Freenet

Florian Baumann
Betreuer: Dr. Holger Kinkelin, Marcel von Maltitz
Seminar Future Internet WS2014
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: derflob@mytum.de

KURZFASSUNG

Das Internet ist aufgrund seiner Architektur und heutigen Implementierung anfällig gegenüber Zensur durch mächtige Parteien wie beispielsweise Regierungen. Zudem bietet es kaum Möglichkeiten für das anonyme Speichern und Abrufen von Daten. *Freenet* hingegen bietet eine anonyme, zensurresistente Peer-to-Peer-Infrastruktur zum Speichern und Aufrufen von Daten. Bei der Konzeption der von *Freenet* verwendeten Algorithmen wurde Wert auf eine effiziente Funktionsweise gelegt. Dazu wurde sich an sozialen Strukturen unserer Gesellschaft orientiert, die sich auch im *Freenet* widerspiegeln.

Schlüsselworte

Anonymität, Peer-to-Peer-Netzwerk, Verteilte Datenspeicherung, Zensurresistenz

1. EINLEITUNG

Im Internet gespeicherte Daten können zensiert werden. Dies ist darin begründet, dass die Herkunft von Daten immer einem Server zuordenbar ist. So können beispielsweise Regierungen durch Beschlagnahmung entsprechender Server die darauf enthaltenen Daten unerreichbar machen und damit zensieren. Es sind außerdem noch andere Arten von Attacken, beispielsweise auf das Domain Name System (DNS) denkbar, bei denen ein Angreifer, der globalen Zugriff und Manipulationsmöglichkeiten im Netzwerk hat, DNS-Anfragen umleitet. Da Kommunikation auch heute noch zu einem Großteil unverschlüsselt stattfindet, ist es solchen Angreifern auch möglich die Konsumenten und Produzenten von Information zu überwachen und deanonymisieren. Um dennoch einen anonymen und zensurfreien, beziehungsweise -resistenten Informationsaustausch zu gewährleisten, sind zusätzliche Maßnahmen vonnöten, die auf den bestehenden Netzwerken aufbauen.

So existiert eine Vielzahl von Anonymisierungsanwendungen, wie etwa *TOR* (The Onion Router)[1], diverse VPN-Dienste, *I2P*[2], *GNUnet*[3] oder *Freenet*[4]. In dieser Arbeit wird *Freenet* betrachtet, welches ein System zum anonymen, verteilten Speichern und Aufrufen von Daten ist. Es wurde 1999 von *Ian Clarke* vorgestellt[1] und wird seitdem ständig erweitert und verbessert. Zur Wahrung der Anonymität der Benutzer im *Freenet* findet einerseits eine Transportverschlüsselung zwischen den einzelnen Peers statt, so dass es einem Eavesdropper nicht möglich ist die Kommunikation passiv abzuhören. Andererseits wird die Herkunft von Anfragen und Daten als Teil des Routing-Protokolls verschleiert, sodass die Quelle nicht eindeutig identifiziert werden kann. Mit der Version 0.7 aus dem Jahr 2005 wurde das verwendete Routing-Protokoll grundlegend verändert, um das Auffinden von Daten effizienter zu gestalten. Gleichzeitig wurde es möglich, *Freenet* in einem *Darknet*-Modus zu verwenden, um die Sicherheit und Anonymität weiter zu erhöhen.

Im Kapitel 2 wird darauf eingegangen, welche Funktionen und Sicherheitsversprechen *Freenet* bietet, sowie worum es sich bei einem *Darknet* handelt und wie es in *Freenet* integriert ist. Kapitel 3 erläutert die verschiedenen Schlüsseltypen, die beim Routing zum Einsatz kommen und gibt einen Überblick, wie Verschlüsselung eingesetzt wird um Benutzer zu schützen und die Sicherheitsversprechen aus Kapitel 2 umzusetzen. Das Kapitel 4 beschreibt die Algorithmen, die für das Routing innerhalb des *Freenet* benutzt werden. In Kapitel 5 wird ein kompakter Überblick gegeben, welche zusätzlichen Diensttypen durch Plugins oder Third-Party Programme implementiert wurden und auf den Konzepten von *Freenet* aufbauen. Das Kapitel 6 zeigt einige Probleme auf, unter denen *Freenet* noch leidet und präsentiert etwaige Lösungsüberlegungen. Abschließend wird in Kapitel 7 ein Überblick über verwandte Arbeiten gegeben, bevor in Kapitel 8 eine Zusammenfassung folgt.

