

Message-Routing in GUNet

Bearbeiter: Hao Qin (qin@in.tum.de)
Betreuer: Vlad Manilici (Vman@net.in.tum.de)

Hauptseminar Internetrouting
Technische Universität München
WS 2004/2005 (Version 29.01.05)

Abstract

“GUNet ist ein Framework für sicheres Peer-to-Peer Networking, das keine zentralisierten oder anderweitig vertrauten Dienste verwendet. Eine erste Implementation, die auf der Netzwerk-Schicht aufsetzt, erlaubt anonymes zensur-resistentes Filesharing. GUNet benutzt ein einfaches, Überschuss-basierendes Modell, um Ressourcen bereitzustellen. Teilnehmer des GUNet überwachen das Verhalten der anderen in Bezug auf Ressourcengebrauch; Teilnehmer, die zum Netzwerk beitragen werden mit besseren Dienstleistungen belohnt.”

Das GNUNet Projekt

1. Einleitung

1.1 Was ist GUNet?

GUNet ist ein Kryptografie-Projekt von den Studenten an der Purdue Universität. Es ist unter der GNU General Public License veröffentlicht. Es wurde versucht die Vorteile verschiedener Systeme wie Napster, Gnutella und Freenet¹ zu verknüpfen. Es unterscheidet sich von den anderen vorher genannten Filesharing-Programmen aber ausser Freenet dadurch, dass es anonym

¹ <http://www.freenetproject.org/>

ist. GUNet gehört zu den P2P² Kommunikationssystemen[2]. Daten können zwischen zwei einzelnen Netzwerkknoten übertragen werden, ohne Unterstützung von einem zentralen Server. Die Knoten brauchen keinen Dritten um sich einander zu identifizieren und eine Verbindung herzustellen. Sie können gleichzeitig als Client und Server funktionieren.

Jeder Netzwerkknoten bildet sich dabei eine "Meinung" über alle anderen Netzwerkknoten, mit denen er Kontakt hat. Auf Basis dieser Einschätzung entscheiden die Knoten, welche Anfragen sie bearbeiten, sofern Kapazität vorhanden ist. Dieses Modell wird sozial-ökonomisch genannt.

GUNet besteht aus den folgenden Komponenten:

- GUNet Server: gnutd;
- GUI: gnut-gtk (Version >= 1.2);
- Kommandozeilen- Tools: gnut-search

Derzeit befindet sich GUNet in der Betaphase. Die aktuelle Version ist 0.7.0, die unter Linux(i386) sowie FreeBSD, OpenBSD, NetBSD und Solaris aber nicht unter Windows lauffähig ist.

1.2 Vergleich zwischen verschiedene Systeme

Netzwerk	GUNet	Napster	Gnutella	Freenet
Verteilte Suchen	ja	nein	ja	ja
Herunterladen von multiplen Quellen	ja	nein	ja	nein
Ökonomie	ja	nein	nein	nein
Anonymität	ja	nein	nein	ja
Plattform	nur Un*x-ähnliche	jede	jede	JVM
Transportprotokolle	UDP, TCP, SMTP, HTTP	TCP	TCP	TCP
Anfrageformat (UI)	Schlüsselwörter oder AFS URIs	Schlüsselwörter	Dateiname, SHA	geheimer Schlüssel
Routing	dynamisch (indirekt, direkt)	immer direkt	immer direkt	immer indirekt
Lizenz	GPL	GPL (knapster)	GPL (gtk-gnutella)	GPL

Tabelle 1-1: Vergleich zwischen verschiedenen Systemen
<http://www.ovmj.org/GUNet/faq.php3#compare>[3]

² peer to peer

2. Kommunikation zwischen Knoten:

2.1 Nachrichtentypen von GUNet

Die folgende Nachrichtentypen müssen in GUNet-Systemen von allen Netzwerkknoten unterstützt werden:

- HELO
- SKEY
- PING
- PONG
- TIMESTAMP
- SEQUENCE
- NOISE
- HANGUP
- CAPABILITY

Netzwerkknoten in GUNet werden durch ihre Public Keys identifiziert. Um Kontakt miteinander aufnehmen zu können, müssen die Netzwerkknoten nicht nur den Public Key von den anderen Netzwerkknoten, sondern auch deren derzeitige IP-Adressen und Ports kennen. Zusätzlich wird ein Zeitstempel gesendet, und alles wird mit dem Private Key signiert. Bei Eintritt in das GUNet-System sendet jeder Netzwerkknoten diese Daten an alle bekannten Peers.

