auf weitere Schutzmassnahmen verzichtet werden.

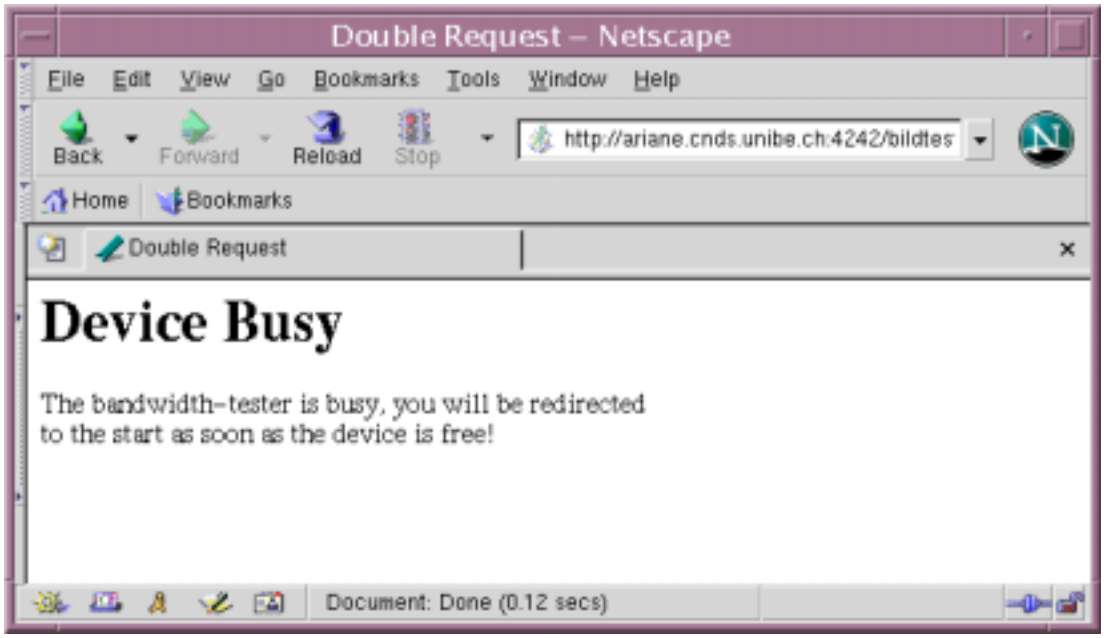


Abbildung 4.10: Fehlermeldung bei mehrfachem gleichzeitigen Zugriff auf den Bandbreitentester

4.3.3 Zeitmessung

Um die Sendedauer möglichst genau und ohne den Einfluss der TCP-Sendepuffergrösse zu bestimmen, wird die Zeit zwischen Auf- und Abbau der Verbindung, über welche die Testdatei übertragen wird, von HTTP-Server zum Client gemessen. Dies erfordert jedoch, dass die Zeitnahme im Webserver durchgeführt wird, während die Daten aufgrund ihrer sich ändernden Grösse von einem CGI-Skript gesendet werden müssen. Um die Bandbreitenmessung möglichst automatisch durchzuführen, wurde auf die Grundeinstellung vieler Browser zurückgegriffen, in HTML-Seiten referenzierte Bilder automatisch zu laden.

4.3.4 Ablauf des Bandbreitentests

Der Bandbreitentest wird automatisch durchgeführt, wenn sich ein Benutzer neu registrieren lässt. Ein bereits registrierter Benutzer kann aber auch einen neuen Test bean-

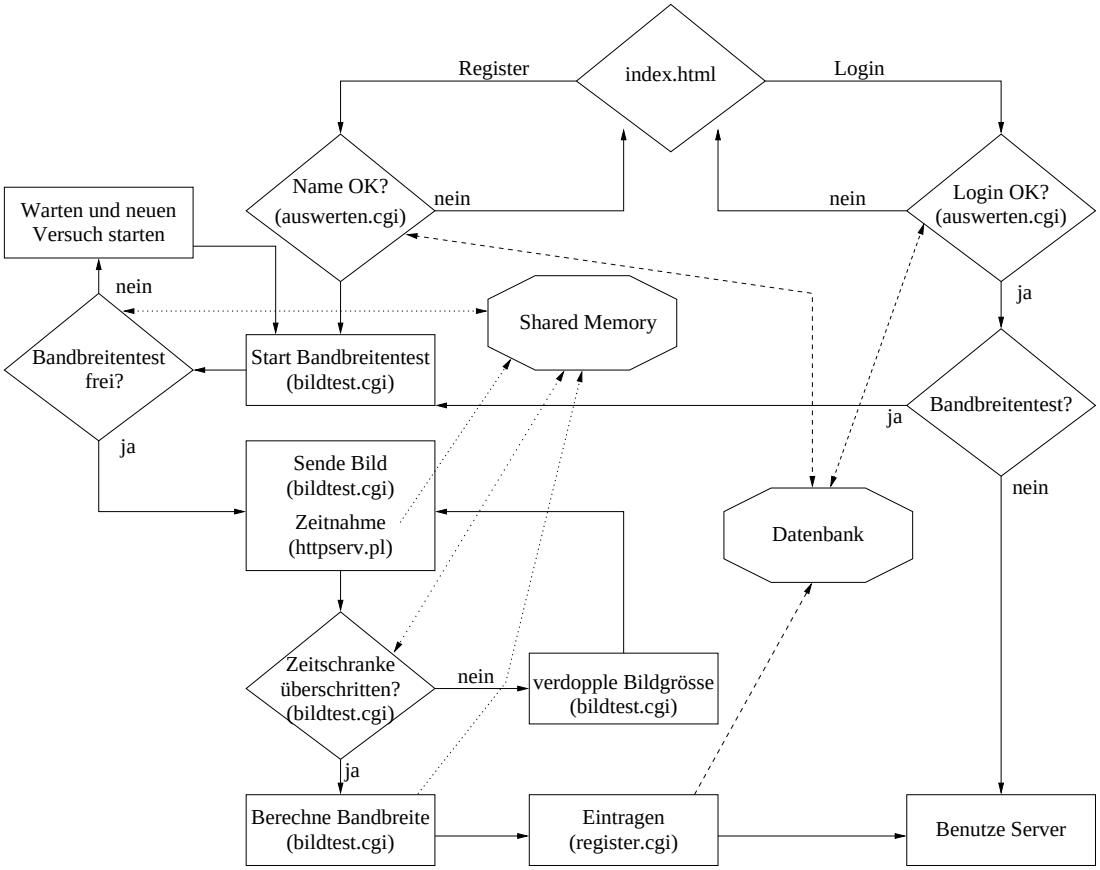


Abbildung 4.11: Zustandsdiagramm zum Ablauf des Bandbreitentests

tragen. Der Gesamtablauf des Bandbreitentests wird im Zustandsdiagramm in Abbildung 4.11 dargestellt. Beispielhaft wird im folgenden der Ablauf einer Registrierung mit anschliessender Bandbreitenbestimmung vorgestellt:

1. Der Benutzer wählt die Hauptseite `index.html` auf dem Server an.
2. Der Benutzer trägt Login-Name und Kennwort ein und drückt ohne Angabe einer Bandbreite den Registrierknopf (siehe Abbildung 4.1)
3. Durch das Drücken des Registrierknopfes wird das CGI-Skript `auswerten.cgi` aufgerufen.
4. `auswerten.cgi` trägt den Benutzer in die Datenbank ein, sofern der Login-Name frei ist. Anderenfalls wird der Benutzer aufgefordert, einen anderen Namen zu wählen.
5. Nach dem Eintragen wird dem Benutzer auf dem Bildschirm die erfolgreiche Registrierung angezeigt (siehe Abbildung 4.12). Nach einer im Konfigurationsfile `httpserv.conf` gesetzten Frist wird der Benutzer an den Bandbreitentest `bildtest.cgi` weitergeleitet.

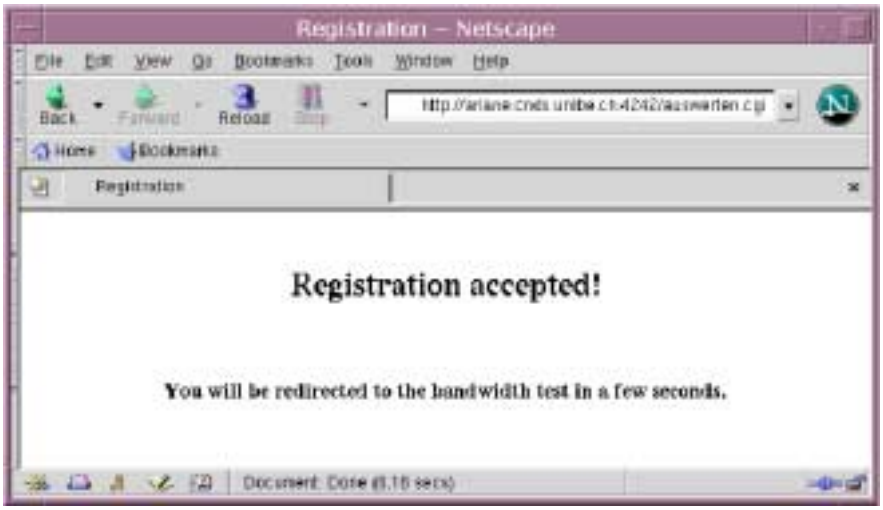


Abbildung 4.12: Registirierung angenommen

6. `bildtest.cgi` übermittelt dem Benutzer eine html-Seite, auf welcher das kleinste Testbild referenziert ist.
7. Der Browser des Benutzers fordert automatisch das Bild an. Beim Eintreffen des Requests beim Server `httpserv.conf` beginnt die Zeitnahme und eine Variable wird gesetzt, so dass nicht mehrere Messungen gleichzeitig stattfinden

können. Die Zeitnahme endet mit dem Eintreffen des Verbindungsabbaurequests des Clients, der die erfolgreiche Übertragung der Datei angibt.
`httpserv.conf` trägt die gemessene Zeit in den gemeinsamen Speicherbereich ein.

8. `bildtest.cgi` wartet eine im Konfigurationsfile `httpserv.conf` angegebene Zeit, danach wird die zur Übertragung benötigte Zeit aus dem gemeinsamen Speicherbereich gelesen. Ist eine (auch in `httpserv.conf` angegebene) Zeit-Schranke nicht überschritten, wird Schritt 6 wiederholt (mit einem doppelt so grossen referenzierten Bild wie zuvor). Anderenfalls wird die Bandbreite mit Hilfe der zuletzt gemessenen Zeit und der Grösse des zuletzt übermittelten Bildes berechnet.

9. Die berechnete Bandbreite wird von `bildtest.cgi` dem Benutzer über eine html-Seite mitgeteilt (siehe Abbildung 4.8).

10. Auch hier wird der Benutzer automatisch nach einer bestimmten Zeit an das Skript `register.cgi` weitergeleitet, wo alle den Benutzer betreffenden Daten angezeigt werden und der Eintrag in die Datenbank gemacht wird (siehe Abbildung 4.13).

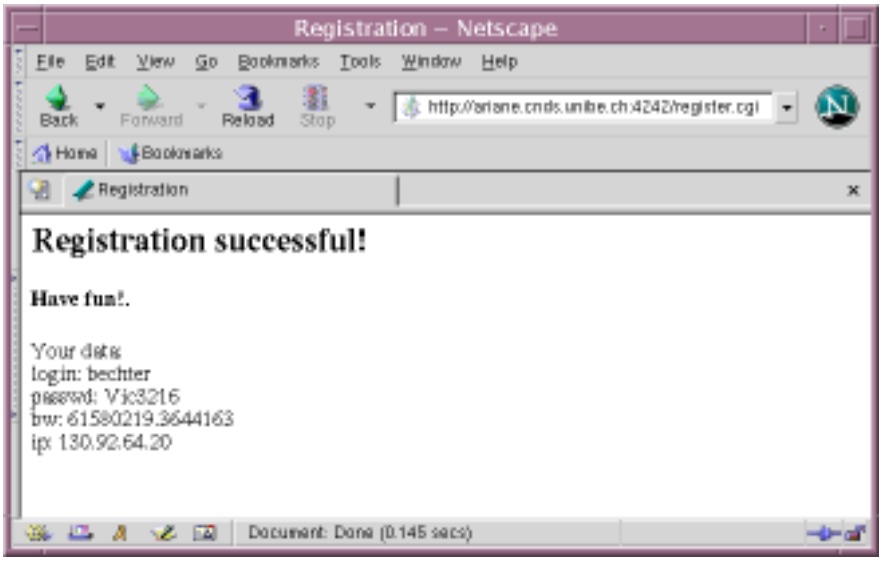


Abbildung 4.13: Ausgabe der gemessenen und eingegebenen Benutzerdaten

11. Der Benutzer kann nun die URL des Webservers als Proxy-Server eintragen und externe Seiten anfordern, die ihm vom Webserserver `httpserv.ps` mit der gemessenen, eingeschränkten Bandbreite gesendet werden.

Bei einem Abbruch der Messung wird sichergestellt, dass die Variable, welche parallele Messungen verhindert, nach Ablauf einer gewissen Zeitspanne wieder zurückgesetzt wird.

4.4 Bandbreitenbeschränkung eines TCP Senders

Die Bandbreitenbeschränkung eines TCP-Senders unterliegt mehreren Einschränkungen, welche die Genauigkeit der Beschränkung betreffen. Zunächst ist zu beachten, dass es *keine* Möglichkeit gibt, den TCP Sendebuffer zu einem bestimmten Zeitpunkt zu leeren (siehe dazu [49] und die Diskussion auf dem Internet [47]). Das liegt daran, dass der TCP-Algorithmus es erlaubt, mehrere Pakete, die von der Anwendung geschickt werden, zu einem Sendefenster zusammenzufassen. Jegliche Bandbreitenbeschränkung auf Applikationsebene limitiert also nur die Bandbreite, mit welcher der TCP-Stack versorgt wird. Messungen zeigen jedoch, dass eine derartige Beschränkung (auf einer gröberen Zeitskala) auch zum gewünschten Effekt einer Beschränkung der TCP Sendebandbreite führt.

Ein möglicher Ansatz für einen Sender, der mit Bandbreite b sendet, ist der sogenannte *token bucket filter*: eine Variable (*token bucket*) gibt an, wieviele Tokens (Bytes) der Sender verschicken darf. Diese Variable wird periodisch um einen festen Betrag vergrössert. Dadurch ist sichergestellt, dass der Sender nicht unbeschränkt senden kann, aber dass kurze Einbrüche in der Sendegeschwindigkeit zu einem späteren Zeitpunkt wieder ausgeglichen werden können.

Die Implementation eines *token bucket filters* mit einer *token bucket* Variable ist jedoch nicht unproblematisch, da zum einen sichergestellt werden muss, dass die Variable periodisch vergrössert wird, zum anderen, dass der Sender die Variable beim Sendevorgang um den entsprechenden Betrag verkleinert, bzw. das Senden solange aussetzt bis die Variable positiv ist. Die parallelen Zugriffe auf die *token bucket* Variable müssen also synchronisiert werden, und der Sendeprozess muss die Variable überwachen oder mittels eines Signals erfahren, ob der *token bucket* gefüllt ist.

Diese Schwierigkeiten lassen sich zum Teil durch eine abgeänderte Implementation vermeiden: Man misst dabei die Zeit t, die benötigt wird um eine Anzahl n von Paketen der Grösse s zu versenden. Anschliessend wird solange (t_1) gewartet, bis die durchschnittliche Bandbreite auf den eingestellten Wert b abgesunken ist:

$$\frac{n \cdot s}{t + t_1} \approx b \tag{1}$$

Die Bestimmung dieser Variablen wird im folgenden im Detail erläutert.

Die Sendedauer t lässt sich mittels des `gettimeofday`-Funktionsaufrufs [33], der in Perl im Modul `Time::HiRes` [53] verfügbar ist, bestimmen. Die dabei erreichbare Genauigkeit ist mit 10^{-6} Sekunden sehr hoch.

Die Zeitspanne t_1, welche der Sendeprozess zwischen zwei Sendevorgängen warten muss, lässt sich auf zwei verschiedene Arten realisieren: zum einen könnte der Sendeprozess in einer Schleife solange die aktuelle Zeit abfragen, bis die erforderliche Zeitspanne verstrichen ist (*busy waiting*). Die dabei entstehende Last auf dem Prozessor ist jedoch sehr hoch, was in einer parallelen Serverarchitektur inakzeptabel ist. Die zweite Möglichkeit besteht darin, den Sendeprozess vom Betriebssystem für eine gewisse Zeitspanne suspendieren (anhalten) zu lassen (z.B. mit dem Befehl `nanosleep` [34]). Dadurch benötigt der Sendeprozess während der Wartezeit keine Prozessorleistung, und diese steht voll anderen Prozessen zur Verfügung. Allerdings unterliegt das Anhalten eines Prozesses durch das Betriebssystem einer Einschränkung: es gibt eine untere Schranke $1/HZ$, die mindestens verstreicht, bis der Prozess vom Betriebssystem erneut ausgeführt wird. Diese Schranke ist von den Einstellungen im Betriebssystemkern abhängig (z.B. $HZ = 100$ für Linux/Intel, $HZ = 1000$ für Linux/Alpha). Ausserdem beträgt die Wartezeit stets ein ganzzahliges Vielfaches von $1/HZ$ (siehe [34]).

