UNIVERSITÄT DUISBURG-ESSEN
FAKULTÄT FÜR INGENIEURWISSENSCHAFTEN
ABTEILUNG INFORMATIK UND ANGEWANDTE KOGNITIONSWISSENSCHAFT

Bachelorarbeit

Evaluation der Effizienz der aggressiven Nutzung von NSEC/NSEC3 Caching

Christian Unger
Matrikelnummer: 3003300

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Abteilung Informatik und Angewandte Kognitionswissenschaft
Fakultät für Ingenieurwissenschaften
Universität Duisburg-Essen

10. Mai 2018

Betreuer:
Prof. Dr.-Ing. Torben Weis
Prof. Dr. Gregor Schiele

Zusammenfassung

Das Internet ist ein viel genutztes Medium und so werden tagtäglich sehr viele Anfragen an das Domain Name System gestellt. Um die Effizienz von DNS Resolvern zu erhöhen, besteht die Möglichkeit der aggressiven Nutzung von NSEC/NSEC3 Resource Records, die im Cache eines Resolvers abgespeichert sind. In dieser Arbeit wird ein Konzept erarbeitet, um diese Nutzung mit aufgezeichneten Netzwerkdaten eines Resolvers zu simulieren und die gesteigerte Effizienz zu ermitteln. Diese Effizienzsteigerung begrenzt sich in dieser Arbeit auf eingesparte Latenzen. Durch eine ereignisorientierte Simulation mit Stream-Ansatz werden die Daten eingelesen und verarbeitet, um einen Resolvercache aufzubauen. Dieser wird verwendet, um Anfragen an den Resolver im Sinne der aggressiven Nutzung von NSEC/NSEC3 Resource Records zu beantwortet. Die Zeit zwischen Anfrage und Antwort wird im Fall einer möglichen Antwort durch den simulierten aggressiven Resolver als Latenzeinsparung dokumentiert. So konnte eine Einsparung der gesamten Latenzzeiten von bis zu 83% festgestellt werden.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Duisburg, 10. Mai 2018 _____

Unterschrift

Inhaltsverzeichnis

1 Einleitung

Im RFC 8198 werden die Vorgehensweise für den *Aggressive Use of DNSSEC-Validated Cache* und die daraus resultierenden Vorteile beschrieben. Eine Implementierung ist jedoch weder bekannt noch wurde das Verfahren getestet oder analysiert.

Ein gewöhnlicher DNS Resolver, der dieses Verfahren nicht nutzt, wird Antworten im Cache speichern und sie bei einer Anfrage, die exakt den gleichen Namen anfragt, aus dem Cache laden. So werden auch NSEC und NSEC3 Resource Records gespeichert und wiederverwendet. Der Resolver nutzt aber die Records nicht 'aggressiv'. Damit ist eine effizientere Nutzung der Informationen aus den Resource Records gemeint. So soll ein DNS Resolver, der dieses Verfahren nutzt, bereits bei Anfragen zu Namen, die zwischen den Grenzen der im Cache gelagerten NSEC/3 Records liegen, diese wiederverwenden und keine Nameserver kontaktieren.

Vergleichbares gilt für die Nutzung von Wildcards. Ein gewöhnlicher Resolver würde bei einer ähnlichen Anfrage, die durch eine bereits im Cache vorhandene Wildcard abgedeckt wäre, eine erneute Anfrage an einen Nameserver stellen. Ein aggressiver Resolver hingegen würde die bereits vorhandene Wildcard aus dem Cache zum Synthetisieren einer Antwort nutzen. Vorausgesetzt ist, dass er die Nichtexistenz des Namens nachweisen kann, was durch die effizientere Nutzung der NSEC/3 Resource Records gegeben sein kann.

Ziel der Arbeit ist es zu ermitteln, ob das im RFC dargestellte Verhalten signifikante Effizienzsteigerungen bewirken würde. Dafür soll ausschließlich die Erweiterung eines Resolvers um das beschriebene Cache Verhalten simuliert und die Unterschiede zu einem gewöhnlichem Resolver analysiert werden. Die Simulation soll einen Datensatz mit realen Anfragen an einen gewöhnlichen Resolver und dessen Antworten zum Einlesen erhalten und basierend darauf Statistiken erheben. Im Fokus steht die Einsparung von Latenzzeiten, die durch die wegbleibenden Anfragen des Resolvers an die Nameserver entsteht. Bei einer beachtlichen Steigerung der Effizienz wäre die Verbreitung und Standardisierung dieses Verfahrens sinnvoll.

Im nachfolgenden Kapitel werden Grundladen, die zum Verständnis der Umsetzung benötigt werden, erläutert. Dazu gehören die grundlegende Funktionsweise des Domain

Name System, dessen Aufbau und das darauf aufbauende Protokoll DNSSEC. Dieses dient der Erhöhung der Sicherheit durch die Nutzung von NSEC und NSEC3 Resource Records. Ebenso erfolgt eine Beschreibung der Vorgehensweise und der genannten Vorteile in RFC 8198.

Der Lösungsansatz zur Durchführung einer Evaluation der Effizienz und dessen Umsetzung sind in Kapitel drei enthalten. Die Aufgabe wird durch gesetzte Ziele, zu erwartende Ergebnisse und angenommene Vereinfachungen abgesteckt. Alle Bestandteile der Umsetzung werden genauer betrachtet, indem erörtert wird, wie eine ereignisorientierte Simulation abläuft, wie sich die DNS Nachrichten unterscheiden lassen und wie sich ein Nameserver verhält, wenn er eine Query beantwortet. Aus letzterem resultiert das zu simulierende Cache Verhalten mit entsprechender Modifikation zur aggressiven Nutzung von NSEC/NSEC3 Resource Records. Außerdem wird beleuchtet, wie die simulierten Antworten verifiziert und die benötigten Daten für die Evaluation in Statistiken erfasst werden.

Danach wird ein Einblick in die Implementierung gegeben. Gewählte Programmiersprache und Programmbibliotheken werden benannt und die Verarbeitung des realen Datensatz wird dargelegt. Weiterhin werden die Kernklassen des Programms durch Diagramme und Auszüge aus dem Programmcode dargestellt und ihre wichtigsten Funktionsweisen geschildert.

Im Kapitel *Auswertung* werden die Ergebnisse der Simulation aufbereitet und visualisiert. Enthalten sind Zählungen relevanter Aspekte der Simulation, Messungen bedeutsamer Parameter zur Erfassung der Effizienzsteigerung und Betrachtung von Fehlern sowie eine Interpretation der Ergebnisse.

Zum Schluss wird der umgesetzte Lösungsansatz zusammengefasst, die Resultate aus dem vorherigen Kapitel werden zur Beantwortung der Frage nach einer Effizienzsteigerung angewendet und es wird ein Ausblick auf zukünftige Arbeiten gegeben.

2 Grundlagen

Dieses Kapitel behandelt grundsätzliche Themen, die Voraussetzung für das Verständnis des Konzepts und dessen Umsetzung sind. Es erläutert den Aufbau und die Funktionsweise des Domain Name System, das darauf aufbauende DNSSEC, sowie Grundsätze zum Thema 'Aggressive Nutzung von NSEC/NSEC3 Caching' aus RFC 8198, welches Ausgangspunkt und Anstoß dieser Arbeit ist.

2.1 Domain Name System

Das Domain Name System (DNS) dient hauptsächlich der Namensauflösung in IP-basierten Netzwerken. Wenn ein Benutzer einen Namen, bspw. example.org, in seinen Browser eingibt, so wird der angefragte Name an das DNS gesendet, verarbeitet und mit der entsprechenden IP-Adresse, unter welcher der angefragte Inhalt zu finden ist, beantwortet. Das gleiche Prinzip gilt für die meisten Webservices, wie Mail, FTP etc.

2.1.1 Aufbau der Datenbank

Das DNS arbeitet mit einer verteilten Datenbank. Alle Informationen sind nicht in einem Ganzen, z. B. auf einem Server, zu finden, sondern auf mehrere Einheiten verteilt. Um eine ständige und weit verbreitete Verfügbarkeit zu gewährleisten, setzt das DNS auf Redundanz. D. h. die gleiche Information ist auf mehreren Servern, die weltweit verteilt sind, vorhanden. Außerdem wird Unicast genutzt. Das bedeutet, dass mehrere Server über die gleiche Adresse erreichbar sind und derjenige mit der kürzesten bzw. schnellsten Route antwortet.

Dieser Aufbau vermindert Probleme, wie unbeabsichtigte oder durch Wartungen verursachte Serverausfälle, sowie schlechte Verbindungen zu bestimmten Servern, während andere besser zu erreichen wären. Außerdem wird die durch viele Anfragen erzeugte Last besser verteilt.

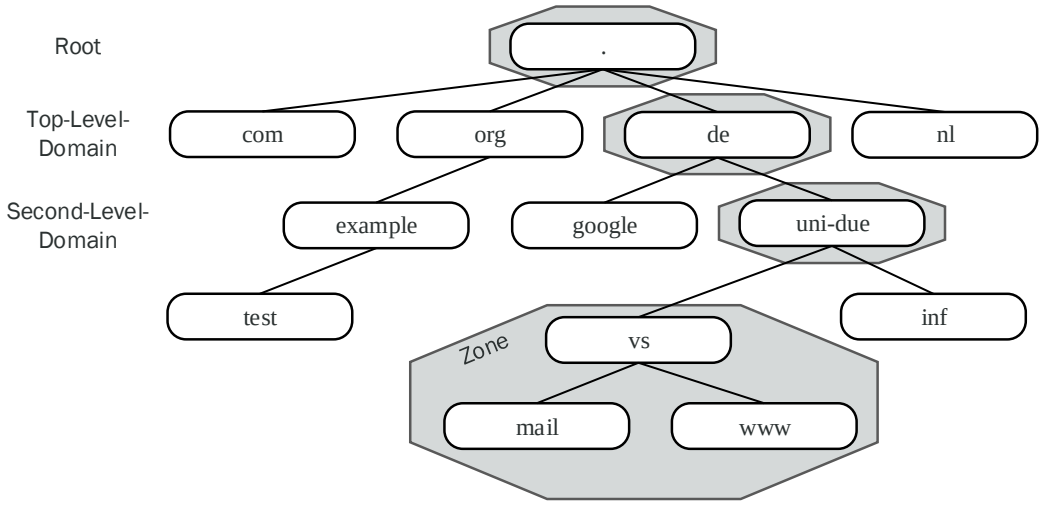

Abbildung 2.1: Baumstruktur der DNS Datenbank mit Zonen

Das DNS ist für die Verwaltung von Namen zuständig. Ein Name besteht aus Labels, die durch Punkte getrennt werden. Zum Beispiel besteht `test.example.org.` aus den Labels `test`, `example`, `org` und einem leeren Label. Man kann je nach Implementierung das letzte, leere Label mitzählen. Nach Definition wird es allerdings nicht mitgezählt.

Dieses spezielle Label wird als *Root* bezeichnet und mit einem einzelnen Punkt dargestellt, wenn es alleine steht. Da es i. d. R. nicht mitgezählt wird, wird es auch häufig bei der Darstellung eines voll qualifizierten Namens weggelassen: `test.example.org`

Man kann die Datenbank des DNS als Baumstruktur auffassen. Dabei repräsentiert jeder Knoten ein Label und der voll qualifizierte Name wird von rechts nach links aufgelöst: Das Root Label ist in Abbildung 2.1 ganz oben wiederzufinden. Danach geht es mit der Top-Level-Domain (TLD) `org` und der Second-Level-Domain (SLD) `example` weiter, gefolgt von der Third-Level-Domain, der Fourth-Level-Domain usw.