2. ZIELE UND EIGENSCHAFTEN

Freenet implementiert eine Art verteiltes Peer-to-Peer-Dateisystem als Overlay-Netzwerk auf dem heutigen Internet. In der ersten Veröffentlichung zum *Freenet* wurden die folgenden Eigenschaften definiert, die dieses Overlay-Netzwerk haben soll[1]:

Dezentralisierung Um keinen Angriffspunkt für (Distributed) Denial of Service Attacken (DDoS) zu bieten, die zum Ziel haben können Information zu unterdrücken, soll das System komplett ohne zentrale Verwaltungs- oder Kontrollmechanismen auskommen. Dies steht zum Beispiel im Gegensatz zu *Napster*, welches zwar eine Peer-to-Peer-Anwendung war, jedoch einen zentralen Server besaß, welcher für die Koordinierung und Vermittlung der Downloads zuständig war.

[1] `https://www.torproject.org/`
[2] `https://geti2p.net/`
[3] `https://gnunet.org/`
[4] `https://www.freenetproject.org/`

doi: 10.2313/NET-2015-03-1_02

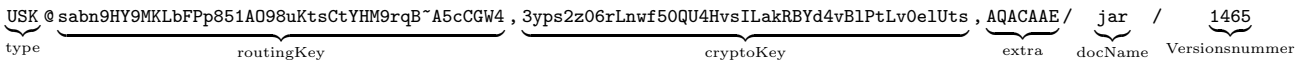

Abbildung 1: Beispiel für einen Updateable Subspace Key, URI für *Freenet* -Updater

Anonymität Sowohl Entitäten, die Information anbieten, als auch jene, die diese aufrufen sollen anonym bleiben.

Robustheit Software- und Hardwarefehler und der damit verbundene Ausfall von Peers sollen das System nicht stark beeinflussen können.

Adaptivität Aufgrund der Natur von P2P-Netzwerken, bei denen Peers ständig verschwinden und Neue beitreten, soll *Freenet* in der Lage sein, sich schnell und effektiv an die neuen Gegebenheiten anpassen zu können.

Performance Die Geschwindigkeit soll bestehenden Informationssystemen gleichen.

Diese Eigenschaften ergeben sich unter anderem aus dem Fokus auf eine Widerstandsfähigkeit gegen Zensur durch Regierungen, die direkten Zugriff und Manipulationsmöglichkeiten auf die bestehende Infrastruktur haben können.

Ein weiteres Ziel ist eine **Plausible Deniability** darüber, was auf dem eigenen Rechner gespeichert ist, um eine Verfolgung von Teilnehmern zu erschweren oder zu verhinden.

Eine wichtige Eigenschaft im Zusammenhang mit Datenspeichern, eine **Lebenszeitgarantie** von Dokumenten, verspricht *Freenet* explizit nicht. Stattdessen wird argumentiert, dass Informationen, die von Interesse sind, deshalb „überleben“, weil sie regelmäßig aufgerufen werden.

Daten werden zum Routing und Auffinden mit einem global eindeutigen Schlüssel versehen. Es sind mehrere Schlüsseltypen in Verwendung, die in Kapitel 3 genauer betrachtet werden. Jeder Peer besitzt zudem eine *Location*, die durch eine Fließkommazahl im Bereich von 0 bis 1 ausgedrückt wird. Sie hat keinen Zusammenhang mit der geographischen Position des Peers, sondern wird beim Beitreten in das Netzwerk zufällig erstellt und dient nur dem Routing. Die Locations der Nachbarn, mit denen ein Node verbunden ist, werden lokal in einer Routing-Tabelle gespeichert.

Freenet ist zudem mit Version 0.7 von einem TCP- auf einen reinen UDP-Transport umgestiegen. Dabei wurde jedoch nachträglich, aufbauend auf UDP, eine Congestion Control und eine zuverlässige Packetvermittlung implementiert.

2.1 Darknet & Opennet

Mit der Veröffentlichung der Version 0.7 von *Freenet* hat das Konzept des *Darknet* Einzug gehalten. Wird *Freenet* im Darknet-Modus betrieben, werden nur Verbindungen zu solchen Peers hergestellt, die man manuell als „Freunde“ eingetragen hat. Dazu müssen beide Parteien ihre *Node References* austauschen, bei welchen es sich um diverse Parameter – wie etwa IP-Adresse oder kryptographische Parameter – handelt, die notwendig sind, eine sichere, verschlüsselte Verbindung aufzubauen. So entsteht nach und nach ein vertrauenswürdiges Netzwerk, das reale, soziale Netzwerke widerspiegelt. Da beim Routing nur Anfragen an direkt verbundene Nachbarn geschickt werden, denen man vertraut die eigene Anonymität nicht zu kompromittieren, erhöht sich die Anonymität und Sicherheit gegenüber böswilligen Angreifern.