Die Paketgrösse s muss also derart bestimmt werden, dass sich der Effekt der Bandbreitenbeschränkung bereits ab der Zeitauflösung $1/HZ$ einstellt. Im Idelalfall versucht man daher gerade so viel zu senden, wie es die Bandbreite zulässt:

$$s_{min} = \frac{1}{HZ} \cdot b \tag{2}$$

Insbesondere für geringe Bandbreite ist s_{min} jedoch unsinnig klein (z.B. 12 Bytes für $b = 9.6\text{kbit}/s$). Ein derart kleiner Wert ist jedoch unsinnig, wenn man bedenkt, dass allein die Header des TCP- und des IP Protokolls zusammen 40 Byte ergeben. In einem solchen Fall wird daher die Genauigkeit der Bandbreitenbeschränkung zugunsten einer Verbesserung des Datendurchsatzes zurückgestellt und die Mindestgrösse eines Sendepaketes auf 50 Bytes gesetzt. Auch im anderen Extrem sehr grosser Bandbreiten ergibt die Formel 2 ungeeignete Werte (z.B. 125000 Byte für 100 Mbit/s), welche grösser als die Sendepuffer des Betriebssystems sind (z.B. 32 kByte für den TCP Sendepuffer). Daher wurde für die Paketgrösse eine obere Schranke von 1000 Byte eingeführt.

$$s = \begin{cases} 50 & : \text{falls} \quad \frac{1}{HZ} \cdot b < 50 \\ 1000 & : \text{falls} \quad \frac{1}{HZ} \cdot b > 1000 \\ \frac{1}{HZ} \cdot b & : \text{sonst} \end{cases} \tag{3}$$

Durch die Bestimmung der Paketgrösse s nach Formel 3 ist klar, dass die Anzahl n der Pakete, die pro Zeiteinheit gesendet werden können nur dann grösser 1 ist, falls die Beschränkung von s auf 1000 Byte eintritt, also $\frac{1}{HZ} \cdot b > 1000$. Es ist allerdings nicht möglich, die Anzahl n einfach durch

$$n = \left\lceil \frac{\frac{1}{HZ} \cdot b}{1000} \right\rceil \tag{4}$$

zu bestimmen. Dagegen spricht, dass Schwankungen in der tatsächlichen Übertragungsgeschwindigkeit des Senders auftreten können. Des weiteren führt die Tatsache, dass die Wartezeit zum einen stets grösser als $1/HZ$ und zum anderen immer aufgerundet wird, zu Fehlern, da immer entweder länger oder kürzer gewartet wird als nötig. Den dadurch entstehenden Fehler kann man vermeiden, indem man nach jedem Sendevorgang eines Paketes die bisher verstrichene Zeit t misst. Falls

$$\frac{n \cdot s}{t} > b \tag{5}$$

müsste eine Sendepause von

$$t_1' = \frac{n \cdot s}{b} - t \tag{6}$$

eingelegt werden (dabei entspricht n der aktuell gesendeten Anzahl der Pakete seit Beginn der Messung von t). Da aber der berechnete Wert von t_1 immer auf die nächste ganzzahlige Vielfache von $1/HZ$ aufgerundet wird, kann man eine statistisch korrekte Rundung dadurch erreichen, dass man vom Ergebnis den Betrag $1/2HZ$ abzieht. Die statistisch korrekte Formel für t_1 lautet demnach

$$t_1 = \frac{n \cdot s}{b} - t - \frac{1}{2 \cdot HZ} \tag{7}$$

und die Bedingung für eine Sendepause ist

$$\frac{n \cdot s}{t + \frac{1}{2 \cdot HZ}} > b \tag{8}$$

Um die Messung gegen Einflüsse des Betriebssystems bei der Suspendierung des Sendeprozesses zu stabilisieren, wird die Sendedauer t nicht nach jeder Pause neu bestimmt, sondern erst nach einer Zeitspanne von mindestens $5/HZ$.

4.5 Synchronisation von Anfragen

Eine Besonderheit von HTML ist es, dass in eine Seite integrierte Teile (wie z.B. Bilder oder Frames) mit einer eigenen URL versehen sind und eigens angefordert werden müssen (siehe 2.3). Dies hat zur Folge, dass die meisten Anfragen an einen Webserver andere unmittelbar nach sich ziehen, da der Browser bereits beim Darstellen der Seite die fehlenden Teile anfordert. Auch wenn die angeforderten Daten auf dem selben Webserver liegen wie die zuvor angeforderte Seite, wird für jede Anfrage eine eigene TCP-Verbindung aufgebaut. Ein Client kann somit mit mehreren parallelen TCP-Verbindungen mit dem Server verbunden sein, welche sich einzig in der Sender-Portnummer unterscheiden.

Mit dem in Kapitel 4.2 vorgestellten Ablauf einer Anfrage, bei dem die IP-Adresse des Clients mit dem Datenbankeintrag verglichen wird und danach die angeforderten

Daten mit der eingetragenen Bandbreite übertragen werden, entsteht somit ein Konflikt mit der Beschränkung der Bandbreite: Wenn über jede der Verbindungen mit der maximalen Übertragungsrate gesendet wird, wird diese überschritten und die ganzen Bemühungen der Bestimmung und Reduzierung sind sinnlos. Aus diesem Grund muss das Senden der Daten an eine IP-Adresse koordiniert werden. Eine verhältnismässig einfache Lösung dafür ist die Benutzung von Semaphoren, welche es den einzelnen Prozessen erlauben, auf eine Ressource zu warten ohne die Datenbank zu belasten. Auf diese Weise wird auch die Sicherheit beim Zugriff gewährleistet. Auch die Benutzung von Semaphoren löst jedoch wie die ersten beiden Ansätze das Problem des *busy waiting* noch nicht, d.h. wartende Prozesse belasten die CPU unnötig. Zur Lösung des Problems muss für jede Semaphore eine Liste mit Prozessen angelegt werden. Die wartenden Prozesse müssen blockiert werden und beim Freiwerden der Semaphore wird ein blockierter Prozess *geweckt* und ausgeführt. Der Nachteil dieser Lösung ist, dass einzelne Anfragen warten müssen, bis die Ressource frei wird. Dies kann bei kleiner Bandbreite und grossen Dateigrössen den Nachteil haben, dass die TCP-Verbindung einer wartenden Anfrage abgebaut wird, bevor die Ressource freigegeben wurde. Auch die Reihenfolge, in der die wartenden Anfragen abgearbeitet werden ist unbestimmt. Diese Probleme treten allerdings auch bei paralleler Abarbeitung der Anfragen auf und werden deshalb nicht weiter behandelt.

Die Lösung des Problems der parallelen Verbindungen lässt sich somit mit Semaphoren gut bewerkstelligen, da Semaphoren mit geringem Aufwand befriedigende Resultate liefern. Die Funktionalität der Semaphoren (inklusive der Lösung des Problems des *busy waiting*) ist bereits im Betriebssystem enthalten (siehe [35]) und kann von Perl genutzt werden. Beliebig viele wartende Prozesse führen somit zu keiner merklichen Belastung des Prozessors.

5 Implementation

In diesem Kapitel wird beschrieben, wie die zuvor vorgestellten Bausteine implementiert wurden.

5.1 Perl

Zur Implementation dieser Arbeit wurde vorwiegend die Programmiersprache Perl verwendet. Perl hat viele Vorteile, von denen besonders die folgenden genutzt wurden:

- Perl ist eine der führenden Skriptsprachen für CGI-Programmierung und die Bibliothek `CGI` sehr umfassend. CGI-Skripte werden zur Bandbreitenbestimmung und als Benutzerinterface benötigt.
- Regular Expressions: Perl bietet einfache und effiziente Befehle an, die es erlauben Daten zu interpretieren und nach Schlüsselworten (Request Methoden, Header, URLs) zu suchen.
- Socketschnittstellen: Perl bietet die Bibliothek `Socket` an, mit deren Hilfe mit einfachen Befehlen die Socketschnittstellen implementiert und verwaltet werden können,
- Datenbankschnittstelle: Perl unterstützt verschiedene SQL-Datenbanken und bietet eine gute objektorientierte Datenbank-Schnittstelle (`DBI`) an.

5.2 Webserver

Die Grundfunktionalität des Webservers ist sehr einfach: Zunächst liest das Programm eine externe Konfigurationsdatei (httpserv.conf) ein, die globale Einstellungen wie Suchpfade, Serverport etc. enthält. Anschliessend werden verschiedene Initialisierungen (z.B. Datenbankverbindungen, TCP-Server-Socketverbindung) vorgenommen. Zuletzt führt der Server die Hauptschleife aus, während der er auf die Anfragen der Benutzer wartet.

```
        # ******************  HAUPTSCHLEIFE  ****************************
1       while(1){
             my $remaddr;
             $SIG{INT} = \&exit_request_handler;
             ACCEPT_CONNECT:
5             {
                ($remaddr = accept(CHILDSOCKET, SERVERSOCKET))
                || redo ACCEPT_CONNECT;
              }
             autoflush CHILDSOCKET 1;
```

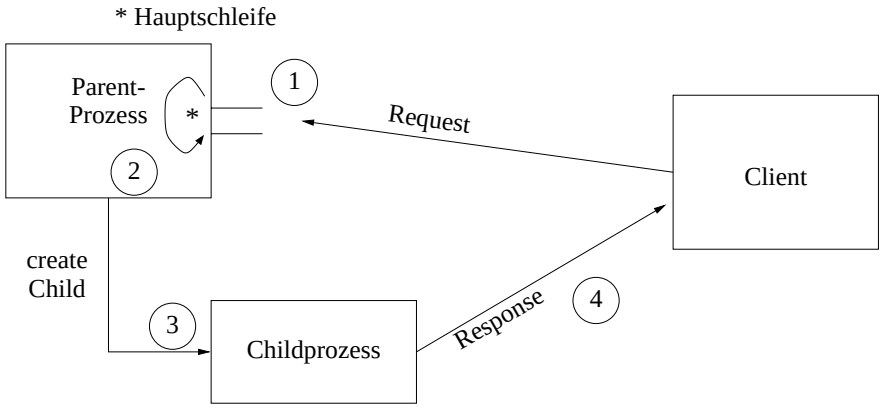


Abbildung 5.1: Übergabe einer Anfrage vom Eltern- zum Kindprozess

```
10          $remip = inet_ntoa((unpack_sockaddr_in($remaddr))[1]);

            my $pid = fork();
            die "Cannot fork, $!" unless defined($pid);
            if ($pid == 0){
15              handle_request($remaddr);
            }
            close(CHILDSOCKET);
        }
        # *****************  ENDE HAUPTSCHLEIFE  **********************
```

5.2.1 Ablauf einer Anfrage (siehe Abbildung 5.1)

Schickt ein Benutzer eine Anfrage an den Webserver werden die folgenden Schritte durchgeführt:

1. die Hauptschleife erhält eine Anfrage
2. die Hauptschleife erzeugt einen neuen Socket (CHILDSOCKET), über den die Kommunikation mit dem Client-Rechner abgewickelt wird, damit der Server weitere Anfragen bearbeiten kann (siehe Code Hauptschleife, Zeile 6).
3. ein neuer (Kind-)Prozess wird erzeugt (Zeile 12), der die Kommunikation übernimmt; der Elternprozess schliesst den Socket für sich (Zeile 17) und kehrt zurück zur Hauptschleife.
4. der Kindprozess empfängt die Anfrage vom Benutzer (Zeile 15), überprüft diese auf syntaktische Korrektheit und führt in einer separaten Funktion (`handle_request`) den entsprechenden Befehl aus.

5.2.2 CGI-Unterstützung

Zur Verarbeitung der Benutzereingaben und insbesondere zur Bandbreitenmessung ist es unabdingbar, dass der Webserver in der Lage ist, eigenständig Programme (CGI-Skripte) auszuführen. Die dazu notwendigen Erweiterungen werden im oben beschriebenen Ablauf in Punkt vier eingebaut (siehe 5.2.1): der Kindprozess erkennt aufgrund der Endung *.cgi*, dass es sich um ein CGI-Skript handelt. In diesem Fall startet der Kindprozess seinerseits einen Prozess, der das Programm ausführt. Falls das Programm etwas auf den Standardausgabekanal schreibt, wird dies vom Kindprozess an den Client-Rechner weitergeleitet:

```
        if($url =~ /\.cgi/){
              $url =~/(.*\.cgi)\?(\d*)/;
              print CHILDSOCKET "HTTP/1.1 200 OK\n";
              if(defined($2)){
5                 print CHILDSOCKET 'env SHMID='$shmid' REQUEST_METHOD='GET'
                         QUERY_STRING=$2
                  $htmlpath$1';
              }else{
                  print CHILDSOCKET 'env SHMID='$shmid' REQUEST_METHOD='GET'
                         $htmlpath$url';
10            }
          }
```

Zur Parameterübergabe vom Webserver an das Programm (z.B. Eingaben, die vom Benutzer getätigt wurden) sind zwei Möglichkeiten vorgesehen: für kleinere Datenmengen kann der Browser des Benutzers die Eingaben an die URL des CGI-Skripts anhängen und diese URL mittels eines GET-Requests an den Server schicken. In diesem Fall muss der Webserver die Eingaben als Umgebungsvariable (`QUERY_STRING` setzen, bevor er das Programm startet. Sollten grössere Datenmengen vom Client an den Server (etwa ganze Dateien) übertragen werden, muss der Browser die POST Methode verwenden und schickt die Daten im Nachrichten-Body seines Requests (siehe dazu Kapitel 2.4). Der Server liest die Daten ein und übergibt sie über den Standardeingabekanal an das CGI-Skript.

```
1       sub handle_POST_request{
            my $url=shift @_;
            $url =~ s/http:\/\/.*?\///g; #remove "http://host:port" from the url
            my($tmp,@tmp,$length,$argument_string,$command);
5           while($tmp = <CHILDSOCKET>) {
                if($tmp =~ /Content-Length/){
                    @tmp= split(" ",$tmp);
                    $length=$tmp[1];
                    read(CHILDSOCKET,$tmp,2); # Zeilenumbruch auslesen ...
10                  read(CHILDSOCKET,$argument_string,$length);
                    last;
                }
            }
            if($url =~ /\.cgi/){
15              $command= "echo \"$argument_string\"|env SHMID=\"$shmid\"
                          REQUEST_METHOD=\"POST\" $htmlpath$url";
```

```
                print CHILDSOCKET "HTTP/1.1 200 OK\n";
                print CHILDSOCKET ‘$command‘;
            }
            else{
                print STDERR "POST Request only allowed for cgi:\n" if $opt_d;
                bad_request(*CHILDSOCKET);
            }
        }
```

Das Skript kann über die REQUEST-METHOD-Umgebungsvariable (siehe 2.4.1) herausfinden, mit welcher Methode es aufgerufen wurde und seine Parameter entsprechend einlesen. Anschliessend muss noch die Kodierung des Parameterstrings (siehe Kapitel 2.4.3) rückgängig gemacht werden. Nun können die Name/Wert-Paare in einem assoziativen Array abgelegt werden.

```
if(uc($ENV{’REQUEST_METHOD’}) eq "GET"){
      $eingaben=$ENV{’QUERY_STRING’};
  }else{
      $eingaben = <>;
  }
  # ’+’-Zeichen durch Leerzeichen ersetzen
  $eingaben =~ s/\+/ /g;
  # ’%xx’-Kodierungen durch ASCII-Zeichen ersetzen
  $eingaben =~ s/%([\d|a-f|A-F]{1,2})/pack("C",hex($1))/eig;
  @eingabe_liste=split(/&/,$eingaben);
  foreach $item (@eingabe_liste) {
      ($name,$wert)=split( /=/,$item);
      $eingabe{$name}=$wert;
  }
```

5.2.3 Proxy-Funktionalität

Da der Webserver auch die Funktion der Bandbreitenbeschränkung für den Benutzer übernimmt, ist es sinnvoll, dass sämtliche Anfragen des Benutzers, das heisst also auch solche Anfragen nach Seiten, die nicht auf dem Webserver selbst gespeichert sind, zunächst an den Webserver geschickt werden. Der Webserver kann dann die benötigte Information holen und mit der entsprechenden Bandbreitenbeschränkung an den Benutzer weiterleiten. Dazu ist es nötig, dass der Webserver auch Client-Funktionalität besitzt.