Dieser Informationsaustausch geschieht in dem HELO-Nachricht. Aus Sicherheitsgründen muss diese Nachricht verschlüsselt werden. Eine HELO-Nachricht wird zu allen neuen Partnern gleich gesendet.

2.1.1 Eintritt in das GUNet System

Das Verfahren mit dem sich ein Knoten in das GUNet System einträgt, funktioniert wie folgt: Es wird ein HELO gesendet zu allen bekannten Netzwerkknoten. Wenn ein Knoten ein HELO empfangen hat, wird er mit einem PING antworten. Mit diesem PING kann der neuer Teilnehmer wissen, dass ein Netzwerkknoten seine Anfrage erhalten hat. Alle PINGs werden mit PONGs bestätigt. Nach diesem Ablauf ist der neue Netzwerkknoten dem GUNet erfolgreich beigetreten.

Die Struktur einer HELO-Nachricht ist wie folgt: A ist der neue Knoten, B ist ein bekannter GUNet Knoten[5]:

1. $A \rightarrow B: E_B(A, S_A(@A, t))$
Dabei ist A der Public Key von A, S_A ist die Signatur von A, @A ist die aktuelle Netzwerkadresse von A, und t ist die Lebensdauer der Adresse von A. E_B ist die Verschlüsselung mit dem Public Key von B.
2. B überprüft die Signatur von A. Wenn sie richtig ist, dann schreibt er die erhaltenen Daten in eine Liste: die Netzwerkadresse, Zeitstempel und Signatur von A.
3. B kann sein Wissen über A einem anderen Knoten C weitergeben:
 $B \rightarrow C: E_C(A, S_A(@A, t))$. Das darf nur im Zeitintervall t passieren.

2.1.2 Initiieren einer Verbindung

Um eine Verbindung zu initiieren, kann jeder Netzwerkknoten anderen Netzwerkknoten ein SKEY und ein PING nach Anforderung senden. Der Empfänger einer SKEY-Nachricht wird diese mit einem PING, einem SKEY und einem PONG bestätigen. Aus Sicherheitsgründen muss der Tunnel mit einem Session Key verschlüsselt werden. Der PING wird mit einem verschlüsselten PONG bestätigt. Wenn keine PING-Nachricht erhalten wurde, soll der Initiator das SKEY noch einmal senden. Ebenfalls, wenn kein PONG erhalten wurde, soll das PING noch einmal gesendet werden. Diese Sequenz ist ähnlich zum dreifachen Handshake von TCP.

Die Session Keys können von jedem Knoten getauscht werden, unabhängig ob er die Kommunikation initiiert hat[5].

1. $B \rightarrow A: B, E_A(SK), t, S_B(H(E_A(SK), t))$
Dabei ist SK die Session Key von B, H ist Hash
2. $A \rightarrow B: E_{SK}(H(SK))$
Empfangsbestätigung

Für die Datenübertragung muss SK sowol in die aktuelle Beziehungstabelle von A, als auch von B abgespeichert werden[5].

1. $A \rightarrow B: E_{SK}(M, t)$.
2. $B \rightarrow A: E_{SK}(M, t)$.

Jetzt können anderen Nachrichten zwischen beiden Netzwerkknoten ausgetauscht werden. Die Attribute TIMESTAMP und SEQUENCE dienen zum Verhindern von Replay-Attacken.