Da ein Darknet jedoch eine erhebliche Einstiegshürde darstellt, bietet *Freenet* weiterhin einen Opennet-Modus an. Dabei werden Verbindungen zu Dritten, die nicht explizit vom Benutzer hinzugefügt wurden, sondern im Laufe der Zeit entdeckt wurden, erlaubt und hergestellt.

Beide Modi lassen sich in Form eines hybriden Modus kombinieren, bei dem Peers aus dem Darknet bevorzugt behandelt werden und nur dann Verbindungen in das Opennet hergestellt werden, falls das Limit für die maximale Anzahl an Verbindungen noch nicht erreicht wurde.

3. PROTOKOLLGRUNDLAGEN

3.1 Schlüsseltypen: CHK, SSK, USK

Es werden hauptsächlich drei verschiedene Typen von Schlüsseln verwendet um Daten aufzufinden. *Content Hash Keys* (CHK), *Signed Subspace Keys* (SSK) und *Updateable Subspace Keys* (USK). Abbildung 1 stellt beispielhaft einen USK mit seinen einzelnen Bestandteilen dar. Bis auf das Versionsnummernfeld, das für USKs einzigartig ist, bestehen alle Schlüsseltypen aus den gleichen Feldern, die in Abhängigkeit des Typs leicht unterschiedliche Bedeutungen haben. Dabei sind die Felder *routingKey*, *cryptoKey* und *extra* Base64 kodiert, mit „-“ und „~“ anstelle der sonst üblichen Zeichen „/“ und „+“ um die URIs als URLs in einem Browser verwenden zu können. Das extra-Feld gibt zusätzliche Parameter, wie etwa den Verschlüsselungsalgorithmus an. *docName* ist der menschenlesbare Dateiname der hochgeladenen Datei.

Der grundlegendste Schlüsseltyp ist der CHK. Er ist vor allem dafür geeignet, große, statische Daten in das *Freenet* einzufügen. Der routingKey ist dabei der SHA-256-Hash über die eingebrachten Daten. Dadurch entsteht für jedes Datum ein global eindeutiger Identifier, da Hashkollisionen, gezielt oder zufällig, als überaus unwahrscheinlich gelten.

SSKs bieten mit Hilfe asymmetrischer Kryptographie eine Möglichkeit private Namespaces im *Freenet* zu schaffen. Nur der Besitzer des privaten Schlüsselteils eines Subspace ist in der Lage Daten hinzuzufügen. Dazu wird zum Erstellen eines neuen Subspaces ein zufälliges Schlüsselpaar generiert. Als routingKey wird der Hash des öffentlichen Schlüsselteils verwendet, der damit gleichzeitig den Subspace identifiziert. Soll nun eine Datei zu einem Subspace hinzugefügt werden, muss der Identifier dieser Daten berechnet werden. Dazu wird der docName gehasht, mit dem Hash des öffentlichen Schlüssels XOR-verknüpft und erneut gehasht. Bevor ein Subspace-Verwalter nun Daten in das *Freenet* hochlädt, fügt er den Daten noch eine Signatur, die nur er mit dem

doi: 10.2313/NET-2015-03-1_02

privaten Schlüssel erstellen kann sowie zusätzlich auch den öffentlichen Schlüssel an. Da mit der Signatur die Authentizität der bereits verschlüsselten Daten gewährleistet wird, kann jeder Node der ein Datum sieht die Signatur auf ihre Gültigkeit überprüfen, ohne den echten Dateninhalt kennen zu müssen.

Versucht nun ein Angreifer Daten in ein Subspace einzufügen, für das er den privaten Schlüssel nicht kennt, ist er nicht in der Lage eine gültige Signatur zu erstellen und die Daten werden von den anderen Nodes abgewiesen.

USKs sind im Grunde SSKs, nur bieten sie eine automatische Möglichkeit der Versionierung von Daten unter dem gleichen Namen. Da bei SSKs der Identifier für ein Datum unabhängig von dessen Inhalt ist und nur vom öffentlichen Schlüssel und dem docName abhängt, könnte ein Subspace-Verwalter versuchen unter gleichem Namen eine neue Datenversion hochzuladen. Diese würde aber trotz gültiger Signatur von den anderen Nodes abgewiesen, da diese bereits ein Datum unter dem angegebenen Identifier besitzen.