Diese Funktionalität wird jedoch nur im Fall einer externen Webseite benötigt. Der die Abfrage behandelnde Prozess erkennt dies daran, dass in der Request-Zeile (siehe 2.3) die URL einen fremden Webserver adressiert. Der Prozess kann nun im Cache die entsprechende Seite suchen, oder gegebenenfalls die Seite neu laden. In Perl gibt es verschiedene Bibliotheksfunktionen, die es ermöglichen eine Seite über ihre URL zu laden. In dieser Implementation wurde dazu die `get`-Funktion des `LWP::Simple` Perl-Moduls verwendet.

```
if(defined($content=get($url))){
        sendbw(*CHILDSOCKET,$content,$bw);
}
```

5.3 Datenbanken

MySQL ist eine Standarddatenbankanwendung unter Linux, deren breite Unterstützung (insbesondere auch durch die Perl Programmiersprache) und Einsatzmöglichkeiten (z.B. im Webserver Apache) sie für diese Anwendung geeignet erscheinen lassen. In dieser Implementation wurde eine Datenbank mit zwei Haupttabellen angelegt, je eine für die Benutzerverwaltung und für das Cache:

```
mysql> show tables;
+--------------------+
| Tables_in_httpserv |
+--------------------+
| proxy              |
| users              |
+--------------------+
2 rows in set (0.00 sec)
```

5.3.1 Benutzerverwaltung

Die Daten des Benutzers, welche von der Datenbank gespeichert werden müssen, umfassen den eingegebenen bzw. ermittelten Wert der Übertragungsbandbreite sowie die IP-Adresse des aktuellen Computers, den der Benutzer verwendet. Die IP-Adresse wird dazu benötigt, die Mobilität des Benutzers zu bemerken und zu unterstützen: wechselt die IP-Adresse eines Benutzers, so wird sich mit grosser Wahrscheinlichkeit auch die ihm zur Verfügung stehende Bandbreite ändern, die Werte in der Datenbank müssen also angepasst werden, damit eine gleichbleibende Qualität gewährleistet werden kann. Dies kann zum einen bedeuten, dass die Sendebandbreite erhöht wird, damit der Benutzer von der grösseren Geschwindigkeit profitieren kann, aber auch eine Reduktion der Bandbreite, da dadurch Paketverluste und Übertragungswiederholungen vermieden werden können. Dieses Verhalten der Anfrage neuer Bandbreitenwerte nach eine IP-Adresswechsel ist jedoch unerwünscht, falls der Benutzer Mobile IP verwendet und mit den jeweiligen fremden Subnetzen eine gleichbleibende Bandbreite ausgehandelt hat. In diesem Fall bleibt die Bandbreite nach einem IP-Adresswechsel konstant und der Benutzer empfindet die Nachfragen des Webservers als störend. In dieser Situation könnte eine Lösung darin bestehen, dass Nachfragen nur nach längerer vorangegangener Inaktivität des Benutzers ausgelöst werden, oder gar vom Benutzer beim login-Vorgang völlig ausgeschlossen werden können. Da aber die Möglichkeit über mehrere Subnetze hinweg Bandbreite zu reservieren noch nicht weit verbreitet ist, wurde auf eine Implementation dieser Massnahmen verzichtet.

Die Tabelle, in der die Benutzerdaten gespeichert werden, sieht folgendermassen aus:

```
mysql> select * from users;
+----------+----------+------------------+--------+------------------+
| login    | passwd   | ip               | screen | bw               |
+----------+----------+------------------+--------+------------------+
| user1    | asdf     | 130.92.66.132    |      0 |  65930564.349425 |
| user2    | qbrdbg   | 130.92.63.24     |      0 | 62477035.3660059 |
|...
```

Die Benutzer-Datenbank verwendet die IP-Adresse des anfragenden Rechners als Index für die Datenbanksuche. Das ist deshalb notwendig, da die Übertragung des Benutzernamens inklusive der notwendigen Autorisierung im normalen HTTP-Protokoll nicht möglich ist. Der Webserver hat somit keine andere Möglichkeit zu erfahren, von welchem Benutzer eine beliebige Anfrage kommt. Da die Verwendung des Proxy-Servers für den Benutzer möglichst transparent geschehen sollte, ist auch die Einkapselung der Benutzer-URL zusammen mit einer Session-ID in eine Anfrage an den Webserver unerwünscht. Die Möglichkeit, dass zwei verschiedene Benutzer die gleiche IP-Adresse bei ihren Anfragen verwenden, ist zwar vorhanden (z.B. falls sich beide Benutzer im selben privaten Netz aufhalten, haben beide Anfragen die IP-Adresse des NAT-Servers) aber im Anwendungsszenario der Applikation eher unwahrscheinlich. In obigem Szenario wird den beiden Benutzern mit grosser Wahrscheinlichkeit auch ein Proxy zur Verfügung stehen, dessen Verwendung in den meisten Fällen sogar obligatorisch ist. Daher ist es den Benutzern gar nicht möglich, den `httpserv` Webserver als Proxy für externe Requests anzugeben. Eine andere Möglichkeit von mehreren Benutzern mit der selben IP-Adresse ist die Benutzung eines gemeinsamen Servers. Auch da ist eine Differenzierung nicht nötig, da alle Benutzer bis zum Server die selbe Bandbreite zur Verfügung haben.

5.3.2 Cache

Einträge in der Cache-Tabelle sind die URL sowie der Inhalt der angeforderten HTML-Seite. Die URL wird dabei als Suchschlüssel verwendet. Zusammen mit der IP-Adresse des anfragenden Rechners können aber auch leicht in einem weiteren Ausbauschritt mehrere Versionen einer HTML-Seite gespeichert werden, die auf die besonderen Anforderungen verschiedener Benutzer zugeschnitten sind. Als zusätzliche Information ist in der Tabelle des weiteren das Datum und die Uhrzeit des letzten Zugriffs vermerkt (in UTC=Sekunden seit dem 01.01.1970). Mit dieser Angabe kann der Webserver entscheiden, ob eine Seite noch aktuell ist, oder vom Ursprungsserver neu geholt werden sollte. Die Dauer während derer der Webserver eine Seite als aktuell betrachtet kann beim Start des Webservers angegeben werden (default-Einstellung: 1 Stunde). Folgend ist die Datenbanktabelle des Caches dargestellt, der Inhalt des Feldes `content` wurde aus Platzgünden weggelassen.

```
mysql> select * from proxy;
+--------------------------+------------------+------------------+
| url                      | time             | content          |
+--------------------------+------------------+------------------+
| http://www.iam.unibe.ch/ | 1079240972.71566 | ...............  |
|...
```

Die Abfrage nach der Gültigkeit des Cache-Eintrages und die eventuelle Aktualisierung des Caches geschieht im Webserver mit folgenden Biblitheksaufrufen der Perl - DBI Bibliothek:

```
#----------------------page already cached? -------------------------
1       if($proxy_disable==0 && $proxy_ignore==0){
            print STDERR "SQL query: SELECT * FROM proxy WHERE url = '$url'\n"
            if $opt_d;
            $sqlquery=$dbh->prepare("SELECT * FROM proxy WHERE url = '$url'");
5           $sqlquery->execute;
            $row=$sqlquery->fetchrow_hashref;

            $time=gettimeofday;
            if($row && ($time-$$row{'time'}>3600) ){
10              print STDERR "Cache content out of date, getting new\n" if $opt_d;
                $row="";
            }
        }
        else{
15          $row="";
        }
        # print HTTP-Header
        print CHILDSOCKET "HTTP/1.1 200 OK\n";
        # ...
20
        if($row){
            print STDERR "Cache hit!\n" if $opt_d;
            semtake($remip);
            sendbw(*CHILDSOCKET,$$row{'content'},$bw);
25          semgive($remip);
        }
        elsif(defined($content=get($url))){
            print STDERR "Cache miss!\n" if $opt_d && !$proxy_disable &&
              !$proxy_ignore;
30          semtake($remip);
            sendbw(*CHILDSOCKET,$content,$bw);
            semgive($remip);
            if($proxy_disable==0){
                # if the cache is disabled permanently, it is not updated
35              print STDERR "Updating Cache.\n" if $opt_d;
                $content =~ s/\'/\\\'/g;       # mask single quotes (') for SQL query
                $time=gettimeofday;
                $sqlquery=$dbh->prepare("INSERT INTO proxy VALUES ('$url',$time,
                  '$content')");
40              $sqlquery->execute;
            }
        }
        else{
            print STDERR "could not get URL $url\n" if $opt_d;
45          bad_request(*CHILDSOCKET);
        }
        $sqlquery->finish;
```

5.4 Implementation der Bandbreitenbestimmung

5.4.1 Zeitnahme im HTTP-Server

Innerhalb des HTTP-Servers wurde zuerst ein neuer Modus definiert: der normale Modus des Servers entspricht der üblichen Arbeitsweise eines HTTP Servers, im Testmodus hingegen wird der Server die zur Zeitmessung nötigen Schritte unternehmen. Der Server dabei ist in der Lage, gleichzeitig Anfragen in beiden Modi zu bearbeiten, was wichtig ist, falls er auch als Proxy-Server eingesetzt wird. Da während einer Messung

mehrere Seiten bzw. Bilder aufgerufen werden können und jede neue Seite bzw. Bild einen neuen Prozess des HTTP Servers verursacht, musste zum Erkennen des Testmodus und zur Übergabe der Variablen ein geteilter Speicherbereich eingerichtet werden:

```
1       #shared memory anlegen und initialisieren
        $shmid=shmget (0,4096,&IPC_CREAT | 0666) or die
          "Couldn't get shared memory segment: $!\n";
        $testphase=0;
5       shmwrite($shmid,$testphase,$testphase_offset,$testphase_size);
        shmwrite($shmid,0,$deltaoffset,$deltasize);
        shmwrite($shmid,0,$bwoffset,$bwsize);
```

Innerhalb des HTTP-Servers existiert eine Unterroutine, die aufgerufen wird, sobald der Benutzer einen GET-Request für das CGI-Skript sendet. Diese Routine überprüft zunächst, ob ein Bandbreitentest derzeit möglich ist (es ist jeweils nur ein Benutzer zugelassen), versetzt anschliessend den HTTP Server in den Testmodus und startet das CGI-Skript.

```
1       #---check if bandwidth testing is requested and possible
        shmread($shmid,$testphase,$testphase_offset,$testphase_size);
        if($url =~ /$testseite/){
            if($testphase=~/0/){
5               print STDERR "entering testphase...\n" if $opt_d;
                shmwrite($shmid,1,$testphase_offset,$testphase_size);
            } else{
                print STDERR "testphase is: $testphase\n";
                double_request(*CHILDSOCKET);
10          }
        }
```

Da sich der Server im Testmodus befindet, startet unmittelbar nachdem das CGI-Skript eines der speziell generierten Bilder (siehe 4.3.1) vom Server anfordert die Zeitmessung. Nachdem das Bild vollständig übertragen ist, beendet der Client die Verbindung, und die Zeitmessung stoppt. Das berechnete Intervall wird dem aufrufenden HTTP-Server Prozess über den geteilten Speicherbereich übergeben. Da die Grösse des Bildes bekannt ist, kann aus dem Zeitintervall ein Durchschnittswert für die Übertragungsbandbreite bestimmt und auf einer Ergebnisseite angezeigt werden.

```
1       #start timing for downloading pics
        if(($testphase=~/1/) && ($url =~ /[$picpath]/)){
            $starttime=gettimeofday;
        }
5       # stop time needed for downloading pics if in testmode
        if(($testphase=~/1/) && ($url =~ /[$picpath]/)){
            $endtime=gettimeofday;
            for($i=0;$i<=$#pics;$i++){
                if($url =~ /$pics[$i]/){
10                  $size_of_pic=$pic_size[$i];
                    last;
                }
            }
```

```
            $delta=$endtime - $starttime;
15          $bw=8*$size_of_pic /$delta;
            shmwrite($shmid,0,$testphase_offset,$testphase_size);
            shmwrite($shmid,0,$deltaoffset,$deltasize);
            shmwrite($shmid,0,$bwoffset,$bwsize);
            shmwrite($shmid,$delta,$deltaoffset,$deltasize);
20          shmwrite($shmid,$bw,$bwoffset,$bwsize);
        }
```

5.5 Beschränkung der Sendebandbreite

Die Funktion **sendbw(SOCKET,$data,$bw)** (siehe Programmcode) implementiert den Algorithmus zur Beschränkung der Sendebandbreite von TCP, der in Kapitel 4.1.5 vorgestellt wurde.

```
#-------------------------------------------------------------------------------
# sendbw($SOCKET,$data,$bw)
# ----------------------
#   * send a string $data over a socket $SOCKET with restricted bandwidth
#     $bw (in bit/s)
#
#-------------------------------------------------------------------------------
1       sub sendbw{
             my($SOCKET)=shift;
             my($data)=shift;
             my($bw)=shift;
5            print "sending with bandwidth $bw bits/s \n" if $opt_d;

             my($starttime,$delta,$sent,$sent_now,$offset,$length,$sendsize);
             my($res) = 0.01; #timing accuracy of select = 1/HZ

10           $offset=0;
             $length = length($data);

             $sendsize = $bw / 8.0 * 2 * $res;
             if($sendsize<50){$sendsize = 50;}
15           if($sendsize>1000){$sendsize = 1000;}

             print "sending $length bytes in packets of $sendsize\n" if $opt_d;

             $starttime=gettimeofday;
20           $sent_now=0;
             do{
                 $sent=syswrite($SOCKET,$data,$sendsize,$offset);
                 $delta=gettimeofday-$starttime;
                 $offset+=$sent;
25               $sent_now+=$sent;
                 if($sent_now*8/($delta+ 0.5*$res) > $bw){
                     select(undef,undef,undef,($sent_now*8/$bw)-$delta-0.5*$res);
                     if($delta >5*$res){
                         $starttime = gettimeofday;
30                       $sent_now=0;
                     }
                 }
             }while($offset<$length);
         }
```

Die Synchronisation paralleler HTTP-Verbindungen geschieht mit Hilfe der Funktionen `semtake` und `semgive` (siehe Programmcode). Beide Funktionen wurden

anhand der in der perlipc-manpage [42] vorgestellten Beispiele entwickelt. Einzige Erweiterung ist der Gebrauch eines Semaphorenpools, der es erlaubt, beliebig viele HTTP-Verbindungen zu verschiedenen Clients zu synchronisieren.

```
1       sub semtake{
            my($ip)=shift;
            my($semid,$semnum,$semop,$semflag,$opstring1,$opstring2,$opstring);

5           $semid = semget($IPC_KEY,  0 , 0 );
            die if !defined($semid);

            $semnum = $semaphore{$ip};
            $semflag = 0;
10          print "taking semaphore $semnum for IP $ip\n" if $opt_d;

            # 'take' semaphore
            # wait for semaphore to be zero
            $semop = 0;
15          $opstring1 = pack("s!s!s!", $semnum, $semop, $semflag);

            # Increment the semaphore count
            $semop = 1;
            $opstring2 = pack("s!s!s!", $semnum, $semop,  $semflag);
20          $opstring = $opstring1 . $opstring2;

            semop($semid,$opstring) || die "$!";
        }

25      sub semgive{
            my($ip)=shift;
            my($semid,$semnum,$semop,$semflag,$opstring);

            $semnum = $semaphore{$ip};
30
            # 'give' the semaphore
            $semid = semget($IPC_KEY, 0, 0);
            die if !defined($semid);

35          $semflag = 0;

            print "give back semaphore $semnum for IP $ip\n" if $opt_d;
            # Decrement the semaphore count
            $semop = -1;
40          $opstring = pack("s!s!s!", $semnum, $semop, $semflag);

            semop($semid,$opstring) || die "$!";
        }
```

Dabei wird jeder Client-IP-Adresse eine Semaphore zugewiesen, die den Zugriff auf die Sendeprozeduren regelt:

```
        semtake($remip);
        sendbw(*CHILDSOCKET,$$row{'content'},$bw);
        semgive($remip);
```

Somit ist sichergestellt, dass zu jeder Zeit nur ein Prozess an eine Client-IP sendet. Man muss also für jede Anfrage eines neuen Clients eine neue Semaphore bereitstellen. Das

Linux Betriebssystem stellt jedoch Semaphoren blockweise in Einheiten bis zu 250 Stück zur Verfügung. Um nicht eine willkürliche Beschränkung auf 250 gleichzeitig zugreifende Clients einzuführen, wird bei Bedarf ein neuer Block von 250 Semaphoren vom Betriebssystem angefordert und bereitgestellt (siehe im folgenden Programmcode Zeile 3).

```
1       if($semindex==$maxsem){
            # allocate new semaphore list
            $semid = semget($IPC_KEY + ($semindex/250), 250, 0666 | IPC_CREAT )
                || die "$!";
5           print(STDERR "semid is: $semid\n") if $opt_d;
            for($i=$semindex;$i<$semindex+250;++$i){
                $sempool[$i]=$i;
            }
            $maxsem+=250;
10      }

        $semaphore{$remip} = $sempool[$semindex];
        print STDERR "semaphore for $remip is $semaphore{$remip}\n" if $opt_d;
        $semindex++;
```

Damit ist sichergestellt, dass immer genügend Semaphoren vorhanden sind. Das Betriebssystem stellt zudem beim Gebrauch der Semaphoren sicher, dass die Prozesse blockiert werden und dass kein *busy-waiting* auftritt.