Da es sich um eine verteilte Datenbank handelt, werden *Zonen* zur Strukturierung des Namensraums verwendet. Eine Zone ist ein abgeschlossener Bereich ohne Überschneidungen mit anderen Zonen, bestehend aus einer oder mehreren Ebenen des Baums. In Abbildung 2.1 sind sie als grauer Bereich zu sehen. Eine Zone wird mit einer anderen Zone durch eine Weiterleitung, *Delegation* genannt, verbunden. [10] [11] [12] [13]

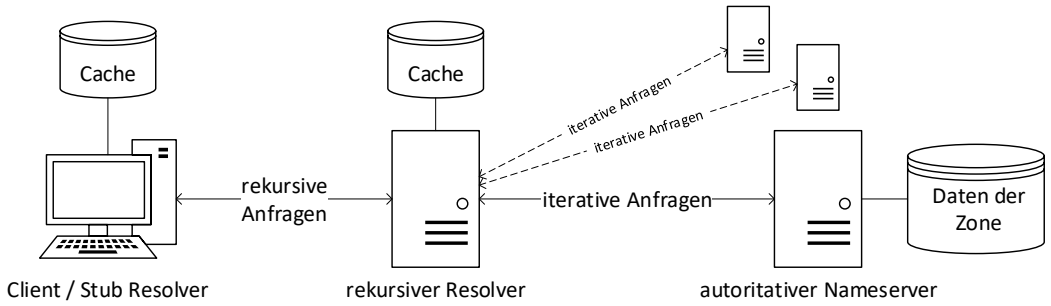

Abbildung 2.2: Lokalisation der Teilnehmer und Ablauf der Kommunikation

2.1.2 Komponenten

Im DNS kommen verschiedene Komponenten und Akteure zusammen. Eine Anfrage wird von einem Client, z. B. einem Browser, gestellt. Sie wird von einem oder mehreren Resolvern verarbeitet und anschließend unter Umständen an einen oder mehrere Nameserver weitergegeben.

2.1.2.1 Resolver

Der Begriff *Resolver* bezeichnet bestimmte Software, die für das Auflösen eines Namens zuständig ist. Resolver stellen, je nach Art des Resolvers, rekursive oder iterative Anfragen. Jeder Resolver kann einen eigenen Cache besitzen. [11]

Stub Resolver Der Client, z. B. ein Browser, stellt eine Anfrage, indem er einen bestimmten Webservice nutzen möchte, wie das Aufrufen einer Internetseite. Als erstes verarbeitet der *Stub Resolver* die Anfrage, in Abbildung 2.2 ganz links zu sehen. Dies ist ein simpler Resolver, meist in das Betriebssystem integriert. Er besitzt eine lokal gespeicherte Liste mit Adressen von rekursiven Resolvern, die, je nach Konfiguration, nach einer bestimmten Zeit aktualisiert wird.
Die Anfrage wird von ihm über einen rekursiven Resolver an einen Nameserver weitergeleitet. Er stellt also rekursive Anfragen an den rekursiven Resolver. [5]

Recursive Resolver Der *rekursive Resolver* ist ein voll funktionsfähiger Resolver, der durch iterative Anfragen an einen oder mehrere Nameserver den angefragten Namen auflöst. Er erhält von ihnen die gewünschten Einträge aus der Datenbank. Er ist in Abbildung 2.2 das Verbindungsstück in der Mitte zwischen Stub Resolver und autoritativem Nameserver. [5]

2.1.2.2 Nameserver

Jede Zone wird durch einen oder mehrere autoritative *Nameserver* (NS) bereitgestellt. Oft sind es mehrere Server, um die in Abschnitt 2.1.1 erwähnte Redundanz zu erhalten. Der NS besitzt alle relevanten Daten aus der entsprechenden Zone und ist am rechten Ende der Abbildung 2.2 zu finden. [11] [12]

2.1.2.3 Einträge

Die vom NS gespeicherten Daten bzw. Einträge werden *Resource Record* (RR) genannt. Ein RR ist also eine Informationseinheit und besteht aus folgenden Feldern: [12]

Owner Name Der Name eines Records wird Besitzername genannt, da er der Identifizierung des Records dient. Dies kann beispielsweise der voll qualifizierte Name `www.vs.uni-due.de.` sein.

Class Dieses Feld gibt die Klasse des RR an, ist aber praktisch fast immer 'IN' für Internet.

Type Ein RR kann unterschiedliche Informationen hinterlegt haben. Der Typ bestimmt die Art der Information. Es gibt viele verschiedene Typen und dies ist ein Überblick über die wichtigsten:

- **A** - Steht für 'address resolution' und sagt, dass eine 32 Bit lange IP-Adresse im Datenfeld zu finden ist.
- **AAAA** - Erfüllt den selben Zweck wie 'A' für IPv6 Adressen. [14]
- **CNAME** - Steht für 'canonical name', also einen Alias. Im Datenfeld ist ein Name zu finden.
- **NS** - Ist für Delegationen zuständig, besagt also, dass der Name eines anderen Nameservers im Datenfeld steht.
- **SOA** - Steht für 'start of authority'. In diesem RR sind relevante Informationen über die Zone zu finden.

Time-to-live Der Wert des Felds 'Time-to-live' (TTL) gibt an, wie lange ein RR maximal im Cache eines Resolvers gespeichert werden soll.

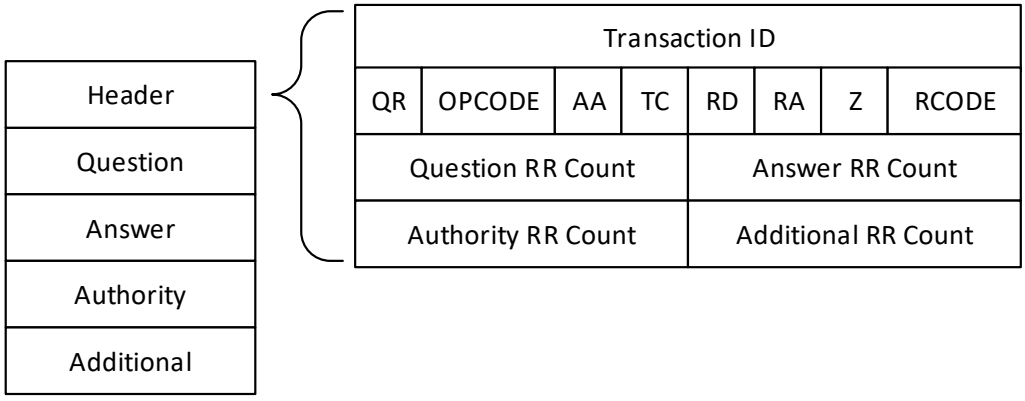

Abbildung 2.3: Format einer DNS Nachricht [12]

Data Das Datenfeld beinhaltet die tatsächlichen Informationen, z. B. die IP-Adresse für den gesuchten Namen oder bei einer Delegation den Namen eines anderen NS. Der Inhalt hängt vom Typ des Records ab.

2.1.3 Protokoll

Das DNS nutzt ein zustandsloses Binärprotokoll, das mit einem Query-Response System arbeitet. Das heißt, dass alle benötigten Informationen stets in einer Nachricht enthalten sind und somit kein Zustand seitens der Akteure festgehalten werden muss.

Abbildung 2.3 zeigt das Format einer solchen Nachricht. Sie besteht aus fünf Bereichen (Sections): Header, Question, Answer, Authority und Additional.

Der Header enthält eine Transaction ID für die Zuordnung von Query und Response, sowie verschiedene Flags. Dazu gehört z. B. das *Query-Response Flag*, kurz QR, zur Unterscheidung von Query und Response. Zum Header gehören auch ein OPCODE, um die Art der Query unterscheiden zu können und ein RCODE für Fehlerhandhabung, sowie je ein Zähler für jeden der vier Bereiche, der die Anzahl der enthaltenen Resource Records angibt.

In der Question Section liegt die Query, auch Question genannt, die stets in den Antworten wieder zu finden ist, wenn denn die Antwort ein Ergebnis enthält. Direkte Antworten, also die gefundenen Resource Records, befinden sich in der Answer Section. Resource Records, welche andere autoritative Server beschreiben oder für andere autoritative Zwecke notwendig sind, liegen in der Authority Section. Hier findet man NS, NSEC[1] und

[1]siehe Abschnitt 2.2.3 NSEC

NSEC3[2] Records. Auch der SOA Record wird in diesem Bereich, wenn benötigt, abgelegt. In der Additional Section haben zusätzliche RR, welche für die anderen Bereiche hilfreich sein können, ihren Platz. [11] [12]

2.1.4 Lookup

Der Client stellt eine Anfrage an einen (Stub-) Resolver, der diese in eine Query transformiert. Eine Query besteht aus einem Tupel von drei Elementen: QNAME, QTYPE und QCLASS. Dies entspricht dem angefragten Namen, dem angefragten Typen und der angefragten Klasse. [11]

2.1.5 Wildcards

Es gibt eine besondere Art von RR, die als *Wildcard* bezeichnet wird. Das Besondere an diesen Records ist, dass der Besitzername mit einem Asterisk '*' beginnt. Eine Wildcard kann als Anleitung zur Synthese von RR verstanden werden. Wenn ein NS eine Anfrage erhält, die die notwendigen Bedingungen für eine Wildcard erfüllt, erstellt der NS den gewünschten RR. Dabei wird der Name aus der Query und der Inhalt aus dem Wildcard Record entnommen. Wie bei anderen Anfragen, müssen QNAME, QCLASS und QTYPE zum RR passen.

Das Asterisk sollte nur als erstes und eigenständiges Label im Besitzernamen stehen. Es passt immer zu einem oder mehreren vollständigen Labels. Einen Besitzernamen wie `test*.example.org` sollte es nicht geben. Hingegen wäre `*.example.org` in Ordnung und würde z. B. zum QNAME `b.example.org`, aber auch zu `b.c.example.org` passen. Eine Anfrage zu `example.org` würde nicht mit dieser Wildcard beantwortet werden.

Wildcards werden nicht angewendet, falls der angefragte Name existiert. Bei eben genanntem Beispiel existiere zusätzlich zur Wildcard der Name `a.example.org`. So würden die zuvor genannten QNAMEs immer noch durch die Wildcard beantwortet werden. Anfragen wie `a.example.org` oder `b.a.example.org` würden jedoch nicht auf die Wildcard zutreffen.

Ein Asterisk innerhalb eines QNAMEs hat keine besondere Auswirkung. Es kann allerdings verwendet werden, um eine Wildcard zu testen. [11]

[2] siehe Abschnitt 2.2.4 NSEC3

2.1.6 Cache Verhalten

Wie in Abbildung 2.2 gezeigt, kann jeder Resolver seinen eigenen Cache besitzen. Jeder RR besitzt eine TTL, die in Sekunden angibt, wie lange der Record im Cache verbleiben darf. Wenn die TTL abgelaufen ist, muss der Record verworfen werden. Bei Anfragen, die diesen Record treffen würden, muss der Resolver eine neue Anfrage an den rekursiven Resolver bzw. die NS stellen. Durch die TTL soll verhindert werden, dass Einträge im Cache zu langlebig sind und Veränderungen in der Zone von den Resolvern nicht rechtzeitig festgestellt werden.