USKs besitzen deshalb am Ende des docName ein zusätzliches Pfadelement, das eine Versionsnummer darstellt. Diese Versionsnummer kann nun entweder eine positive oder negative Ganzzahl sein. Hiervon ist der Suchalgorithmus für neue Versionen des Datums abhängig. Ist die Version positiv, sucht der Node in einer lokalen Datenbank, der *USK registry* nach dieser Version. Findet er dort eine neuere Version als angefragt wurde, liefert er diese aus. Gleichzeitig startet der Node eine Hintergrundsuche im *Freenet* nach neueren Versionen. Diese Suche funktioniert ebenso wie die Suche, die ausgeführt wird, wenn eine negative Versionsnummer angegeben wurde, mit der Ausnahme, dass das Aufrufen einer negativen Versionsnummer im Vordergrund passiert und dem Benutzer somit nicht sofort eine Version der von ihm angefragten Daten geliefert wird.

Für den Aktualisierungsalgorithmus versucht der Node die angefragte, sowie die vier nächsthöheren Versionen zu finden. Ist die Suche nach einer der fünf Versionen erfolgreich, sucht der Node nach fünf neueren Versionen. Dies geschieht so lange, bis er vier aufeinanderfolgende Versionen nicht finden kann. Ist zum Beispiel nach der Version -10 gefragt, sucht der Node nach den Version 10 bis 14. Findet er eine davon – unabhängig welche – fährt er mit der Suche nach 15 bis 19 fort. Wird dem Node nun explizit vom *Freenet* mitgeteilt, dass die Versionen 15 bis 18 nicht existieren, beendet er die Suche und liefert dem Benutzer die nun aktuellste Version aus der USK registry.

Auch beim Hochladen von Daten unter einer USK kann sich der Benutzer bei der Versionierung unterstützen lassen. Gibt man als Versionsnummer 0 an, sucht der Node die niedrigste, freie Nummer und benutzt diese beim Hochladen. Gibt man eine Version größer 0 an, versucht der Node erst das Datum unter dieser Version einzufügen. Ist sie bereits besetzt, wird automatisch nach der nächsten Freien gesucht und unter dieser eingefügt.

3.2 Einsatz von Kryptographie

Alle Daten, die in das *Freenet* hochgeladen werden, sind mit einem zufälligen, symmetrischen Schlüssel, dem cryptoKey, verschlüsselt. Da dieser aber nicht zusammen mit den Daten auf einem Node gespeichert wird, kann ein Node plausibel widerlegen, dass er Wissen über den Inhalt der von ihm gespeicherten Daten hatte. Inwiefern diese Unwissenheit vor Strafe schützt, kann wohl je nach Rechtssystem stark unterschiedlich angesehen werden. Der cryptoKey muss auf einem anderen Weg zusammen mit dem Identifier zugänglich gemacht werden, damit die Daten entschlüsselt werden können. Dazu kann beispielsweise die URI einer Datei ausgetauscht werden, die alle nötigen Informationen enthält.

Weitere Schutzmaßnahmen bestehen darin, dass auch die Kommunikation zwischen zwei Nodes verschlüsselt wird.

Es bestehen zudem Überlegungen, steganographische Transports zu implementieren. Dabei wird *Freenets* Datenverkehr in den Verkehr einer anderen, unauffälligen Anwendung eingebettet oder als solche "verkleidet". So soll verhindert werden, dass *Freenet* -Verkehr als solcher erkannt wird und damit blockiert werden kann.

Als Verschlüsselungsalgorithmus kommt AES mit einer Blockgröße von 256 Bit, anstelle der sonst üblichen 128 Bit zum Einsatz. Es gibt jedoch Überlegungen ebenfalls auf eine Blockgröße von 128 Bit umzusteigen, da diese Version von AES besser untersucht ist.

4. PROTOKOLLALGORITHMEN

4.1 Joining & Leaving

Der Beitritt in ein Peer-to-Peer-Netzwerk stellt immer eine sehr große Hürde da, insbesondere, wenn es wie bei *Freenet* keine zentralen Vermittlungsstellen geben soll. Gerade bei einem Dienst, der seinen Benutzern Anonymität anbieten möchte, muss verhindert werden, dass sich Nodes mit böswilligen Angreifern verbinden. Im Darknet-Modus ist dies dadurch sichergestellt, dass sich nur zu beidseitig manuell hinzugefügten Nodes verbunden wird, von denen man sich sicher ist, dass diese nicht böswillig sind.