Nach dem Abbruch der Verbindung muss noch dafür gesorgt werden, dass die Semaphore wieder in den Pool zurück gegeben wird. Dazu gibt der Kindprozess die Semaphorennummer als Rückgabewert an: `exit($semaphore{$remip});`
Der Hauptprozess des Webservers empfängt bei Beendigung des Kindprozesses ein SIGCHLD Signal, was durch Installation eines entsprechenden Signalhandlers aufgefangen werden kann und entsprechende Aktionen durchgeführt werden können. Hier wird die vom Kind zurückgegebene Semaphore wieder in den Pool eingefügt:

```
        #Signalhandler f"ur SIGCHLD
1         sub child_handler{
              wait;
              my($sem)=$?>>8;
              print STDERR "child returned semaphore number $sem\n" if $opt_d;
5             $semindex--;
              $sempool[$semindex]=$sem;
          }
```

6 Resultate

In diesem Kapitel werden die Resultate der Messungen der einzelnen Komponenten aufgezeigt und analysiert. Als erste Messung wurde die Auflösung der Zeitnahme des Servers getestet, um die Genauigkeit der darauffolgenden Messungen bestimmen zu können. Danach wurde die Zuverlässigkeit des Bandbreitentesters gemessen, wobei die vom Programm berechneten Übertragungsraten mit dem effektiven Netzverkehr verglichen wurden. In Kapitel 6.3 wurde gemessen, ob bei einer Beschränkung der Sendebandbreite die eingestellte Übertragungsrate erreicht wurde. In Kapitel 6.4 schliesslich wurde untersucht, ob eine Bandbreitenreduzierung zu besseren Übertragungsraten oder stabileren Verhältnissen während der Übertragung führen kann.

6.1 Genauigkeitsmessung der Zeitnahme des Servers

Wie in 5.4 beschrieben ist das Anhalten eines Prozesses für kurze Zeitspannen mit Problemen verbunden. Um eine möglichst genaue Anpassung der Senderate zu erreichen, wird der Sendeprozess Bruchteile von Sekunden unterbrochen (suspendiert). Dies geschieht mit Hilfe des Befehls `select`. `select` ist ein relativ portabler Weg, um einen Prozess mit Subsekunden-Genauigkeit zu suspendieren. Die Genauigkeit, mit der das beschränkt werden kann, ist die minimale Zeitdauer, die `select` einen Prozess suspendieren kann.

Mit dem Hilfsprogramm *timingtest.pl* wurde daher zuerst gemessen, mit welcher Genauigkeit die Bandbreite angepasst werden kann. Erst wurde gemessen, wie gross die Zeitspanne zwischen zwei Zeitaufrufen (`gettimeofday`) ist. Danach wurde gemessen, wie lange die effektive Wartezeit zwischen Aufruf und Rückkehr des `select`–Befehls dauert, wobei dem Befehl jeweils eine andere Verzögerung übergeben wurde. Um die Verzögerung genau zu bemessen, wurde dem gemessenen Wert der durchschnittliche Wert des Zeitaufrufbefehles *gettimeofday* abgezogen. Diese Messungen haben ergeben, dass die Zeit, die `select` einen Prozess warten lässt, nicht der gerundete Wert auf die nächsten 10 ms ist, sondern dass in jedem Fall auf die nächsten 10 ms aufgerundet wird (die Auflösung von 10 ms ist wie bereits in Kapitel 4.4 beschrieben, abhängig vom Kernel). Die in der Tabelle 6.1 angegebenen Resultate sollen dies illustrieren (Mittelwerte aus 1000 Messungen und daraus schliessbare effektiver Verzögerung, alle Werte in Sekunden).

Aus den ersten beiden Ergebnissen sieht man, dass es nicht möglich ist, eine kleinere Auflösung als 10 ms zu erreichen, egal ob man eine Verzögerung von 0.00001 s oder von 0.009 s angibt. Beide Angaben bewirken eine effektive Verzögerung von 0.01 Sekunden. Die dritte Messung bezeugt, dass eine kleine Überschreitung einer Zehntelsekunde bereits zu einer effektiven Verzögerung von 20 ms führt (siehe dazu [34]). Um diesem Effekt entgegenzuwirken wird bei jeder Verzögerungsangabe 5 ms abge-

angegebene Verzögerung	gemessene Verzögerung	effektive Verzögerung
0.00001	0.0101594579219818	0.01
0.009	0.0098338959217072	0.01
0.011	0.0199930448532104	0.02

Tabelle 6.1: Interne Verzögerung von *select*

zogen. Dadurch erreicht man, dass ein Prozess so lange suspendiert wird, wie es dem statistisch gerundeten Wert entspricht.

6.2 Bandbreitenbestimmung

Bei der Bandbreitenbestimmung wurden (wie bereits in Kapitel 4.3.1 beschrieben) speziell generierte Bilder übertragen. Nach jeder Übertragung wurde die gemessene Zeit betrachtet und falls die im Konfigurationsfile gesetzte Schranke s von 2 Sekunden nicht überschritten wurde, wurde das nächst grössere Bild gesendet, welches in etwa die doppelte Grösse des zuvor gesendeten Bildes hatte (siehe auch Kapitel 4.3). Zwischen den Übertragungen wurde jeweils $s + 1$ Sekunden gewartet bevor das nächste Bild übertragen wurde.

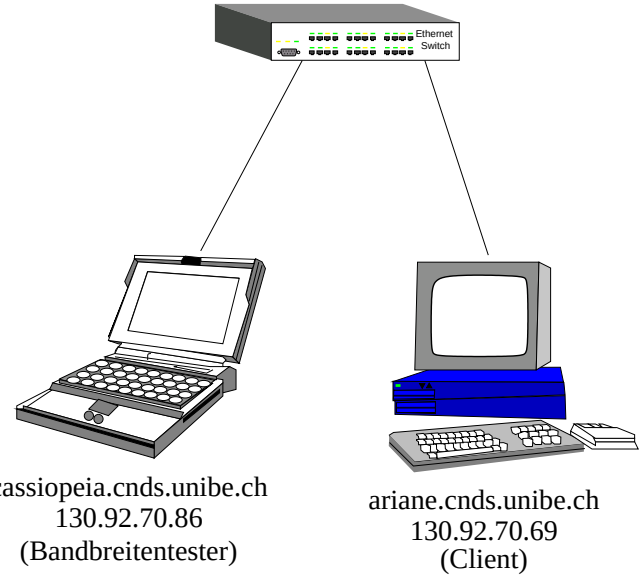


Abbildung 6.1: Architektur der Testumgebung für 100 Mbit Ethernet

Dabei wurden wieder folgende Bildgrössen verwendet: 8753 Bytes, 16’856 Bytes, 32’963 Bytes, 65’252 Bytes, 130’242 Bytes, 259’792 Bytes, 518’593 Bytes, 1’036’612 Bytes und 2’073’621 Bytes. Um die Messungen des Bandbreitentesters zu überprüfen,

wurde gleichzeitig auf dem Client der gesamte Verkehr auf dem entsprechenden Link gemessen. Dies geschah mit der Funktion `tcpdump`. Nachträglich wurden die beiden Messergebnisse verglichen.

6.2.1 100 Mbit/s Ethernet

Dateigrösse in Bytes	Mittelwert in bits/s	Sendedauer in s	S in %
8’753	99’327’882.32	0.0007	0.58
16’856	57’319’535.33	0.0024	1.59
32’963	83’427’135.44	0.0032	0.32
65’252	88’418’681.14	0.0084	0.14
130’242	92’114’270.01	0.0115	0.22
259’792	64’474’068.03	0.0323	0.15
518’593	64’850’861.28	0.0640	1.10
1’036’612	63’145’072.59	0.1313	0.11
2’073’621	62’022’355.89	0.2676	0.26

Tabelle 6.2: Mittelwerte der Ergebnisse des Bandbreitentests auf einer 100 Mbit Ethernetleitung mit Standardabweichung S

In Tabelle 6.2 wird die erreichte Bandbreite für die Übertragung der einzelnen Bilder auf einer 100 Mbit/s Ethernetleitung aufgelistet. Der Wert ergibt sich jeweils aus der Mittelung über 20 Messungen. Dabei wurde die Bandbreite bei allen übertragenen Bildern gemessen, um die Resultate besser auswerten zu können. Dabei zeigt es sich, wie wichtig es ist, eine untere Schranke einzuführen, ab der die Messung als hinreichend genau betrachtet werden kann. Erst ab dem sechsten Bild (oder einer Übertragungsdauer von mindestens 30 ms) wurden konstante Messungen für verschiedene Dateigrössen erzielt. Dies ist wichtig, da sonst nicht davon ausgegangen werden kann, dass der gemessene Wert der tatsächlich erreichten Bandbreite entspricht. Die sehr geringe Standardabweichung im Promillebereich gibt die geringe Streuung der Messungen auch für kleine Dateigrössen wieder. Allerdings darf in diesem Fall nicht aus einer geringen Standardabweichung auf die Korrektheit der Messung geschlossen werden, da weitere Faktoren das Ergebnis negativ beeinflussen können. Zum einen ist die bereits besprochene Zeitauflösung ein Problem: Wie in 4.4 beschrieben, hat der Kernel eine Zeitauflösung von 0.01 Sekunden. Alle Messungen, die diesen Wert nicht klar überschreiten, sind daher unzuverlässig. In Tabelle 6.2 betrifft diese Ungenauigkeit die ersten fünf Messungen.

Eine weitere Quelle für Messungenauigkeiten stellt der Socket-Puffer dar. Dessen Grundeinstellung liegt im Linuxkernel bei 64 kbytes. Die Messungen haben nun ergeben, dass genau ab dieser Bildgrösse die Messschwankungen abnehmen. Zwar liegt die Messung des nächsthöheren Bildes noch über dem effektiv gemessenen, dies kann aber

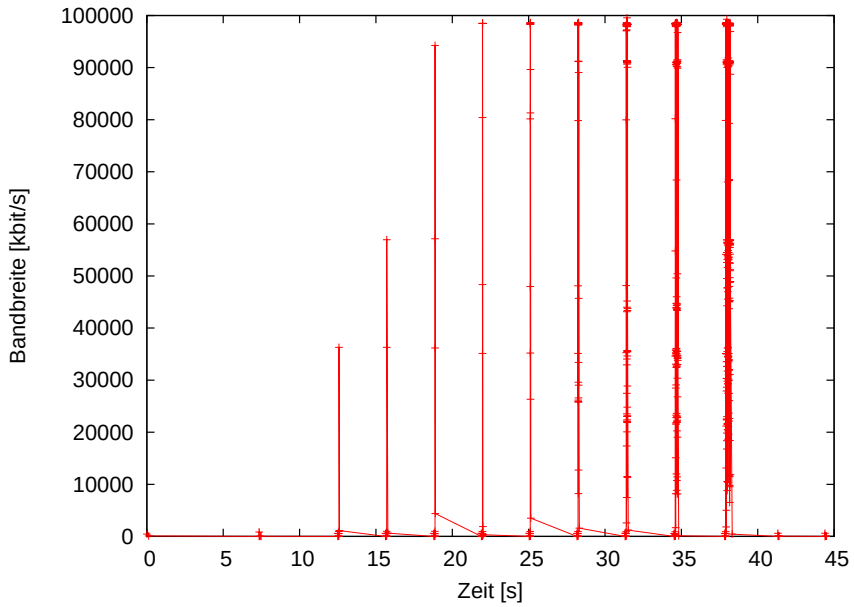


Abbildung 6.2: Auswertung von `tcpdump` während der Bandbreitentests auf 100 Mbit Ethernet, Auflösung 1ms

auf das Zeitauflösungsproblem im Kernel zurückgeführt werden, da sich die Sendedauer im kritischen Messbereich von 10 ms befindet. Das Problem des Socket-Puffers fällt bei kleineren Übertragungsraten mehr ins Gewicht, da bei diesen Messungen die kritische Socket-Puffergrösse aufgrund der Mindestmessdauer ohnehin überschritten wird.

Als Vergleich zu den Messresultaten des Bandbreitentesters wurde zunächst mit der `tcpdump`-Funktion der gesamte Netzverkehr auf dem Server protokolliert. In einem zweiten Schritt konnte aus diesen Daten die tatsächlich übertragene Datenmenge an den Client herausgefiltert (mit `tcpdump -e -n -tt -r <Dateiname> tcp and src <senderip>`) und die Bandbreite mit unterschiedlicher Auflösung berechnet werden. Abbildung 6.2 zeigt die Ergebnisse der Auswertung, falls die Bandbreite über eine Millisekunde gemittelt wurde. Auf der Graphik, auf der der ganze Testlauf dargestellt ist, erkennt man die Übertragung der einzelnen Bilder im Abstand von jeweils drei Sekunden. Die volle Sendebandbreite wurde ab der vierten Übertragung zumindest temporär erreicht. Abbildung 6.3 zeigt das selbe Experiment, wobei die Mittelung von `tcpdump` über 10 ms berechnet wurde. Daraus kann man ablesen, dass im Durchschnitt nur eine Bandbreite von etwas über 60 Mbit/s erreicht wurde, was eine gute Übereinstimmung mit den Messungen des Bandbreitentesters ergibt (siehe Tabelle 6.2).

Aus den gemessenen Werten lassen sich folgende Schlussfolgerungen ziehen: Die Messdauer muss eine gewisse Grenze überschreiten, damit der Bandbreitentest ei-

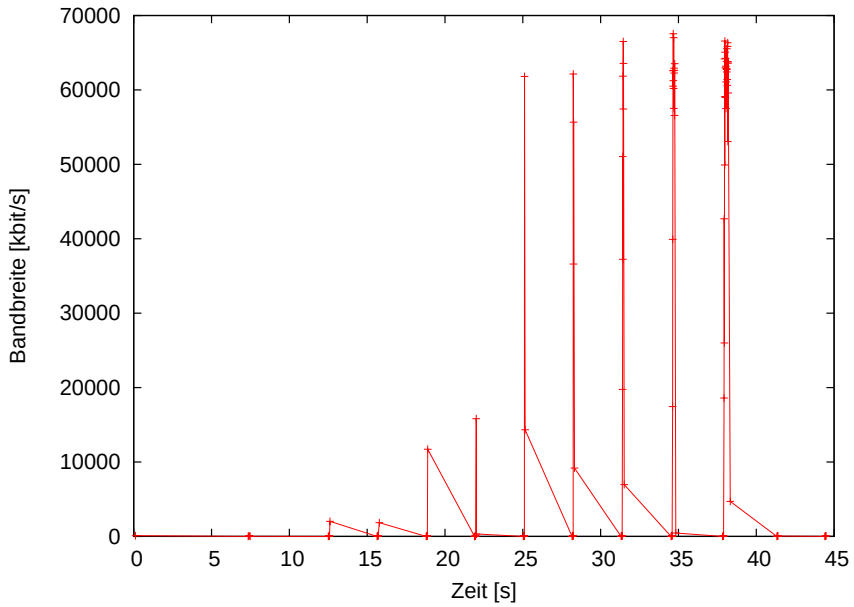


Abbildung 6.3: Die selbe Messung mit `tcpdump` wie in Abbildung 6.2 in 10ms Auflösung

ne ausreichende Genauigkeit erreicht. Das Setzen einer Mindestübertragungsdauer ist deshalb unerlässlich und der Wert sollte auf jeden Fall ein Mehrfaches der Kernel-auflösungszeit (in diesem Fall ein Mehrfaches von 10 ms) betragen.

6.2.2 11 Mbit/s Wireless LAN

Alle Messungen bei Wireless LAN wurden in der in Abbildung 6.4 dargestellten Umgebung gemacht. Die Bandbreitentests ergaben sehr unterschiedliche Messergebnisse: Es wurden als erstes Messungen während der Arbeitszeit durchgeführt, wobei ein durchschnittlicher Wert von 1.6375 Mbit/s erreicht wurde. Weiter wurde ausserhalb der Arbeitszeit gemessen, wobei eine mittlere Bandbreite von 5.697 Mbit/s erreicht wurde, mit einem Spitzenwert von über 6 Mbit/s.

Die unterschiedlichen Messresultate können zwei Gründe haben, welche auf Eigenheiten von Wireless LAN zurückzuführen sind: Einerseits ist bei Mehrfachbenutzung des Mediums die Bandbreite kleiner als bei einer alleinigen Nutzung, da das drahtlose Medium als *Shared Medium* die verfügbare Bandbreite unter allen Benutzern aufteilt. Andererseits kann allein die Anwesenheit eines Teilnehmers mit beschränkter Bandbreite (z.B. 2 Mbit/s) die Bandbreite aller anderen Teilnehmer auf diesen Wert reduzieren (siehe Kapitel 1.3).

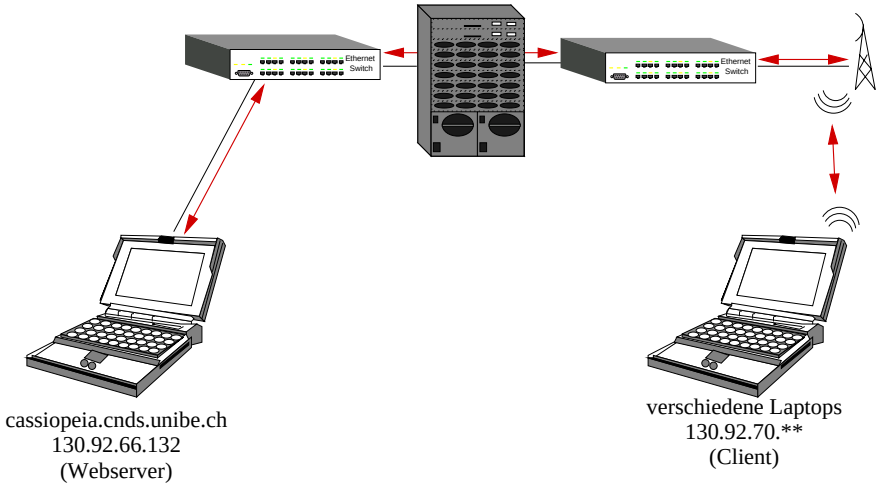


Abbildung 6.4: Architektur der Testumgebung für Wireless LAN

Die Resultate der Messungen des Servers stimmen in beiden Fällen sehr gut mit den Daten überein, die man durch das Protokollieren des gesamten Netzverkehrs erhält. Auch die Verlässlichkeit der Messung ist mit einer durchschnittlichen Standardabweichung bei der berechneten Bandbreite von 0.0019 % sehr genau.

