Martin Richtarsky

# Analyse von Freenet und Implementierung einer Applikation für sicheren, asynchronen Nachrichtenaustausch über Freenet

Studienjahresarbeit Informatik
Fachgebiet Telematik
Betreuer: Dipl.-Inf. Thorsten Strufe

**Richtarsky, Martin:**

# Analyse von Freenet und Implementierung einer Applikation für sicheren, asynchronen Nachrichtenaustausch über Freenet

Studienjahresarbeit Informatik
Fachgebiet Telematik
Betreuer: Dipl.-Inf. Thorsten Strufe



Tag des Druckes: 24. Mai 2003

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

## 1.1 Was ist Freenet?

> "Freenet is a large-scale peer-to-peer network which pools the power of member computers around the world to create a massive virtual information store open to anyone to freely publish or view information of all kinds." [1]

Freenet ist unter anderem:

- ein komplett dezentralisiertes Peer-to-Peer-Netzwerk
- ein adaptives Netzwerk
- eine WWW-Alternative
- eine Plattform für Veröffentlichungen, die Lesern, Autoren und Serverbetreibern Anonymität bietet
- eine sichere Internet-Emulation, die auf dem Internet selbst basiert ("rewiring the net")

## 1.2 Was bisher geschah

Als das Forschungsnetz Internet anfing, populär zu werden, wurde es oftmals als ein anonymer, relativ sicherer Kommunikationskanal missverstanden. Viele Provider warben mit der unbegrenzten Spielwiese, die keine Wünsche offen lässt. Neuen Nutzern fehlte das technische Verständnis; Wissen um Risiken und Gefahren wurden unzureichend vermittelt. Heute ist das Internet aus dem Alltag nicht mehr wegzudenken und wird von breiten Teilen der Bevölkerung genutzt, sowohl von computertechnisch versierten Nutzern als auch von Laien. Jedoch fehlt dem Großteil der Nutzer immer noch das Wissen, um sich ausreichend

vor Gefahren zu schützen. Der bedenkenlose Umgang mit persönlicher Information kann in Verbindung mit der Architektur des Internet sehr leicht für Überwachung genutzt werden ([2], [3], [4]).

Datennetze ermöglichen den schnellen, billigen und unkomplizierten Austausch von Texten, Programmen, Musik und Filmen. Mit der Killerapplikation Napster wurde dies der Content-Industrie schlagartig bewusst: Zum Höhepunkt des Booms tauschten 17 Millionen Nutzer (meist illegal) MP3s über das Netzwerk. Peer-to-Peer-Netze sollten in den nächsten drei Jahren eine Renaissance erleben: Leicht zu bedienende Clients wie Napster, Gnutella oder Limewire ermöglich(t)en schnelle Suche und direktes Herunterladen von Multimedia-Daten. Nutzer dieser Dienste fühlen sich oft anonym und glauben, nicht mit Konsequenzen rechnen zu müssen [5]. Zwar gab es schon vorher, vor allem im universitären Bereich, viele Designs für Peer-to-Peer-Netzwerke, doch erst durch benutzerfreundliche Clients wurde dieses Modell populär. Konzeptuell waren die verfügbaren Clients den schon vorhandenen Designs meist unterlegen.

Auch Napster wurde Opfer eines eher schlechten Designs: Der Suchmechanismus zum Auffinden von Peers, die eine bestimmte Datei zum Download anbieten, lief hier zentralisiert über einen Server. Ohne einen solchen Server war das System nutzlos. Interessenverbände der Film- und Musikindustrie hatten letztendlich Erfolg mit einer Klage, Napster ist mittlerweile bankrott.

Nun begann die Entwicklung von dezentralisierten Systemen, die widerstandsfähig gegen Gerichtsurteile oder Zensur sein sollten. Eines dieser Systeme ist Gnutella, vorgestellt im März 2000. Der Single Point of Failure von Napster wurde hier eliminiert und durch eine Broadcast-Suche ersetzt: Ein Client fragt benachbarte Peers nach den gewünschten Inhalten, diese leiten die Anfrage weiter oder öffnen eine Verbindung zum Anfrager, wenn sie die gesuchte Datei anbieten. Auch dieser Ansatz beinhaltet Schwachstellen. Durch den entstehenden Netzwerktraffic bei der Suche skaliert Gnutella schlecht, es kam schon zum völligen Zusammenbruch des Netzes [6]. Die Herausforderungen und Schwierigkeiten beim Entwurf von Peer-to-Peer-Netzen werden im folgenden noch zur Sprache kommen. Am Beispiel von Freenet soll in dieser Arbeit gezeigt werden, wie viele dieser Probleme gelöst werden können.

Auf der anderen Seite sind Inhalteanbieter nicht untätig geblieben. Neben dem Gerichtsweg gegen Peer-to-Peer-Anbieter nutzen sie Gesetzesinitiativen und Digital Rights Management (DRM) zur Durchsetzung ihrer Interessen.

Die bekannteste legislative Maßnahme ist der US-amerikanische DMCA (Digital Millennium Copyright Act), verabschiedet 1998. WIPO-Verträge [7] und eine EU-Richtlinie ([8], [9]) zur Umsetzung eines ähnlichen Gesetzes werden in den nächsten Monaten auch in

Deutschland in einem neuen Urheberrechtsgesetz mit vergleichbaren Bestimmungen münden [10].

Ein Kritikpunkt am DMCA ist das Verbot, Kopierschutzmechanismen zu umgehen. Nicht nur der Content ist geschützt, schon der Versuch des Umgehens der Schutzmechanismen ist eine Straftat. Es wird verfügt, dass die Anbieter Möglichkeiten schaffen müssen, um dann trotz Kopierschutz "Fair Use" zu ermöglichen. Damit ist beispielsweise das auszugsweise Kopieren von Büchern an Schulen gemeint, auch Privatkopien von Musik fallen in diese Kategorie. Die Industrie wird eher wenig Interesse an solchen Möglichkeiten zeigen; "Fair Use" im Zeitalter des Internet könnte durch Bürokratie und Einschränkungen unbrauchbar werden. Die Auswirkungen fasst Armin Medosch zusammen:

> "Denn was der DMCA tut, ist eine Barriere für die Verbreitung von Wissen und für die Erzeugung neuer Erkenntnisse aufzustellen. Das Gesetz ermöglicht nicht nur die Verfolgung kriminell gesinnter Kopierschutzknacker, sondern behindert etablierte und auf ihre Reputation bedachte Wissenschaftler wie Felten und Ferguson in ihrer Arbeit und bei der Verbreitung ihrer Forschungsergebnisse. Es schützt schlechten Code, indem es diejenigen kriminalisiert, die diesen fehlerhaften Code analysieren. Und es stellt eine ganz grundsätzliche Balance in Frage - die zwischen dem Schutz geistigen Eigentums und dem Recht auf fair use." [11]

Der DMCA ist so allgemein gehalten, dass er Schutz vor Musikpiraten ebenso ermöglicht wie Zensur von missliebiger Information. Dazu zwei Beispiele:

- Hewlett Packard drohte mit einer Klage gegen ein Team von Computersicherheitsexperten, nachdem ein Mitglied Quellcode veröffentlicht hatte, mit dem sich eine Schwachstelle im Tru64 Unix-Betriebssystem ausnutzen ließ [12].
- Die Suchergebnisse auf `http://google.com` werden zensiert. Dies wurde bekannt, nachdem die Scientology-Kirche mit einer Klage wegen Copyright-Verletzung drohte, weil Suchergebnisse von Google Links auf Material der Kirche enthielten, die unter Copyright standen. Begründet wurde dies mit dem DMCA. Um einem Prozess zu entgehen, schloss Google die beanstandeten Adressen aus [13].

Der Kopierschutz selbst wird über Digital Rights Management (DRM) realisiert. Ein Konsortium führender Hardware- und Softwarehersteller arbeitet weiterhin an einem für Inhalteanbieter vertrauenswürdigen System, der Trusted Computing Platform. Applikationen und Dokumente können auf diesem System nur auf die vorhergesehene Weise genutzt

werden. Der Nutzer wird praktisch komplett von Applikationen, Betriebssystem und Hardware ausgegrenzt. Vorteile gibt es durchaus, doch die Nachteile wiegen schwer.

Vorteile:

- Wahrung von Anbieterrechten
- stabiles System, da nicht beliebiger Code installiert und ausgeführt werden kann

Nachteile:

- Implementierungsfehler in der Plattform oder Kompromittierung von Schlüsseln können verheerende Folgen haben
- Gefahr von Hintertüren
- vielfältige Möglichkeiten zur Überwachung und Zensur
- Einschränkung/Bürokratisierung des Fair Use

Zweifellos ermöglichen solche Kopierschutzsysteme eine sehr gute Kontrolle des Nutzers, und können leicht für andere Zwecke missbraucht werden: Nutzer-Profiling und Data Mining, Ausgrenzen von Konkurrenz-Software, bis hin zur Zensur bestimmter Inhalte für einzelne Nutzer. Programmierer des Freenet-Projektes arbeiten aus verschiedenen Gründen mit: Viele sehen Freenet als Schutz vor Zensur, manche wollen das Copyright im digitalen Zeitalter überflüssig machen.

Im Internet wird im Moment der Konflikt zwischen Copyright und freier Verfügbarkeit von Daten sowie freier Meinungsäußerung ausgetragen. Protagonisten sind Rechteinhaber, Programmierer, und die Nutzer. Auch hier wird die "Wahrheit" irgendwo in der Mitte, und nicht bei einem der Extreme (völlig freie Verfügbarkeit von Daten - völlige Überwachung des Nutzers) liegen. Die Zukunft von Technologien wie Freenet wird nicht nur durch technische Parameter entschieden.

# Kapitel 2

# Ähnliche Systeme

## 2.1 World Wide Web

Das WWW ist neben Email der populärste Teil des Internet und ermöglicht das schnelle Veröffentlichen von Informationen. Auf diese kann dann per eindeutiger URL [1] zugegriffen werden. Neben Veröffentlichung beliebiger Daten ist vor allem das leichte Referenzieren von Websites untereinander durch URLs interessant.

Das im WWW verwendete Protokoll HTTP [2]bietet keinerlei Anonymität für Nutzer und Serverbetreiber. Der Autor eines Dokuments kann zwar anonym bleiben, jedoch kann der Serverbetreiber durch Gerichtsbeschluss o.Ä. gezwungen werden, dessen Identität preiszugeben (z.B. durch Logs, die den Upload der Datei durch einen Nutzer belegen).

Eine weitere Schwachstelle ist die Skalierbarkeit einzelner Server; aufgrund der steigenden Zahl von Internet-Nutzern können viele Server einer plötzlichen Flut von Anfragen nach einem Dokument nicht standhalten, Mirrors sind nicht rechtzeitig verfügbar (Slashdot-Effekt [14]).

## 2.2 Akamai EdgeSuite

Dieses kommerzielle System ergänzt das WWW um die Komponente des Mirroring. Akamai betreibt weltweit etwa 16000 Server in mehr als 60 Ländern. Anbieter können diese Server nutzen, um dort statischen und dynamischen Content zu hosten. Der Inhalt ist also nicht nur auf einem zentralen Server verfügbar, sondern weltweit verteilt. Auf Nutzerseite ändert sich nichts - beim Aufruf der Website des Anbieters wird der nächstgelegene Akamai-Server ermittelt und der Browser des Nutzers darauf umgeleitet, auf diese Wei-

[1] Uniform Resource Locator
[2] Hypertext Transfer Protocol

se bekommt jeder Nutzer unabhängig vom Aufenthaltsort den für ihn schnellsten Server zugewiesen [15].

## 2.3 Web MIXes / JAP

JAP [3]([16], [17]) ist ein Anonymisierungsdienst für Nutzer von Websites. Es gibt drei Komponenten: Einen Client (JAP), der auf dem Rechner des Benutzers installiert wird; Kaskaden von Rechnern im Internet, sogenannte "Web-Mixe"; sowie ein Informationsdienst.

JAP basiert auf den 1981 von David Chaum vorgestellten Mixes [18]: Hier wird eine Email-Nachricht durch eine vordefinierte Kette von Rechnern geschickt, der letzte Rechner liefert die Email an den Empfänger aus. Der Absender verschlüsselt die Nachricht mit den Public Keys dieser beteiligten Mix-Server, beginnend mit dem letzten. Die letzte "Schale" der Verschlüsselung wird mit dem Public Key des Servers gebildet, an den der Absender die Nachricht direkt schickt. Dieser dekodiert sie mit seinem Private Key und schickt das Ergebnis an den nächsten Server der Kette. So werden die "Schalen" entfernt, bis die Originalnachricht auf dem letzten Server wiederhergestellt ist. Es können keine Rückschlüsse auf die Identität des Absenders gezogen werden, solange mindestens einer der Server aus der Kette "richtig" arbeitet. Erst wenn alle Server kollaborieren, ist das System unsicher.

Es können auch Antworten auf dem umgekehrten Weg versandt werden - es bietet sich an, ein Protokoll wie HTTP über Mixes zu implementieren. Genau dies versucht JAP.