Es gibt positives und negatives Caching. Positives Caching bedeutet, dass die positive Antwort, also das Set aus den gelieferten Records, für die Dauer der TTL gespeichert wird. Mit negativem Caching ist gemeint, dass negative Antworten, die die Nichtexistenz von angefragten Records wiedergeben, gespeichert werden. Dies sind im Fall von DNSSEC NSEC bzw. NSEC3 RR.

Im Fall des negativen Caching gelten bestimmte Berücksichtigungen. So sollen NSEC bzw. NSEC3 RR maximal drei Stunden im Cache bleiben. Die negative TTL soll aus dem Minimum und dem TTL Feld des SOA Records gebildet werden. Dies bedeutet, dass die eigentliche TTL kleiner sein kann als im TTL Feld des NSEC/3 Resource Records angegeben ist. Denn dieses ergibt sich ausschließlich aus dem Minimum Feld des SOA Resource Records. [1] [3] [6] [8]

2.2 DNSSEC

Gewünschte Sicherheitsziele bei Internetprotokollen sind Integrität, Authentizität, Vertraulichkeit und Verfügbarkeit. DNSSEC steht für *Domain Name System Security Extensions* und ist eine Erweiterung um Sicherheitsmechanismen für das DNS. Digitale Signaturen sorgen für Integrität und Authentizität. Es wird gewährleistet, dass die übertragenen Daten nicht manipuliert wurden und die Identität des Autors korrekt und bekannt ist.

Inhalte der DNS Nachrichten werden unverschlüsselt übertragen. Grund dafür ist, dass die übertragenen Inhalte der Datenbank des DNS für jedermann über Anfragen erhältlich sind und somit eigentlich keine geheimen Informationen beinhalten. Ein Dritter könnte die gleiche Anfrage stellen und würde die selbe Antwort erhalten. Später wurde der Aspekt der Vertraulichkeit jedoch überdacht und in NSEC3 teilweise ergänzt. Siehe dazu Abschnitt 2.2.4.

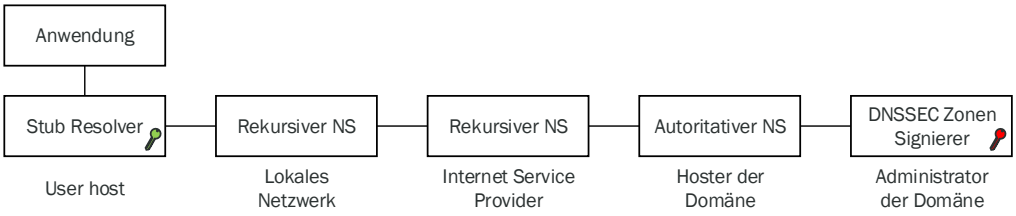

Abbildung 2.4: Ende-zu-Ende Sicherheit durch Signatur

Eine wichtige Voraussetzung von DNSSEC war die Abwärtskompatibilität zum allgemeinen DNS. Es sollten nur Ergänzungen an Stellen geben, an denen Integrität und Authentizität gewünscht sind.

2.2.1 Signatur

Zu Grunde liegt ein asymmetrisches Kryptosystem. Abbildung 2.4 zeigt die Verteilung der Schlüssel und die damit einhergehende Ende-zu-Ende Sicherheit zwischen NS und Stub Resolver. Mit einem privaten Schlüssel werden die RR auf dem Server vom sogenannten *Zone Signer* signiert. Die daraus resultierende Signatur ist ein RR vom Typ RRSIG, der mit dem angefragten RR im entsprechenden Bereich der Nachricht mitgeschickt wird.

Dieser RR hat den selben Besitzernamen wie der signierte RR. Ferner enthält er weitere Informationen, die für die Validierung der Signatur relevant sind. Im Zusammenhang dieser Arbeit spielt allerdings nur das Feld 'Labels' eine elementare Rolle. Es gibt an aus wie vielen Labels der Besitzername besteht. Dies ist besonders für Wildcards erheblich, da das Asterisk hier nämlich nicht mitgezählt wird. So kann ein synthetisierter RR von einem gewöhnlichen RR unterschieden werden.

Nachdem beide RR den Empfänger erreicht haben, kann er die Signatur mit dem öffentlichen Schlüssel validieren. Dabei wird eine Vertrauenskette genutzt, denn jede Zone hat ihren eigenen öffentlichen Schlüssel, der von der Elternzone signiert und somit authentifiziert wird.

Auf diese Weise benötigt der Resolver nur den öffentlichen Schlüssel eines Vertrauensankers, also einer Einheit, dessen Vertrauenswürdigkeit angenommen und nicht abgeleitet ist. Die Vertrauenskette muss nicht bis ganz nach oben aufgelöst werden, da z. B. schon die SLD ein Vertrauensanker sein kann. Sobald in der Kette ein Vertrauensanker erreicht wird, gilt die Signierung als valide. [2]

2.2.2 Closest Encloser und Next Closer Name

Closest Encloser Sowohl für NSEC als auch für NSEC3 ist das Prinzip des *Closest Encloser* von relevanter Bedeutung. Er wird häufig als Bestandteil der 'Source of Synthesis' bezeichnet und aus dem Namen der Anfrage und den Namen der entsprechenden Zone abgeleitet. Es ist der Knoten in der Baumstruktur, der, von rechts ausgehend, die meisten übereinstimmenden Labels mit dem QNAME hat.
Abbildung 2.5 zeigt die Anfrage von `bfxz.a.example.org`. Der Closest Encloser ist hierbei `a.example.org`, weil es für alle drei Labels eine Übereinstimmung mit mindestens einem aus den vorhanden Namen der Zone gibt. [9]

Next Closer Name Der *Next Closer Name* basiert auf dem Closest Encloser. Es wird lediglich das nächste Label des QNAME hinzugenommen.
Beim genannten Beispiel in Abbildung 2.5 würde der Next Closer Name dem vollständigen, angefragten Namen, also `bfxz.a.example.org`, entsprechen. [8]

2.2.3 NSEC

Es werden nur Records signiert und nicht die gesamte Antwort des Servers. Daraus folgt das Problem, dass die Nichtexistenz eines Records nicht signiert werden kann. Gelöst wird es durch einen neuen Typ, dem NSEC Resource Record. Alle existierende Namen der Zone werden in alphabetischer Reihenfolge in einem Ring angeordnet und für jedes dieser Namenspaare wird ein NSEC Record angelegt. Der Record beinhaltet als bezeichnenden Namen den ersten Namen des Paares (NAME) und als Information im Datenfeld den nächsten Namen der Zone (NEXT) sowie eine Liste aller Record Types, die unter diesem Besitzernamen vorhanden sind. [4]

2.2.3.1 Sortierung

Bei der Sortierung der Namen gelten folgende Regelungen: Jedes Label der beiden Namen wird von rechts nach links einzeln betrachtet. Wenn beide Labels gleich sind, so wird nach dem voranstehenden (linken) Label sortiert. Also werden erst die beiden letzten Labels verglichen, dann die vorletzten, danach die davor usw. Nicht vorhandene Labels werden allen anderen vorangestellt. Großbuchstaben werden wie Kleinbuchstaben behandelt. [4]

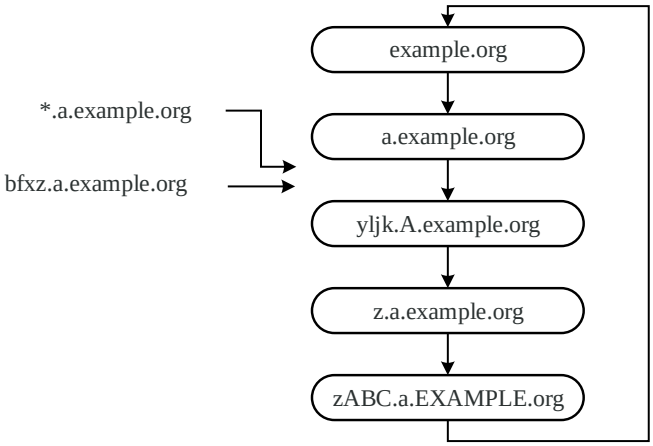

Abbildung 2.5: Ein Ring aus NSEC Records stellt sicher, dass die Nichtexistenz von Namen nachgewiesen werden kann

2.2.3.2 Denial of Existence

Um die Existenz eines angefragten Namens zu dementieren, müssen zwei Aussagen bestätigt werden:
1. Es gibt keinen Record unter exakt diesem Namen.
2. Es kann kein Record auf Basis einer Wildcard synthetisiert werden.
Dies kann auf zwei Arten geschehen:

Direkter Treffer Für den QNAME wird ein authentifizierter NSEC Record gefunden, jedoch ist der QTYPE nicht in der Liste der existierenden Typen des Records vorhanden. Die Existenz einer Wildcard wird ausgeschlossen, indem das Feld 'Labels' des dazugehörigen RRSIG Records mit der Anzahl der Labels im Namen des NSEC Records abgeglichen wird. Wenn sie gleich sind, existiert keine Wildcard. [3]

Abgedeckter Treffer Wenn der QNAME zwischen NAME und NEXT liegt, sagt man, dass der Record den angefragten Namen abdeckt. Der Server antwortet mit diesem abdeckenden NSEC Record. Ein weiterer ist nötig, um auch die Existenz einer Wildcard zu dementieren.[3] Dabei wird die zu überprüfende Wildcard aus dem Asterisk und dem Closest Encloser[3] zusammengesetzt. [9]

Das Beispiel in Abbildung 2.5 zeigt die Überprüfung von `bfxz.example.org`. Dieser Name würde zwischen `a.example.org` und `yljk.A.example.org` liegen. Außerdem wird

[3]siehe Abschnitt 2.2.2 Closest Encloser

überprüft, ob die Wildcard *.a.example.org existiert. Sie würde an der selben Stelle liegen.

2.2.4 NSEC3

NSEC löst zwar ein Problem, wirft aber auch ein neues auf, welches *Zone Walking* genannt wird. Dadurch, dass bei einer Anfrage eines nicht existierenden Namens zwei existierende Namen zurück gegeben werden, könnte man durch iterative Abfragen alle Namen einer Zone erfahren und somit eine Kopie der gesamten Datenbank erhalten.

Als Lösung des Problems dienen Hashfunktionen. Dadurch können NSEC3 Records die Nichtexistenz, ohne alle vorhandenen Namen offen zu legen, bestätigen. Für den Ring in alphabetischer Reihenfolge wird für alle Namen ein Hashwert berechnet. Der Besitzername für einen Record setzt sich aus dem Hashwert als einzelnes Label und dem Namen der Zone zusammen. Der Hashwert ist mit einem erweiterten Hex-Alphabet in Base32 kodiert. Der Hashwert des nächsten Namens (NEXT) ist jedoch im Binärformat und ohne den Namen der Zone gespeichert.