2.1.3 Abbruch einer Verbindung

Wenn entweder ein Netzwerkknoten die Verbindung abbrechen muss, oder wenn seit langer Zeit (c.a. 15 Minuten) keine Daten übertragen worden sind, wird die Verbindung mit einer HANGUP-Nachricht abgebrochen. HANGUP benötigt keine Antwort. Die Session Keys verlieren ihre Gültigkeit.

2.1.4 Fazit

Die Anzahl von Nachrichten in einem Paket ist nur von der MTU des Transportlayers begrenzt.

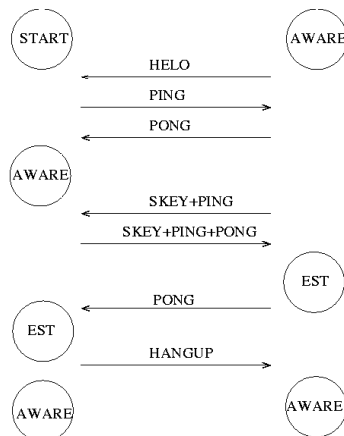


Bild 2-1: Vorgang von Nachrichten[3] Dabei ist EST „Established“

3. Das Protokoll

Der Datenübertragungsmechanismus von GNUet ist ähnlich wie SSH. Session-Keys werden mittels RSA ausgetauscht. Alle Knoten sind durch ihre Public Keys identifiziert. UDP kann anstatt von TCP von dem niedrigen Layer benutzt, aber es kann zu Verlust von Datenpaketen führen. Wenn z.B. eine Search-Query zu 10 Knoten gesendet werden muss, ist es unwichtig, ob alle 10 Knoten die Query bekommen haben oder nicht. Wesentlich ist, dass mindestens ein Knoten antwortet. Sonst wird der Knoten eine neue Query ins Netz senden. Bei GNUet überwacht der höhere Layer (Application Layer) den Verlust von Datenpaketen[5].

3.1 Dynamische IPs

Netznutzer, die Dialup benutzen, wechseln oft die IP-Adresse. Diese ist genauso wichtig wie der Port und der Public Key für die Identifizierung. Sie wird mit einem Zeitstempel versetzt, um Angriffe zu vermeiden. Abgelaufene Identifizierungsdaten können nicht mehr verwendet werden.

3.2 UDP

TCP hat den Vorteil, dass es zuverlässig ist, da nach der Übertragung Veränderungen in der Paketreihenfolge anhand der Segmentnummer rückgängig gemacht werden können. Ein weiteres Protokoll, das auf IP aufbaut, ist das User Datagram Protocol (UDP), das aber in Sachen Sicherheit dem TCP um einiges nachsteht. UDP ist ein verbindungsloses Protokoll. Es nutzt das Internet-Protokoll (IP) ähnlich wie TCP zur Übertragung von Daten, kümmert sich aber nicht darum, ob die Daten auch ihr Ziel erreichen. Der Vorteil von UDP ist eindeutig seine Schnelligkeit im Gegensatz zu TCP. UDP wird immer häufiger für Real-time-Anwendungen genutzt. Der Nachteil ist aber, dass öfters ein Datenpaket verloren geht. Das kann passieren z.B. wenn ein Knoten eine neue Session benutzt, aber zweite die ältere Session noch benutzt. Bei Datenübertragung ist das Datenpaket von dem zweiten Knoten nicht akzeptiert. Beide Knoten müssen dieselbe Session benutzen.

4. Routing

Immer wenn ein Netzwerkknoten eine Anfrage empfängt, muss er ein paar Entscheidungen treffen, was er damit machen soll: Verwerfen oder Weitergeben.

4.1 Nachrichten verwerfen

Wann müssen wir eine Query verwerfen? Die Leistung eines Computers ist beschränkt. Wenn keine Kapazität frei ist, können wir den Computer nicht mehr benutzen. Deswegen muss man Querys verwerfen, wenn die „local load“ zu hoch ist. Für alle Querys gibt es Zeitstempel, die die „time to live“ zeigen. Wenn eine Query abgelaufen ist, muss diese auch verworfen werden. Solche abgelaufenen Querys sind vermutlich schon lange Zeit im Netzwerk geblieben, oder es sind Angriffe. Außer diesen zwei Fällen gibt es noch die Möglichkeit, dass die Querys schon von irgendjemandem abgearbeitet wurden. Natürlich sollen wir solche Query verwerfen.