Auch bei diesen Messungen wird deutlich, dass eine zu kurze Messung unzuverlässig ist. Realistische Werte wurden auch hier ab einer Messdauer von einem Mehrfachen der Kernelauflösezeit erreicht. Bei den Messungen während der Arbeitszeit war die erste Messung nicht aussagekräftig (siehe Tabelle 6.3), bei freiem Medium dauerten die ersten beiden Messungen nicht lange genug (siehe Tabelle 6.4). Die Abnahme der Bandbreite bei grösseren Dateien auf der drahtlosen Leitung ist auf vermehrte Kolli-

Dateigrösse in Bytes	Mittelwert in bits/s	Sendedauer in s
8’753	95’018’422	0.000797
16’856	3’317’614	0.0406
32’963	3’135’635	0.0840
65’252	2’088’932	0.2498
130’242	2’408’888	0.4325
259’792	1’830’342	1.1354
518’593	1’636’231	2.5365

Tabelle 6.3: Mittelwerte der Ergebnisse des Bandbreitentests auf einer 11 Mbit W-LAN Leitung während der Arbeitszeit

Dateigrösse in Bytes	Mittelwert in bits/s	Sendedauer in s
8’753	111’025’842.20	0.000778
16’856	9’833’696.87	0.0169
32’963	6’349’597.09	0.0514
65’252	7’467’849.03	0.0864
130’242	6’854’186.98	0.1878
259’792	6’000’212.18	0.4277
518’593	5’851’927.73	0.8753
1’036’612	5’767’906.42	1.7751
2’073’621	5’696’822.19	3.5953

Tabelle 6.4: Mittelwerte der Ergebnisse des Bandbreitentests auf einer 11 Mbit W-LAN Leitung ausserhalb der Arbeitszeit

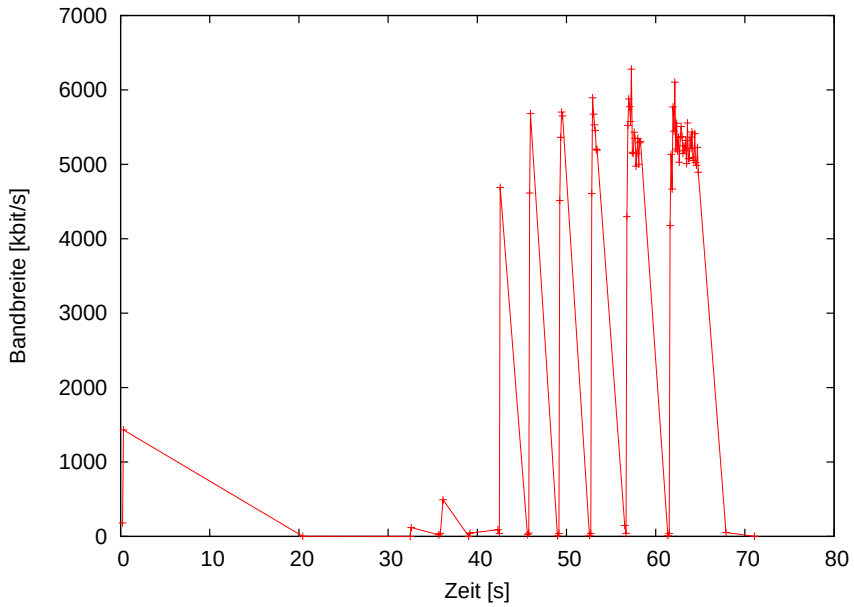


Abbildung 6.5: W-LAN Messung in 100ms Auflösung

sionen und die entsprechende Reaktion von TCP zurückzuführen. Bei einem Vergleich der beiden Messungen bemerkt man auch, dass bei einer grösseren vorhandenen Bandbreite mehr Übertragungen notwendig sind, um aussagekräftige Resultate zu erhalten. Die beiden Messungen verdeutlichen, dass bei einem geteilten Medium wie Wireless LAN die Messung der Bandbreite nur eine Momentaufnahme darstellt und dass sich die tatsächlich verfügbare Bandbreite in Abhängigkeit der Benutzeranzahl verändern kann. Bei einer genügend grossen Benutzerzahl wird dieser Effekt, dass die verfügbare Bandbreite sich in diesem Mass verändert, jedoch geringer.

Zur Überprüfung der vom Bandbreitentester gemessenen Bandbreite wurde wiederum mit `tcpdump` der Verkehr aufgezeichnet. Bei Wireless LAN wird laut dieser Aufzeichnungen die theoretisch mögliche Höchstbandbreite von 11 Mbit/s nicht erreicht. Dies kann man in Abbildung 6.7 erkennen, wo der Netzverkehr mit der geringsten Auflösung von 1 ms bei freiem Medium dargestellt ist. Spitzenwerte erreichen nur eine Bandbreite von 7 Mbit/s. Mittelt man den Verkehr über 10 ms (siehe Abbildung 6.6), kann man ein Absinken der höchsten erreichten Bandbreite feststellen. Bei einer Auflösung von 100 ms (siehe Abbildung 6.5) erkennt man eine durchschnittliche Bandbreite zwischen 5 Mbit/s und 6 Mbit/s, was genau den vom Bandbreitentester berechneten Daten entspricht. Es kann somit wieder festgehalten werden, dass der Bandbreitentester dieselben Resultate berechnet, die sich aus der direkten Beobachtung des gesamten Netzverkehrs ergeben.

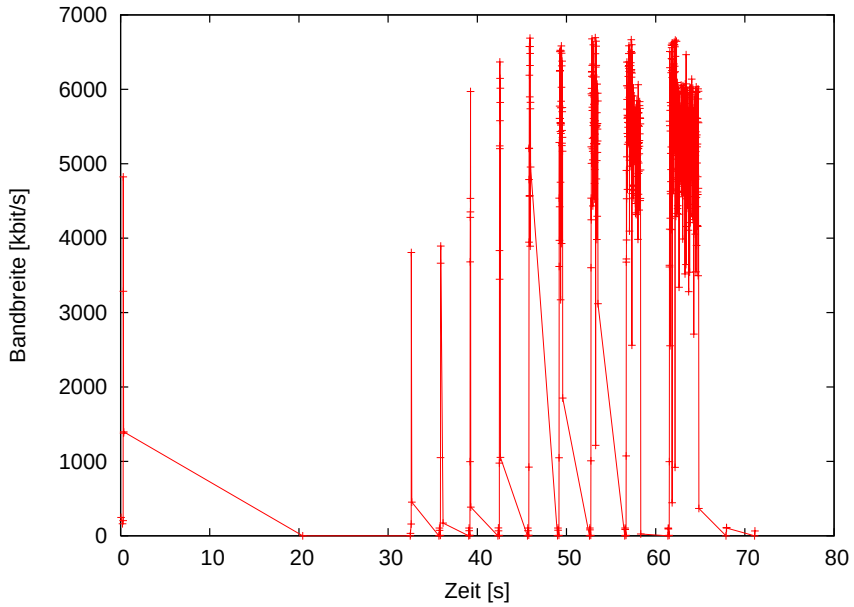


Abbildung 6.6: W-LAN Messung in 10ms Auflösung

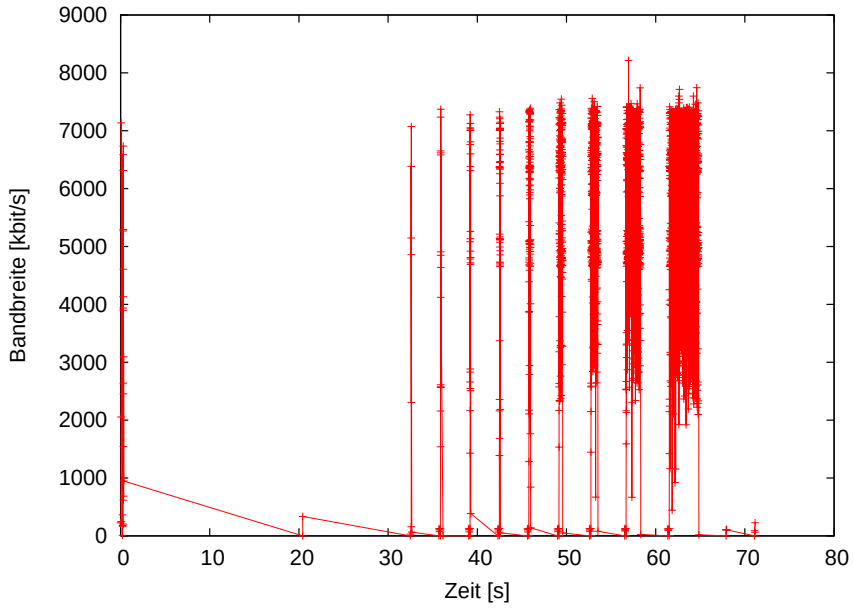


Abbildung 6.7: W-LAN Messung in 1ms Auflösung

6.2.3 150 kbit/s ADSL

Das Universitätsnetz erlaubt aus Sicherheitsgründen keine Anfragen von ausserhalb auf beliebige Portnummern. Anfragen, die als Zielport keinen offiziell zugelassenen Server (z.B. Webserver mit Port 80) oder Dienst (z.B. ssh auf Port 23) angeben, werden nicht weitergeleitet. Aus diesem Grund musste man den HTTP-Server mit der Zeitmessung ausserhalb des Universitätnetzes aufstellen und die Messungen auf dem *Up-Link* der ADSL-Leitung durchführen. Der *Up-Link* ist in der Regel mit weniger Bandbreite versehen als der *Down-Link*, da angenommen wird, dass der Benutzer mehr Daten vom Internet herunterlädt als dass er Daten in das Internet sendet.

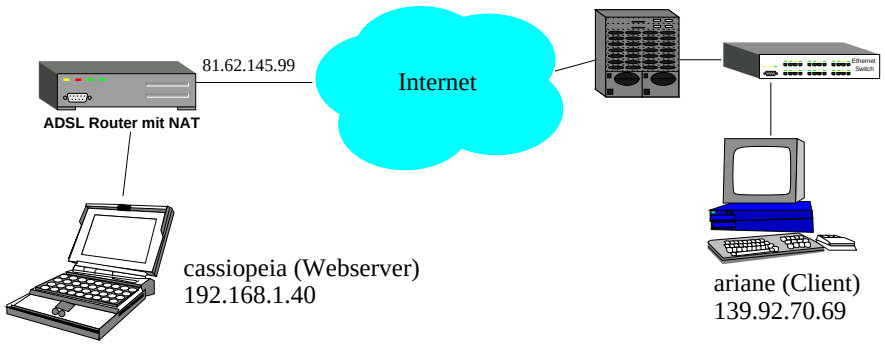


Abbildung 6.8: Architektur der Testumgebung für ADSL

Die folgenden Messungen wurden auf einer Leitung gemacht, auf der 150 kbit/s Bandbreite als Up-Link zur Verfügung stehen (siehe Abbildung 6.8). Erste Messungen über eine kurze Zeitdauer von maximal zwei Sekunden, welche sich bei den vorangegangenen Messungen als hinreichend herausgestellt hat, ergaben sehr unbefriedigende Messresultate: unwahrscheinliche Bandbreiten, die über der vom Provider angegebenen Leitungskapazität von 150 kbit/s lagen. Aus diesem Grund wurde die Mindest-Messzeit zu Testzwecken auf 100 Sekunden erhöht, die Wartezeit zwischen der Übertragung zweier Bilder wurde weggelassen, so dass nach Beendigung einer Übertragung gleich das nächste Bild angefordert wurde. Die vom Webserver gemessenen Resultate sind in Tabelle 6.5 aufgelistet, der Gesamtnetzverkehr wird in den Abbildungen 6.9 und 6.10 dargestellt.

In den Resultaten der Messungen finden sich die beiden zuvor gemachten Beobachtungen wieder: Es braucht einerseits eine gewisse Übertragungszeit, andererseits beeinflusst der Socketpuffer von 65’000 Bytes die Genauigkeit der Messungen. Hier spielt die Kernelauflösungszeit nur eine zweitrangige Rolle, da nur das kleinste Bild eine geringere Übertragungszeit benötigte, was sich in der unrealistischen Bandbreite von 241 Mbit/s wiederschlägt (siehe Tabelle 6.5). Anders sieht es beim Puffer aus: Die

Dateigrösse in Bytes	Mittelwert in bits/s	Sendedauer in s
8’753	241’541’657	0.0002
16’856	306’045	0.44
32’963	205’849	1.28
65’252	256’500	2.03
130’242	118’027	6.15
259’792	169’090	12.29
518’593	127’144	32.63
1’036’612	117’424	70.61
2’073’621	111’466	148.82

Tabelle 6.5: Mittelwerte der Ergebnisse des Bandbreitentests auf einer 150 kbit ADSL-Leitung

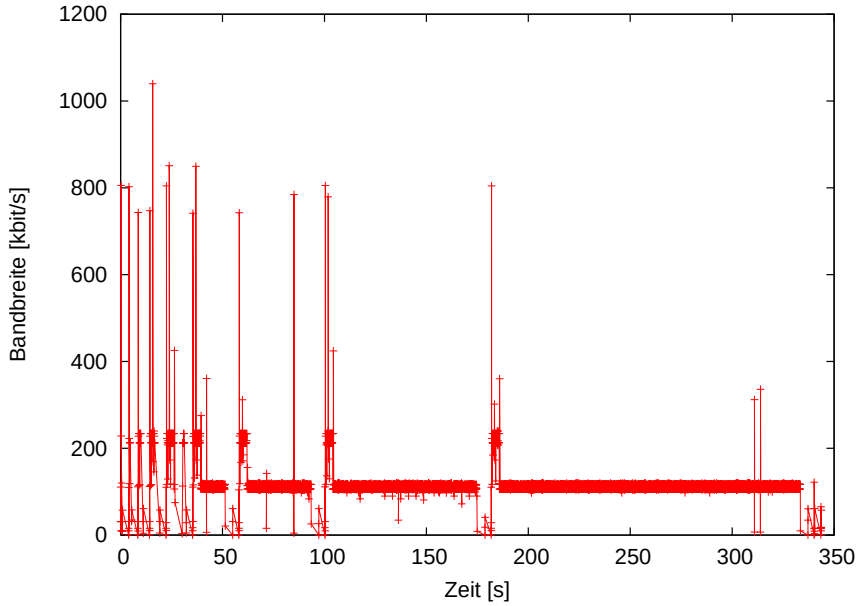


Abbildung 6.9: Auswertung von `tcpdump` während der Bandbreitentests auf einer ADSL Leitung, Auflösung 10 ms

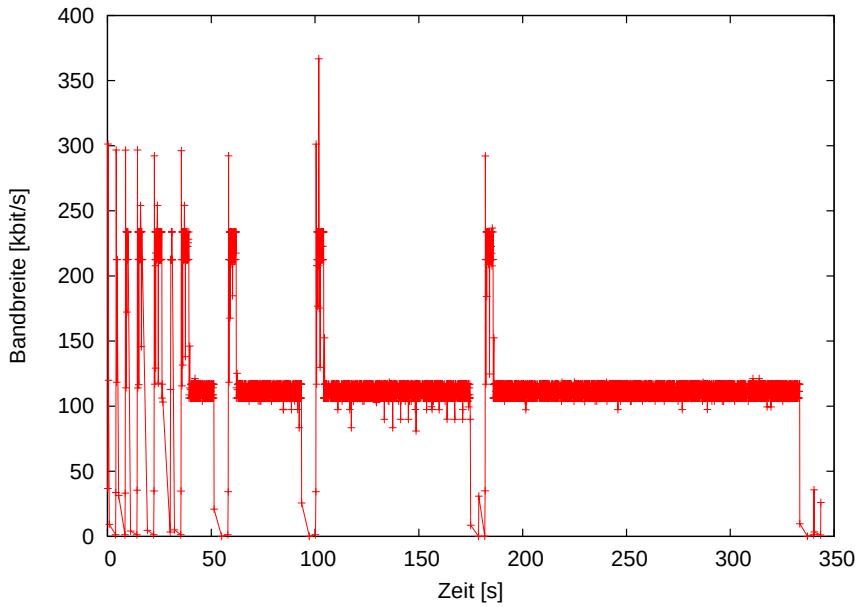


Abbildung 6.10: Auswertung von `tcpdump` während der Bandbreitentests auf ADSL, Auflösung 100ms

grosse Puffergrösse einerseits und das grosse Delay von ADSL andererseits führen beim Übertragen von kleineren Bildgrössen zu ungenauen Resultaten. Betrachtet man den Netzverkehr mit `tcpdump` (siehe Abbildung 6.10), kommt man auf ähnliche Resultate wie der Bandbreitentester. In beiden Abbildungen (6.9 und 6.10) erkennt man (abgesehen von den hohen Werten zu Beginn einer jeden Übertragung) eine durchschnittliche Bandbreite zwischen 100 kbit/s und 120 kbit/s. Aufgrund der unterschiedlich langen Übertragungszeiten wurden die Bilder nicht in regelmässigen Abständen gesendet, der Sendebeginn jedes Bildes ist jedoch gut ersichtlich. Die teilweise hohen Übertragungsraten über kürzere Dauer müssen deshalb auf eine ADSL-interne Regulierung der Übertragungsrate zurückgeführt werden. Dieser Mechanismus wird erst bei einer Messung mit grossen Übertragungsmengen umgangen, erst ab einer Bildgrösse von über einem Megabyte ergibt der Bandbreitentest als Resultat einen der effektiv erreichten Durchschnittsbandbreite entsprechenden Wert. Diese Ergebnisse stimmen mit den Beobachtungen des Netzverkehrs überein.