Der Client auf Benutzerseite leitet HTTP-Anfragen verschlüsselt nach dem beschriebenen Verfahren an eine der verfügbaren Mix-Kaskaden weiter. Die Anfragen werden durch die Kaskade geleitet, vom letzten Mix an einen Cache-Proxy und von dort an den Web-Server gesendet, mit dem der Benutzer kommunizieren möchte. Die letzte Übertragungsstrecke findet unverschlüsselt statt, dies kann mit SSL behoben werden.

Ein Angreifer, der die Absender und die Länge aller in die Kaskade hineingehenden und die Länge der am Ende der Kaskade ausgehenden Nachrichten filtert, könnte mit einer gewissen Wahrscheinlichkeit feststellen, wer welche Seite abruft. Dies wird durch Message Padding und Dummy-Anfragen verhindert.

JAP wahrt damit die Anonymität der Nutzer, jedoch nicht die der Server.

[3] Java Anon Proxy

## 2.4 Rewebber

Rewebber [19] hingegen wahren die Anonymität von Serverbetreibern und Autoren. Die Idee: Ein Autor verschlüsselt ein Dokument mit einem symmetrischen Schlüssel und legt das Chiffrat auf einem Server ab. Es soll möglich sein, dieses Dokument zu lesen, ohne die URL zu kennen (sonst wäre es ein leichtes, den Betreiber des Servers ausfindig zu machen, und entweder ihn für den Inhalt verantwortlich zu machen oder die Identität des Autors in Erfahrung zu bringen). Dazu veröffentlicht der Autor einen Locator, zusammengesetzt aus der Adresse des Dokuments und dem symmetrischen Key. Dieser Locator wird nun noch mit dem Public Key des zu benutzenden Rewebber-Proxies verschlüsselt und kann wie in Abbildung 2.1 dargestellt an den Proxy geschickt werden, der das Dokument vom eigentlichen Server herunterlädt und an den Client schickt.

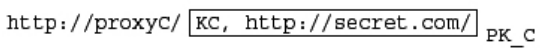


**Abbildung 2.1:** Ein Rewebber-Locator mit nur einem Proxy-Server

Nun liegt die Schwachstelle beim Rewebber-Proxy: Wird er kompromittiert oder von einer nicht vertrauenswürdigen Instanz betrieben, besteht kein Schutz, denn bei nur einem Rewebber ergibt sich die Zuordnung unverschlüsseltes Dokument zu URL auf diesem Server. Deshalb werden - vergleichbar den Web MIXes - mehrere Rewebber-Proxies in Reihe geschaltet, so dass nur der letzte die richtige URL und der erste (der mit dem Client kommuniziert) das unverschlüsselte Dokument, nicht aber dessen URL sieht. Hier ein Beispiel:

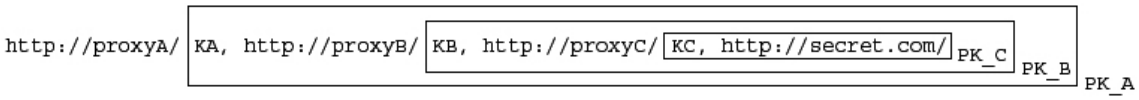


**Abbildung 2.2:** Ein Rewebber-Locator mit drei Proxy-Servern

Proxy A bekommt die Anfrage vom Client und dechiffriert (KA, URL_A) mit seinem Private Key, anschließend wird die Anfrage an Proxy B weitergeleitet, dieser dekodiert (KB, URL_B), selbiges für C. C kontaktiert den für den A, B und den Client anonymen Server secret.com, der ein mehrfach verschlüsseltes Dokument zurückliefert. C dekodiert dieses mit KC und liefert es an B, dort wird mit KB und anschließend von A mit KA dechiffiert, das Originaldokument ist damit wiederhergestellt und wird an den Client gesendet.

Der Autor bestimmt bei diesem System, wie viele Rewebber-Server er einsetzen will. Er muss sein Dokument dann mit entsprechend vielen symmetrischen Keys verschlüsseln und einen korrekten Locator bilden. Der große Nachteil liegt darin, dass schon der Ausfall eines Servers eine URL ungültig macht.

Anonymität für Autoren ist damit gesichert, Goldberg und Wagner weissen darauf hin [19], dass der Rewebber gegebenenfalls mit anderen Systemen kombiniert werden muss, wenn Anonymität für Nutzer gewünscht ist. Ein mögliches Gegenstück wäre das vorher besprochene JAP (Abschnitt 2.3).

## 2.5 Crowds

Crowds [20] zielt wie JAP auf den Schutz der Privatsphäre des Nutzers ab und ermöglicht anonymes Surfen im WWW. Dazu formieren sich Nutzer zu einer "Crowd". Grundgedanke ist, in der Masse unterzutauchen, damit nur Aussagen über die Grundgesamtheit der Beteiligten, nicht aber über einen einzelnen Nutzer getroffen werden können.

Jeder Crowds-Nutzer startet einen Proxy auf seinem Rechner. WWW-Anfragen werden nun von diesem Proxy an ein zufällig ausgewähltes Mitglied der Crowd geschickt. Dieses ermittelt über eine Zufallsfunktion, ob der Request erneut weitergeleitet oder der gewünschte Server kontaktiert wird. Die Wahrscheinlichkeit für eine Weiterleitung ist größer 50%, mit höherer Wahrscheinlichkeit steigt die Sicherheit des Systems, die Latenz wird allerdings größer.

Aufgrund des Proxy-Mechanismus tauchen im Log des Webservers nicht die Daten des Nutzers auf. Die Antwort des Webservers wird auf dem Weg der Anfrage zurück geroutet, jede Station kennt nur den unmittelbaren Vorgänger und Nachfolger auf dieser Route.

Der Empfänger einer Anfrage kann damit nicht feststellen, ob die Nachricht vom Absender generiert oder nur weitergeleitet wurde. Um das System zu kompromittieren, müsste ein Angreifer alle Proxies, ausgenommen den des zu überwachenden Nutzers, unter seine Kontrolle bringen. Erst dann lässt sich feststellen, welche Anfragen von diesem Nutzer stammen.

Schutz für Serverbetreiber oder Autoren bietet Crowds nicht.

## 2.6 FastTrack

Hierbei handelt es sich um eine Peer-to-Peer-Technologie, die in verschiedenen Programmen, wie z.B. KaZaA [21], eingesetzt wird. Die Idee von Gnutella wurde aufgegriffen und verbessert: Gnutellas Broadcast-Suche konnte nur sehr schlecht skalieren, was unter anderem an den unterschiedlich schnellen Internetanbindungen der Nutzer lag - Modemverbindungen wurden mit Suchanfragen saturiert, sie waren der Flaschenhals des Netzes.

KaZaA beugt dem vor, indem Rechner mit relativ vielen Ressourcen (vor allem Bandbreite, aber auch Rechenleistung) zu sogenannten "Supernodes" ernannt werden. Supernodes bilden ein mit Gnutella vergleichbares Netzwerk - sie verwalten Indizes von freigegebenen Dateien, führen Suchanfragen darüber durch und leiten sie an andere Supernodes weiter. Jeder KaZaA-Client meldet sich beim Start an eine Supernode an.

Supernodes kommunizieren nur untereinander und mit den Clients, für die sie zuständig sind; Clients kontaktieren nach einer entsprechenden Suchanfrage die Anbieter der Dateien selbst.

Durch dieses Verfahren erreicht KaZaA die beste Skalierbarkeit unter den P2P-Netzen, und ist auch am populärsten. Eine weitere Verbesserung des Netzes könnte durch das in Version 2 eingeführte "Integrity Rating" eintreten: Nutzer sind angehalten, die Qualität der von ihnen angebotenen Dateien zu bewerten. Ein "Participation Level" bewertet, ob ein Client nur Dateien herunterlädt oder auch zu den Inhalten beiträgt. Ein hoher Participation Level führt zur Bevorzugung bei Downloads.

Anonymität gehört nicht zu den Designzielen von KaZaA; so wurden in Dänemark bereits Nutzer abgemahnt, weil sie Dateien zum Download anboten [22]. Das System ist allerdings robust gegen die Abschaltung von Supernodes und damit nur sehr schwer zu schließen. KaZaA und eDonkey [23] sind im Moment die beliebtesten P2P-Systeme.

## 2.7 Usenet

Usenet ermöglicht Diskussionen in moderierten oder unmoderierten Gruppen. Die Server sind weltweit verteilt, befinden sich also in verschiedenen Jurisdiktionen. Jeder Server abonniert eine vom Administrator konfigurierte Auswahl von Gruppen, die die Benutzer des Servers dann lesend oder schreibend nutzen können. Die Beiträge der Gruppen werden auf dem Server gespeichert. In regelmäßigen Abständen kommunizieren die Server und tauschen neue Nachrichten aus.

Aufgrund dieser Struktur wird ein gewisser Schutz gegenüber Zensur erreicht - auch wenn bestimmte Inhalte in einem Land illegal sind, hat dies keinen Einfluß auf die restlichen Server des Usenet. Durch die automatische Verbreitung von Nachrichten wäre somit ein sehr guter Mechanismus zum Publizieren von sonst leicht zensierbarem Material vorhanden. Allerdings ist es möglich, Artikel im Usenet durch sogenanntes Canceln zu löschen. Dazu muss lediglich der Absender-Header gefälscht werden - Zensur stellt somit kein Problem dar.

## 2.8 Publius

Dieses System bietet vor allem Schutz vor Zensur. Publius wurde als ein CGI-Skript implementiert, das auf Servern installiert wird, die Inhalte hosten. Zugriff auf diese Inhalte bietet ein clientseitiger HTTP-Proxy, somit ist eine leichte Bedienung per Web-Browser möglich.

Zur Veröffentlichung wird eine Datei mit einem symmetrischen Schlüssel K verschlüsselt. Anschließend wird dieser Schlüssel in Fragmente aufgeteilt, so dass nur einige dieser Fragmente nötig sind, um den Schlüssel wiederherzustellen [24]. Aus einer vordefinierten Liste von Publius-Servern werden nun einige ausgewählt und auf jedem die verschlüsselte Datei und genau ein Schlüsselfragment gespeichert. Nun wird eine URL für das veröffentlichte Dokument gebildet, die u.a. die Adressen der verwendeten Server enthält.

Um das Dokument herunterzuladen, muss der Publius-HTTP-Proxy aus dieser URL die Liste der verwendeten Server extrahieren. Von einem dieser Server werden die Datei und ein Schlüsselfragment angefordert, eine Teilmenge der anderen Server liefert weitere Fragmente. Der Schlüssel wird zusammengesetzt und das Dokument entschlüsselt. Ist die darauffolgende Integritätsprüfung nicht erfolgreich, war entweder die Datei oder ein Teil des Schlüssels korrupt. Dann wird die Datei von einem anderen Server heruntergeladen und der Schlüssel von einer anderen Teilmenge von Fragmenten generiert. Im Worst Case müssen alle möglichen Kombinationen getestet werden.

Durch die Replikation des Dokuments über mehrere Server ist Zensur nur sehr schwer möglich. Ein Serverbetreiber kann Verantwortung für illegale Inhalte theoretisch von sich weisen, da er nur verschlüsselten "Datenmüll" speichert, der ohne andere Schlüsselfragmente wertlos ist. Ob diese etwas blauäugige Herangehensweise vor Gericht Bestand hat, ist allerdings ungeklärt, aber auch für Freenet von Bedeutung.

Weitere Features von Publius sind das Löschen und Ändern von veröffentlichten Dokumenten. Kritiker sehen hierin eine Schwachstelle, die den Autor unmittelbar gefährdet - er könnte gezwungen werden, unliebsame Inhalte selbst zu zensieren.

Der Autor muss beim Einstellen eines Dokuments selbst für die Anonymisierung seiner Verbindung sorgen, genau wie ein Client, der das Dokument über mehrere HTTP-Verbindungen herunterlädt.

Für ersteres empfehlen die Publius-Autoren Crowds (Abschnitt 2.5). Auf Client-Seite könnte JAP (Abschnitt 2.3) zum Einsatz kommen.

## 2.9 Free Haven

Free Haven basiert auf einem vielversprechenden Design, ist aber leider noch nicht lauffähig. Im Gegensatz zu anderen Systemen kann ein Autor hier ein Dokument mit der gewünschten Verfügbarkeitsdauer versehen.

Die Server innerhalb eines sogenannten Servnets stellen bei Free Haven Festplattenkapazität zur Verfügung. Vergleichbar den Schlüsseln bei Publius werden die Dateien in redundante Fragmente aufgesplittet und auf verschiedenen Servern gespeichert.

Das zentrale Konzept von Free Haven ist das "Trading", der regelmäßige Austausch von Dateifragmenten zwischen den Servern. Der Autor übergibt die Fragmente des Dokuments, die im System gespeichert werden sollen, an seinen lokalen Server. Dieser tritt mit anderen Servern in Kontakt, bietet die Fragmente an und speichert im Gegenzug Fragmente dieser Server. Parameter wie Größe des Fragments und gewünschte Speicherdauer beeinflussen die Transaktion.