Abgesehen von den Eigenschaften eines NSEC Records, beinhaltet ein NSEC3 Record noch wichtige Parameter für die Hashfunktion: Der verwendete Hashalgorithmus, wobei bisher nur SHA1 registriert ist, die Länge und den Wert des Salt, sowie die Anzahl der Iterationen der Hashfunktion. Außerdem gibt es ein Feld für bestimmte Flags, das bisher nur für das Opt-Out Flag verwendet wird. Mehr zu diesem Flag in Abschnitt 2.2.6. [8]

2.2.4.1 Sortierung

Die Regeln für die Sortierung sind die selben wie bei NSEC, jedoch kann die Reihenfolge eine andere sein. So könnte bspw. der Hashwert des zweiten Namens a.example.org aus Abbildung 2.5 dem vierten Namen d0b1.example.org in Abbildung 2.6 entsprechen.

2.2.4.2 Denial of Existence

Um die Existenz eines angefragten Namens zu dementieren, sind zwei Überprüfungen notwendig: Ein so genannter Closest Encloser Proof und, wie bei NSEC auch, der Nachweis der Nichtexistenz einer Wildcard.

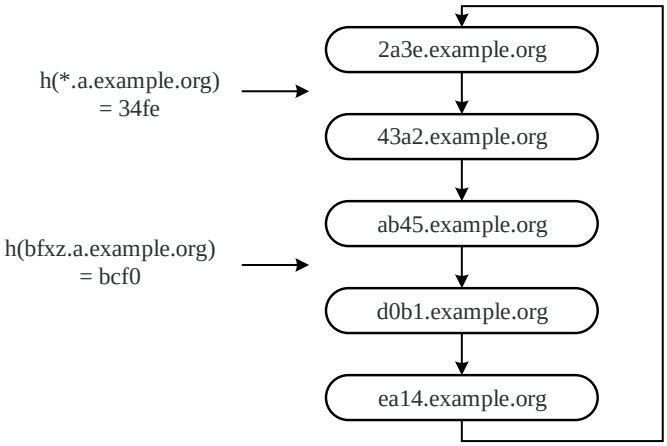

Abbildung 2.6: Ein Ring aus NSEC3 Records stellt sicher, dass die Nichtexistenz von Namen nachgewiesen werden kann und erschwert dabei das Zone Walking. (vereinfachte Darstellung mit gekürzten Hashwerten)

Closest Encloser Proof Der *Closest Encloser Proof* besteht aus zwei Schritten. Zuerst wird der Closest Encloser ermittelt, indem der Hashwert des angefragten Namens berechnet und dieser mit den vorhandenen Namen der NSEC3 Records der Zone abgeglichen wird. Wenn es keinen Treffer gab, wird der Name von links um ein Label gekürzt und die Prozedur wiederholt. Sobald ein Treffer erzielt wurde, ist der aktuelle Name der Closest Encloser. Daraus kann geschlossen werden, dass der angefragt Name nicht existiert und der am nächsten liegende Name der gefundene ist.
Im zweiten Schritt wird überprüft, ob der Next Closer Name von einem Record abgedeckt wird. Damit ist gezeigt, dass der angefragt Name nicht existiert und somit der Closest Encloser wirklich der am nächsten liegende Name ist.
Wenn beide Schritte erfolgreich waren, ist der Closest Encloser Proof abgeschlossen. [7]

Nichtexistenz einer Wildcard Eine Wildcard für NSEC3 wird ebenso ermittelt, wie bei NSEC: Asterisk plus Closest Encloser. Danach wird überprüft, ob ein NSEC3 Record existiert, der die Wildcard abdeckt. Dafür wird der Hashwert der Wildcard berechnet und nach einem NSEC3 Record gesucht, dessen NAME und NEXT diesen Wert umschließen.

2.2.5 Negativer Cache

Bei einem negativen Cache speichert der Resolver die negativen Antworten der NS, also die NSEC bzw. NSEC3 RR. Er kann so bei einer Anfrage zu dem exakt selben Namen dem Cache entnehmen, dass kein Eintrag existiert und dem Client eine Antwort liefern, ohne erneute Anfragen an einen NS stellen zu müssen.

2.2.6 Opt-Out

DNSSEC benötigt viel mehr Rechen- und Speicherressourcen, da für jeden Namen ein NSEC bzw. NSEC3 Record erstellt werden muss. Neben den zusätzlichen Records, werden auch die Berechnung und Speicherung von Hashwerten benötigt. Gerade für Top Level Domains, wie `com`, wäre der Ressourcenbedarf zu groß. Um dieses Problem zu umgehen, bietet DNSSEC ab NSEC3 eine Besonderheit namens *Opt-Out*. Der Administrator einer signierten Zone kann durch das Setzen des Opt-Out Flags unsichere Delegationen ohne dazugehörige NSEC3 RR zulassen. [8] Damit ändert sich aber leicht die ursprüngliche Spezifikation von NSEC, da die Bedeutung der Nichtexistenz verändert wurde: ,,Opt-Out NSEC3 records are not able to prove or deny the existence of the insecure delegations. In other words, those delegations do not benefit from the cryptographic security that DNSSEC provides." (RFC5155, Abschnitt 5.1) [7]

2.3 Aggressive Nutzung von NSEC/NSEC3 Caching

Durch RFC 8198 "Aggressive Use of DNSSEC-Validated Cache" wird RFC 4035 "Protocol Modifications for the DNS Security Extensions" aktualisiert. Das erste Mal wurde aggressives negatives Caching in RFC 5074 "DNSSEC Lookaside Validation (DLV)" vorgeschlagen.

2.3.1 Negativer Cache

Bisher wurde nur auf exakte Treffer im Cache reagiert, also wenn der QNAME mit dem Namen des im Cache vorhandenen RR übereinstimmt. Nun werden NSEC und NSEC3 RR im Cache aggressiver genutzt, indem auch Anfragen zu Namen, die zwischen dem Besitzernamen und dem Namen im Next Feld des im Cache gespeicherten RR liegen, negativ beantwortet werden. Es erfolgt keine Anfrage mehr an einen NS.

Wenn beispielsweise *apple.example.com* und *cherry.example.com* als NSEC RR im Cache vorliegt und ein Client *banana.example.com* anfragt, so soll der Resolver dem Client mitteilen, dass es den angefragten Namen nicht gibt, anstatt eine Anfrage an einen NS zu stellen.

Die aggressive Nutzung von NSEC3 RR gestaltet sich etwas komplexer und rechenaufwendiger, da der Resolver selbstständig den Hashwert des angefragten Namens berechnen muss, um einen Abgleich mit dem Cache durchführen zu können.

2.3.2 Berücksichtigung der Time-to-live

Besonders im Fall der aggressive Nutzung spielt die TTL eine wichtige Rolle. Gegeben durch die geringere Anzahl an neuen Anfragen an die NS, besitzen Resolver mit einer aggressiven Nutzung von NSEC/NSEC3 Caching einen länger gleichbleibenden Stand im Cache. So können neu angelegte Namen zunächst nicht aufgelöst werden. Nach obigem Beispiel müsste man nach Erstellung des RR für *banana.example.com* etwas warten bis ein Resolver mit besagtem Cache Eintrag eine Anfrage stellt und bemerkt, dass der Name ab diesem Zeitpunkt doch existiert.

Aus dem Grund soll die TTL kleiner als das Minimum-Feld im SOA Eintrag sein und nicht 3 Stunden, als Wert 10800, übersteigen. Verwaltern von Zonen sollte bereits bewusst sein, dass sich Änderungen nicht sofort bemerkbar machen, da es schon beim ursprünglichen DNS nicht zu erwarten war. [6]

2.3.3 Vorteile

Das beschriebene aggressive Caching soll zu folgenden Vorteilen führen. Begründet wird dies darauf, dass in einer im Juli 2017 veröffentlichten Statistik 65% aller Queries mit der Nichtexistenz der angefragten Domain beantwortet wurden. [6]

2.3.3.1 Ressourceneinsparungen

Durch den Verzicht auf zusätzliche Anfragen des Resolvers an die NS ergeben sich verschiedene Vorteile: Die Latenz, also die Zeit zwischen Anfrage und Antwort aus Sicht des Clients, wird verringert. Die Last auf die rekursiven und somit auch auf die autoritativen Server wird reduziert. Sie können Bandbreite und Rechenleistung einsparen. [6]

2.3.3.2 Sicherheit

Es gibt so genannte *Random QNAME Attacks*, bei denen ein potentieller Angreifer viele Anfragen für zufällige Namen an einen Resolver sendet. Wenn die Antworten nicht im Cache des Resolvers vorliegen, stellt dieser wiederum Anfragen an autoritative Nameserver. Dadurch könnten sie überlastet werden.

Wenn der Resolver unterdessen bereits nach ein Paar Anfragen die entsprechenden NSEC/NSEC3 RR zwischengespeichert hat, kann er daraus die Nichtexistenz nachfolgender angefragter Namen herleiten und direkt dem Client antworten. So wären die autoritativen Server weniger angreifbar. [6]

3 Konzept

In diesem Kapitel geht es um die Lösung des anfangs dargestellten Problems. Es beinhaltet eine kurze Beschreibung der Aufgabe bzgl. der verwendeten Daten, gesetzten Ziele, vereinbarten Vereinfachungen und erwarteten Ergebnisse. Im Hauptteil, der Umsetzung, wird näher beleuchtet, wie die Lösung durch eine ereignisorientierte Simulation mit Hilfe eines Streams und Sliding Window erreicht wird. Es wird auch erklärt, wie die Filterung der Nachrichten funktioniert. All diese Verfahren sind Bestandteil der Simulation und tragen zur Gesamtlösung bei.

Danach ist eine schrittweise Übersicht der Durchführung der Simulation und eine Erläuterung der Verifizierung der vom simulierten aggressiven Resolver ermittelten Antwort zu finden.

Im letzten Abschnitt steht eine Erläuterung zur Erhebung der Statistiken. Es wird erklärt, welche Statistiken erfasst und welche nicht erfasst wurden bzw. welche nicht erfasst werden können. Außerdem enthält dieser Abschnitt Informationen darüber, an welcher Stelle die entsprechenden Messungen bzw. Berechnungen stattfinden.

3.1 Aufgabenbeschreibung

Bevor der eigentliche Lösungsansatz erläutert wird, erfolgt eine kurze Beschreibung der Rahmenbedingungen der zu lösenden Aufgabe, um festzuhalten mit welchen Annahmen und zu welchem Zweck die Software entwickelt werden soll.

3.1.1 Daten

Zur Verarbeitung und anschließender Analyse liegt ein Netzwerkmitschnitt eines Resolvers vor. Es wird angenommen, dass dieser Resolver innerhalb des gleichen Netzwerks wie die Clients liegt und ihm eine statische IP-Adresse zugeordnet wurde. Er stellt Anfragen an Nameserver, die wiederum außerhalb des Netzwerks zu finden sind.

3.1.2 Ziele

Es soll eine Software zur Simulation und Analyse des genannten Verfahrens entworfen werden. Sie soll auf Basis der gegebenen Daten eine Simulation eines solchen aggressiven Resolvers durchführen und dabei die Unterschiede messen und protokollieren. Des Weiteren soll das Verhältnis zwischen 'Hit' und 'Miss' bei Zugriffen auf den aggressiv genutzten Cache untersucht werden. Gewöhnliches Cache Verhalten soll nicht simuliert werden, da der gewöhnlicher Resolver, dessen Netzwerkverkehr überwacht wurde, dies bereits abbildet.