6.2.4 Modemleitung

In den vorherigen Kapiteln wurde festgestellt, wie wichtig die Mindestsendedauer ist. Aus diesem Grund wurde bei den Messungen über das Modem die Messzeit auf der bei ADSL auf hundert Sekunden gesetzten Schranke gelassen. Trotzdem wurden die grossen Bilder nicht benötigt, die erforderliche Genauigkeit wurde schon nach dem

vierten Bild (und einer Zeit von 16 Sekunden) erreicht. Die Messungen wurden nur über Daten bis 518 kBytes durchgeführt (die ersten sieben Bilder), da grössere Bilder keine substantielle Verbesserung der Genauigkeit mehr bringen und eine Messung über die Telephonleitung pro Bild fast eine Viertelstunde gedauert hätte. Der Aufbau der Messumgebung ist in Abbildung 6.11 dargestellt.

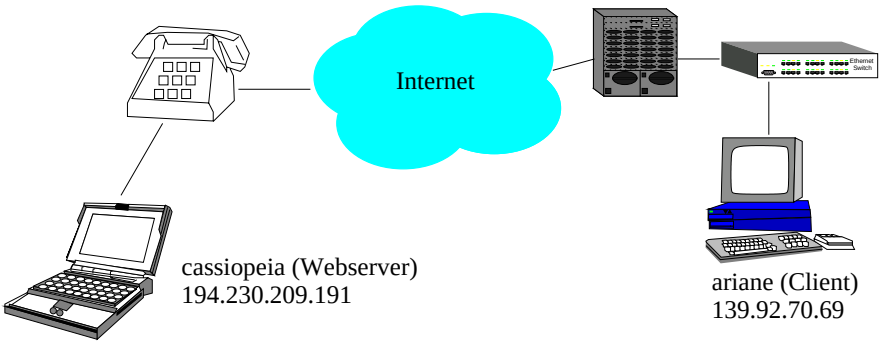


Abbildung 6.11: Architektur der Testumgebung bei Messungen über das Modem

Dateigrösse in Bytes	Mittelwert in bits/s	Sendedauer in s
8’753	235’001’718	0.00029
16’856	83’795	1.60
32’963	80’254	3.49
65’252	34’307	16.41
130’242	32’724	31.84
259’792	26’122	79.59
518’593	26’532	156.35

Tabelle 6.6: Mittelwerte der Ergebnisse des Bandbreitentests mit einem 56kbit/s Modem

Die Resultate sind in Tabelle 6.6 und in den Abbildungen 6.12 und 6.13 dargestellt. Wie bei den vorher gemachten Messungen ist die Übertragungszeit des kleinsten Bildes zu gering, um aussagekräftig zu sein. Bei den Bildern unter der Puffergrösse von 65’000 Bytes kann man auch wieder den Einfluss des Puffers erkennen, die Pufferung der gesamten zu übertragenden Daten führt zu einer zu grossen Bandbreite. Überschreitet die Datenmenge die Puffergrösse, werden Übertragungsraten berechnet, die wiederum mit den bei `tcpdump` gemachten Beobachtungen übereinstimmen. Interessant sind hier die Schwankungen der Bandbreite, welche in den Messbeobachtungen von `tcpdump` erscheinen. Diese sind nur durch eine zwischenzeitliche Pufferung der Daten zu erklären.

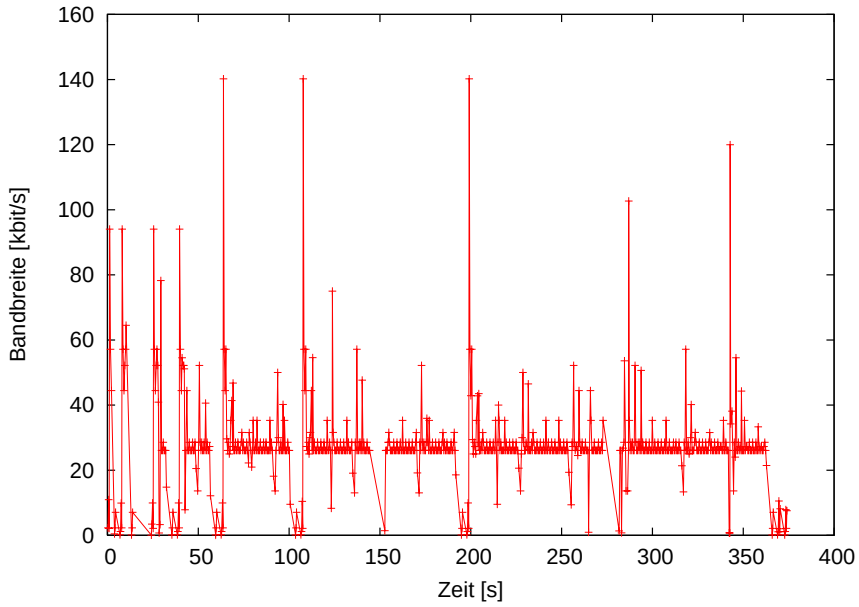


Abbildung 6.12: Auswertung von `tcpdump` während der Bandbreitentests über ein Modem, Auflösung 100 ms

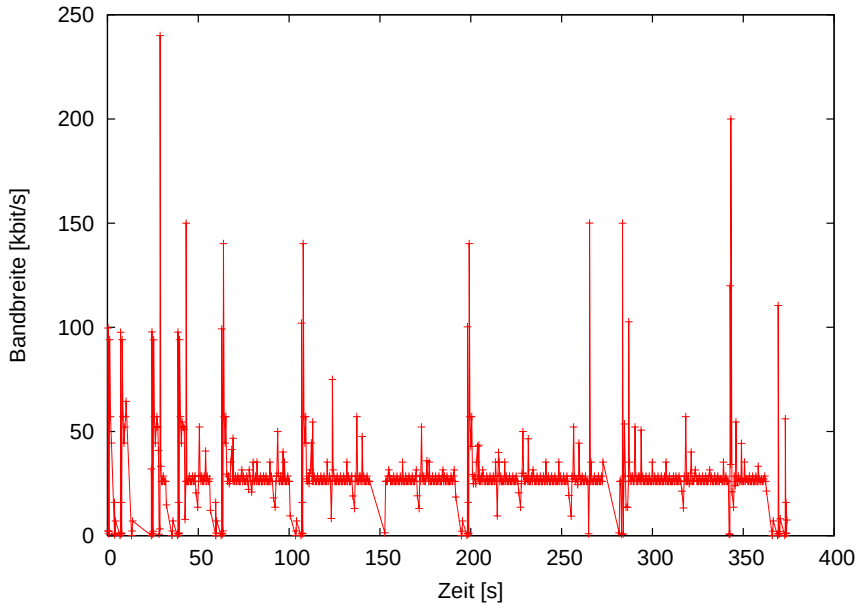


Abbildung 6.13: Auswertung von `tcpdump` während der Bandbreitentests über ein Modem, Auflösung 10 ms

6.2.5 Zusammenfassung der Bandbreitentests

Messungen auf allen Leitungen haben zu einer wichtigen Beobachtung geführt: Es erfordert eine bestimmte Mindestsendezeit und eine Mindestdatenmenge, um ein zuverlässiges Messresultat zu erhalten. Diese liegt bei geringen Bandbreite über einem Mehrfachen der Kernelauflösezeit, da die Datenpufferung des Socket-Puffers mit einwirkt. Bei grossen Bandbreiten genügt es, lediglich die Kernelauflösung zu berücksichtigen. Dies genügt, weil in einer Übertragungszeit, die grösser ist als die Kernelauflösung, bereits Paketgrössen benötigt werden, welche die normal eingestellte Puffergrösse überschreiten. Die Einstellung der Zeitschranke hängt somit direkt vom zu messenden Medium ab. Aufgrund der gemessenen Resultate wurden die beiden kleinsten Bilder (8’753 Bytes und 6’856 Bytes) aus dem Bandbreitentester entfernt, die Messung beginnt nun bei einer Bildgrösse von 32’963 Bytes. Bei Leitungen mit geringer Bandbreite führt die lange Übertragung bei Messungen leider zu einer gewissen Belastung des Netzwerkes und einer nicht zu vermeidenden Wartezeit für den Benutzer.

Die berechneten Resultate des Servers stimmen sehr gut mit den Ergebnissen überein, die man durch direkte Beobachtung des gesamten Netzverkehrs mit Hilfe von `tcpdump` erhält, wie man bei allen Messungen in den vorangegangenen Kapitel beobachten konnte. Dies lässt auf eine sehr zuverlässige Funktionsweise des Bandbreitentesters schliessen.

6.3 Bandbreitenbeschränkung

Um die Genauigkeit der Anpassung an eine beschränkte Übertragungsrate zu testen, wurde der Server `httpserv.pl` auf cassiopeia jeweils fest auf die zu sendende Bandbreite eingestellt. Danach wurden Dateien von einem externen Webserver (asterix.unibe.ch, anderes IP-C-Netzwerk) über den eigenen HTTP-Server (cassiopeia.cnds .unibe.ch) angefordert und an den Client (atlas.cnds.unibe.ch) mit der entsprechenden beschränkten Bandbreite weitergereicht (siehe Abbildung 6.14). Die Messung wurde mit Hilfe des Zusatzprogrammes *webget_client.pl* beim Client durchgeführt. Wie auch bei der Bandbreitenbestimmung wurde die Gesamtheit der Daten und die zur Übertragung benötigte Zeit gemessen. Durch Division erhielt man dann die effektive Bandbreite.

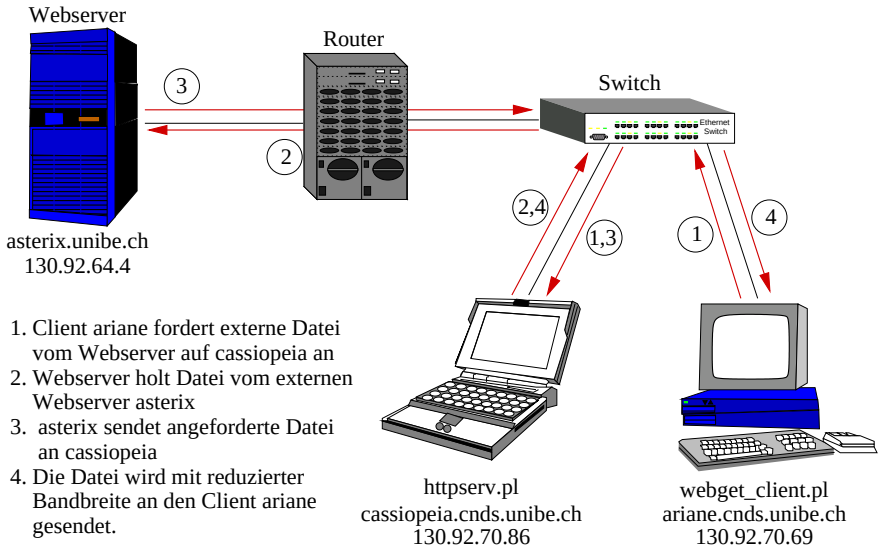


Abbildung 6.14: Architektur der Testumgebung mit Ablauf

Für die Messungen wurden bis zu acht verschiedene Dateien angefordert. Die Grösse der kleinsten Datei betrug 8794 Bytes, bei den folgenden Dateien wurde die Grösse jeweils in etwa verdoppelt. Um dabei aussagekräftige Werte zu erhalten, wurde für jede Bandbreite und jede Dateigrösse zwanzig Mal gemessen und der jeweilige Mittelwert berechnet. Bei den ersten Messungen wurde die Bandbreite auf die relativ kleinen Werte 64 kbit/s, 256 kbit/s und 512 kbit/s eingeschränkt. Die gemessenen Übertragungsraten sind in der Tabelle 6.7 aufgelistet und in der Abbildung 6.15 graphisch dargestellt, wobei die x-Achse die Dateigrösse logarithmisch in Bytes und die y-Achse die Übertragungsrate in kbit/s darstellt.

Schon bei der kleinsten Datei wurde bei einer Einstellung der Bandbreite auf 64 kbit/s eine Annäherung an den eingestellten Wert von über 96% erreicht. Je grösser die Bandbreite allerdings eingestellt wurde, desto grösser musste die übertragene Datei sein, um

Dateigrösse in Bytes	64 kbit/s	256kbit/s	512 kbit/s
8’794	61’903	226’543	403’581
16’897	62’512	238’147	453’480
33’004	63’427	247’206	480’897
65’293	63’712	251’250	493’084
130’283	63’777	253’121	495’947
259’833	63’893	254’029	504’775
518’634	63’924	254’491	506’042
1’036’653	63’964	254’952	507’693

Tabelle 6.7: Durchschnittlich erreichte Übertragungsrate bei eingestellten Sendebandbreiten von 64 kbit/s, 256 kbit/s und 512 kbit/s

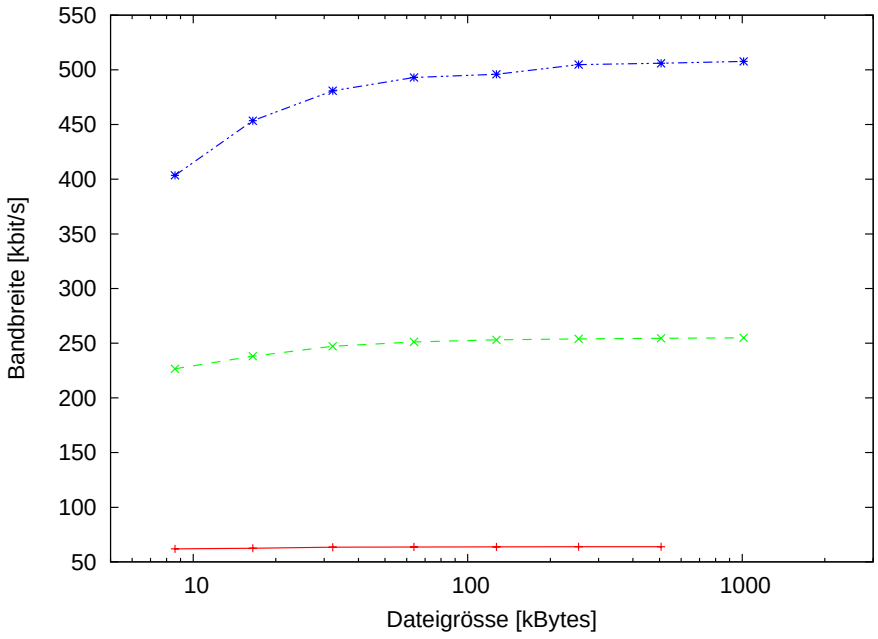


Abbildung 6.15: Anpassung an die eingestellte Bandbreite mit wachsender Grösse der übertragenen Datei (Sendebandbreite: 64 kbit/s, 256 kbit/s und 512 kbit/s)

eine Annäherung zu erreichen. Bei 256 kbit/s wurde bei der ersten Messung nur ein Wert von 88% erreicht, bei einer Bandbreite von 512 kbit/s wurden sogar nur 79% erreicht. Je länger die Übertragung jedoch dauert, desto genauer stimmt die eingestellte Bandbreite mit der tatsächlich erreichten überein, was in dem Graphen in Abbildung 6.15 besonders gut sichtbar wird.

Die Messungen bei grösseren Bandbreiten ergaben ähnliche Resultate, wobei die Annäherung an die gewünschte Bandbreite erst mit grösseren Dateien d.h. mit einer längeren Sendedauer erreicht wird, wie aus Tabelle 6.8 ersichtlich wird. Die effektive Sendedauer vergrössert sich jedoch nicht, eine Annäherung von über 95 % an die eingestellte Bandbreie wurde bei allen Messungen nach mindestens drei Sekunden erreicht, wobei es je länger dauerte je grösser die eingestelle Bandbreite war.