Mittels eines Broadcast-Mechanismus können alle Teile einer Datei wieder aufgefunden werden, Server verschicken sie dann über einen anonymen Kommunikationskanal (per Remailer-Reply-Block) an den Anfrager.

Server in Free Haven haben Pseudonyme und können darüber anhand ihrer Erfüllung von Verpflichtungen bewertet werten. Sollte ein Server z.B. einen abgeschlossenen Vertrag zur Speicherung von Daten nicht einhalten, wird er abgewertet. Dadurch werden "schädliche" Systeme ausgegrenzt, die Speicherung von Dokumenten für eine definierte Zeitspanne kann durchgesetzt werden.

Free Haven bietet Anonymität bzw. Pseudonymität für Autor, Leser und Server. Aufgrund mangelnder Effizienz, vor allem durch die Broadcast-Suche nach Dokumentfragmenten, ist das System noch nicht nutzbar.

# Kapitel 3

# Ziele des Freenet-Projekts

Der kleinste gemeinsame Nenner aller oben besprochenenen Systeme und auch Freenet ist die Vernetzung von Computern, um über ein spezifisches Protokoll Daten auszutauschen. In diesem Kapitel soll erläutert werden, welche Ziele darüber hinaus beim Entwurf der Freenet-Architektur eine Rolle gespielt haben, und wie sie umgesetzt wurden. Viele Konzepte der in Kapitel 2 diskutierten Systeme werden wieder sichtbar.

Begriffe:

- Node: ein Knoten des Freenet-Netzes, der auf einem vernetzten Rechner läuft, mit anderen Knoten kommuniziert und das Freenet bildet

## 3.1 Anonymität für Leser

Keiner, auch nicht andere Freenet-Teilnehmer, soll feststellen können, wer welches Dokument abfragt.

Eine Anfrage nach einer Datei wird vom Client mittels des Freenet Client Protocol (FCP) an eine Node geschickt. (Jeder Nutzer sollte eine eigene "erste Node" betreiben, da der Kommunikationskanal zwischen Client und Node nicht gesichert ist. Dies stellt aber kein Problem dar, wenn der Nutzer selbst für die erste Node zuständig ist und ihr vertrauen kann.) Hat diese Node das Dokument nicht gespeichert, wird die Anfrage weitergeleitet. Die Route wird von einem Routingkey bestimmt, der Teil der URL ist und beim Insert des Dokuments generiert wurde (siehe Abschnitt 4.4). Dabei kann jedoch keine der an der Route beteiligten Nodes feststellen, ob der Übermittler der Request-Nachricht die erste Node, also die des Nutzers, oder eine andere beteiligte Zwischenstation ist - hierin liegt die Anonymität.

Weiterhin werden Nachrichten zwischen Nodes verschlüsselt. Somit kann ein Angreifer, der das Netzwerkinterface der Nutzer-Node kontrolliert, zwar nachweisen, dass eine

Nachricht gesendet wurde, obwohl keine Nachricht einging (also keine Weiterleitung stattfand). Er kann aber keine Aussagen über den Inhalt der Nachricht treffen.

## 3.2 Anonymität für Autoren

Jeder Nutzer soll Daten in Freenet veröffentlichen können, ohne seine Identität preisgeben zu müssen.

Nach demselben Prinzip wie die Requests funktionieren auch Inserts. Vor dem Einfügen des Dokuments wird ein Hash-Wert über den Inhalt berechnet. Der Insert wird dann zu einer Node geroutet, die für dieses Dokument "zuständig" ist. Alle Nodes auf dieser Route speichern das Dokument, keine Node weiß, ob die Vorgängernode den Insert-Request generiert oder weitergeleitet hat. Nur die erste Node, die der Autor selbst kontrolliert, hat diese Information.

## 3.3 Abstreitbarkeit für Serverbetreiber

Jede Node stellt Speicherplatz für Dokumente ("Datastore") zur Verfügung, der in folgenden Fällen genutzt wird:

- Speichern von neuem Content bei Insert-Nachrichten
- Speichern von altem Content bei Insert-Nachrichten mit Schlüssel-Kollision. Tritt eine Schlüssel-Kollision auf, so wird nicht der neue Content unter dem bereits existierendem Schlüssel gespeichert, sondern der alte Inhalt in Richtung des Ausgangspunktes der Nachricht zurückgesendet. Damit wird der alte Content weiterverbreitet, dieser Mechanismus soll vor Attacken schützen.
- Speichern von Content, der von einer Request-Nachricht abgefragt wurde. Auf diese Weise adaptiert das Netzwerk an die Anfragen, Dokumente wandern in Richtung der Nachfrager (automatisches Mirroring).

Es besteht die Gefahr, dass ein Angreifer einen Zensurversuch unternimmt, indem er Node-Betreiber aufgrund der Inhalte des Datastore verfolgt. Freenet versucht dies mit Kryptographie zu verhindern: Jedes Dokument wird verschlüsselt gespeichert, Schlüssel werden nicht auf dem Server gehalten. Somit befindet sich auf einer Node nur "Datenmüll", einzig ein Nutzer mit passendem Schlüssel kann ein verschlüsseltes Dokument abfragen und dekodieren.

## 3.4 Replikationsmechanismus

Das Freenet-Protokoll beinhaltet einen Mirroring-Mechanismus, der in den in Abschnitt 3.2 genannten Fällen greift. Das Netz passt sich den gestellten Anfragen an: Content "wandert" in Richtung der Nachfrager und kann in Zukunft schneller abgerufen werden.

## 3.5 Verteilter Datenspeicher

Freenet lässt sich mit einer riesigen Festplatte vergleichen, jede Node stellt einige Sektoren des Speicherplatzes zur Verfügung. Allerdings ist die Festplatte redundant und damit weitgehend immun gegen den Ausfall von einzelnen Sektoren. Weiterhin können selten benutzte Dokumente durch neue oder populärere verdrängt werden, dies geschieht nach dem Least Recently Used-Verfahren.

In diesem Zusammenhang stellt sich die Frage, wie sich Freenet mit diesen Eigenschaften entwickeln wird - können alternative Inhalte überleben, oder verdrängt die Masse alles andere?

## 3.6 Adaptives, kooperatives Routing

Nodes können nicht alle Dateien des Netzwerks speichern. Deshalb spezialisiert sich jede Node auf eine bestimmte Teilmenge. Wenn eine Datei in Freenet eingefügt werden soll, muss zunächst ein SHA-1 Hashwert (160 Bit) gebildet werden, der mit sehr hoher Wahrscheinlichkeit eindeutig ist. Jede Node spezialisiert sich auf einen Abschnitt des SHA-1-Wertebereiches und kann Anfragen nach Dateien innerhalb dieses Bereiches nach einer Lernphase sehr gut beantworten. Von der Node erfolgreich bearbeitete Requests führen zu Einträgen in die Routing-Tabelle anderer Nodes - das Netzwerk adaptiert.

Die Entscheidung, an welche Node eine Anfrage weitergegeben wird, wenn eine Node diese nicht bearbeiten kann, wird mit Hilfe eines einfachen Abstandsmaßes getroffen. Kann die "beste" Node die Anfrage nicht beantworten, wird die nächstbeste kontaktiert. Bleibt auch dies erfolglos, sendet die Node ein "Request Failed" an den Anfrager, der seinerseits die zweitbeste Node kontaktiert. Die Anfrage wandert somit auf dem schnellstmöglichen Weg zur richtigen Node ("steepest-ascent hill-climbing with backtracking").

## 3.7 Keine zentralen Elemente

Freenet darf keinen Single Point of Failure beinhalten, Abschaltung oder Zensur wäre sonst leicht möglich.

Um dem Schicksal von Napster und Gnutella zu entgehen, wurden im Zweifelsfall Features weggelassen, statt eine Schwachstelle zu schaffen. So gab es im ursprünglichen Design noch keinen Suchmechanismus, dafür existiert mittlerweile ein Entwurf ([25], [26]).

# Kapitel 4

# Architektur, Implementierung

Vorbemerkung: Freenet befindet sich noch im Alpha-Stadium (aktuelle Version: 0.5) und ist damit ein schwer zu beschreibendes "Moving Target".

## 4.1 Topologie des Netzwerks

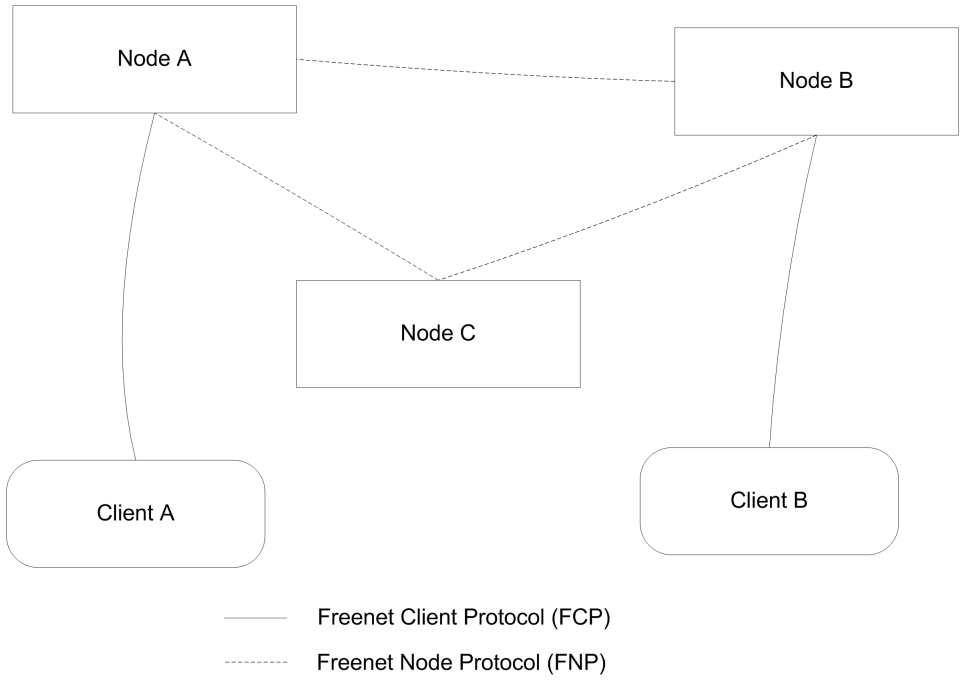


**Abbildung 4.1:** Freenet-Topologie

Es werden Nodes und Clients unterschieden. Nodes kommunzieren über das verschlüsselte Freenet Node Protocol (FNP) miteinander und bearbeiten Anfragen von Clients. Anfragen und Antworten zwischen Client und Node werden über das unverschlüsselte Freenet Client Protocol abgewickelt. Es wird klar, warum jeder Client eine eigene (lokale) No-

de betreiben sollte - nur so kann gewährleistet werden, dass die unverschlüsselte FCP-Verbindung nicht abgehört wird.

## 4.2 Adressraum

Aufgrund der geforderten Dezentralisierung kann es keine URIs [1] wie im Internet geben, die auf Server verweisen. Dadurch wäre u.a. Forderung 3.3 verletzt. Statt dessen wird ein Adressraum definiert, in den jede Datei mittels einer Hashfunktion (SHA-1) abgebildet wird. Jede Node spezialisiert sich auf bestimmte Bereiche des Adressraumes und speichert Dateien daraus; welche Bereiche das sind, wird durch NodeAnnouncements (Abschnitt 4.5.3) und indirekt durch Anfragen anderer Nodes festgelegt. Es lässt sich dennoch nicht beweisen, dass eine auf einen bestimmten Adressbereich spezialisierte Node eine Anfrage nach einem darin enthaltenen Schlüssel beantwortet hat, denn es können sich mehrere Nodes auf ähnliche Bereiche spezialisieren (es gibt keine eindeutige Abbildung).

Dieser "Low-Level-Adressraum" wird vom Freenet Node Protocol (Abschnitt 4.5) verwendet. Darauf aufbauend wird ein URI-Namensraum definiert (Abschnitt 4.4), den Clients einsetzen.

Zum Auffinden von Dateien werden Nachrichten anhand des berechneten Hashwerts geroutet, daher wird er auch als Routingkey bezeichnet.

## 4.3 Nodes

Wichtigste Komponenten einer Node sind der Datastore und die Routingtabelle. Jede Node im Netzwerk hat eine Nodereferenz, die sie eindeutig identifiziert. Diese besteht aus Public Key, den unterstützten Protokollen, ihrer Transportschicht-Adresse und einer Signatur über diese Daten.

### 4.3.1 Datastore

Der Datastore verwaltet den einer Node zur Verfügung stehenden Festplattenplatz. Im Gegensatz zu klassischen Peer-to-Peer-Systemen werden nicht lokale Dateien freigegeben, sondern die von anderen Nodes empfangene Dateien gespeichert. Zu jeder Datei wird neben einem Schlüssel, der die Datei eindeutig identifiziert (SHA-1-Hash) die Quelle in Form einer Nodereferenz gespeichert. Wie noch deutlich werden wird (Routing, InsertRequest, Abschnitt 4.5.3), entspricht diese Quelle nicht zwingend der Node, über die die Datei veröffentlicht wurde.