3.1.3 Vereinfachungen

Um den Umfang dieser Arbeit nicht zu überschreiten und die Komplexität der Aufgabe im Rahmen zu halten, wurden folgende Vereinfachungen angenommen:

1. Es kann in der Praxis vorkommen, dass ein Resolver neu gestartet wird. In der Simulation wird dies missachtet. Sie geht davon aus, dass der Resolver durchgängig eingeschaltet gewesen ist.

2. Ein realer, sicherheitsbewusster Resolver würde die mitgelieferten RRSIG RR nutzen, um die Antwort zu validieren. Diese Validierung ist nicht Teil der Simulation. Es wird die Annahme getroffen, dass die Antworten der NS valide sind.

3. Die Anfragen des Resolvers an einen Nameserver werden außer Acht gelassen. Lediglich der Verkehr zwischen dem internen Netzwerk und dem Resolver wird für Berechnungen beachtet. Die Antworten der NS an den Resolver werden für den Aufbau des Caches ausgewertet. Mehr dazu ist im Abschnitt 3.2.5 *Nachrichten filtern* zu finden.
   Die missachteten Anfragen wären insofern wichtig, weil sie Auskunft darüber geben könnten, wie viele Anfragen der Resolver, die er von den Clients erhält, aus dem Cache beantwortet und für wie viele er einen NS kontaktieren muss. Man könnte daraus ableiten, wie viele Anfragen seitens des Resolvers eingespart und somit dem NS als Last erspart wurden.

3.1.4 Erwartete Ergebnisse

Aggressive Use of DNSSEC-Validated Cache beschreibt eine Einsparung von Latenzzeiten und Rechen- bzw. Speicherressourcen. In dieser Arbeit soll allerdings nur die Zeiteinsparung untersucht werden. Es ist zu erwarten, dass eine markante Einsparung eintreten wird.

3.2 Umsetzung

Um das Problem zu lösen und das gewünschte Cache Verhalten zu simulieren, werden unterschiedliche Verfahren genutzt. Die Lösung baut auf Konzepten auf, wie u. a. der ereignisorientierten Simulation und dem Sliding Window. Außerdem wurden verschiedene Software Komponenten, die benötigt werden, entwickelt.

Verwendete Konzepte werden kontextbezogen erörtert und das Verhalten der entworfenen Software beschrieben. Dabei wird auf das Verhalten zum Beantworten einer Query von Nameservern und des simulierten aggressiven Resolvers noch ein Mal genauer eingegangen. Die Struktur des Netzwerks und der Bezug der einzelnen Komponenten zu diesem werden grafisch dargestellt. Ebenso wird die Durchführung der Simulation Schritt für Schritt behandelt und eine Verifizierung der erhaltenen Ergebnisse des simulierten aggressiven Resolvers erläutert.

3.2.1 Ereignisorientierte Simulation mit Stream-Ansatz

Damit die Zeit während der Simulation nicht gemessen werden muss, handelt es sich um eine ereignisorientierte Simulation. D. h. die Verarbeitung erfolgt auf Basis von Ereignissen. In diesem Fall ist jedes Paket, das verschickt wurde bzw. eingelesen wird, ein Ereignis. Der Zeitstempel vom zuletzt eingelesenen Paket gilt dann in dem Moment als aktueller Zeitpunkt der Simulation. Dadurch ist die Zeit nicht von der Ausführungsdauer abhängig, sondern deterministisch.

Zum Einlesen der Daten wird ein Stream verwendet. Die Datei wird sequentiell Paket für Paket eingelesen und ausgewertet. Auf diese Weise muss nicht die vollständige Datei in den Arbeitsspeicher geladen werden, was bei zu großen Dateien zu Komplikationen führen würde.

3.2.2 Cache Verhalten simulieren

Um das Cache Verhalten eines aggressiven Resolvers simulieren zu können, muss zunächst die Beantwortung einer Query mit der Vorgehensweise eines Nameservers und anschließend basierend auf der Ausgangslage eines Resolvers mit aggressivem Cache Verhalten analysiert und verstanden werden.

3.2.2.1 Beantwortung einer Query durch einen Nameserver

Inhalt und Anzahl der RR in der Response des Nameservers weichen bei NSEC und NSEC3 voneinander ab, weil die Struktur der Zone und der Vorgang der Ermittlung des richtigen Records Unterschiede aufweisen. Daher erfolgt eine differenzierte Betrachtung von NSEC und NSEC3. Später, bei der Betrachtung des simulierten aggressiven Resolvers, werden beide Ansichten vermischt, da die Bedeutung von Inhalt und Anzahl der Records in der Antwort geringer und für die Simulationszwecke nur die Art der Antwort relevant ist.

Die erfolgreiche Bestimmung eines Records, in dieser Arbeit *Answer Response* genannt, ist in beiden Fällen vereinfacht dargestellt, weil der Fokus auf den Fall gerichtet ist, dass keine direkte Übereinstimmung ermittelt werden kann.

NSEC Die nachfolgende Beschreibung ist anhand der Abbildung 3.1 nachzuvollziehen. Der NS erhält die Query und sucht nach einem RR Set unter QNAME, QTYPE und QCLASS. Eine vollständige Übereinstimmung führt zu einer *Answer Response*, die den gewünschten Record bzw. die gewünschten Records mit dazugehörigen RRSIG Record(s) enthält.

Wenn es keine Übereinstimmung gibt, erfolgt eine Überprüfung auf einen direkten Treffer. D. h. es wird unter dem QNAME nach einem NSEC Record gesucht. Bei Erfolg ist es eine *No Data Response*. Diese besteht aus dem NSEC Record, dessen Name gleich dem QNAME ist, der aber den angeforderten QTYPE nicht im Type Bit Maps Feld aufweist. Der entsprechende RRSIG Record ist in der Antwort inbegriffen.

Im Fall keines direkten Treffers, muss es ein abgedeckter Treffer sein. Der QNAME liegt zwischen dem Besitzernamen und dem Namen im Next Feld des NSEC Records. Somit ist die Existenz eines RR unter dem QNAME ausgeschlossen. Es kann aber eine Wildcard geben, die den QNAME abdeckt. Also wird überprüft ob eine passende Wildcard vorliegt.

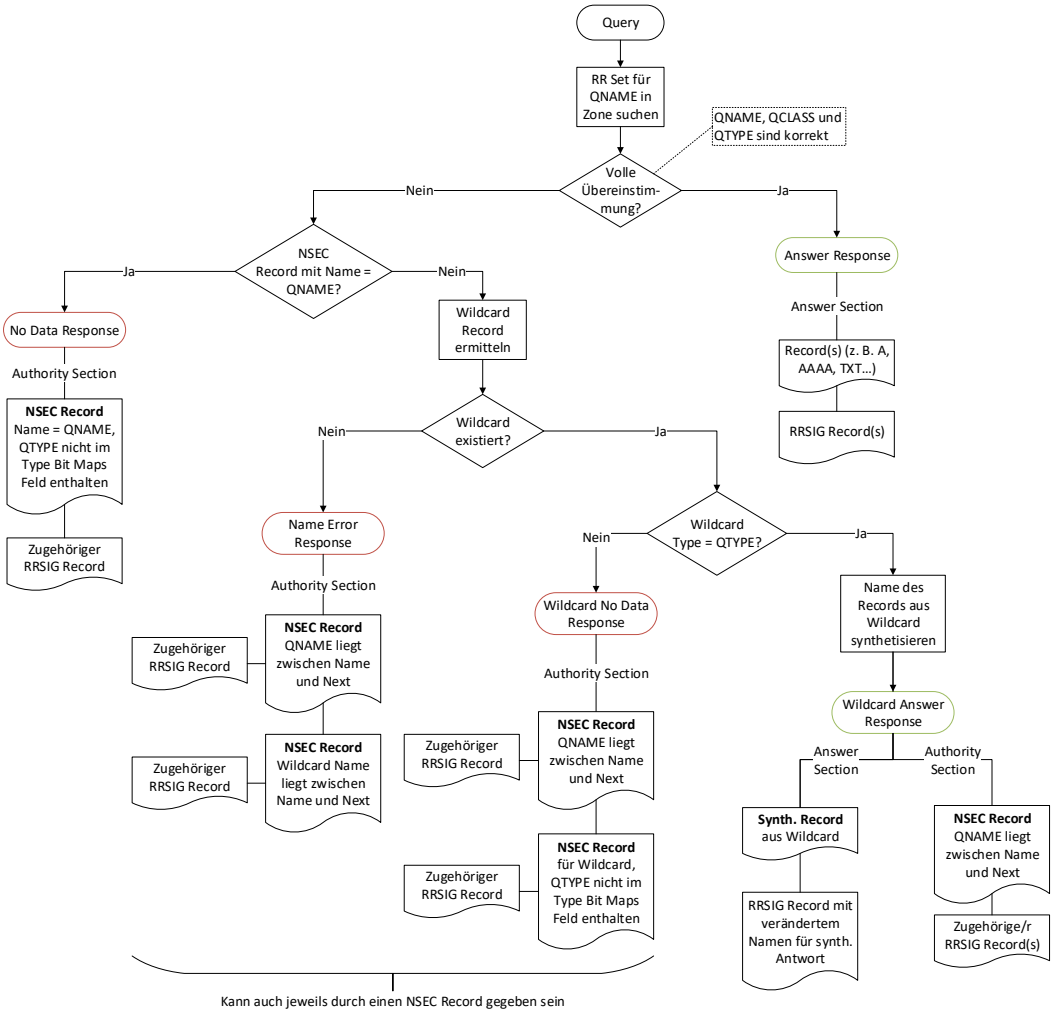

Abbildung 3.1: Beantwortung einer Query aus Sicht eines Nameservers mit NSEC Resource Records

Sofern keine Wildcard existiert, muss der NS zusätzlich zum NSEC Record, der den QNAME abdeckt, einen weiteren NSEC Record schicken. Dieser deckt, ähnlich wie der erste Record, den Namen der Wildcard ab. Dies ist eine *Name Error Response*.

Falls eine Wildcard vorhanden ist, gilt das gleiche Prinzip, wie bei einem direkten Treffer. Der Typ des Wildcard Records muss dem QTYPE gleichen. Bei Übereinstimmung synthetisiert der NS anhand des gefundenen Records eine Antwort. Zusammen mit dem RRSIG Record, der auch den abgeänderten Namen erhält, wird eine *Wildcard Answer Response* gesendet.

Wenn der QTYPE nicht dem Typ des Wildcard Records entspricht, sendet der NS einen NSEC Record, der den QNAME abdeckt, und einen zweiten NSEC Record, der den gleichen Name wie die Wildcard besitzt. Jedoch enthält er nicht den QTYPE im Type Bit Maps Feld. Dies wird *Wildcard No Data Response* genannt. [4]

NSEC3 Zur nachfolgenden Beschreibung ist Abbildung 3.2 zu beachten.
Auch bei NSEC3 sucht der NS nach dem Erhalt der Query nach einem passenden RR Set, mit gleicher Antwort bei einem Erfolg. Bei einem Misserfolg wird auch bei Nutzung von NSEC3 nach einem direkten Treffer gesucht, mit dem Unterschied, dass der Hashwert des QNAME berechnet und mit einem Punkt mit dem Zonennamen verbunden wird. Auch bei NSEC3 handelt es sich bei einem direkten Treffer um eine *No Data Response* mit fehlendem QTYPE im Type Bit Maps Feld und zugehörigem RRSIG RR.