Im Opennet-Modus, also wenn der Benutzer keine oder nur wenige Nodebetreiber kennt, denen er vertraut, ist dies ein erhebliches Problem. Um dennoch am *Freenet* teilnehmen zu können, wird eine Liste von sogenannten *Seed Nodes* angeboten. Die Liste kann beim Start eines Nodes über die *Freenet* -Webseite bezogen werden. Verbindet man sich mit einem Seed Node, vermittelt dieser anderen Nodes, die sich in der Umgebung der eigenen Location befinden, die nötigen Verbindungsdaten. Nodes, die an neuen Verbindungen interessiert sind, senden dann Verbindungsanfragen an den neu beitretenden Node.

Zur Optimierung der Netzwerktopologie und des Routings werden in Abhängigkeit des Modus, in dem der Node läuft, verschiedene Techniken angewandt. Im Darknet-Modus wird ein *Location Swapping*-Algorithmus angewendet, der auf [6] basiert. Dabei vergleichen zwei Nodes zu zufällig gewählten Zeitpunkten die mittlere Distanz zu den Nachbarn, mit denen sie jeweils verbunden sind. Wäre die mittlere Distanz geringer, wenn sie ihre Locations vertauschen würden, wird dieser Tausch vorgenommen. Dabei ändert sich nicht die Netzwerktopologie, also die Verbindungsstruktur zu anderen Nodes, da diese ja statisch konfiguriert wurde. Location Swapping kommt bei neuen Nodes noch relativ häufig vor;

die Frequenz lässt aber mit der Zeit nach, sofern sich die Verbindungen nicht ändern.

Da im Opennet eine dynamische Netzwerktopologie möglich ist, werden die Locations von Nodes nicht getauscht, sondern lebenslang beibehalten. Stattdessen werden wenig genutzte Verbindungen im Laufe der Zeit durch neue ersetzt.

Um aus dem Netzwerk auszutreten, sind keine speziellen Schritte notwendig. Da *Freenet* darauf ausgelegt ist widerstandsfähig gegenüber Soft- und Hardwareausfällen zu sein, verträgt es plötzlich verschwindende Nodes ohne Probleme.

4.2 Speichern & Abrufen von Daten

Als eine grundlegende Maßnahme gegen Profiling und Traffic Analysis[2], wurde die Datenblockgröße von CHKs auf 32 kB sowie von SSKs und damit auch USKs auf 1 kB beschränkt. Dies hat zudem den Vorteil, dass große Datenmengen parallel angefragt werden können und durch die zufällige Natur der Hashes auf viele Nodes verteilt werden.

Will ein Node nun doch eine Datei in das *Freenet* einbringen, welche größer als 32 kB ist, muss er diese aufteilen und das letzte Element gegebenenfalls auch auf 32 kB padden. Nun werden die einzelnen Teilstücke unabhängig voneinander hochgeladen. Zudem werden redundante Daten mit Hilfe von Vandermode Forward Error Correction Codes[4] eingefügt, um aufgrund der fehlenden Lebenszeitgarantie verlorene Teilstücke rekonstruieren zu können. Dabei bekommt jedes Teildatum einen eigenen CHK. Sind alle Teile im *Freenet*, erstellt der Node eine weitere Datei, die alle CHKs der eingefügten Teildaten beinhaltet und fügt diese zuletzt ein. So benötigen Benutzer lediglich diesen indirekten Dateizeiger, um die ursprüngliche Datei aufzurufen. Der *Freenet* - Node löst diese Indirektion für den Benutzer automatisch auf. Werden für den Dateizeiger USKs verwendet, kann zudem eine automatische Versionierung von Daten vorgenommen werden.

Um ein Datum im *Freenet* aufzufinden, wird eine *Greedy Search* nach dessen Identifier ausgeführt (bei SSKs und USKs Hash aus XOR-verknüpften routingKey und Hash des docName, bei CHKs der routingKey). Die Vorgehensweise ist hierbei wie folgt:

1. Der Node prüft, ob sich das Datum zum routingKey bereits in seinem lokalem Speicher befindet. Falls ja, ist die Suche trivialerweise beendet. Falls nicht, fährt er mit Schritt 2 fort.
2. Der Node schickt eine Suchanfrage an einen ihm bekannten Node, bei dem die Distanz zwischen dessen Location und dem routingKey am geringsten ist. Hierzu wird der Identifier ebenfalls als eine Fließkommazahl aus dem Intervall zwischen 0 und 1 dargestellt.
3. Der Node, welcher die Anfrage bekommen hat, prüft nun in seinem Speicher nach, ob er das Datum mit entsprechendem Schlüssel kennt. Ist dies nicht der Fall, fährt er ebenfalls wie in Schritt 2 fort.

Damit dies nicht zu einer unendlich langen Kette von Anfragen führt, enthält jede Anfrage ein *Hop-To-Live*-Feld (HTL).