Dateigrösse in Bytes	1 Mbit/s	2 Mbit/s	5.5 Mbit/s
8’794	0.654784	0.999881	1.475534
16’897	0.800622	1.252066	2.289807
33’004	0.883377	1.546027	2.956199
65’293	0.925114	1.708270	3.791384
130’283	0.958103	1.835428	4.407880
259’833	0.972007	1.894451	4.749657
518’634	0.980432	1.922652	4.953940
1’036’653	0.983988	1.918754	5.051624
2’073’662	0.986442	1.940153	5.107641

Tabelle 6.8: Durchschnittlich erreichte Übertragungsrate bei eingestellten Sendebandbreiten von 1 Mbit/s, 2 Mbit/s und 5.3 Mbit/s

Schuld an dieser langsamen Anpassung ist unter anderem der in Kapitel 2.1 beschriebene *slow-start* von TCP. Bei grosser Bandbreite und kleinen zu übertragenden Dateien wird die volle Übertragungsrate gar nicht erreicht. Diese Problematik wird auch in [28] behandelt, wo unter anderem beschrieben wird, dass TCP für eine Erholung von einem Paketverlust auf einer 100 Mbit Leitung (und einer Round-Trip-Time von 0.2 sec) 2 min 50 sec benötigt. Da diese Messungen über einen sehr viel kleineren Zeitraum gemacht wurden, kann man im Graphen der Abbildung 6.16 die Charakteristik des Slow-Starts gut beobachten. Ein anderer Grund für die erst bei grösseren Übertragungsmengen erzielte Genauigkeit sind rechnerinterne Ungenauigkeiten, welche bei kurzen Übertragungszeiten zu Messungenauigkeiten führen. Wie in Kapitel 6.1 bereits gezeigt, kann zum Beispiel die Verzögerung nur mit einer Genauigkeit von 10 ms berechnet werden. Daraus summieren sich Fehler. Aufgrund der in Kapitel 5.5 beschriebenen Implementation ergaben die Messungen sehr gleichmässige Resultate, was die Berechnung der Standardabweichung (siehe Tabelle 6.9) zeigt. Als Illustration sind die Standardabweichungen der grössten und geringsten Beschränkung (64 kbit/s und 5.5 Mbit/s) angegeben. Aus Gründen der Übersichtlichkeit ist die Standardabwei-

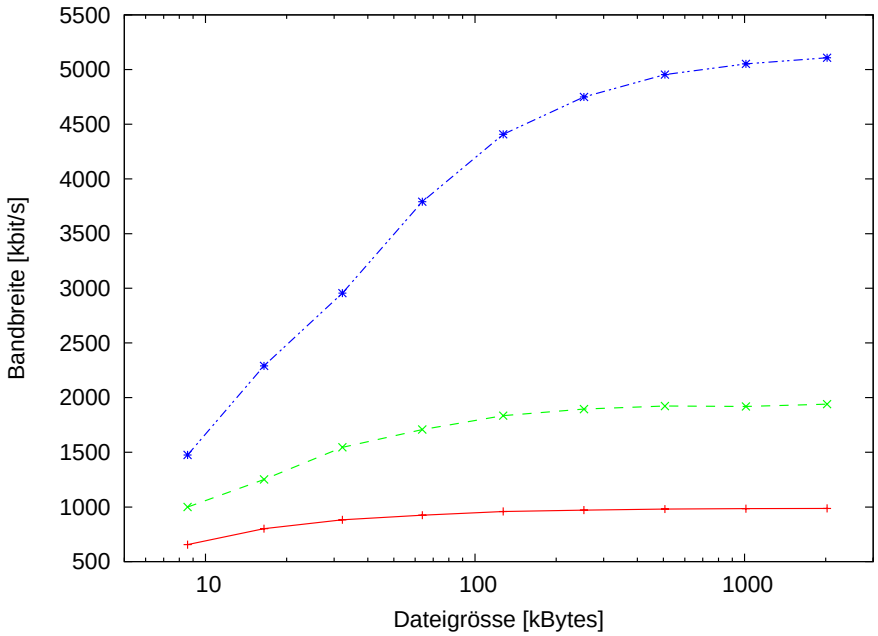


Abbildung 6.16: Anpassung an die eingestellte Bandbreite mit wachsender Grösse der übertragenen Datei: 1 Mbit/s, 2 Mbit/s, 5.5 Mbit/s

chung in % angegeben. Auch hier bemerkt man bei der Messung mit der kleinsten Bandbreite eine kontinuierliche Verbesserung bei steigender Übertragunsdauer, Extremwerte kommen praktisch keine vor. Bei einer Bandbreite von 5.5 Mbit/s werden bereits höhere Abweichungen gemessen, was auf die bereits zuvor diskutierten rechnerinternen Rundungsprobleme zurückgeführt werden kann.

Dateigrösse in Bytes	64 kbit/s	5.5 Mbit/s
8’794	1.087	0.688
16’897	1.481	1.429
33’004	0.219	0.730
65’293	0.002	0.526
130’283	0.192	0.290
259’833	0.013	0.064
518’634	0.001	0.164

Tabelle 6.9: Standardabweichung von den eingestellten Sendebandbreiten in %:

6.4 Vergleich zur unbegrenzten Übertragung

In diesem Kapitel wurde der Effekt der Beschränkung auf die vom Bandbreitentester berechnete Bandbreite untersucht. Dabei wurde vor allem die Stabilität der Verbindung und die Übertragungsrate beobachtet.

Auf unbelasteten Links mit hoher Bandbreite (100Mbit/s) ergab eine Einschränkung der Sendebandbreite keine Verbesserung. Auf Leitungen mit einer hohen Kapazität macht deshalb eine Begrenzung nur dann Sinn, wenn eine Kostenerhebung oder eine einfache Art von Quality of Service erwünscht ist.

Um zu vergleichen, ob eine Bandbreitenbeschränkung auf belasteten Links zu einem Vorteil führt, wurden daher nur Messungen auf einer Modemleitung und auf einer ADSL-Leitung durchgeführt.

6.4.1 Modemleitung

Um auf der Modemleitung zu messen wurde eine Datei der Grösse 130'242 Bytes mehrfach mit und ohne Beschränkung der Sendebandbreite übertragen. Die Grösse der Datei wurde aufgrund der Bandbreitentests gewählt, ab diesem Wert ergab der Bandbreitentest verlässliche Resultate, welche weder von zu geringer Übertragungsdauer noch von der Puffergrösse beeinflusst werden. Die Messumgebung war dabei die selbe wie die in Kapitel 6.2.4 vorgestellte. Um die Beschränkung zu bestimmen wurde zuerst der Bandbreitentest durchgeführt. Dabei wurde eine zur Verfügung stehende Bandbreite von (gerundet) 25000 bit/s gemessen. Nun wurde die Bandbreite auf diesen Wert beschränkt. Die Zeit wurde auf die selbe Art gemessen wie bei der Bestimmung der Bandbreite. Mit Zeit und Dateigrösse wurde die effektiv erreichte Übertragungsrate berechnet. Für diese Messungen wurde ein Hilfsskript (*web_client.pl*) benutzt, welches mit einem HTTP-Request eine gewünschte Datei anfordert. Der Stichprobenumfang betrug jeweils 20 beschränkte und unbeschränkte Messungen. Die gemessenen und berechneten Resultate sind in Tabelle 6.10 dargestellt.

	Beschränkt	Unbeschränkt
Durchschnitt	23'951.79	22'860.09
Höchster Wert	24'037.83	24'712.66
Tiefster Wert	23'912.44	19'560.50
Standardabweichung in %	0.307	8.45
Durchschnitt 2	23'959.66	22'936.252
Standardabweichung 2 in %	0.218	7.98
Median	23'954.37	23'423.64

Tabelle 6.10: Vergleich der gemessenen Bandbreite über eine Modemleitung

Die Messresultate ergaben, dass für die Übertragung ohne Limite ein grösserer Maximalwert erreicht wurde als für die limitierte Übertragung. Im Durchschnitt allerdings erreichte man bei der beschränkten Übertragung einen um 4.7% höheren Wert und vor allem eine viel kleinere Schwankung. Besonders deutlich zeigt dies die berechnete Standardabweichung der gemessenen Bandbreite: bei der beschränkten Übertragung liegt sie bei 0.3 %, bei der unbeschränkten Übertragung bei dem 28-fach höheren Wert von 8.45 %. Um mögliche Messfehler zu eliminieren, wurden Durchschnitt und Standardabweichung noch einmal berechnet (in der Tabelle jeweils mit Durchschnitt 2 und Standardabweichung 2 angegeben), nachdem die höchsten und tiefsten Messwerte entfernt wurden. Dies hat allerdings nur wenig am Resultat geändert, die limitierte Übertragung erreichte noch immer bessere Durchschnittswerte. Zum gleichen Ergebnis kommt man bei der Betrachtung eines weiteren statistischen Kennwertes, des Medians: Auch hier liegt der Wert der beschränkten Übertragung über dem der normalen, unlimitierten Übertragung, wenn auch etwas weniger deutlich.

Die Betrachtung der Verteilung der Messresultate der unbeschränkten Bandbreite lässt darauf schliessen, dass die Modemleitung grossen Schwankungen unterliegt. Dies kann man aus der grossen Standardabweichung und dem über dem Durchschnitt liegenden Median entnehmen. Um solche Schwankungen besser in das Resultat einbeziehen zu können, wurde eine erneute Messung unter gleichen Bedingungen mit doppeltem Stichprobenumfang gemacht. Aus den Resultaten ersieht man, dass auch die Übertragung mit beschränkter Bandbreite diesen Schwankungen unterliegt, allerdings weniger stark. Auch hier wurden je die beiden höchsten und tiefsten Messungen zur Berechnung des zweiten Durchschnitts weggelassen. Die Zusammenfassung der zweiten Messreihe ist in der Tabelle 6.11 dargestellt.

	Beschränkt	Unbeschränkt
Durchschnitt	23’470.91	23’344.46
Höchster Wert	24’094.26	24’724.39
Tiefster Wert	20’197.03	20’465.70
Standardabweichung in %	6.04	8.40
Durchschnitt 2	23’703.58	23’531.81
Standardabweichung 2 in %	4.66	7.90
Median	24’045.79	24’474.34

Tabelle 6.11: Vergleich der gemessenen Bandbreite über eine Modemleitung, grösserer Stichprobenraum

Die Schwankung der Leitung fällt für den Durchschnittswert bei der beschränkten Übertragung auch bei grösserem Stichprobenumfang weniger ins Gewicht, die Standardabweichung ohne Extremwerte ist jedoch noch immer um einen Faktor 1.7 kleiner als bei unbeschränkter Senderate. Obwohl der Mittelwert der beschränkten Übertragung auch hier über dem der unbeschränkten Übertragung liegt, kann man doch aus

dem Median ersehen, dass die Mehrzahl beider Übertragungen über dem Mittelwert liegen, da dieser noch immer durch Extremwerte beeinflusst wird.

6.4.2 150 kbit ADSL

Die selben Messungen wie zuvor bei der Modemleitung wurden bei ADSL mit einer Datei der Grösse 518’593 Bytes durchgeführt. Wiederum wurde vor der Messung mit dem Bandbreitentester eine Messung durchgeführt, bei der eine (aufgerundete) Übertragungsrate von 105’000 kbit/s erreicht wurde. Zusätzlich zu dieser Beschränkung wurde noch getestet, wie sich eine Beschränkung auf die vom Provider angegebene höchste Bandbreite auswirkt. Die Resultate sind in Tabelle 6.12 zusammengefasst.

Beschränkung:	105kbit/s	150kbit/s	Unbeschränkt
Durchschnitt	102’926.14	103’756.76	103’349.14
Höchster Wert	103’376.67	104’955.22	105’029.03
Tiefster Wert	102’301.35	101’309.06	101’504.55
Standardabweichung in %	0.46	1.23	1.13
Durchschnitt 2	102’984.23	103’964.97	103’369.73
Standardabweichung 2 in %	0.38	1.05	0.91
Median	103’198.32	104’766.34	103’313.54

Tabelle 6.12: Vergleich der gemessenen Bandbreite über eine 150Kbit ADSL-Leitung

Bei diesen Messungen liegt die durchschnittliche Übertragungsrate der ersten Beschränkung unter der gemessenen Bandbreite der unbeschränkten Übertragung. Die Schwankung allerdings ist wiederum kleiner, wenn auch die Standardabweichung bei der unbeschränkten Bandbreite nur noch gut doppelt so gross ist (im Vergleich: Bei den ersten Resultaten der Modemleitung war die Standardabweichung der ungebremsten Senderate 28 mal grösser). Ein Versuch von Messungen mit einer Beschränkung auf die vom Provider angegebenen Höchstrate von 150 kbit/s hat erstaunlicherweise ergeben, dass Durchschnitt wie auch Median deutlich über den berechneten Werten der unbeschränkten Übertragung liegen – dafür sind die Standardabweichungen vergleichbar.

6.4.3 Zusammenfassung der Vergleichs-Versuchsreihen

Eine Kombination von einem Bandbreitentest gefolgt von einer entsprechenden Beschränkung kann als sinnvoll angesehen werden, wenn man als Ziel eine stabilere Verbindung hat, die weniger Schwankungen in der Bandbreite unterliegt. Eine Vergrösserung der durchschnittlichen effektiv erreichten Bandbreite kann nicht garantiert werden. Auch wenn Messungen bei einer Modemleitung einen um 1.7 % bis 4.7 %

höheren Durchschnittswert ergeben, liegt der erreichte Durchschnittswert bei ADSL bei einer Beschränkung auf 105 kbit/s 1 % unter dem Wert der unbeschränkten Übertragung. Wenn man jedoch die Schwankung der erreichten Bandbreite betrachtet, ergaben alle Messungen, dass die begrenzte Übertragung stabiler ist und kleineren Schwankungen unterliegt.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein HTTP-Server mit Zusatzfunktionen vorgestellt. Dieser ermöglicht es, die Bandbreite vom Server zum Benutzer bestimmen zu lassen und entsprechend die Senderate des Servers dynamisch anzupassen. Diese beiden Bestandteile sind nicht in einem normalen Webserver zu finden und können auch nicht mit externen Skripts alleine realisiert werden. Weiter ist mit Hilfe einer Datenbank eine Benutzerverwaltung und ein Cache im Webserver integriert. Diese Funktionalitäten erlauben es, den Webserver auch als Proxy-Server zu benutzen.

7.1 Bandbreitenbestimmung

Im Webserver integriert ist eine Funktion zum Messen der Bandbreite vom Webserver zum Benutzer. Dabei muss der Benutzer ausser einem normalen Webbrowser nichts zur Verfügung stellen. Gemessen werden auf Applikationsebene versendete Nutzdaten, das Resultat der Messung gibt also an, wieviel Bandbreite zur Übertragung von Nutzdaten zur Verfügung steht, nicht wie sonst die maximale Kapazitätsgrenze einer Leitung. Messungen haben ergeben, dass die Bestimmung der Bandbreite sehr gut mit der effektiv erreichten Übertragungsrate übereinstimmt. Dabei wurde der gesamte Netzverkehr während der Messungen beobachtet und mit den Messergebnissen verglichen. Es wurde festgestellt, dass ein zuverlässiges Ergebnis nur erhalten wurde, wenn die Messdauer nicht zu kurz war. Die Mindestmessdauer ist abhängig von der zur Verfügung stehenden Bandbreite und muss mindestens ein Vielfaches der Kernelauflösezeit betragen.

7.2 Bandbreitenbeschränkung

Auch im Webserver integriert ist eine Funktion zur Bandbreitenbeschränkung. Die Sendebandbreite kann aufgrund der zuvor gemachten Messung oder aufgrund von Benutzereinstellungen (eingetragen vom Provider oder Benutzer) beschränkt werden. Auch hier haben Messungen ergeben, dass die Beschränkung der Bandbreite ab einer gewissen Sendedauer einwandfrei funktioniert, die Abweichungen der erreichten Übertragungsrate von der tatsächlich eingestellten bewegen sich im unteren Promillbereich. Bei hohen Bandbreiten brauchte es eine gewisse Zeit, bis die Höchstbandbreite erreicht wurde, was allerdings nicht an dem Bandbreitenbeschränker sondern am *Slow Start* Mechanismus von TCP liegt.

7.3 Auswirkungen auf die Übertragungsrate

Nach den Messungen zur Bandbreitenbestimmung und Bandbreitenbeschränkung wurde die Auswirkung einer Bandbreitenbeschränkung auf Übertragsraten beobachtet.

Dabei konnte festgestellt werden, dass bei Verbindungen mit geringem Durchsatz und grossen Schwankungen der Bandbreite (z.B. bei einer Modemleitung) durch das Reduzieren der Sendebandbreite auf den zuvor berechneten Wert eine stabilere Übertragungsrate erreicht werden konnte. Eine daraus entstandene Vergrösserung der Bandbreite konnte auf einer Modemleitung in der Grössenordnung von bis zu 5 % beobachtet werden. Die durch die Banbreitenbeschränkung errungene Stabilität der Verbindung ist von Vorteil für den Benutzer, der sich so auf eine konstantere Übertragungsrate verlassen kann. Auch Provider können den Beschränker benutzen, z.B. um den Verkehr auf Links besser vorauszusagen und zu kontrollieren.

7.4 Mögliche Anwendungsgebiete

Die Architektur von Webserver, Bandbreitenbestimmung und Bandbreitenbegrenzung kann in Endpunkten zu drahtlosen Netzen (z.B. W-LAN Hubs oder Basisstationen von Mobilanbietern) als Proxy-Server eingesetzt werden. Dabei können auch nur Teile der Architektur benutzt werden, z.B. der Bandbreitenbeschränker, um eine einfachste Form von Quality of Service anzubieten. Die Benutzung des Bandbreitenbeschränkers ist auch sinnvoll für Benutzer, die über eine Modem-Verbindung das Internet benutzen und eine stabilere Übertragungsrate wünschen. Für manchen Benutzer des Internets mag es auch interessant sein zu wissen, wie gross die Bandbreite ist, die ihm effektiv zum Herunterladen von Nutzdaten zur Verfügung steht.