Wenn es keinen direkten Treffer gibt, erfolgt ein Closest Encloser Proof, bei dem der Closest Encloser und der Next Closer Name für den QNAME bestimmt werden. Beide werden für anschließende Schritte benötigt. Wie bei NSEC, wird als nächstes überprüft, ob eine Wildcard vorhanden ist.

Falls keine Wildcard vorliegt, sendet der NS eine *Name Error Response*, die bis zu drei NSEC3 RR umfasst: Ein Record, der ein direkter Treffer für den Closest Encloser ist, ein Record, der den Next Closer Name abdeckt und damit bestätigt, dass der Closest Encloser tatsächlich der nächstmögliche Treffer ist und ein Record, der den Namen der Wildcard abdeckt. Damit ist die Nichtexistenz des QNAME nachgewiesen. Wildcard und Next Closer Name können ggf. durch einen gemeinsamen Record abgedeckt sein.

Im Fall, dass eine Wildcard existiert, muss überprüft werden, ob der QTYPE in der Liste der vorhandenen Record Types enthalten ist. Bei einer *Wildcard No Data Response* ist dies nicht der Fall. Die Antwort des NS setzt sich ähnlich wie bei NSEC zusammen, nur dass zwei NSEC3 Records für die Nichtexistenz des QNAME benötigt werden, wie bei der Name Error Response zuvor. Auch hier können allerdings insgesamt zwei statt drei Records genügen.

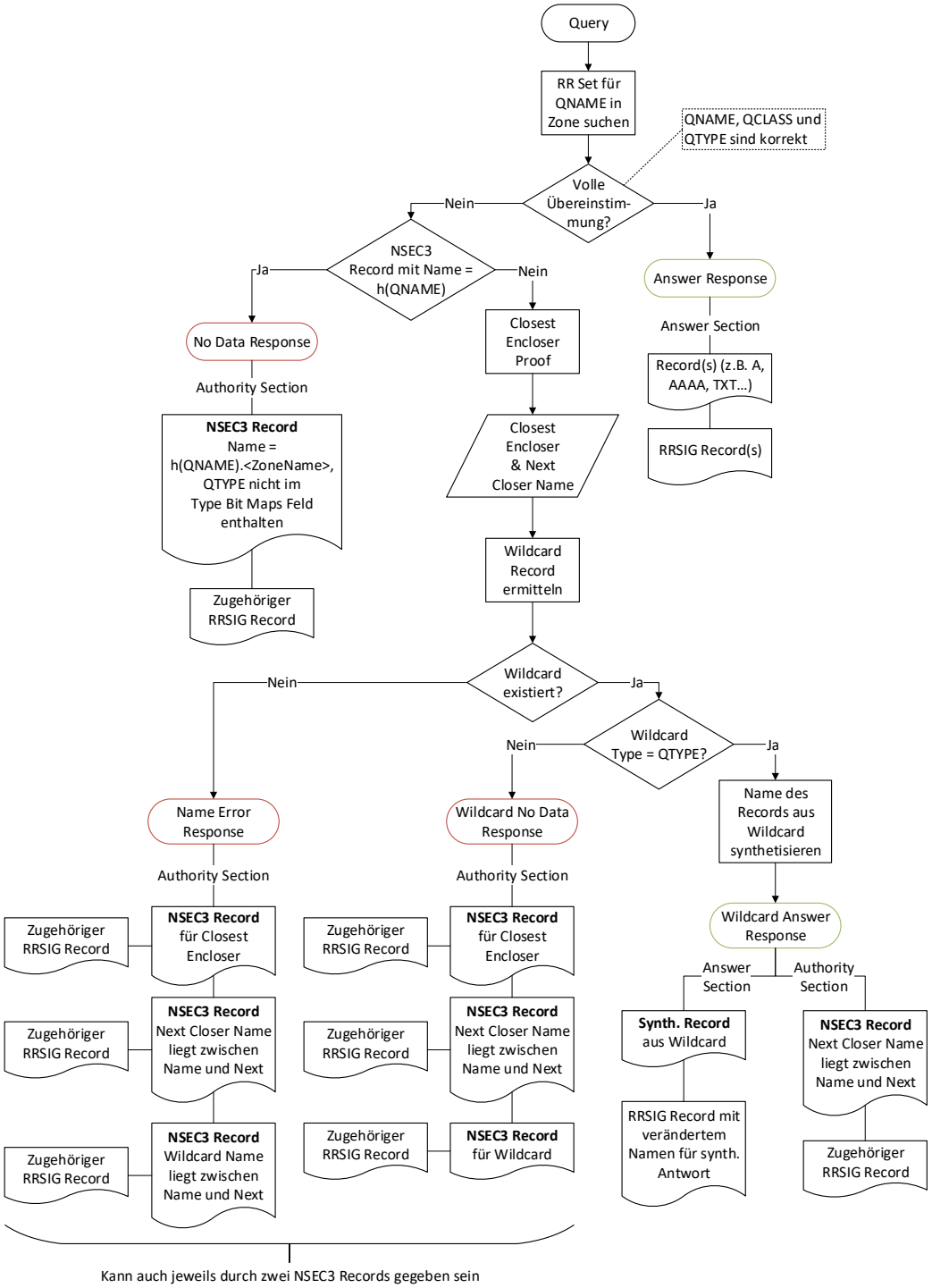

Abbildung 3.2: Beantwortung einer Query aus Sicht eines Nameservers mit NSEC3 Resource Records

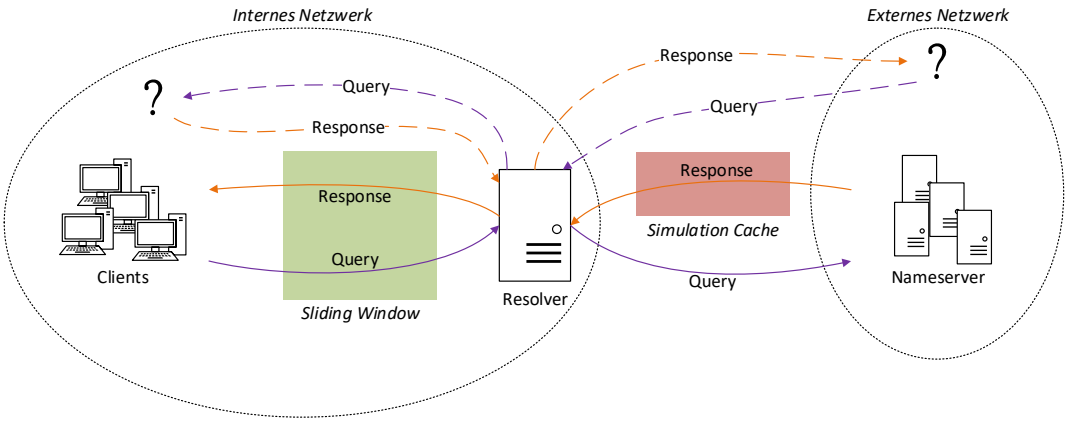

Abbildung 3.3: Netzwerktopologie mit den Zugriffspunkten von Sliding Window und Simulation Cache

Wenn der QTYPE mit dem Typ des Wildcard Records übereinstimmt, wird die resultierende Antwort *Wildcard Answer Response* genannt. Der NS synthetisiert einen RR als Antwort aus dem Wildcard RR und einen RRSIG RR aus dem RRSIG RR, der zum Wildcard RR gehört. Er sendet sie zusammen mit dem NSEC3 Record, der den Next Closer Name abdeckt, und dessen zugehörigen RRSIG RR an den Resolver. [8]

3.2.2.2 Beantwortung einer Query durch einen Resolver mit aggressivem Cache Verhalten

Abbildung 3.4 veranschaulicht die Beantwortung einer Query durch den simulierten aggressiven Resolver. Ausgegraute Endpunkte werden von ihm nicht behandelt und mit 'No Response' beantwortet, da dies in den Aktionsbereich eines gewöhnlichen Resolvers fallen oder eine Anfrage an einen NS erfordern würde.

Grundlegende Überprüfung Nach dem Erhalt der Query wird über den QNAME nach einem Eintrag im negativen Cache gesucht. Wenn keiner existiert, die TTL abgelaufen ist oder es sich um einen NSEC3 RR mit Opt-Out handelt, kann keine Antwort gebildet werden.
Danach findet die Überprüfung des Namens des gefundenen NSEC/3 Records statt. Sofern er mit dem QNAME übereinstimmt und der Record den QTYPE nicht im Type Bit Maps Feld aufweist, kann mit dem Labelsmatch fortgefahren werden. Im Fall, dass der QTYPE vorhanden ist, lautet die Antwort 'No Response'.

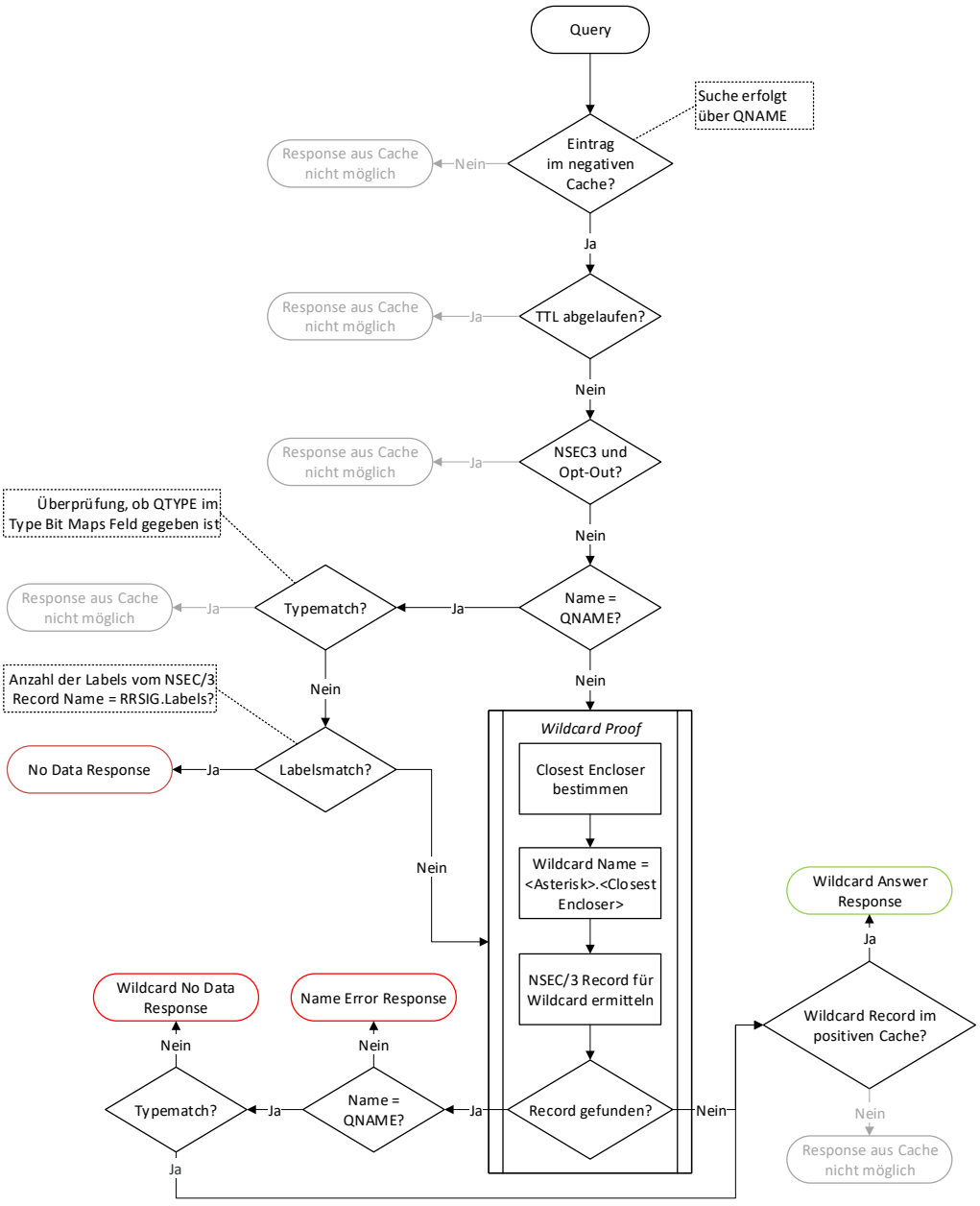

Abbildung 3.4: Ermittlung einer Response aus Sicht eines Resolvers mit aggressivem Cache Verhalten

Labelsmatch Bei diesem Test, der in dieser Arbeit *Labelsmatch* genannt wird, wird die Anzahl an Labels im Namen des NSEC/3 Records mit dem Labels Feld des zugehörigen RRSIG Record verglichen. Wenn sie gleich ist, antwortet der simulierte aggressive Resolver mit 'No Data Response'. Ansonsten erfolgt der Wildcard Proof.