[1] Uniform Resource Identifier

Wenig nachgefragte Dateien werden bei Platzmangel nach dem Least Recently Used-Prinzip gelöscht. Die Referenz auf die Quelle bleibt zunächst erhalten und kann später zum erneuten Anfordern des Dokuments genutzt werden.

### 4.3.2 Routingtabelle

Hier werden Referenzen auf andere Nodes zusammen mit einer Adresse, die eine Spezialisierung der Node angibt, gespeichert. Anfragen nach Dokumenten "in der Nähe" dieser Adresse werden an diese Node weitergeleitet (siehe Abschnitt 4.5.3).

Eine Besonderheit sind Transient Nodes für nichtpermanente Verbindungen: Die Adressen dieser Nodes werden nicht in die Routingtabellen aufgenommen; es treten keine Verzögerungen durch Netzwerk-Timeouts auf, wenn die Nodes bereits offline sind.

## 4.4 URIs

URIs sind in Freenet folgendermaßen definiert:

```
freenet:[KeyType@]Routingkey[,CryptoKey][/DocName][//MetaString]
```

- `KeyType`: CHK, SSK, KSK, SVK [2]
- `Routingkey`: Wird zum Auffinden des Dokuments innerhalb des SHA-1-Adressraums verwendet
- `CryptoKey`: Entschlüsseln des Dokuments auf Client-Seite
- `DocName`: Nur für SSK und KSK, spezifiziert das Dokument innerhalb des SSK/KSK-Adressraumes (siehe Abschnitt 4.4.3)
- `MetaString`: Definiert weitere Verarbeitung nach Erhalt des Dokuments

Im folgenden soll auf die drei wichtigsten Schlüsseltypen eingegangen werden. Es wird jeweils das Vorgehen beim Veröffentlichen einer Datei D beschrieben. Am Ende jedes Verfahrens stehen der Routingkey, eine verschlüsselte Datei und die URI.

### 4.4.1 Content Hash Key (CHK)

- Generieren eines symmetrischen Schlüssels `SymKey`
- `FileEnc` = `File` verschlüsselt mit `SymKey`

[2] Content Hash Key, Signed Subspace Key, Keyword Signed Key, Signature Verifiable Key

- `Routingkey` = `SHA-1_Hash(FileEnc)`
- Veröffentlichen von `FileEnc` unter `Routingkey`
- URI: `freenet:CHK@Routingkey,SymKey`

SHA-1-Hash-Kollisionen sind sehr unwahrscheinlich, deshalb kann `Routingkey` als eindeutige ID für das Dokument angesehen werden. Scheitert der Insert, weil der Schlüssel schon vergeben ist, befindet sich diese Datei bereits im Freenet. Damit gleiche Dateien gleiche Routingkeys erzeugen, muss auch `SymKey` deterministisch generiert werden. Es ist aber nicht möglich, diesen Schlüssel anhand von `FileEnc` oder `Routingkey` zu ermitteln.

Da `SymKey` nicht auf den Nodes gespeichert wird, besteht für Serverbetreiber keine Möglichkeit, auf die Daten zuzugreifen. Damit ist die Abstreitbarkeit (Abschnitt 3.3) erfüllt. Ein Client kann die Datei über den `Routingkey` anfordern und sie mittels `SymKey` dekodieren. Löschung oder Änderung ist nicht möglich. Wenn die Datei nicht mehr nachgefragt wird, verschwindet sie nach und nach von den Nodes.

### 4.4.2 Keyword Signed Key (KSK)

Im Gegensatz zu CHKs werden bei den Keyword Signed Keys einfache, nutzerfreundliche URIs generiert.

- `DocName` = Name, der die Datei innerhalb von Freenet identifiziert (z.B. /studium/sja-freenet.pdf)
- Generieren eines asymmetrischen Schlüsselpaars (`PubKey`, `PrivKey`) aus `DocName` (deterministisches Verfahren)
- `Routingkey` = `SHA-1_Hash(PubKey)`
- `FileEnc` = `File` verschlüsselt mit `DocName`
- Veröffentlichen von `FileEnc` unter `Routingkey`
- URI: `freenet:KSK@DocName`

Der Nachteil ist, dass KSK-Routingkeys abhängig vom `DocName`, nicht aber vom Inhalt des Dokuments sind. Dadurch lassen sich - im Gegensatz zu CHKs - verschiedene Dateien unter denselben KSKs veröffentlichen.

### 4.4.3 Signed Subspace Key (SSK)

SSKs beheben dieses Manko, indem der `Routingkey` aus `DocName` und dem privaten Teil eines asymmetrischen Schlüsselpaares generiert wird. Es wird ein Namensraum aufgespannt, in dem nur der Besitzer dieses Schlüssels veröffentlichen kann.

- Generieren eines asymmetrischen Schlüsselpaares (`PubKey`, `PrivKey`)
- `DocName` = Name, der die Datei innerhalb des Subspace identifiziert (z.B. /studium/sja-freenet.pdf)
- `FileEnc` = `File` verschlüsselt mit `DocName`
- `PubKeyHash` = `SHA-1_Hash(PubKey)`
- `DocNameHash` = `SHA-1_Hash(DocName)`
- `Routingkey` = `SHA-1_Hash(PubKeyHash XOR DocNameHash)`
- URI: `freenet:SSK@PubKey,DocName`

Zusätzlich wird ein Dokument beim Veröffentlichen signiert und kann mittels `PubKey` von jeder Node validiert werden. Auf diese Weise funktionieren auch SSK-Updates. Dazu wird das geänderte Dokument mit `DocName` verschlüsselt (siehe oben), signiert und neu eingefügt. Die Nodes erkennen die Kollision des Schlüssels, die gültige Signatur beweist aber, dass die neue Datei vom Besitzer des Namensraumes stammt (oder aber der Schlüssel gestohlen wurde). Die Nodes ersetzen die alte Kopie. Der Update-Mechanismus ist zur Zeit noch nicht implementiert, es fehlt eine Möglichkeit, alle Nodes mit der alten Version aufzufinden, um sie mit der neuen Version zu überschreiben.

Clients können nach Berechnung des `Routingkey` (mittels SHA-1 und XOR) auf die Datei zugreifen und sie mit `DocName` entschlüsseln.

## 4.5 Freenet Node Protocol (FNP)

FNP setzt im OSI-Schichtenmodell auf der Transportschicht auf und ist bis zur Applikationsschicht definiert. Es existiert bislang nur eine auf TCP aufsetzende Implementierung, andere Protokolle sind erst für eine 1.x-Version geplant.

### 4.5.1 Crypto Session Layer (FNP/S)

Hier wird eine verschlüsselte Verbindung etabliert. Zuerst erfolgt ein Handshake der Nodes, dann wird mit dem Diffie-Hellman-Algorithmus von beiden Nodes ein gemeinsamer Wert K berechnet. Vor einem das Netzwerk abhörenden Dritten bleibt dieser Wert verborgen. Nun generieren beide Seiten mittels K einen initialen Schlüssel für die Rijndael-Stromchiffre, mit der die weitere Kommunikation verschlüsselt wird. Um die Sicherheit weiter zu erhöhen, wird der Periodic Cipher Feedback Mode (PCFB [27]) eingesetzt, auf den hier nicht näher eingegangen werden soll.

### 4.5.2 Presentation Layer (FNP/P)

FNP/P ist ein nachrichtenbasiertes Protokoll zum Datenaustausch. Eine Nachricht besteht aus einem Befehl, Parametern und optional Binärdaten am Ende. Befehle und Parameter werden als UTF-8-Strings kodiert. Der Textteil einer Nachricht darf maximal 64KiB groß sein.

### 4.5.3 Application Layer

Hier werden die möglichen Nachrichten und das Verhalten der Node darauf definiert. Jede Nachricht enthält eine zufällig generierte, mit hoher Wahrscheinlichkeit eindeutige ID und die Nodereferenz des Absenders (siehe Abschnitt 4.3). Zusätzlich gibt ein Hops-to-Live-Wert (HTL) an, wie viele Nodes die Nachricht maximal passieren darf.

Wichtigstes Element einer Nachricht ist der Routingkey, der eine Adresse im schon erwähnten Freenet-Namensraum (Abschnitt 4.2) darstellt. Über diesen Key und die Routingtabellen der Nodes erreicht eine Anfrage schnell die Node, die dafür zuständig ist, deren Spezialisierung innerhalb des Adressraumes also in der Nähe dieses Routingkeys liegt. Dafür wird ein Operator definiert, der ermittelt, welcher von zwei Keys "näher" an einem Basiskey liegt. Auf welche Weise dieser Vergleich durchgeführt wird (lexikografisch etc.) ist unerheblich, muss aber für alle Nachrichtentypen gleich sein.

Es sind Nachrichten für drei Typen von Anfragen definiert: InsertRequest (Daten veröffentlichen), DataRequest (Daten abfragen) und NodeAnnouncement (Integrieren neuer Nodes ins Netzwerk).

#### Routing

Das Routing auf Application-Layer-Ebene funktioniert für alle Nachrichtentypen gleich. Eine Nachricht, die Node A nicht allein bearbeiten kann, wird geroutet. Als Absender wird die Nodereferenz von A angegeben, auch wenn die Nachricht von einer anderen Node kam. Der Empfänger, Node C, kann nun nicht mehr feststellen, ob die Anfrage von Node A

ausging oder von ihr geroutet wurde (Anonymität für Leser, Abschnitt 3.1, ist erfüllt). Hops-to-Live wird bei jedem Hop dekrementiert.

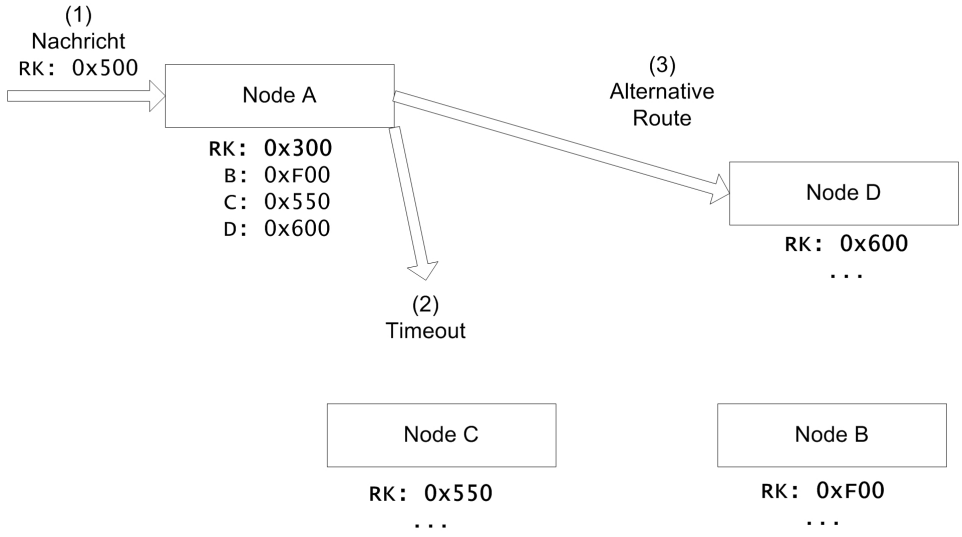


**Abbildung 4.2:** Routing einer Nachricht

In diesem vereinfachten Beispiel empfängt Node A eine Nachricht mit Routingkey 0x500. A hat die angeforderten Daten nicht, oder es handelt sich um einen InsertRequest bzw. ein NodeAnnouncement. Die Nachricht muss geroutet werden. Die "beste" Node wäre C (Spezialisierung 0x550), hier tritt aber ein Timeout auf, also wird zur zweitbesten Node geroutet. Dies wird wiederholt, bis die Nachricht bearbeitet oder Hops-to-Live abgelaufen ist.

#### InsertRequest

Node A empfängt von einem Client oder einer anderen Node eine InsertRequest-Nachricht mit einem Routingkey, der nach einem der in Abschnitt 4.4 beschriebenen Verfahren generiert wurde. Damit die Daten von einem späteren DataRequest gefunden werden, müssen sie auf einer Node gespeichert werden, die sich auf diesen oder einen "nahen" Routingkey spezialisiert hat.

Ist Hops-to-Live>0, wird die Nachricht an eine Node B aus der Routingtabelle weitergeleitet, deren Routingkey dem Routingkey der Nachricht (im einfachsten Fall dem CHK-Key) am nächsten liegt. Kann B die Anfrage nicht bearbeiten (Timeout oder QueryRejected), wird die nächstbeste Node kontaktiert. Kann keine erreicht werden, wird der Fehler an die anfragende Instanz (Client oder Node) zurückgegeben. Andernfalls endet der Request nach einigen Hops mit Hops-to-Live=0 bei einer Node C, deren Routingkey in der Regel nahe beim Routingkey der Nachricht liegt. Entlang dem entstandenen Pfad werden nun die zu speichernden Daten gesendet. Zur Minimierung von Latenzen fangen die Nodes mit senden

an, bevor die Datei komplett vorliegt (forwarding/streaming). Jede Node speichert das Dokument unter dem Routingkey in seinem Datastore ab. Zuletzt passen die beteiligten Nodes ihre Routingtabellen an, so dass genügend Referenzen auf die neuen Daten existieren.