Das GAP (GUNet anonymous protocol) benutzt TTLs³ im Millisekundenbereich. Beim Empfang einer Query wird diese in der Routing-Tabelle gesucht. Falls diese Query schon gesehen wurde, und die neuempfangene älter ist, wird diese verworfen. Falls sie neuer ist, wird sie weiterbearbeitet.⁴

Die Anzahl von Weiterleitungen von Daten an andere Knoten ist abhängig von der aktuellern Netzwerklast und der Priorität der Daten:

- $0\% < \text{Networkload} < 50\%$: answer/forward/indirect
- $50\% < \text{Networkload} < 50\% + \text{priority}$: answer/forward/indirect
- $50\% + \text{priority} < \text{Networkload} < (90 + 10 * \text{priority})\%$: answer/forward
- $(90 + 10 * \text{priority})\% < \text{Networkload} < 100\%$: answer

4.2 Nachricht weiterleiten

Außer bei der obengenannten Gelegenheit werden die Querys weiterbearbeitet. Dafür ist ein „local lookup“ nötig. In der Funktion *execQuery()* von *routing.c* wird festgelegt, ob eine Query bearbeitet wird. Wenn ja, dann wird vorerst der Inhalt auf der lokalen Maschine gesucht. Unabhängig davon ob der Inhalt gefunden wird oder nicht, wird die Query geroutet. Potenzielle Antworten werden zurückgeschickt, mit einer zufälligen Verzögerung, wenn die Bandbreite es zulässt. Dann wird diese Query mit verringerter „time to live“ zu Nachbarknoten weitergeleitet. Diese Knoten werden nach einem Algorithmus gewählt⁵. Der Knoten entweder behält die Absenderadresse vom Vorgängerknoten oder ersetzt es mit seiner eigenen Adresse. Im zweiten Fall fügt der Knoten seinem lokal Routing-Tabelle die Query hinzu. Außerdem, falls Bandbreite reichlich vorhanden ist, soll es auch möglich sein zu mehren Netzwerkknoten, die Kostenlos sind, die Query weiterleiten⁶.

Für eine Query, egal ob sie in der Routing Tabelle vorhanden ist oder nicht, werden weitergeleitete Antworten eventuell lokal gespeichert. Aber in der Praxis hängt das von der Größe des lokalen Speichers ab. Meist werden Antworten für Querys, die nicht in der Routing-Tabelle vorhanden sind, verworfen. Warum sollen Daten lokal gespeichert werden, wenn die Query unbekannt ist? Die Netzwerkknoten erhalten oft viele Querys, die dasselbe Ziel haben. Diese Querys müssen nicht alle geroutet werden, wenn die Antworten schon lokal gespeichert sind. So ist es ökonomisch.

4.3 Die Kodierung

Dateien werden innerhalb des Netzwerks durch den 3-fachen Hash des für das Dokument vergebene Schlüsselwortes in Form vom $H(H(H(\text{Keyword})))$ indiziert. Dieses Verfahren wird aus Sicherheitsgründen angewandt, da es verschiedene Sicherheitsprobleme mit der Indexierung nur durch $H(H(\text{Keyword}))$ oder nur durch den einfachen Hash gibt. Bei GUNet ist Datenübertragung

³ time-to-live

⁴ Der entsprechende Quellecode befindet sich in *routing.c*, Funktion *needsForwarding()*.

⁵ Der entsprechende Quellecode findet sich in *querymanager.c* Funktion *forwardQuery()*.

⁶ Der entsprechende Quellecode findet sich in *querymanager.c* Funktion *fillInQuery()*.

immer in 1KB-Blocks.