Wildcard Proof Zwei Schritte zurück: Der Name des Records entspricht nicht dem QNAME, daher muss auf einen abgedeckten Treffer überprüft werden. Sofern ein NSEC/3 Record gegeben ist, muss ein Wildcard Proof durchgeführt werden. Ebenso falls der bereits erwähnte Labelsmatch fehlgeschlagen ist. Der abgedeckte Treffer entspricht dem 'Nein' bei 'Name = QNAME?' in Abbildung 3.4. Denn wenn die Nichtexistenz des QNAME weder durch einen direkten, noch durch einen abgedeckten Treffer, nachgewiesen werden kann, ist die Antwort eine 'No Response'. Dieser dritte Fall ist zur Vereinfachung der Abbildung nicht dargestellt.
Der Wildcard Proof beginnt mit der Bestimmung des Closest Encloser. Zusammen mit dem Asterisk kann daraus der Wildcard Name gebildet werden. Anschließend wird für den gebildeten Namen ein NSEC/3 Record ermittelt. Dabei erfolgt die Suche nach einem direkten oder abdeckenden NSEC/3 Record im negativen Cache wie für jeden anderen QNAME.

Verarbeitung des Wildcard Proof Es wurde ein NSEC/3 Record für den Wildcard Name gefunden. Entspricht der Name dessen nicht dem Wildcard Name, so ist es ein abgedeckter Treffer. Der Resolver antwortet mit einer 'Name Error Response'. Ansonsten ist es ein direkter Treffer. Es wird, wie bereits beim direkten Treffer für den ursprünglichen QNAME, der Typ überprüft. Im negativen Fall lautet die Antwort 'Wildcard No Data Response'. Dies bedeutet, es ist eine Wildcard vorhanden, aber mit dem falschen Typ.
Im positiven Fall des Typematch, sowie wenn kein NSEC/3 Record für den Wildcard Name gefunden wurde, findet eine Suche nach einem Wildcard Record im positiven Cache statt. Denn entweder gab es einen direkten Treffer, aber mit Übereinstimmung des Typs oder es gab keinen Treffer. Also könnte es potenziell eine Wildcard geben. Jedoch kann der Resolver nur mit 'Wildcard Answer Response' antworten, sofern auch der entsprechende Record im positiven Cache gespeichert ist. 'Entsprechend' heißt hier, dass es einen Record unter diesem Wildcard Name gibt, die TTL des Records nicht abgelaufen ist und sowohl der QTYPE mit dem Typ des Records übereinstimmt, als auch der QNAME zum Namen passt.

Response Type	NSEC	NSEC3
No Data	direkter Treffer + QTYPE nicht in Type Bit Maps	direkter Treffer + QTYPE nicht in Type Bit Maps (+ QTYPE ist nicht DS)
Name Error	abgedeckter Treffer für QNAME abgedeckter Treffer für Wildcard	Closest Encloser Proof abgedeckter Treffer für Wildcard
Wildcard No Data	abgedeckter Treffer für QNAME direkter Treffer für Wildcard + QTYPE nicht in Type Bit Maps	Closest Encloser Proof direkter Treffer für Wildcard Name + QTYPE nicht in Type Bit Maps
Wildcard Answer	abgedeckter Treffer für QNAME Wildcard Record existiert für QNAME und QTYPE	abgedeckter Treffer für Next Closer Name Wildcard Record existiert für QNAME und QTYPE
Answer Response	*gilt nur für den gewöhnlichen Resolver* *übereinstimmende Antwort in Answer Section*	
No Reponse	*gilt nur für den simulierten aggressiven Resolver* *keine Antwortermittlung möglich*	

Tabelle 3.1: Übersicht über die Response Types [8] [12]

3.2.2.3 Response Types

Aus den beschriebenen Verhalten eines Nameservers und des simulierten aggressiven Resolvers ergeben sich vier definierte *Response Types*. Zusätzlich zu diesen wurden zwei weitere Types für die Simulation definiert. Ihr Namen lauten *Answer Response* und *No Response*. In Tabelle 3.1 ist eine genaue Übersicht über die Response Types und ihre Bestimmungsregeln zu finden.

Eine *Answer Response* liegt vor, wenn der oder die angefragte/n RR gefunden werden konnten. Dieser Response Type kann nicht durch den simulierten aggressiven Resolver erzeugt werden. Eine *No Response* gilt für den Fall, dass der simulierte aggressive Resolver keine Antwort auf Basis des Caches bilden konnte.

3.2.3 Sliding Window

Ein Sliding Window ist i. A. ein Fenster, dass momentan vorgehaltene Datenpakete beinhaltet. Es 'schiebt' (engl. slide) sich so nach und nach über die Pakete. Abbildung 3.5 zeigt beispielhaft, wie das Sliding Window von Links nach rechts verschoben wird. Zuerst wird Paket #3 entfernt und Paket #7 eingelesen. Danach wird Paket #5 entfernt und Pakete #8 und #9 werden hinzugefügt.

Demnach hat das hier genutzte Sliding Window keine strenge Reihenfolge, in der die Pakete abgearbeitet werden müssen. In einem anderen Zusammenhang oder in anderen

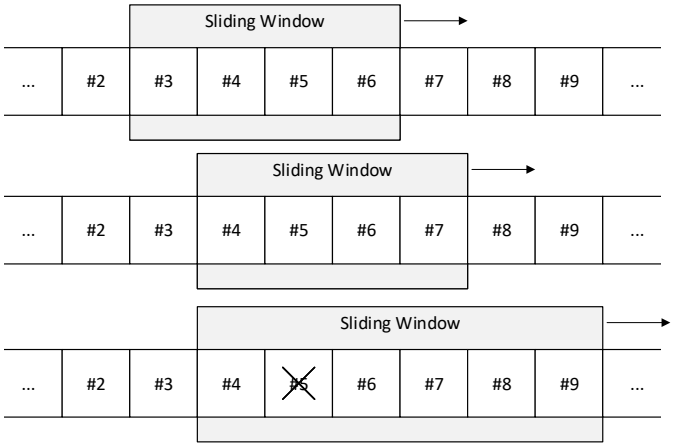

Abbildung 3.5: Vorgang eines Sliding Window

Konzepten kann es Abweichungen zu dieser Definition geben. Für das hier gegebene Szenario genügt sie.

Im Kontext dieser Simulation speichert das Sliding Window die Anfragen der Clients an den Resolver zusammen mit der auf Grundlage des simulierten Caches generierten Antwort. Wenn zu einem späteren Zeitpunkt die zugehörige Antwort des Resolvers an den Client eingelesen wird, evaluiert das Programm die zuvor ermittelte Antwort, vergleicht sie mit der des realen Resolvers und entfernt das Paket aus dem Sliding Window. Das Sliding Window befindet sich sozusagen zwischen den Clients und dem Resolver, wie es der grüne Bereich in Abbildung 3.3 veranschaulicht. Falls nach einer gewissen Zeit, welche anhand der Zeitstempel der Pakete gemessen wird, keine Antwort zu einer Anfrage eingelesen wird, sollte das entsprechende Element entfernt werden, da mit großer Wahrscheinlichkeit keine Antwort mehr kommen wird. Dadurch wird verhindert, dass das Sliding Window zu stark anwächst.

3.2.4 Simulation Cache

Der Cache dieser Simulation, schlichtweg Simulation Cache genannt, betrachtet alle Response Nachrichten, die vom externen Netzwerk an den Resolver gesendet werden und speichert die gewünschten RR. Er ist durch einen roten Bereich in Abbildung 3.3 erkennbar. Der Simulation Cache setzt sich aus einem negativen und einem positiven Cache zusammen.

3.2.4.1 Negativer Cache

Im negativen Cache werden alle NSEC und NSEC3 RR, die der Resolver vom externen Netzwerk als Response erhält, gespeichert. Diese RR, falls sie in der DNS Nachricht vorhanden sind, werden der Answer und der Authority Section entnommen.

Schlüssel Der Schlüssel für einen Eintrag im Cache ergibt sich im Fall von NSEC aus dem Besitzernamen und dem Namen im Next Feld und im Fall von NSEC3 aus dem Namen der Query und dem des NSEC3 Records. Er wird für jeden RR neu erzeugt und aus dem gemeinsamen Suffix der beiden Namen gebildet. Als Beispiel sei der QNAME `b.example.org` und die Response enthalte einen abdeckenden NSEC3 RR mit dem Namen `E7Q4EUC85B.example.org`. Daraus ergibt sich der Schlüssel `example.org`. Mehrere Records können den selben Schlüssel haben, daher besteht jeder Eintrag aus einer Liste von Records.

Eintrag Ein Eintrag im negativen Cache kann als Liste von Quadrupeln verstanden werden. Ein Quadrupel wird aus NSEC/3 RR, dem Zeitstempel seiner Ankunft in Unixzeit, seiner ermittelten TTL und dem zugehörigen RRSIG RR zusammengesetzt. Beispielsweise könnte basierend auf Abbildung 2.5 unter `example.org` folgender Eintrag gegeben sein:

(1)	`example.org`	`NSEC`	`a.example.org`	`...`
	Arrival: 1379053930			
	TTL: 10800			
	`example.org`	`RRSIG`	`...`	
(2)	`z.a.example.org`	`NSEC`	`zABC.a.EXAMPLE.org`	`...`
	Arrival: 1379054516			
	TTL: 7200			
	`z.a.example.org`	`RRSIG`	`...`	

(Reduzierte Darstellung der RR)

3.2.4.2 Positiver Cache

Der positive Cache speichert positive Antworten, in dem Fall ausschließlich Wildcard Responses, die vom externen Netzwerk an den Resolver gesendet worden sind.