#### DataRequest

Nach dem gleichen Schema funktioniert auch die Abfrage von Daten aus dem Netz. Es liegt wieder ein Routingkey vor, der aus einer URI generiert wurde. Ziel ist es nun, die Nachricht zu einer Node zu routen, die sich auf diesen Routingkey spezialisiert, und daher mit hoher Wahrscheinlichkeit diese Datei im Datastore hat. Hops-to-Live gibt hier die "Suchtiefe" an.

Hat eine Node das Dokument, antwortet sie mit einer DataFound-Nachricht und beginnt mit dem Transfer. Auch hier speichern alle Nodes auf dem Pfad das Dokument in ihrem Datastore und aktualisieren ihre Routingtabellen. Auf diese Weise "wandern" nachgefragte Dateien in Richtung der Nachfrage - und bleiben dort erhalten, wenn sie nicht von beliebteren Dokumenten verdrängt werden. Dies ist der angesprochene Replikationsmechanismus (Abschnitt 3.4). Kann die Datei nicht gefunden werden, sendet die letzte Node (bei der die Nachricht HTL=0 erreicht) ein DataNotFound über die Route an die initiale Node zurück. Diese (bzw. der Client) kann nun versuchen, HTL zu erhöhen und die Anfrage zu wiederholen.

In einem adaptierten Netz, in dem sich die Nodes untereinander "gut kennen" (über die Routingtabellen), genügen geringe HTL-Werte zum Auffinden der Daten. Mit der Anzahl der Nodes steigt allerdings auch die benötigte Pfadlänge. In [28] wird die mittlere Pfadlänge in Abhängigkeit von der Anzahl der Nodes mit `Pfadlänge = N^0.28` angegeben.

#### NodeAnnouncement

Eine neue Node hat noch keine Spezialisierung im Adressraum. Ausserdem ist sie anderen Nodes nicht bekannt und wird nicht für das Routing verwendet. Über NodeAnnouncements weisen bereits etablierte Nodes der neuen Node eine Spezialisierung zu und tragen sie in die eigenen Routingtabellen ein.

Zunächst generiert eine Node ein asymmetrisches Schlüsselpaar, durch das sie eindeutig identifiziert ist. Der öffentliche Schlüssel ist Teil der Nodereferenz (Abschnitt 4.3). Weiterhin benötigt die Node mindestens eine externe initiale Nodereferenz. Freenet liefert dazu mit jeder Distribution die Datei `seednodes.ref` aus. Zu Beginn wird ein NodeAnnouncement an eine zufällig ausgewählte Node gesendet. Nun läuft ein Protokoll ab, an dem sich mehrere Nodes beteiligen, um gemeinsam die Spezialisierung für die initiierende Node zu bestimmen. Das Besondere dabei: Kein Teilnehmer kann das Endergebnis vorhersehbar beeinflussen. Zusätzlich erhält die Node Routingkeys von Dateien, die in der Nähe der Spe-

zialiserung liegen. Es wird empfohlen, diese Schlüssel abzufragen, um eine gute Adaption zu ermöglichen.

## 4.6 Freenet Client Protocol (FCP)

Zur Kommunikation zwischen Client und Node ist ein einfaches textbasiertes Nachrichtenprotokoll definiert. Ressourcen werden über die bekannten URIs (Abschnitt 4.4) referenziert. Die wichtigsten Nachrichten sind ClientGet und ClientPut sowie Nachrichten zur Generierung von CHK- und SSK-Schlüsseln (GenerateCHK, GenerateSVKPair).

## 4.7 Metadaten

Jedes Dokument in Freenet enthält Metadaten. Diese bestehen aus Versionsinformationen und einer beliebigen Anzahl Parts, die Informationen über jeweils ein Dokument enthalten. Im folgenden werden die wichtigsten Einsatzgebiete von Metadaten dargestellt.

### 4.7.1 Metadaten für ein einzelnes Dokument

Im einfachsten Fall werden Informationen über das zum Schlüssel gehörige Dokument gespeichert. Es genügt ein Part.

```
Version
Revision=1
EndPart
Document
Info.Format=text/html
Info.Description=file
End
```

Zeilen 1-3 sind Versionsinformation. Die letzten vier Zeilen bilden den einzigen Part und definieren MIME-Typ und Kurzbeschreibung. Sämtliche Elemente des Dublin Core Metadata Standard [29] dürfen innerhalb des `Info.*`-Namensraumes verwendet werden. Die Metadaten werden von Spidern und Content-Filtern (z.B. Freenets Webinterface) ausgewertet.

### 4.7.2 Date Based Redirect

Da Freenet noch keine SSK-Updates beherrscht (siehe Abschnitt 4.4.3), wurde ein anderer Mechanismus eingebaut, um trotzdem aktualisierbare Dateien zu ermöglichen. Die Idee

ist, in den Metadaten eine Weiterleitung zu definieren, die abhängig von der Zeit ist. Viele Freenet-Websites (Freesites) benutzen dieses DBR-Format für regelmäßige Updates. Ein entsprechender Metadaten-Part sieht folgendermaßen aus:

```
Document
DateRedirect.Increment=1c20
DateRedirect.Target=SSK@QuL53nJqScrNwE-GNpUGNrZoZGwPAgM/ssk1
End
```

`DateRedirect.Increment` gibt an, wie oft das Dokument aktualisiert wird (hier alle 0x1c20 = 7200 sec = 2 h). Aktualisierte Dokumente werden unter abgewandelten Dokumentnamen eingefügt, die Zeitslots darstellen:

`SSK@QuL53nJqScrNwE-GNpUGNrZoZGwPAgM/`*`TIME`*`-ssk1`

TIME ist die Unix-Zeit (Sekunden seit 1.1.1970), ab der dieses Dokument gültig wird. Autoren veröffentlichen nun für jeden Zeitslot (also im Abstand von `DateRedirect.Increment` Sekunden) das Dokument unter den entsprechenden Adressen. Dies kann natürlich schon für viele Tage im Voraus geschehen. Ein Client sieht bei der Abfrage den DateRedirect, berechnet die aktuell gültige Adresse und ruft das Dokument darunter ab. Ältere Versionen bleiben bei entsprechender Nachfrage erhalten. Allerdings muss die Seite immer rechtzeitig aktualisiert werden (ein cronjob bietet sich hier an).

### 4.7.3 Mapfiles

Eine weitere Optimierung für Freesites sind Mapfiles. Eine Freesite, die jeden Tag eine neue Ausgabe veröffentlichen möchte, müsste für jede Datei (.html, .png, ...) eine neue Version im aktuellen Zeitslot einfügen. Auf Clientseite steigen die Latenzen, denn für jede Datei muss ein Redirect auf die aktuelle Version bearbeitet werden.

Mapfiles hingegen fassen die Metainformationen (Redirects, MIME-Typen usw.) aller Dokumente einer Freesite zusammen. Ein Beispiel (lange Zeilen wurden mit \ umgebrochen):

```
Document
Redirect.Target=freenet:CHK@K7Bj2EhCBqKo25d-RO4YDcduAN0KAw\
 I,k-6yQ8o8lBHrSQPwXYrk-A
Name=activelink.png
Info.Format=image/png
EndPart
Document
Redirect.Target=freenet:CHK@bhG9LAhfq0FUod6Lq45R3ypJResKAw\
```

```
 I,Y363NiUNzr4dnVYXSlhEZw
Name=description.txt
Info.Format=text/plain
EndPart
Document
Redirect.Target=freenet:CHK@m4joe~JIlwqxSi947I5j~t3U9HQKAw\
 I,LZIKhCiJhNtV~n3YgwJTKA
Name=index.html
Info.Format=text/html
EndPart
Document
Redirect.Target=freenet:CHK@Bs4dkQe~cc~BXYj3zp4maoJJhxoQAw\
 I,5w2E7tvhNk~1UDmQkRer9A
Name=nba.jpg
Info.Format=image/jpeg
EndPart
```

Für jedes Dokument gibt es genau einen Part (`Document` bis `EndPart`). Es wird jeweils per `Redirect.Target` auf einen CHK verwiesen. Möglich wäre auch eine Weiterleitung zu einem anderen Key, auch zu einem weiteren DBR.

## 4.8 Implementierung

Die Referenzimplementierung von Freenet erfolgt in Java. Solange Konzepte und Protokolle noch verändert werden, ist eine optimierte Version in C++ wenig sinnvoll.

# Kapitel 5

# Eigenschaften von Freenet

Freenet ist ein verteilter, redundanter, nicht persistenter Datenspeicher. Mit Tests sollen hier einige Eigenschaften dieses Speichers ermittelt werden.

## 5.1 Abrufbarkeit von Daten in Abhängigkeit von Zeit und Speichertiefe

Es soll ermittelt werden, wie lange in Freenet gespeicherte Dateien verfügbar sind. Unmittelbar nach dem erfolgreichen Speichern ist die Datei auf mehreren Nodes vorhanden (auf wie vielen ist abhängig vom gewählten Insert-Hops-to-Live-Wert ("Speichertiefe")). Wenn niemand diese Datei anfordert, verschwindet sie nach und nach von den Nodes, Anfragen benötigen mehr Hops.

Um dies zu verifizieren, wurden 116 4KiB große Dateien in Freenet gespeichert und 12 Tage lang im Intervall von 12 Stunden jeweils fünf dieser Dateien abgefragt. Die benötigten Hops für jede Anfrage wurden gespeichert und über jedes Intervall gemittelt. Dabei muss auch der Fall einer nicht mehr verfügbaren Datei berücksichtigt werden. Der maximale Hops-to-Live-Wert ist 25, kann das Dokument damit nicht gefunden werden, gilt es als verloren und es wird ein HTL-Wert von 40 angenommen, um eine Unterscheidung zum Erfolgsfall mit HTL=25 zu treffen. Ein Durchschnittswert von 40 innerhalb eines Zeitslots bedeutet demnach, dass keine Datei verfügbar war. Dieser Versuch wurde mit drei unterschiedlichen Speichertiefen durchgeführt (HTL=5, 12, 25).

Beim ersten Versuch (Insert-HTL=5, Abbildung 5.1) ist ein Ansteigen der Abfrage-HTL zu beobachten, bis nach 120 Stunden keine Datei mehr gefunden werden kann. Bei 168 und 204 Stunden sind einige Dateien vorhanden. Dieses Verhalten ist ein Indiz für ein funktionierendes Routing, denn die Dateien werden auf den verschiedenen Rechnern in Freenet gespeichert, die alle eine unterschiedliche Erreichbarkeit haben. Bei einer größeren Stich-

probe (Abfrage von 20 Dateien pro Intervall) wäre das Verhalten ähnlich, die Übergänge jedoch "glatter" gewesen.

Die Dateien sind mit einer hohen Wahrscheinlichkeit bis zu vier Tage verfügbar.

Wie Abbildung 5.2 zeigt, ist bei steigenden Insert-HTL die Verweildauer der Dateien schwerer vorhersagbar. Der erste "Komplettausfall" aller fünf Dateien ist nach 84 Stunden zu sehen, dann folgen 4 Abfrageintervalle mit relativ gutem Erfolg, ab 144 Stunden sind nur noch einzelne Dateien vorhanden. Die höhere Insert-HTL führt zu einem diffusen Ausfallverhalten. Während im ersten Versuch während der letzten sechs Intervalle keine Dateien mehr vorhanden waren, ist hier zu diesem Zeitpunkt die Erfolgsquote höher. Das liegt daran, das beim Insert die Dateien auf mehr Rechnern verteilt werden.

Dieses Bild verstärkt sich beim dritten Versuch. In jedem Intervall ist mindestens eine Datei verfügbar. Die Abfrage-HTL ist im Mittelteil maximal und gegen Ende deutlich besser als bei den ersten Versuchen. Als Grund kann hier wiederum die größere Verbreitung beim Insert gesehen werden. Bei HTL=25 ist die Wahrscheinlichkeit hoch, dass die Datei auf Nodes gespeichert wird, die längere Zeit verfügbar sind.