Es gibt insgesamt drei Typen von Blocks:

- *Data Blocks (DBlock)*: die eigentlichen Daten
- *Indirect Blocks (IBlock)*: werden gebraucht, um DBlocks anzufordern. Ein IBlock wird gebraucht um die Nachfrage nach einem Inhalt zu initiieren.
- *Root Blocks (RBlock)*: RBlock ist IBlock + Keywords. Ein IBlock ist ausreichend um alle Datenblöcke in eine Datei zu referenzieren. Damit ist es möglich nach Inhalten zu suchen.

4.4 Suchen und Runterladen

Ein Inhalt wird in GUNet in zwei Schritte geholt:

4.4.1 Entdeckung des RBlocks

Es wird zuerst ein Root-Block gefunden, der Indirect-Blocks indiziert. RBlocks werden damit entdeckt, dass die Suchanfragen enthalten eine Liste von dreifach hashed Stichwörtern $H(H(H(Kj)))$. Ein Knoten, der die Frage herausgegeben hat, ist in der Lage, das RBlock mit $H(Kj)$ zu entschlüsseln.

4.4.2 Herunterladung

Nachdem ein RBlock gefunden wird, kann der Knoten die entsprechende Dateien herunterladen. Der Knoten benutzt zuerst die Query, das im RBlock umfasst wird, um der Wurzel - IBlock zu holen und zu entschlüsseln. Diese Indirect-Blocks indizieren weitere Data-Blocks, woraus eine Datei besteht.

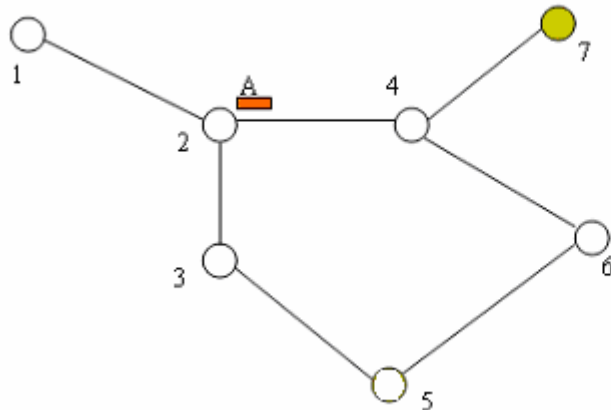
So gibt es zwei Arten von Querys: Such-Query und Download-Query. Beide Arten von Querys enthalten drei zusätzliche Werte, die verwendet werden, um die Frage zu verarbeiten: eine Antwortadresse, eine Priorität und time-to-live (TTL).

4.5 Flooding Algorithmen

Routing basiert auf einem Flooding Algorithmus. Nach diesem Algorithmus wird ein Paket über mehrere ausgehenden Leitung gesendet. Die Nachteile sind: viele Duplikate und eine hohe Netzlast. Die Vorteile sind: Sehr robust, selbst bei Ausfall mehrerer Router und Optimal, weil der kürzeste Weg immer dabei ist.

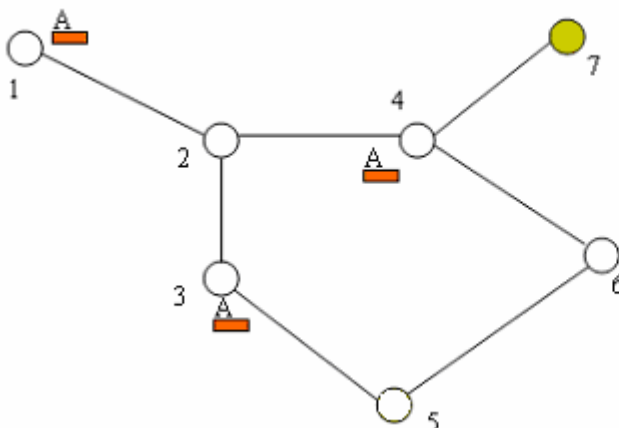
GUNet unterscheidet sich vom reinen Floodingdaruech, dass die Anzahl der gefloodeten Knoten begrenzt wird. Die Auswahl der Knoten ist kritisch, wie auch die Tatsache der Begrenzung. Der Vorteil gilt dadurch aber schon nicht mehr, da der kürzeste Weg nicht mehr dabei ist. Weiterhin sind natürlich Umwege gewünscht wegen Anonymität, allerdings nur in Grenzen. Es wird aus der Routing-Tabelle der Knoten gewählt, der am ehesten den gesuchten Schlüssel in seinem Data Store verwaltet. An diesen Knoten wird dann eine Data Request-Nachricht versendet.