Um den genauen Ursprung einer Anfrage zu verschleiern, wird der HTL nur mit einer bestimmten Wahrscheinlichkeit dekrementiert. Besitzt auch der Node die angefragten Daten nicht, bei dem der HTL auf 0 fällt, meldet er dies an seinen Vorgänger zurück. Der Vorgänger setzt dann mit dem Node fort, der die nächstgeringste Distanz zum routingKey hat. Sollte es zu einem Kreis im Routingpfad kommen, kann dies erkannt werden, da der angefragte Node selbst noch auf eine Antwort für den entsprechenden Identifier wartet.

Dies wird solange fortgesetzt, bis die Daten entweder gefunden wurden oder das gesamte Netzwerk mit einer Tiefe, die etwa der HTL entspricht[5], durchsucht wurde. Wurden die Daten gefunden, nehmen diese den umgekehrten Weg, der für die Anfrage zustande gekommen ist. Auf dem Weg zum Anfragensteller entscheidet jeder Node, ob er die Daten speichert, anhand dessen, ob er noch freien Speicherplatz hat. Ist dies der Fall, speichert er das Datum. Ansonsten, falls kein freier Platz mehr vorhanden ist, prüft er, ob der Eintrag, dessen Aufruf am längsten zurückliegt, eine größere Distanz zu seiner Location hat. Ist dies der Fall, wird der alte Eintrag durch den neuen ersetzt. Abbildung 2 zeigt dabei beispielhaft einen möglichen Suchablauf.

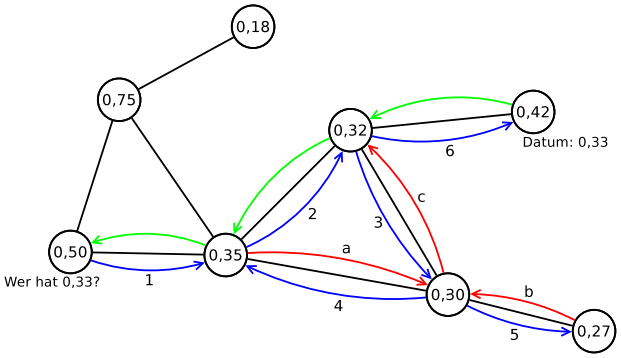

Abbildung 2: Ein möglicher Suchablauf. Der Node mit der Location 0,50 sucht nach einer Datei mit Schlüssel 0,33. Die blauen Pfeile geben die Folge der Anfragen an. Die roten sind Fehlermeldungen: *a*) Routingloop *b*) ist eine Sackgasse, Node 0,27 kann nicht weitersuchen *c*) der Node hat alle seine Nachbarn erfolglos befragt. Die grünen Pfeile zeigen den Rückweg der bei 0,42 erfolgreich gefundenen Daten. Die überquerten Nodes können sich entscheiden die Daten zu speichern.

Durch das Duplizieren der Daten auf dem Rückweg und des Überschreiben von unbenutzten Daten beziehungsweise Daten mit größerer Entfernung zur eigenen Location, werden mehrere Ziele erreicht. Einerseits verbreiten und verteilen sich „interessante", also oft aufgerufene Daten, im *Freenet*. Dies steigert die Zensurresistenz des Systems, da es beispielsweise nicht mehr nur einen zentralen Server gibt, auf dem die Daten gespeichert sind. Außerdem "spezialisieren" sich Nodes im Laufe der Zeit auf Daten, deren Identifier nahe ihrer Location liegen. Dadurch wird es wahrscheinlicher, dass Daten schneller, also über wenigere Hops gefunden werden können.

[5] Aufgrund der probabilistischen Dekrementierung wird tiefer als HTL gesucht

doi: 10.2313/NET-2015-03-1_02

Zum Hochladen von Daten startet ein Node genau so als würde er nach den Daten mit dem Schlüssel, den er einfügen möchte, suchen. Der Unterschied liegt darin, dass der letzte Hop nicht etwa eine Fehlermeldung zurückliefert, dass der Schlüssel nicht gefunden wurde, sondern ein O.K. gibt, dass der Schlüssel noch nicht existiert und der Einfügende beginnen kann die Daten zu senden. Auch hier speichern die Nodes auf dem Weg zwischen erstem und letzten Hop die Daten.