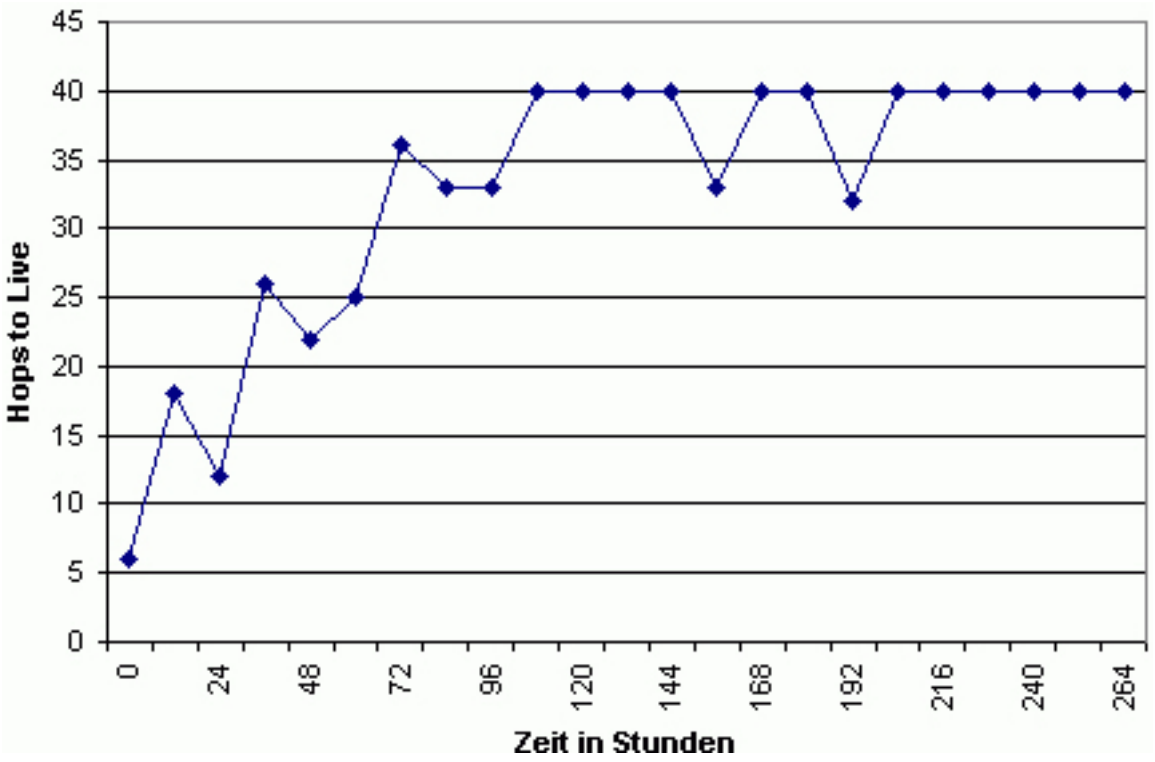


**Abbildung 5.1:** Speicherverhalten bei Insert-HTL 5

Die drei Versuche wurden nacheinander durchgeführt, um Unabhängigkeit zu gewährleisten. Dadurch entsteht aber eine Abhängigkeit vom Zustand von Freenet zum Zeitpunkt des Versuchs (neue Features, neue Versionen etc.). Das schränkt die Aussagekraft teilweise ein.

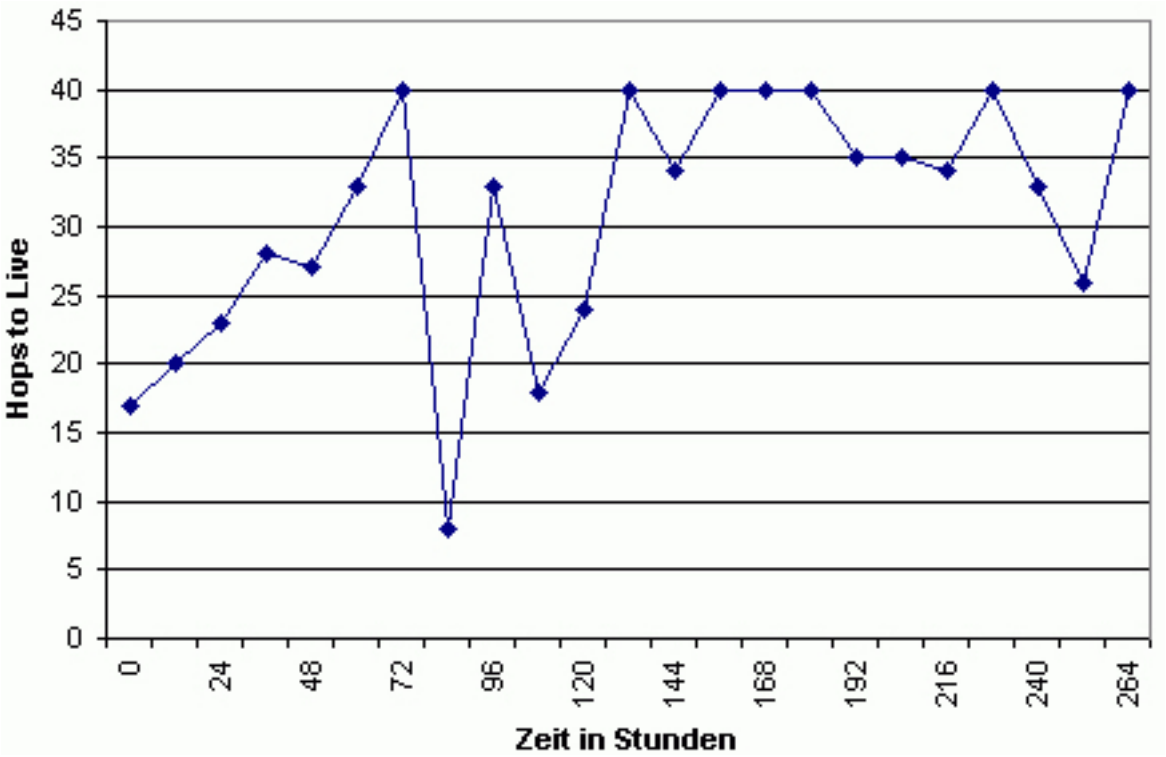


**Abbildung 5.2:** Speicherverhalten bei Insert-HTL 12

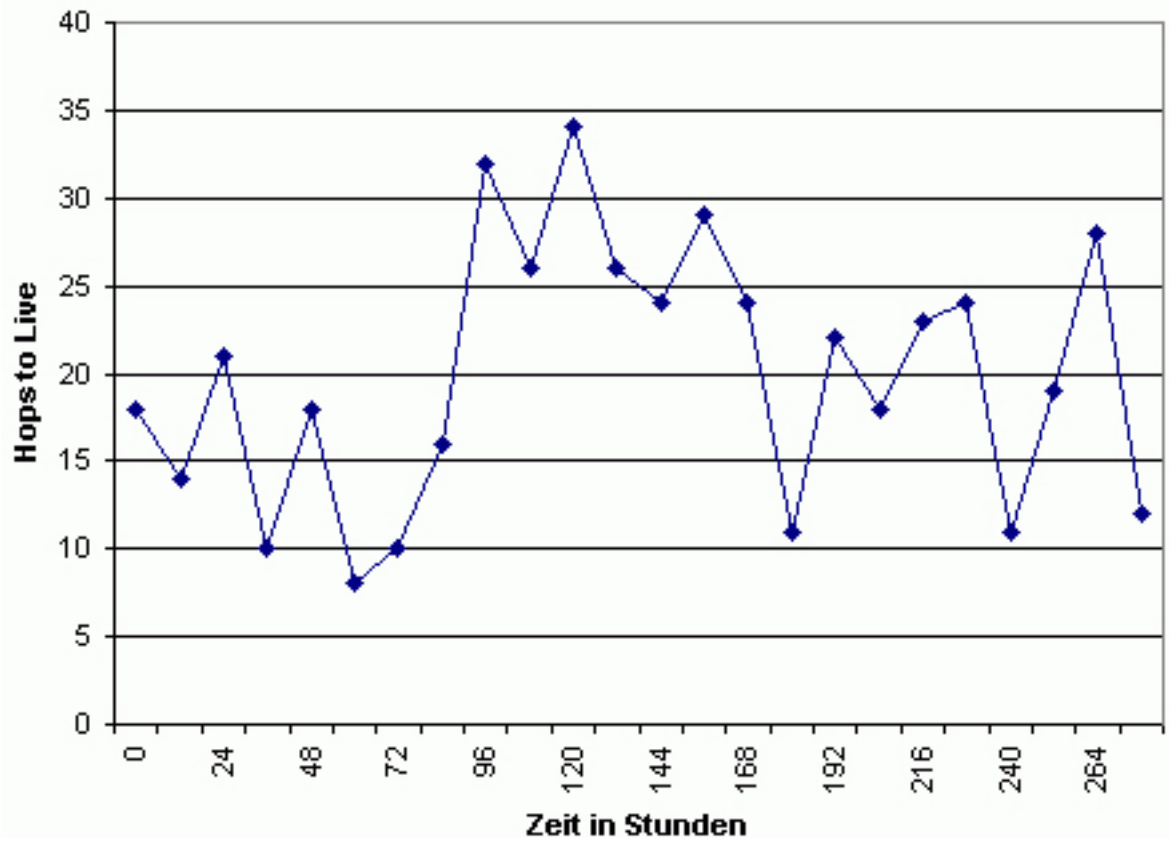


**Abbildung 5.3:** Speicherverhalten bei Insert-HTL 25

## 5.2 Caching-Eigenschaften

Hier soll untersucht werden, welchen Einfluss die automatische Weiterverbreitung von Inhalten bei verstärkter Nachfrage hat (siehe Abschnitt 3.4). Dazu wurden zehn Dateien im Netz gespeichert. In Intervallen von acht Stunden wurden sie abgefragt und ermittelt, wie viele Hops-to-Live dafür nötig waren. Vor jeder Abfrage wurde der Datastore gelöscht und die Node neu ins Netzwerk integriert. Hops-to-Live sollte mit jeder Abfrage sinken.

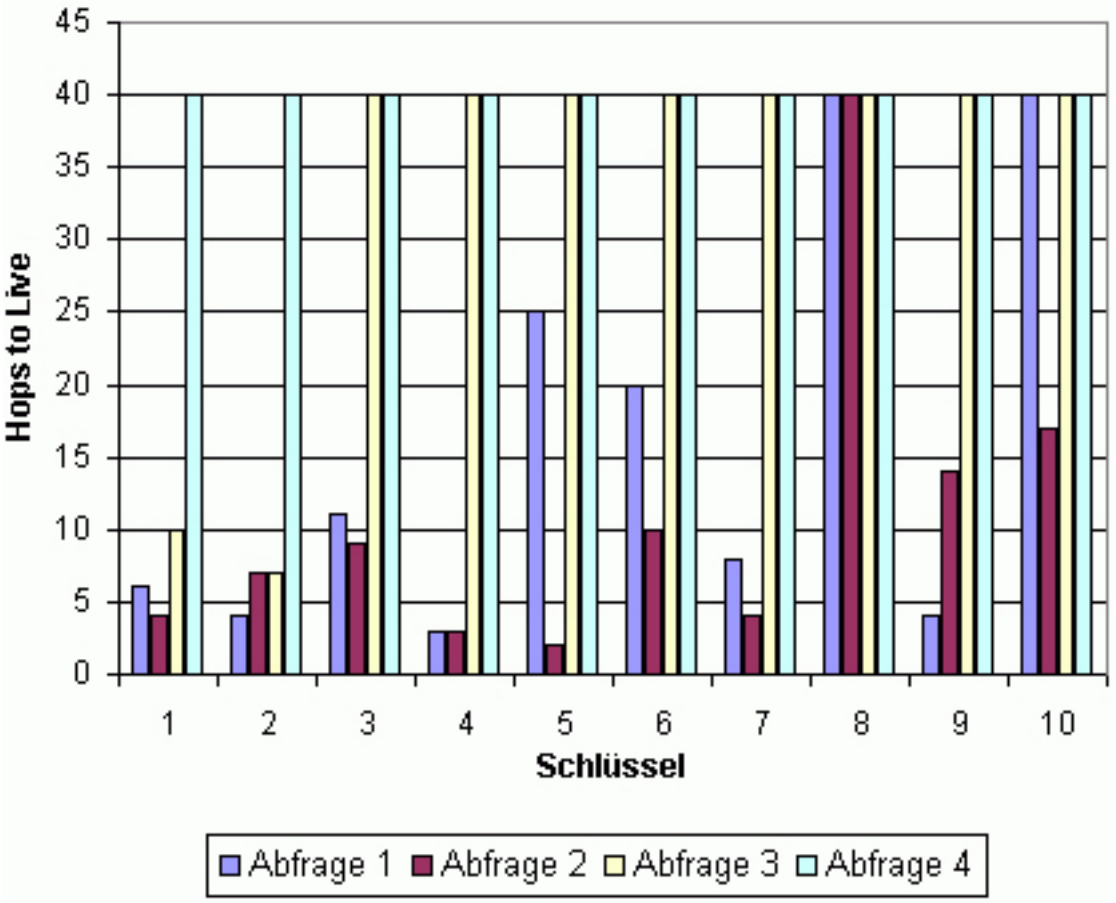


**Abbildung 5.4:** Caching-Verhalten bei zehn verschiedenen Schlüsseln

Bei sechs Schlüsseln (1, 3, 5, 6, 7, 10) war bei der zweiten Abfrage eine Verbesserung zu beobachten, die erste Abfrage hat hier also zu einer Verbreitung im Netzwerk geführt. Im Gegensatz dazu benötigten zwei Schlüssel mehr Hops, einer genauso viel und ein weiterer konnte nicht mehr gefunden werden. Bei der dritten Abfrage ist eine dramatische Verschlechterung bei den meisten Schlüsseln zu sehen, nur noch zwei werden überhaupt gefunden. Bei der vierten Abfrage ist kein Schlüssel mehr verfügbar. Wie lässt sich dieser plötzliche Einbruch erklären? Möglich ist, dass die Node nach dem Löschen des Datastores (wobei auch die Routingtabelle gelöscht wurde, und somit alle Informationen über nützliche Nodes verloren gingen) und der Reintegration in das Netzwerk noch nicht genügend Informationen über andere Nodes gesammelt hatte und nicht effizient routen konnte.