Die folgenden Diagramme zeigen die Funktionsweise:



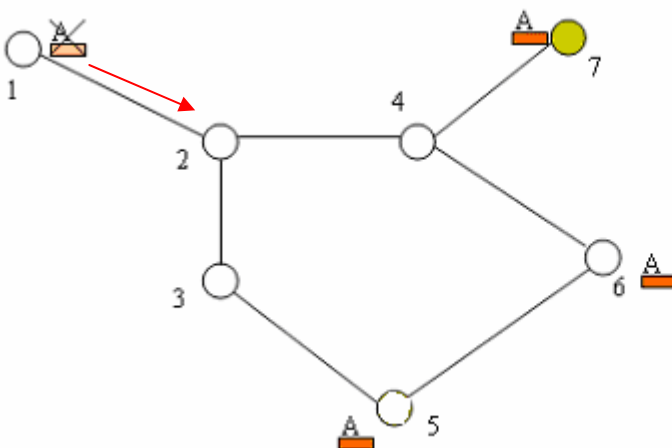
Knoten 2 sucht Datei "A", aber er weiss nicht, wo "A" ist.

Beim Empfang einer Nachricht wird das Feld TTL dekrementiert und die Nachricht verarbeitet. Werden die Daten nicht durch den lokalen Data Store verwaltet wird die Data Request-Nachricht an den nächsten geeigneten Knoten weitergeleitet.



Knoten 2 initiiert "A" und sendet die Query zu allen Nachbarn.

Kennt der Knoten keinen weiteren Knoten, antwortet er an den anfragenden Knoten mit einer Request Failed-Nachricht. Der anfragende Knoten kann dann die Request-Nachricht an einen anderen Knoten weiterleiten.



Nachbarn leiten die Query weiter.