5. NUTZBARKEIT UND ANWENDUNGEN

Dem *Freenet* kann jeder durch das Betreiben eines *Nodes* beitreten. Der Installer lässt sich von *Freenets* Internetauftritt beziehen. Dabei handelt es sich um einen in Java geschriebenen Daemon, den man im Hintergrund laufen lassen kann. Auf das *Freenet* kann man mit einem handelsüblichen Browser zugreifen, da der Daemon einen Proxy, der standardmäßig auf Port 8888 des lokalen Rechners läuft, startet. Durch den Betrieb stellt man dem Netzwerk eine frei wählbare Speicherkapazität und Bandbreite zur Verfügung. Auf dem Konzept von *Freenet*, statische Daten zu speichern und abzurufen, wurden einige zusätzliche Dienste implementiert, die andere bereits bekannte Diensttypen bereitstellen.

Eine für andere Dienste grundlegende Anwendung ist das **Web of Trust** (WoT). Dieses hat keine Verbindung zum *Web of Trust*, das im Zusammenhang mit *OpenPGP* bekannt ist, welches aber wohl als Inspiration gedient haben wird. Es wird benutzt, um Vertrauen zwischen Identitäten oder Pseudonymen im *Freenet* herzustellen. Das WoT ermöglicht es, beliebige Identitäten zu erstellen, repräsentiert durch ein asymmetrisches Schlüsselpaar. Um andere Identitäten kennen zu lernen, veröffentlicht jeder Benutzer eine Liste von ihm bekannten Pseudonymen. Zusätzlich kann jedem Pseudonym ein Trustlevel zugeteilt werden, das angibt, für wie vertrauenswürdig man eine Identität hält. Das WoT wird in den anderen Diensten vor allem zur Erkennung von Spammern benutzt.

Weiter gibt es einen Email-Service, genant **Freemail**, der es erlaubt vertrauliche Email zu verschicken.

Frost und das **Freenet Message System** (FMS) sind zwei newsgroup-ähnliche Dienste, wobei Frost das ältere System ist und FMS zum Ziel hatte, viele Probleme zu beheben die Frost plagen. So hat Frost kein System um Spammer zu erkennen und zu blockieren. Im FMS funktioniert dies ähnlich wie im WoT über das Veröffentlichen von Trust- Listen. Unterschreitet ein Pseudonym ein bestimmtes Trustlevel, werden dessen Nachrichten ignoriert und nicht vom Node abgefragt.

Sone ist ein Facebook oder Twitter ähnelndes Social Network, auf dem kurze Beiträge gepostet werden. Es benutzt ebenfalls WoT zum Auffinden und Verwalten von Identitäten.

6. PROBLEME UND AUSBLICK

Ein großes Problem ist die Suche nach für den jeweiligen Nutzer relevanten Inhalten und das Auffinden der zum Entschlüsseln verwendeten cryptoKeys. So existiert etwa ein Crawler, welcher versucht *Freenet* -Seiten und darauf enthaltene Keys zu indizieren um diesen Index dann durchsuchen zu können. Zudem existieren mehrere manuell gepflegte Seitenindizes, welche regelmäßig aktualisiert werden.

Leider kann das Hoch- und Herunterladen von größeren Datenmengen mehrere Stunden oder Tage in Anspruch nehmen. Einfache Webseiten mit nur wenigen Bildern laden in der Regel innerhalb von einigen 10 Sekunden. Außerdem gibt es aufgrund der Architektur keine Garantie, dass Daten, die man heute einfügt, auch einige Zeit später noch abrufbar sind.

Eine Evaluierung des Routingalgorithmus findet in [3] statt. Es wird zudem eine Attacke auf das Location Swapping demonstriert, die es einem beliebigen Angreifer erlaubt, das Netzwerk so stark zu stören, dass es zu signifikantem Datenverlust kommen kann.

Roos et al. haben erfolgreich versucht das bestehende *Freenet* zu vermessen[5]. Damit wurde gezeigt, das einige Obfuscationmaßnahmen in *Freenet* nicht halten, was sie versprechen, da sie solche Untersuchungen verhindern hätten sollen.

Damit *Freenet* wirklich effizient und mit größtem Vertrauen in dessen Netzwerk benutzt werden kann, ist es vonnöten, das Konzept des Darknets weiter auszubauen und zu verbreiten. Wie sich dieses Bootstrapping-Problem für Menschen, die beispielsweise von staatlicher Seite verfolgt werden und deshalb auf einen sicheren und anonymen Kommunikationskanal angewiesen sind, beheben lässt, ist noch offen. Dabei kann auch schon das vertrauensvolle Beziehen der Software problematisch sein, sollte der Angreifer in der Lage sein, Man-in-the-Middle-Attacken auszuführen.