## 5.3 Netzwerktraffic bei Initialisierung einer Node

Abschließend soll gezeigt werden, welchen Verlauf der Outgoing-Traffic bei der Installation und Integration einer neuen Node ins Netzwerk nimmt. Dazu wurde ein aktueller Stable-Snapshot installiert, konfiguriert und gestartet. Anschließend wurde jede Stunde geloggt, wieviele Bytes gesendet wurden. Der Incoming-Traffic wurde nicht analysiert, entspricht aber aufgrund der Arbeitsweise von Freenet bis auf geringe Abweichungen dem Outgoing-Traffic.


**Abbildung 5.5:** Verlauf des Outgoing-Traffics einer neuen Node

Die Node braucht fast 15 Stunden, bis sie im Netzwerk bekannt ist und Anfragen eintreffen, die beantwortet werden. Vorher werden nur NodeAnnouncements gesendet, die aufgrund des geringen Datenaufkommens kaum zu sehen sind. In den nächsten 22 Stunden steigt das Datenvolumen an und bewegt sich danach im Bereich von 100-120 MiB/Stunde.

# Kapitel 6

# Implementierung einer Applikation zum sicheren, asynchronen Nachrichtenaustausch über Freenet (FreenetMTA)

Aufbauend auf den vorgestellten Primitiven soll nun ein Konzept für eine sichere, asynchrone Kommunikation über Freenet erarbeitet werden. Anschließend wird eine Implementierung vorgestellt.

## 6.1 Ziele

Es soll ein leicht benutzbarer Kommunikationskanal geschaffen werden, der die Eigenschaften von Freenet optimal nutzt. Dazu gehören:

- Anonymität für den Absender und Empfänger von Nachrichten
- Verschlüsselte Übertragung und Speicherung von Nachrichten
- Redundanz (Nachrichten werden auf mehreren Nodes gespeichert, auch bei Ausfall einzelner Nodes ist der Abruf noch möglich)

Auf Nutzerseite sollen keine zusätzlichen Programme vorausgesetzt werden. Es bietet sich an, bewährte Standards zu nutzen - hier die Internet-Protokolle für Email-Kommunikation (POP [30], SMTP [31]). Dann lassen sich beliebige Email-Clients einsetzen.

## 6.2 Erster Lösungsansatz

### 6.2.1 Konzept

Mit einem Signed Subspace Key (Abschnitt 4.4.3) lassen sich Daten innerhalb eines Namensraumes leicht auffindbar speichern. Ein Email-Account entspricht dann einem SSK-Namensraum, in dem Nachrichten an definierten Adressen eingefügt werden. Das Einfügen entspricht dem Versenden einer Email und ist nur mit dem privaten Schlüssel möglich, dieser muss also veröffentlicht werden. Für den durch das Schlüsselpaar

```
Public=EQCIHRp72uEB6QPsMyLFulDSCUA
Private=AMb8HP9Q6BjgEDWVh7FNtAmsAfS0
```

definierten Namensraum würden Nachrichten in dieser Reihenfolge eingefügt:

```
SSK@AMb8HP9Q6BjgEDWVh7FNtAmsAfS0/1
SSK@AMb8HP9Q6BjgEDWVh7FNtAmsAfS0/2
SSK@AMb8HP9Q6BjgEDWVh7FNtAmsAfS0/3
usw.
```

Die Schnittstelle zwischen Email-Client und Freenet bilden ein POP- und ein SMTP-Server. Zum Verschicken einer Nachricht wird als Email-Adresse der private Teil des SSK-Schlüssels angegeben, der SMTP-Server versucht dann, die Nachricht in den nächsten freien Slot, wie oben dargestellt beginnend von eins, zu speichern. Da aus dem privaten Schlüsselteil der öffentliche berechnet werden kann, ist der Account für jedermann lesbar. Deshalb sollte zusätzlich PGP-Verschlüsselung eingesetzt werden.

Der POP-Server speichert für jeden Account folgende Daten:

- Username
- Passwort
- öffentlicher Schlüssel des SSK-Namensraumes
- zuletzt gelesene Nachricht

Nach dem Login werden die neuen Nachrichten, beginnend nach der letzten gelesenen, abgefragt. Anschließend kann sie der Email-Client abrufen.

### 6.2.2 Implementierung

Da die schnelle Entwicklung eines lauffähigen Programms im Vordergrund stand und Effizienz praktisch keine Rolle spielt, erfolgte die Implementierung in der Skriptsprache Python. Hier stehen leistungsfähige Bibliotheken zur Verfügung, die den Entwicklungsaufwand drastisch reduzieren und die Konzentration auf die eigentliche Aufgabe ermöglichen.

Die Kommunikation mit der Freenet-Node erfolgt über den Kommandozeilen-Java-Client. Dadurch ist eine hohe Kompatibilität und Korrektheit gewährleistet, denn neue Entwicklungen (z.B. FEC oder Splitfiles) liegen dort meist als Referenzimplementierung vor.

Als Schnittstelle zum Netzwerk werden die beiden Python-Klassen `asyncore` und `asynchat` eingesetzt. Sie ermöglichen eine asynchrone, event-basierte Kommunikation auf Basis der `select()/poll()` - Systemrufe. `asynchat` erbt von `asyncore` und stellt Methoden zur einfachen Verwaltung der Ein- und Ausgabepuffer zur Verfügung. So lässt sich ein Line Terminator definieren (der vom verwendeten Netzwerkprotokoll abhängt, bei SMTP und POP "<CR><LF>"). Für jede durch diesen Terminator beendete gesendete Zeile ruft `asynchat` eine Methode auf, die die weitere Bearbeitung übernimmt. Für die Ausgabe gibt es Producer-Objekte, die in einer FIFO-Queue gesammelt werden. Mit diesen Mitteln lassen sich einfach und schnell Applikationen für Internet-Protokolle entwickeln.

#### Hauptprogramm

Es werden zwei Sockets für eingehende Verbindungen (POP und SMTP) geöffnet und mit `asyncore` überwacht. Bei eingehenden Verbindungen wird der entsprechende Server gestartet. Die Server sind als `asynchat`-Kindklassen implementiert. Die verschiedenen Server-Objekte laufen nicht in einzelen Threads, sondern blockieren sich gegenseitig, was die Bearbeitung von Requests anbelangt (z.B. beim Kommandozeilen-Aufruf des Java-Clients).

#### add-account

Dieses Skript erstellt neue Accounts. Es wird ein SSK-Schlüsselpaar generiert sowie Benutzername und Passwort abgefragt. Diese Informationen werden in der Accounts-Datei gespeichert.

Der Server muss auf einem Rechner mit aktiver Freenet-Node gestartet werden. Natürlich muss auch hier wieder der Nutzer dem Administrator vertrauen können, idealerweise betreibt jeder Nutzer einen eigenen FreenetMTA-Server in Verbindung mit einer Freenet-Node.

### 6.2.3 Diskussion

Diese Implementierung ist einsetzbar, hat allerdings zwei Schwachstellen. Zum einen verschwinden alte Nachrichten nach einiger Zeit aus dem Netz. Dies kann natürlich auch bei ungelesenen Nachrichten passieren. Bei regelmäßigem Abholen der Mails tritt dies aber nur selten auf. Wie in Abschnitt 5.1 festgestellt wurde, sind 4 KiB große Dateien (was einer typischen Email entspricht) schon bei einer Insert-HTL von 5 mit hoher Wahrscheinlichkeit bis zu vier Tage verfügbar. Problematisch ist, dass durch den verwendeten Insert-Mechanismus wieder frei gewordene Slots erneut benutzt werden. Da der POP-Server diese Nachrichten als schon gelesen markiert hat, gehen sie verloren.

Ein zweiter Nachteil: Inserts dauern umso länger, je voller die Mailbox ist.

## 6.3 Verbesserter Lösungsansatz

### 6.3.1 Konzept

Es bietet sich an, in Freenet zu speichern, in welchem Slot die nächste Nachricht eingefügt werden soll. Der SMTP-Server kann dann ab dieser Position beginnen - die oben genannten Nachteile fallen weg. Zum Speichern muss eine fixe Adresse verwendet werden, die auf veränderbare Information zeigt. Es bietet sich ein Date Based Redirect (Abschnitt 4.7.2) an. Dabei verweist der Redirect abhängig von der Zeit auf wechselnde URIs, unter denen rechtzeitig die aktuelle Slotnummer eingefügt wird. Es kann jedoch nicht der Email-SSK-Namensraum verwendet werden, da dessen privater Schlüssel jedem bekannt ist. Ein Angreifer könnte dann für beliebige Zeiträume in der Zukunft die Slotnummer bestimmen und den Account unbrauchbar machen. Deshalb wird ein zweiter SSK-Namensraum verwendet, dessen privater Schlüssel geheim bleibt. Eine fixe Adresse im ersten Namensraum verweist auf die DBR-URI des zweiten. Die Schwachstelle ist dann der Verweis im ersten Namensraum, der beim Erstellen des Accounts erzeugt wird. Allerdings ist die Gefahr einer Manipulation hier deutlich geringer, da zum einen der Key schon existiert (im Gegensatz zu DBR-Zeitslots in der Zukunft) und zum anderen jedesmal abgefragt wird, wenn eine Nachricht an den Account geschickt wird. Dadurch verbreitet sich der Schlüssel, und es ist praktisch unmöglich, ihn zu ersetzen.

### 6.3.2 Implementierung

Auch hier wurde Python mit den beiden Klassen `asyncore` und `asynchat` eingesetzt. Der Java-Client konnte nicht mehr benutzt werden, da er keine DBR-Inserts unterstützt. Statt dessen wurde pyFreenet [32] verwendet. Diese Python-Bibliothek befindet sich noch in der Entwicklung, ist aber schon einsatzfähig.

#### freenetmtad

Beim Start werden die Accountinformationen gelesen. Neben den oben genannten Informationen wird zusätzlich eine Refresh-Zeit gespeichert, die angibt, nach wievielen Sekunden die Slotnummer in Freenet aktualisiert wird. Diese Zeit sollte ungefähr dem Abfrageintervall der Mailbox entsprechen. freenetmtad besteht aus zwei Threads. Der InsertManager-Thread kümmert sich um das rechtzeitige Aktualisieren der Zeitslots für alle Accounts. Im zweiten Thread werden die Server gestartet. Die Threads müssen kommunizieren, da der InsertManager den Slot der zuletzt gelesenen Nachricht kennen muss.

Als Feature wurde neben der normalen Authentisierung über USER/PASS-Kommandos, bei der das Passwort unverschlüsselt übertragen wird, das APOP-Kommando implementiert. Bei diesem Challenge-Response-Verfahren sendet der POP-Server gleich nach dem Verbindungsaufbau einen eindeutigen String (z.B. Hostname, Process-ID und Timestamp). Der Client verknüpft diesen String mit dem Passwort, berechnet den md5-Hash und sendet das Ergebnis an den Server. Dieser kennt das Passwort, kann also die gleiche Berechnung durchführen und bei Übereinstimmung der Hashwerte den Zugriff erlauben.

freenetmtad kann als Daemon-Prozess im Hintergrund laufen (Start mit Option -D: `./freenetmtad.py -D`).

Die genaue Konfiguration von Freenet und FreenetMTA ist im Anhang A beschrieben.

#### add-account

Ähnlich wie bei der ersten Implementierung dient dieses Skript zur Generierung von Accounts. Hier müssen allerdings zwei SSK-Schlüsselpaare generiert und ausserdem der DBR-Redirect angelegt werden. Dies soll an einem Beispiel verdeutlicht werden.

Zunächst werden Benutzername und Passwort abgefragt:

```
telpc4:/home/martin/freemail# ./add-account.py
POP username: martin
POP password: HZ74jrio
```

Für das Refreshintervall gibt es drei vordefinierte Werte:

```
Refresh interval (how often do you retrieve your email?)
This is just an optimization hint for the SMTP server, and won't
limit how often you can check your inbox.
1) 25 minutes
2) 1 hour
3) 1 day
Choose one: 1
```

Nun wird der Account initialisiert. Die beiden SSK-Schlüsselpaare (PubA, PrivA) und (PubB, PrivB) werden generiert, der erste dient als "Email-Account", PrivA wird veröffentlicht. Im Beispiel gilt:

```
PubA = Jw6N4yH0BNaHXJgVgPV-5BXhm1s
PrivA = AJN~ghMk~LfTwb3WI1OnxQNdX9g5
PubB = EX-jZ5BTqFRqPuBTm5X299id3WM
PrivB = ALabJHTIA4Wvn7yId~aMdfGl8us7
```

Die Email-Adresse ist also `AJN~ghMk~LfTwb3WI1OnxQNdX9g5@dummy-url.com`.