Request Faild

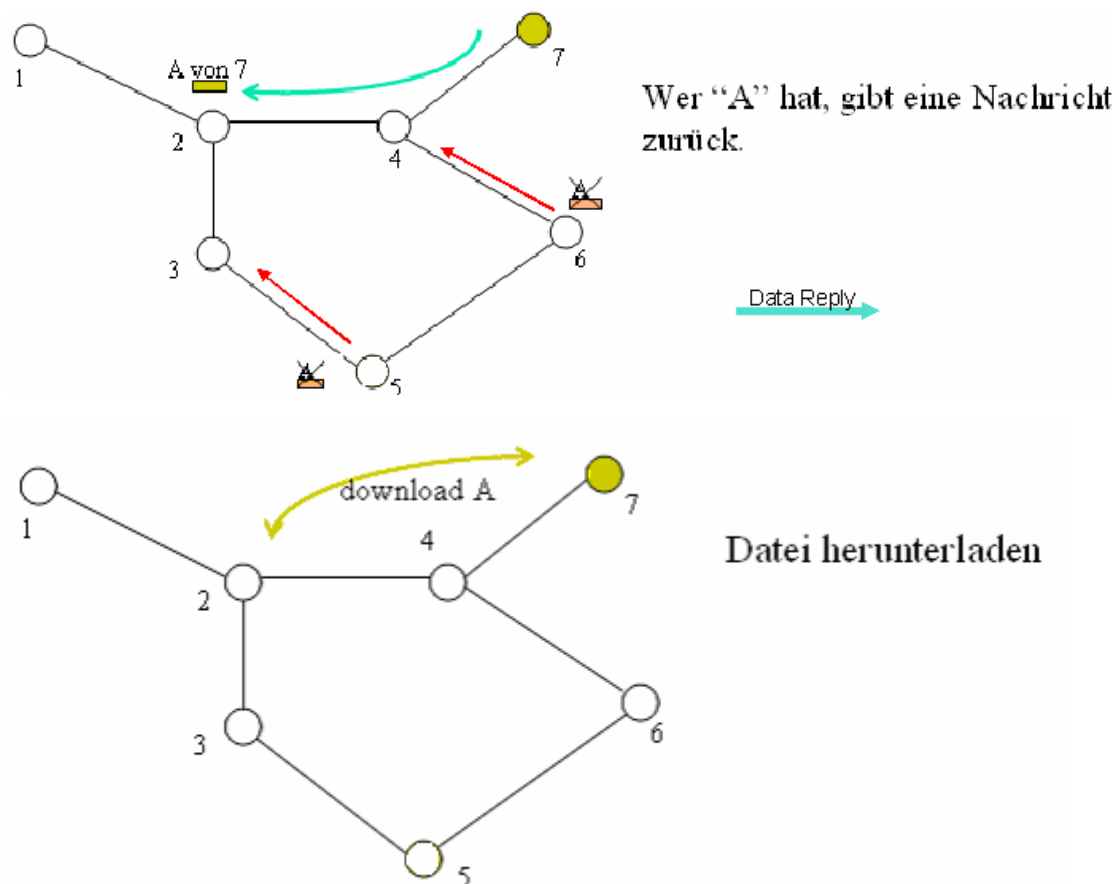


Bild 4-1: Flooding-Algorithmus

Wenn eine Query mehrmals erhalten wird, wird sie nur einmal betrachtet.

4.6 Begrenzung durch TTL

Eine gesendete Anfrage besitzt zusätzlich noch eine Priorität und eine „time to live“. Eine zu hoch erscheinende TTL wird automatisch heruntergesetzt. Bei Flooding-Algorithmus hängt die Suchweite und die Suchtiefe von der TTL ab. Wenn eine Datei gefunden wird, oder der Wert von TTL auf „0“ sinkt, dann wird die Suche beendet.

- Bei Initiierung einer Query wird eine positive TTL gewählt.
- Bei Weiterleitung wird die TTL x ($x > 1$) verringert.
- Es wird weitergeleitet bis Dateien gefunden sind oder die $TTL = 0$ ist.

Der Absender, der angefragte Hash und die TTL werden in einer lokalen Query-Tabelle notiert und falls sie dort schon bekannt sind erfolgt keine Weiterleitung. Abhängig von der Priorität der Anfrage und der Last auf dem Server werden zufällig n bekannte Hosts ausgewählt, an die die Anfrage mit einer Verringerung der TTL und der Priorität ($P=P/n+1$) weitergesendet wird[5].

5. Fazit

GNUet ist ein Peer-to-Peer (P2P) System das Anonymität anstrebt. Einzelne Nachrichten werden immer verschlüsselt weitergeleitet, sodass Antworten denselben Weg nehmen müssen. Das Routing basiert auf begrenzten Flooding Algorithmen. Sie werden zweifach beschränkt, einmal in die Breite durch die Auswahl von weiteren Router und einmal in die Länge durch das Time To Live (TTL).

Literatur

[1] Doxygen von GNUet:

<http://www.ovmj.org/GNUet/doxygen/html/index.html>

[2] Offizielle GNUet-Seite:

<http://www.ovmj.org/GNUet/index.php3?xlang=German>

[3] Dokumentation von GNUet:

<http://www.ovmj.org/GNUet/documentation.php3?xlang=German>

[4] Sam Joachim: *Ähnliche Ziele, Verschiedene Wege: Freehaven & GNUet*

http://www.wiwi.hu-berlin.de/~fis/p2pe/paper_S_Joachim.pdf

[5] Krista Bennett, Christian Grothoff, Tzvetan Horozov, Ioana Patrascu und Tiberius Stef:

The GNUet Whitepaper

<http://www.ovmj.org/GNUet/download/main.pdf>

[6] Dennis Kügler: *An Analysis of GNUet and the Implications for Anonymous, Censorship-Resistant Networks*

http://www.ovmj.org/GNUet/papers/GNUet_pet.pdf