Zudem muss sich zeigen, wie gut *Freenet* in der realen Welt skalierbar ist, sollte es größere Adaption erfahren. Noch wird laut [5] *Freenet* vorallem in westlichen Ländern wie den USA, Deutschland oder Großbritannien verwendet, die noch nicht so stark von staatlicher Zensur betroffen sind. In Ländern, deren Regierungen öfter Zensur und staatliche Unterdrückung vorgeworfen wird, ist *Freenet* noch nicht sehr populär.

7. VERWANDTE ARBEITEN

Freenet lässt sich nur eingeschränkt mit Anonymisierungsdiensten wie *TOR* vergleichen. Im Gegensatz zu TOR gibt es keine Möglichkeit Daten von außerhalb des Netzwerkes aufzurufen. TOR bietet über sogenannte *Exit Nodes* die Möglichkeit das “normale”, öffentliche Internet aufzurufen. Zusätzliche Webseiten, die *Hidden Services* sind nur innerhalb von TOR routbar. Im Gegensatz zu *Freenet*, sind dies aber keine verteilten Datenspeicher, sondern nur speziell konfigurierte Server. Dies macht sie, sobald man deren echte IP-Adresse ermittelt hat, anfällig für Zensur und DDoS-Attacken.

Ein weiteres anonymes Netzwerk ist das *Invisible Internet Project* (I2P). Wie TOR besitzt es *Hidden Services*, die nur aus dem I2P-Netzwerk erreichbar sind. Es findet aber auch hier keine Duplizierung von Daten auf verschiedenen Nodes statt. Im Gegensatz zu *Freenet* bietet aber auch I2P die Möglichkeit das normale Internet anonym zu benutzen. Somit ist seine Ähnlichkeit zu TOR bedeutend grüßer als zu

doi: 10.2313/NET-2015-03-1_02

Freenet.

Ein Projekt mit anfangs ähnlichen Zielen wie *Freenet* ist *GNUnet*. Es wurde als anonymes, zensurresistentes Filesharingprogramm entworfen, ist aber seither stark gewachsen und legt den Fokus nun mehr darauf ein allgemeines, dezentrales Netzwerk aufzubauen. So bietet *GNUnet* ein alternatives DNS, genannt *GNU Name System*, sowie eine dezentrale Public Key Infrastruktur (PKI).

8. ZUSAMMENFASSUNG

In dieser Arbeit wurde ein Überblick über *Freenet* gegeben. Es handelt sich hierbei um ein anonymes Peer-to-Peer-Netzwerk, das zum Ziel hat, eine verteilte, zensurresistente Möglichkeit zum Speichern und Abrufen von Daten im Internet zu bieten.

Die Zensurresistenz beruht darauf, dass Daten ständig, auf viele Nodes verteilt, dupliziert werden. Dadurch ist *Freenet* resistent gegen gezielte Attacken auf einzelne Nodes. Dies hat aber auch zur Folge, dass es keine Garantie gibt, dass wenig genutzte Daten im *Freenet* für längere Zeit aufrufbar bleiben.

Für das Routing im Darknet-Modus, welcher der bevorzugte Modus gegenüber dem Opennet ist, wird argumentiert, dass sich ein sognanntes *Small World Network* bildet. Der Grund hierfür ist, dass sich die sozialen Bekanntschaftsstrukturen im *Freenet* widerspiegeln, wenn nur Verbindungen mit vertrauenswürden Freunden hergestellt werden. Dies wurde bei der Implementierung des Routingalgorithmus beachtet, weshalb dieser insbesondere in einem *Small World Network* effizient arbeiten kann.

Zusätzlich zum *Freenet* existieren teils von diesem unabhängige Projekte, um zusätzliche Dienste wie Email oder Newsgroups anzubieten.

9. LITERATUR

[1] I. Clarke and S. D. C. Mellish. A distributed decentralised information storage and retrieval system. Technical report, 1999. `https://freenetproject.org/papers.html`.

[2] I. Clarke, O. Sandberg, M. Toseland, and V. Verendel. Private communication through a network of trusted connections: The dark freenet. `https://freenetproject.org/papers.html`.

[3] N. S. Evans, C. Gauthierdickey, and C. Grothoff. Routing in the dark: Pitch black. In *In Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC). IEEE Computer Society*, 2007.

[4] L. Rizzo. Effective erasure codes for reliable computer communication protocols, 1997.

[5] S. Roos, B. Schiller, S. Hacker, and T. Strufe. Measuring freenet in the wild: Censorship-resilience under observation. In E. De Cristofaro and S. Murdoch, editors, *Privacy Enhancing Technologies*, volume 8555 of *Lecture Notes in Computer Science*, pages 263–282. Springer International Publishing, 2014.

[6] O. Sandberg. Distributed routing in small-world networks, 2005. `https://freenetproject.org/papers.html`.