In den ersten Namensraum wird nun unter der URI `SSK@AJN~ghMk~LfTwb3WI1OnxQNdX9g5/Inbox` der Date Based Redirect eingefügt. Das Dokument dafür besteht aus einem Dummy-Text und den Metadaten mit dem Redirect:

```
FreeMail
Version
Revision=1
EndPart
Document
DateRedirect.Target=\
    SSK@EX-jZ5BTqFRqPuBTm5X299id3WMPAgM/Inbox/next
DateRedirect.Offset=0
DateRedirect.Increment=5dc
End
```

Bei der Abfrage dieses Dokuments über `SSK@Jw6N4yH0BNaHXJgVgPV-5BXhm1sPAgM/Inbox` geschieht folgendes:

- Auswerten der Metadaten, Erkennen des DBR
- Berechnen des aktuellen Zeitslots T
- Redirect zu URI der Form `SSK@PubB/T-Inbox/next`

Das Skript initalisiert den aktuellen und den nächsten Zeitslot mit dem Wert 2. Im Beispiel:

```
SSK@ALabJHTIA4Wvn7yId~aMdfGl8us7/3e93478c-Inbox/next
SSK@ALabJHTIA4Wvn7yId~aMdfGl8us7/3e934d68-Inbox/next
```

Anschließend wird eine Begrüßungsmail mit den wichtigsten Informationen verschickt. Damit ist der Account funktionsfähig.

### 6.3.3 Diskussion

Im Vergleich zur ersten Implementierung ist diese Version wie erwartet beim Versenden von Mails schneller. Nachteilig wirkt sich die höhere Belastung des Netzwerkes durch die höhere Zahl an eingefügten Schlüssel aus. Das verwendete Freenet-Modul hat noch einige Schwachstellen. So werden einige Fehlermeldungen auf der Konsole ausgegeben, einige Bugs sind vorhanden (z.B. in der URI-Klasse für öffentliche und private SSK-Schlüssel).

## 6.4 Ausblick, Verbesserungen

Die zweite Implementierung wurde unter dem Namen FreenetMTA auf SourceForge veröffentlicht sowie auf der Freenet-Devel-Mailingliste vorgestellt ([33], [34]). Hier noch einige Ideen für Verbesserungen, die unter anderem von der Devel-Liste stammen.

Ein wichtiger Punkt ist die gleichzeitige Nutzung der POP- und SMTP-Server durch mehrere User. Dies ist im Moment nicht möglich, kann aber relativ einfach durch eine Implementierung mit fork() oder Threads realisiert werden. Wünschenswert wäre auch eine intelligentere Fallback-Strategie, falls der Verweis auf den aktuellen Slot nicht abrufbar ist. In diesem Fall versucht der SMTP-Server, die Mail ab Slot 1 zu speichern, das Verhalten entspricht also der ursprünglichen Implementierung. Besser wäre, ältere Slot-Verweise zu suchen.

Wie schon angesprochen werden Mails im Klartext gespeichert. Für eine private Kommunikation sollte PGP-Verschlüsselung eingesetzt werden. Um dies zu vereinfachen, wird der öffentliche PGP-Schlüssel eines Accounts in Freenet gespeichert. Der SMTP-Proxy verschlüsselt dann jede Nachricht, die an den Account geschickt wird, automatisch. Ebenso entschlüsselt der POP-Proxy die Emails. Die Passphrase wird als Passwort für den POP-Account übertragen. Damit wäre auch diese Verschlüsselung transparent für den Nutzer.

Im Vergleich zu normalen POP- und SMTP-Servern entstehen immer noch relativ lange Wartezeiten beim Versenden und Abrufen von Mails. Der POP-Server sollte vergleichbar dem InsertManager versuchen, im Hintergrund die jeweils nächste Mail für die Accounts zu empfangen. Die Intervalle sollten vom Mailaufkommen des Accounts abhängig sein. Bei diesem Verfahren wäre ein Großteil der Mails schon auf dem Server gespeichert, wenn der Nutzer sie abfragt.

Der SMTP-Server sollte Nachrichten annehmen und bestätigen und parallel zu den Sitzungen versuchen, sie zuzustellen. Dann muss der Nutzer nicht auf den erfolgreichen Insert warten. Bei einer gewissen Anzahl von Fehlversuchen wird eine Bounce-Nachricht generiert.

Mit diesen Änderungen wäre FreenetMTA eine einsetzbare Alternative zu normaler

Email-Kommunikation. Der gravierendste Nachteil soll aber noch einmal genannt werden: Nachrichten können aufgrund des Konzeptes von Freenet verlorengehen.

# Anhang A

# Installation und Konfiguration von Freenet und FreenetMTA

## A.1 Freenet

Auf der Website des Freenet-Projekts können der aktuelle Stable-Build oder tägliche Snapshots der Stable- und Unstable-CVS-Branches heruntergeladen werden. Um die aktuellste Version zu bekommen, ist ein Checkout der CVS-Quellen nötig. Hier soll die Installation eines Stable-Builds beschrieben werden.

Zunächst wird eine Java Virtual Machine benötigt. Am stabilsten läuft Freenet auf den Sun-VMs. Bei dieser Arbeit wurde diese Version verwendet:

```
java version "1.4.1_02"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_02-b06)
Java HotSpot(TM) Client VM (build 1.4.1_02-b06, mixed mode)
```

Nun wird die aktuelle Freenet-Version, in diesem Fall 0.5.1, heruntergeladen, entpackt und gestartet.

```
martin@telpc4:~/fn1$ tar xfz freenet-0.5.1.tar.gz
martin@telpc4:~/fn1$ cd freenet
martin@telpc4:~/fn1/freenet$ sh start-freenet.sh
```

Es werden einige Fragen zur Konfiguration gestellt, die alle mit der Default-Einstellung beantwortet werden können.

Sollen mehrere Nodes auf einem Rechner laufen, müssen diese Einstellungen bei jeder Node anders sein:

- `listenPort`
- `clientPort`
- `distribution.port`
- `mainport.port`

## A.2 FreenetMTA

Die aktuelle Version kann unter `http://sourceforge.net/projects/freenetmta` heruntergeladen werden. Es muss Python 2.2 (`http://www.python.org/`) oder eine dazu kompatible Version installiert sein. Dann kann pyFreenet (`http://www.freenet.org.nz/python/pyFreenet/` installiert werden:

```
martin@telpc4:~/fn1$ tar xfz pyFreenet-0.2.tar.gz
martin@telpc4:~/fn1$ cd pyFreenet
martin@telpc4:~/fn1/pyFreenet$ su
Password:
telpc4:/home/martin/fn1/pyFreenet# python2.2 setup.py install
running install
running build
running build_py
not copying freenet.py (output up-to-date)
running install_lib
copying build/lib/freenet.py -> /usr/lib/python2.2/site-packages
byte-compiling /usr/lib/python2.2/site-packages/freenet.py\
    to freenet.pyc
```

Anschließend wird FreenetMTA entpackt. An der Default-Konfiguration `freenetmta.conf` muss nichts geändert werden. Mit `./freenetmtad.py -D` wird der MTA als Daemon gestartet.

Bei Problemen sollte in `freenetmta.conf` die Einstellung `verbosity` auf 4 gesetzt und dann das Logfile überprüft werden.

# Literaturverzeichnis

Alle Links wurden am 13. Mai 2003 zum letzten Mal abgerufen. Wenn eine Seite nicht verfügbar ist, kann auf `http://web.archive.org/` oder `http://google.com/` danach gesucht werden. Am Ende jedes Eintrags ist eine Referenz auf die Zitierstelle zu finden.


[1] Freenet Project. What is Freenet?, 2002.
`http://web.archive.org/web/20020202085612/http://freenetproject.org/cgi-bin/twiki/view/Main/WhatIs`. 1

[2] ACLU. Echelonwatch, 2002.
`http://www.echelonwatch.org/`. 2

[3] Federal Bureau of Investigation. Carnivore Diagnostics Tool, 2000.
`http://www.fbi.gov/hq/lab/carnivore/carnivore.htm`. 2

[4] Heise Newsticker. US-Verteidigungsministerium plant weltweite Internet-Überwachung, 2002.
`http://www.heise.de/newsticker/data/anw-12.11.02-008/`. 2

[5] Holger Bleich, Jörg Heidrich. Ach wie gut, dass niemand weiß..., 2002.
`http://www.ct.heise.de/ct/02/19/124/default.shtml`. 2

[6] Erik Möller. Treffen unter Gleichen oder die Zukunft des Internet, Teil II, 2001.
`http://www.heise.de/tp/deutsch/inhalt/te/7051/3.html`. 2

[7] WIPO. Intellectual Property Protection Treaties, 1996.
`http://www.wipo.int/treaties/ip/wct/index.html`. 2

[8] Axel Metzger. Der Euro-DMCA kommt!, 2001.
`http://www.ifross.de/ifross_html/art17.html`. 2

[9] Europäische Kommision. Kommission begrüßt Richtlinie zum Urheberrecht in der Informationsgesellschaft.
`http://europa.eu.int/comm/internal_market/de/intprop/news/copyright.htm`. 2

[10] Regierung der Bundesrepublik Deutschland. Regierungsentwurf eines Gesetzes zur Regelung des Urheberrechts in der Informationsgesellschaft, 2002. http://www.bmj.bund.de/ger/themen/urheberrecht_und_patente/10000595/. 3

[11] Armin Medosch. Der DMCA muss fallen, 2002. http://www.heise.de/tp/deutsch/inhalt/te/11721/1.html. 3

[12] Declan McCullagh. Security warning draws DMCA threat , 2002. http://news.com.com/2100-1023-947325.html. 3

[13] Dave Roos. Should Google censor itself?, 2002. http://tinyurl.com/bjii. 3

[14] Eric S. Raymond. Jargon File, 2002. http://www.elsewhere.org/jargon/html/entry/slashdot-effect.html. 5

[15] Akamai. Akamai EdgeSuite, 2002. http://www.akamai.com/en/html/services/edgesuite.html. 6

[16] Hannes Federrath. JAP, 2002. http://anon.inf.tu-dresden.de/. 6

[17] Oliver Berthold, Hannes Federrath, Stefan Köpsell. Web MIXes: A system for anonymous and unobservable Internet access, 2001. 6

[18] David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms, 1981. Communications of the ACM, v. 24, n. 2, pp. 84-88. 6

[19] I. Goldberg, D. Wagner. TAZ Servers and the Rewebber Network: Enabling Anonymous Publishing on the World Wide Web, 1998. 7, 8

[20] M. Reiter, A. Rubin. Anonymity loves company: Anonymous Web transactions with Crowds, 1999. 8

[21] KaZaA. KaZaA Homepage, 2002. http://www.kazaa.com/. 8

[22] Kasper Larsen. Anti-pirates hit Danish P2P users with huge bills, 2002. http://www.theregister.co.uk/content/6/28286.html. 9

[23] eDonkey 2000, 2003. http://www.edonkey2000.com/. 9

[24] A. Shamir. How to share a secret., 1979. Communications of the ACM, 22:612-613. 10

[25] Ian Clarke, Oskar Sandberg, Brandon Wiley und Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46+, 2001.
`http://citeseer.nj.nec.com/clarke00freenet.html`. 16

[26] Amr Z. Kronfol. FASD: A Fault-tolerant, Adaptive, Scalable, Distributed Search Engine, 2002.
`http://citeseer.nj.nec.com/571354.html`. 16

[27] Adam Langley. The Freenet Protocol.
`http://www.firenze.linux.it/~marcoc/index.php?page=protocol`. 22

[28] Clarke et al. Protecting Free Expression Online with Freenet, 2002. IEEE Internet Computing 6(1), 40-49 (2002). 24

[29] DCMI. Dublin Core Metadata Initiative, 2003.
`http://www.dublincore.org/`. 25

[30] J. Myers, M. Rose. Post Office Protocol - Version 3, 1996.
`ftp://ftp.rfc-editor.org/in-notes/rfc1939.txt`. 35

[31] J. Klensin. Simple Mail Transfer Protocol, 2001.
`ftp://ftp.rfc-editor.org/in-notes/rfc2821.txt`. 35

[32] David McNab. pyFreenet - Easy Python API for Freenet, 2003.
`http://freenet.org.nz/python/pyFreenet/`. 38

[33] Martin Richtarsky. FreenetMTA, 2003.
`http://sourceforge.net/projects/freenetmta`. 41

[34] Martin Richtarsky. Announce: FreeMail 0.01, 2003.
`http://hawk.freenetproject.org:8080/pipermail/devl/2003-April/005054.html`. 41