7.5 Mögliche Erweiterungen

- Die Bandbreite, die dem Benutzer zur Verfügung steht, kann je nach Tageszeit oder Benutzeranzahl auf dem Link stark schwanken (siehe Resultate der Bandbreitenmessung von W-LAN in Kapitel 6.2.2). Somit kann sich die zur Verfügung stehende Bandbreite kurz nach der Messung bereits wieder verändert haben. In dieser Arbeit wird die Messung einmal durchgeführt, entweder auf Wunsch des Benutzers oder beim Registrieren. Wiederholte Messungen werden nur durchgeführt, wenn der Benutzer dies explizit wünscht (d.h. wenn er den Bandbreitentest erneut startet). Eine Möglichkeit zur Fortführung dieser Arbeit wäre nun eine Implementation, die konstant den Datenverkehr zwischen Benutzer und Webserver analysiert und die Bandbreite entsprechend neu berechnet.

- Interessant wäre ein Werkzeug, welches das Verhältnis zwischen den tatsächlich übertragenen Nutzdaten, dem HTTP-Overhead und dem gesamten Verkehr auf einem Link berechnet. Ein Programm wurde im Verlauf dieser Arbeit dazu entwickelt, hat aber bisher nicht zu den gewünschten Ergebnissen geführt und wurde deshalb nicht in die Arbeit eingebunden.

- Der hier implementierte Webserver beinhaltet nicht alle möglichen Funktionen, die ein Apache Webserver zur Verfügung stellt. Da Apache aber als Open Source zur Verfügung steht, könnte man die hier implementierten Bestandteile auf einfache Art und weise in einen Apache Server integrieren
- Durch eine Verbindung mit anderen Techniken (z.B. dem Snoop-Protokoll, welches in [39] vorgestellt wurde) kann der Proxy-Server erweitert werden, so dass auch der TCP-Durchsatz erhöht wird.

Abkürzungen / Glossar

ADSL Asymmetric Digital Subscriber Line: Datenübertragungstechnik zur Hochgeschwindigkeitsübertragung digitaler Informationen über die Kupferkabel des Telefonnetzes

CGI Common Gateway Interface: Schnittstelle zwischen einem Webserver und den dort abgelegten Programmen

CPU Central Processing Unit: Die CPU (dt.: Prozessor) ist die zentrale Rechen- und Steuereinheit eines Computers. Sie besteht aus einem oder mehreren Mikroprozessoren (Chips), die die Befehle der Programme interpretieren und ausführen.

CSMA Zugriffsverfahren, bei dem mehrere Netzstationen zugleich Zugriff auf das Übertragungsmedium (Carrier) haben. CSMA gehört zu Schicht 2 des OSI-Schichtenmodells.

CSMA/CA Carrier Sense Multiple Access/ Collision Avoidance: CSMA-Verfahren mit Kollisionsvermeidung, für die Datenübertragung in Funknetzen, vom IEEE als Standard 802.11 genormt

CSMA/CD Carrier Sense Multiple Access/ Collision Detection: Zugriffsverfahren mit Kollisionserkennund in verkabelten LANs eingesetzt, vom IEEE als Standard 802.3 genormt.

DFÜ Datenfernübertragung: Allgemeine Bezeichnung für die Datenübertragung zwischen Computern mit einem Modem oder einer ISDN-Karte

ECN Explicit Congestion Notification: Bit, welches von Routern gesetzt wird, wenn diese kurz vor einer Überlastung stehen. TCP kann daraufhin die Senderate reduzieren und einer Netzüberlastung vorbeugen.

Ethernet Gebräuchlichste Technik für die Verbindung lokaler Netzwerke, umgangssprachlich für CSMA/CD

FTP File Transfer Protocol: ein Datei-Übertragungsprotokoll des Internets, das auf TCP/IP aufbaut.

GPRS General Packet Radio Service: Eine auf dem Internet-Protokoll basierende Datenübertragungstechnik für GSM-Mobilfunk-Netze, die auf Paketvermittlung basiert

GSM Global System for Mobile Communication: Mobilfunk-Standard für Mobiltelephone

HTML HyperText Markup Language: die Sprache in der viele Seiten des WWW aufgebaut sind. HTML bietet insbesondere die Möglichkeit, verschiedene Dokumente miteinander zu verknüpfen.

HTTP HyperText Transfer Protocol: das meist verwendete Protokoll zum Transfer von HTML Dateien im WWW

IAB Internet Architecture Board: Zusammenschluss von Wissenschaftlern und Programmierern, die sich der technisch-organisatorischen Entwicklung des Internet widmen

IANA Internet Assigned Numbers Authority: Organisation, die die Vergabe von IP-Adressen, Top Level Domains und IP-Protokollnummern regelt

IEEE Institute of Electrical and Electronical Engineers:

IETF The Internet Engineering Task Force: Gremium des Internet Architecture Board zur Umsetzung technischer Standards. Die IETF definiert und beschließt neue Internet- Protokolle und Änderungen der aktuellen Standards

IP Internet Protocol: Das Basis-Protokoll für die Datenübertragung im Internet, das seit 1974 nahezu unverändert in Gebrauch ist.

IPC Interprozesskommunikation

ISO International Organization for Standardization: arbeitet an der weltweiten Vereinheitlichung technischer Standards

ISOC Internet Society: Organisation zur Weiterentwicklung des Internet

JPEG Joint Photographic Experts Group: beliebter Komprimierungsstandard für unbewegte Bilder

LAN Local Area Network: Lokales Kommunikationsnetzwerk innerhalb eines relativ kleinen geographischen Gebiets (ca. 1-2 km)

MIME Multipurpose Internet Mail Extensions: Erweiterung textbasierter E-Mails, Verfahren für den Versand von Binärdateien, Grafiken, Video- und Audiodaten als Attachment, ohne dass die Daten zuvor in das ASCII-Format umgewandelt werden müssen

MySQL Eine relationale SQL-Datenbank, wohl populärste Open-Source-Datenbank der Welt

NAT Network Address Translation: Verfahren, bei dem private IP-Adressen auf öffentliche IP-Adressen abgebildet werden

NTP Network Time Protocol: Standard zur Synchronisierung von Uhren in Computersystemen über paketbasierte Netzwerke

OSI Open Systems Interconnection: eine Arbeitsgruppe der ISO, die zuständig für die Ausarbeitung von Standards und Protokollen für die Datenübertragung in digitalen Netzen ist

PDA Personal Digital Assistant: Tragbare Kleinstrechner zur Verwaltung und Abfrage von Termin- und Adressdaten, häufig auch als Organizer bezeichnet

PEP Performance Enhancing Proxy: Performance enhancing proxies

Perl Practical Extraction and Reporting Language: Eine interpretierte Skriptsprache, die vor allem für die flexible und effiziente Manipulation von textbasierten Dateien geeignet ist. CGIs werden häufig in Perl geschrieben. Perl ist jedoch viel mehr als eine CGI-Spracheünd wird vor allem auf UNIX-basierten Systemen für Aufgaben der Dateien- und Systemverwaltung, aber auch für exotischere Zwecke wie die Entwicklung von mathematischen Werkzeugen eingesetzt.

Proxy Ein Proxy oder Proxyserver (vom engl. proxy = Stellvertreter) ist ein Programm, das zwischen Server und Client vermittelt.

QoS Quality of Service: Priorisierung von Datenpaketen anhand bestimmter Merkmale und Eigenschaften

RFC Request For Comments: Beschreiben sämtliche Internet- Protokolle sowie Standards, Verfahren, Algorithmen, Regeln und Strategien der Kommunikations- und Netzwerktechnik

SQL Structured Query Language: Strukturierte Abfragesprache für Datenbanken mit relationaler Algebra

SSL Secure Socket Layer: Ein offener Standard für die gesicherte Datenübertragung (DFÜ) im Internet

TCP Transmission Control Protocol: Ein zuverlässiges, verbindungsorientiertes Transportprotokoll

TLS Transport Layer Security: TLS ist eine neue Variante von SSL. Das neue Protokoll verspricht noch mehr Sicherheit bei der Kommunikation im Internet.

UDP User Datagram Protocol: Ein unzuverlässiges, verbindungsloses Transportprotokoll

URI Uniforme Recource Identifier: Obermenge für URLs und URNs

URL Uniform Resource Locators: Standard zur Adressierung beliebiger Objekte im Internet

URN Uniform Resource Names: Dauerhafter, ortsunabhängiger Bezeichner für eine Ressource (noch nicht in Betrieb)

VPN Virtual Private Network: In einem VPN werden die öffentlich zugänglichen Leitungen des Internet in einer Weise genutzt, als wären sie Teil eines privaten Leitungsnetzes. Die zu einem VPN gehörenden Internet-Rechner tauschen ihre Daten untereinander nur in verschlüsselter Form aus, so dass diese Rechner gewissermassen ein privates Netz innerhalb des öffentlichen Internet bilden

WAN Wide Area Network: Weitverkehrsnetz, bildet häufig den Backbone zwischen mehreren lokalen Netzen

WAP Wireless Application Protocol: Übertragungsstandard für die Interaktion von kabellosen Endgeräten mit externen Dienstleistungen und Anwendungen

WWW World Wide Web: Interaktives Informationssystem, das den weltweiten Austausch digitaler Dokumente ermöglicht.

Literatur

[1] M. Allman, V. Paxson, and W. Stevens. RFC 2581: TCP Congestion Control, April 1999. Obsoletes RFC2001 [48].

[2] H. Alvestrand. RFC 1766: Tags for the Identification of Languages, March 1995. Obsoleted by RFC3066 [3].

[3] H. Alvestrand. RFC 3066: Tags for the Identification of Languages, June 2001. Obsoletes RFC1766 [2].

[4] H. Alvestrand. RFC 3282: Content Language Headers, May 2002. Obsoletes RFC1766 [2].

[5] T. Berners-Lee, R. Fielding, and H. Frystyk. RFC 1945: Hypertext Transfer Protocol – HTTP/1, May 1996.

[6] T. Berners-Lee, R. Fielding, and L. Masinter. RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, August 1998.

[7] N. Borenstein and N. Freed. RFC 1341: MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies, June 1992. Obsoleted by RFC1521 [8].

[8] N. Borenstein and N. Freed. RFC 1521: MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies, September 1993. Obsoletes RFC1341 [7].

[9] M. Brown. *Perl Annotated Archives*. Osborne / McGraw-Hill, 1999. ISBN 0-07-882557-1.

[10] T. Christiansen and N. Torkington. *Perl Cookbook*. O' Reilly, 1998. ISBN 1-56592-243-3.

[11] D. Connolly and L. Masinter. RFC 2854: The 'text/html' Media Type, June 2000. Obsoletes RFC2070 [56].

[12] B. D. Davison, K. Komaravolu, and B. Wu. A Split Stack Approach to Mobility-Providing Performance-Enhancing Proxies. *Technical Report LU-CSE*, December 2002.

[13] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. RFC 2068: Hypertext Transfer Protocol – HTTP/1, January 1997. Obsoleted by RFC2616 [14].

[14] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext Transfer Protocol – HTTP/1, June 1999.

[15] S. Floyd, S. Rarnasamy, and S. Shenker. Modifying TCP's Congestion Control for High Speeds. *preliminary draft*, 2002. http://www.icir.org/floyd/papers/hstcp.pdf.

[16] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. RFC 2617: HTTP Authentication: Basic and Digest Access Authentication, June 1999.

[17] J. A. Garcia-Marcias, F. Rousseau, G. Berger-Sabbatel, L. Toumi, and A. Duda. Mobility management for providing qos in local area wireless networks. *Proc. First ACM Wireless Mobile Internet Workshop*, 2001.

[18] S. Guelich, S. Gundavaram, and G. Birznieks. *CGI Programmierung mit Perl*. O' Reilly, 2. edition, 2001. ISBN 3-89721-167-X.

[19] A. Gurtov and S. Floyd. Modeling Wireless Links for Transport Protocols. *under submission*, 2004.

[20] T. J. Hacker, B. D. Noble, and B. D. Athey. Improving Throughput and Maintaining Fairness using Parallel TCP. *Infocom*, March 2004.

[21] B. B. Housel, G. Samaras, and D. B. Lindquist. WebExpress: A client/intercept based system for optimizing Web browsing in a wireless environment. *Mobile Networks and Applications*, pages 419–431, 1998.

[22] N. Hunn. The Evolution of the Wireless LAN and the role of the Hot-Spots. *White Paper*, 2002.

[23] Internet Assigned Numbers Authority. http://www.iana.org/.

[24] Wireless Local Area Networks. http://grouper.ieee.org/groups/802/11/, März 2004.

[25] The Internet Engineering Task Force. http://www.ietf.org/.

[26] A. Kamara, V. Misra, and D. Towsley. Achieving High Throughput in Low Multiplexed, High Bandwidth, High Delay Environments. *PFLDnet*, 2003.

[27] D. Katabi, M. Handley, and C. Rohrs. Internet Congestion Control for Future High Bandwidt-Delay Product Environments. *Proceedings of ACM/SIGCOMM*, 2002.

[28] T. Kelly. A scalable TCP with adaptive congestion detection at routers. *Proceedings of IEEE Computer Communications Workshop*, 2002.

[29] T. Kelly. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *PFLDnet*, 2003.

[30] S. Koch. *JavaScript*. dpunkt Verlag, 2. edition, 1999. ISBN 3-932588-26-6.

[31] Linux Programmer's Manual: exec — execute a file, November 1993.

[32] Linux Programmer's Manual: fork — create a child process, Juni 1995.

[33] Linux Programmer's Manual: gettimeofday — get the date and time, Mai 2000.

[34] Linux Programmer's Manual: nanosleep — high resolution sleep, Juni 1998.

[35] Linux Programmer's Manual: semop — semaphore operations, November 1993.

[36] Linux Programmer's Manual: wait, waitpid - wait for process termination, Juli 2000.

[37] M. Margaritidis and G. C. Polyzos. MobiWeb: Enabling Adaptive Continuous Media Application over 3G Wireless Links. *IEEE Network*, 2000.

[38] R. Maurer and O. Paukstadt. *HTML und CGI Programmierung*. dpunkt Verlag, 2. edition, 1998. ISBN 3-920993-79-9.

[39] C. Ng, J. Chow, and L. Trajkovic. Performance Evaluation of TCP over WLAN 802.11 with the Snoop Performance Enhancing Proxy. *Opnetwork*, August 2002.

[40] Wachstum des Seitenangebots für Mobiltelephone in Japan. http://www.nttdocomo.com/companyinfo/subscriber.html, März 2004.

[41] V. Paxson and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, pages 226 – 244, June 1995.

[42] Perl Programmers Reference Guide: perlipc - perl interprocess communication (signals, fifos, pipes, safe subprocesses, sockets, and semaphores), März 1999.

[43] J. Postel. RFC 0793: Transmission Control Protocol, April 1989.

[44] K. Ramakrishnan and S. Floyd. RFC 2481: A Proposal to add Explicit Congestion Notification (ECN) to IP, January 1999. Obsoleted by RFC3168 [45].

[45] K. Ramakrishnan, S. Floyd, and D. Black. RFC 3168: The Addition of Explicit Congestion Notification (ECN) to IP, September 2001. Obsoletes RFC2481 [44].

[46] J. Song and L. Trajkovic. Enhancements and Performance Evaluation of Wireless Local Area Networks. *Opnetwork*, August 2003.

[47] R. Stevens. How can I force a socket to send the data in its buffer? http://www.unixguide.net/network/socketfaq/2.11.shtml.

[48] W. Stevens. RFC 2001: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, January 1997. Obsoleted by RFC2581 [1].

[49] W. R. Stevens. *UNIX Network Programming*, volume 1. Prentice Hall PTR, 2. edition, 1998. ISBN 0-13-490012-X.

[50] K. Thompson, G. J. Miller, and R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics (Extended Version). *IEEE Network*, 1997.

[51] R. Tolksdorf. *Die Sprache des Web: HTML 4*. dpunkt Verlag, 3. edition, 1997. ISBN 3-920993-77-2.

[52] L. Wall, T. Christiansen, and R. L. Schwartz. *Programmieren mit Perl*. O' Reilly, 2. edition, 1997. ISBN 3-930673-48-7.

[53] D. Wegscheid, R. Schertler, J. Hietaniemi, and G. Aas. Perl Library: Time::HiRes - High resolution alarm, sleep, gettimeofday, interval timers. http://www.perldoc.com/perl5.8.0/lib/Time/HiRes.html, März 1999.

[54] E. Wilde. *Wilde's WWW*. Springer Verlag, 1999. ISBN 3-540-64285-4.

[55] R. J. Yarger, G. Reese, and T. King. *MySQL & mSQL*. O' Reilly, 1999. ISBN 1-56592-434-7.

[56] F. Yergeau, G. Nicol, G. Adams, and M. Duerst. RFC 2070: Internationalization of the Hypertext Markup Language, January 1997. Obsoleted by RFC2854 [11].