Schlüssel Für den Schlüssel des Positiven Caches werden der Wildcard Record, der vom NS synthetisiert wurde, und der zugehörige RRSIG Record benötigt. Das Labels Feld des RRSIG Records gibt an, wie viele Labels vom Namen des Records von rechts ausgehend für den Schlüssel übernommen werden. Er wird auf der linken Seite noch um ein Asterisk und einen Punkt ergänzt.
Als Beispiel: Der Name des synthetisierten Records lautet `a.example.org` und im Labels Feld des RRSIG Records steht eine Zwei. Der Schlüssel für den positiven Cache ergibt sich aus den letzten beiden Labels `example` und `org` zusammen mit einem voranstehenden Asterisk. Er lautet damit `*.example.org`, wie der ursprüngliche Name der Wildcard.

Eintrag Die Einträge im positiven Cache sind im Vergleich zum negativen Cache keine Listen. Es wird lediglich der Wildcard Record zusammen mit dem Zeitstempel seiner Ankunft abgespeichert. Mehr wird nicht benötigt.

3.2.4.3 Im Cache suchen

Negativer Cache Um im negativen Cache einen Eintrag zu finden, wird der QNAME verwendet. Zunächst wird der vollständige Name als Schlüssel genutzt. Wenn kein Treffer erzielt wurde, wird der QNAME um ein Label von links gekürzt und erneut gesucht. Diese Prozedur wird so lange wiederholt bis es einen Treffer gab oder der QNAME leer ist. Im letzten Schritt wird das leere Label überprüft. Denn es kann auch unter '.' einen Eintrag für die TLD geben. Am Ende erhält man eine List von NSEC/3 Resource Records.

Positiver Cache Ein Suche im positiven Cache findet erst statt, wenn es einen Treffer im negativen Cache gegeben hat. Basierend auf dem gefundenen NSEC/3 RR wird dann der Name der Wildcard ermittelt. Da der Schlüssel für den positiven Cache auf der Grundlage von Wildcard Names generiert wird, genügt eine direkte Suche über den zuvor ermittelten Namen der Wildcard. Das Ergebnis der Suche ist, bei einem Treffer, ein einzelner Wildcard Record.

3.2.5 Nachrichten filtern

Abbildung 3.3 visualisiert, wie Sliding Window und Cache den Netzwerkverkehr an unterschiedlichen Stellen im Netzwerk betrachten. Jedoch sind alle Pakete zusammen in einer Datei und in der Reihenfolge, in der sie über das Netz versendet wurden, dokumentiert. Deshalb müssen sie gefiltert werden. Unabhängig von den Betrachtungspunkten

Verbindung	QR Flag	Ursprung	Ziel
Query Client an Resolver	0	interne Adresse	Resolver Adresse
Response Resolver an Client	1	Resolver Adresse	interne Adresse
Query Resolver an Nameserver	0	Resolver Adresse	externe Adresse
Response Nameserver an Resolver	1	externe Adresse	Resolver Adresse

Tabelle 3.2: Filterkriterien für relevante Nachrichten

lassen sich vier Verbindungsstränge ausmachen: Internes Netzwerk Richtung Resolver, Resolver Richtung externes Netzwerk, externes Netzwerk Richtung Resolver und Resolver Richtung internes Netzwerk. Kriterien für die Unterscheidung sind IP-Adresse des Ziels und des Ursprungs. Jeder Strang kann außerdem noch anhand der DNS Nachricht in Query und Response unterteilt werden, dargestellt in Abbildung 3.3 durch violette bzw. orange Pfeile. So kommt man auf insgesamt acht Verbindungsstränge

3.2.5.1 Relevante Nachrichten

Vier der insgesamt acht Verbindungsstränge werden im Normalfall erwartet. Sie sind in Abbildung 3.3 an den durchgängigen Pfeilen zu erkennen. Drei von ihnen sind für die Analyse bedeutsam: Clients $\overset{\text{Query}}{\longrightarrow}$ Resolver, Nameserver $\overset{\text{Response}}{\longrightarrow}$ Resolver und Resolver $\overset{\text{Response}}{\longrightarrow}$ Clients. Über die IP-Adressen von Ziel und Ursprung können sie voneinander unterschieden werden.

Tabelle 3.2 besagt, dass für eine Query von Client an Resolver folgendes gilt: Das QR Flag im Header der DNS Nachricht ist nicht gesetzt, die IP-Adresse des Ursprungs wird als intern erkannt und die IP-Adresse des Ziels stimmt mit der des Resolvers überein.

Die direkte Response von Resolver an Client wird umgekehrt erkannt, also gilt: Das QR Flag ist gesetzt, die IP-Adresse des Ursprungs stimmt mit der des Resolvers überein und die IP-Adresse des Ziels wird als intern erkannt.

Auf die gleiche Art und Weise wird eine Response von einem Nameserver an den Resolver identifiziert. Das QR Flag ist gesetzt, die IP-Adresse des Ursprungs wird als extern erkannt und das Ziel ist die Adresse des Resolvers.

Eine Query von Resolver an Nameserver gehört zu den erwarteten Verbindungen, wird aber von der Simulation nicht genauer betrachtet und ist daher in Tabelle 3.2 grau.

Verbindung	QR Flag	Ursprung	Ziel
Response Intern an Resolver	1	interne Adresse	Resolver Adresse
Query Resolver an Intern	0	Resolver Adresse	interne Adresse
Response Resolver an Extern	1	Resolver Adresse	externe Adresse
Query Extern an Resolver	0	externe Adresse	Resolver Adresse

Tabelle 3.3: Filterkriterien für irrelevante Nachrichten

3.2.5.2 Irrelevante Nachrichten

Die übrigen Verbindungsstränge sind für die Simulation uninteressant. Vier von ihnen werden im Normalfall nicht erwartet und sind in Abbildung 3.3 an den gestrichelten Pfeilen auszumachen. Diese sind aus Sicht des Resolvers Verbindungen in die gleichen Richtungen wie zuvor, jedoch mit umgekehrtem QR Flag.

Genauer gesagt bedeutet das, es wäre theoretisch möglich, dass der Resolver eine Query in das interne Netzwerk verschickt. Hierbei könnte es sich um interne, netzwerkspezifische Dienste oder einen weiteren (sekundären) Resolver handeln. Bei letzterem könnten auch Antworten zurück kommen, sprich eine Response aus dem internen Netzwerk an den Resolver. Genauso kann die Verbindung von Resolver nach außen erneut betrachtet werden: Es kann eine Query von außerhalb an den Resolver mit entsprechender Response geben.

Für die eigentliche Simulation sind alle vier Fälle irrelevant, da nur das Cache Verhalten in Bezug auf Anfragen aus dem internen Netzwerk untersucht werden soll. Darüber hinaus sind es ungewöhnliche Fälle die im Normalfall nicht auftreten sollten, insbesondere in Richtung des internen Netzwerks. Dennoch werden alle Fälle gefiltert und zur Überprüfung der Filterkriterien protokolliert. Die genauen Kriterien für irrelevante Nachrichten können der Tabelle 3.3 entnommen werden.

3.2.6 Durchführung der Simulation

Nachdem nun die wichtigsten Komponenten und der Ablauf der Überprüfung des simulierten Caches erklärt wurden, folgt ein Überblick über die einzelnen Schritte, die für die Durchführung der Simulation notwendig sind. Somit wird auch das Zusammenwirken der einzelnen Komponenten deutlich.

Schritt 1 - Einlesen Die Datei, die den Mitschnitt des Netzwerkverkehrs enthält, wird über einen Stream eingelesen. Wie dies konkret geschieht, ist vom Dateiformat und

der Implementierung abhängig. Jedes Paket wird einzeln verarbeitet und folgende Informationen werden eingelesen: Ziel- und Ursprungsadressen und -ports, sowie Ankunftszeit und die DNS Nachricht.

Schritt 2 - Filtern Anhand der vorhandenen Informationen und den in Abschnitt 3.2.5 genannten Kriterien, erfolgt eine Unterscheidung der Pakete. Daraus resultieren verschiedene Abhandlungen des momentan vorliegenden Pakets.

Schritt 3 - Abhandeln Je nachdem aus welchem Bereich des Netzwerks das Paket stammt, wird es, wie in Abbildung 3.3 zu sehen, anders behandelt.

Schritt 3a - Simulation Cache aufbauen Wenn es sich bei dem momentan zu verarbeitenden Paket um eine Antwort von außerhalb an den Resolver handelt, so werden alle NSEC und NSEC3 Records, soweit in der DNS Nachricht vorhanden, samt benötigten Zusatzinformationen zum negativen Cache hinzugefügt. Im Fall einer Wildcard Response, wird der Record zum positiven Cache hinzugefügt.
Bei bereits vorhandenen Records, also Duplikaten, werden in beiden Fällen die alten Records durch die neuen ersetzt.

Schritt 3b - Sliding Window aufbauen Das aktuelle Paket ist eine Query aus dem internen Netzwerk an den Resolver. Daher wird überprüft, ob der Simulation Cache eine Antwort für die Question liefern kann. Da das Ergebnis bei der Suche im negativen Cache eine Liste ist, wird über diese Liste iteriert, bis die Kriterien in Abschnitt 3.2.2.2 erfüllt sind und einer der in Abschnitt 3.2.2.3 erläuterten Response Types als Antwort geliefert wird. Erst wenn kein passender NSEC/3 Record gefunden wurde und die Liste abgearbeitet ist, lautet die Antwort 'No Response'.
Die Question wird dann zusammen mit dem ermittelten Response Type dem Sliding Window hinzugefügt.

Schritt 3c - Sliding Window abbauen Das Sliding Window wird wieder abgebaut, wenn das derzeitige Paket eine Antwort vom Resolver an das interne Netzwerk ist. Die zuvor gespeicherte Query wird herausgesucht, um an dieser Stelle die Latenzberechnung durchzuführen. Zudem werden hier relevante Statistiken erfasst und auch der Response Type der Antwort ermittelt und mit der Antwort des simulierten aggressiven Resolvers verglichen.

3.2.7 Verifizierung der Antwort

Der Response Type, den der simulierte aggressive Resolver als Antwort liefert, sollte verifiziert werden. Es ist möglich, dass der simulierte aggressive Resolver und der reale gewöhnliche Resolver zu verschiedenen Ergebnissen gelangen, da sich z. B. die Daten auf dem NS geändert haben können und sich nun zu denen im Cache des simulierten aggressiven Resolvers unterscheiden.

Um eine Verifizierung durchführen zu können, muss der Response Type der Antwort ermittelt und anschließend mit dem des simulierten aggressiven Resolvers abgeglichen werden.

3.2.7.1 Response Type der Antwort ermitteln

Ähnlich wie der Response Type, der im Simulation Cache auf Basis der gefunden Resource Records ermittelt wird, kann der Response Type auch anhand der Antwort des gewöhnlichen Resolvers bestimmt werden:

- Ein Wildcard Record befindet sich in der Answer Section der Nachricht. Zu erkennen ist dies durch das Labels Feld des RRSIG Records. Der Response Type lautet 'Wildcard Answer Response'.
- In der Authority Section ist ein NSEC/3 RR zu finden, der ein direkter Treffer für den QNAME ist und dessen Type Bit Maps Feld nicht den QTYPE enthält. Es handelt sich um eine 'No Data Response'.
- Einer der NSEC RR hat einen Wildcard Name als Besitzernamen oder einer der NSEC3 RR ist ein direkter Treffer für den generierten Wildcard Name, allerdings passt in beiden Fällen der Typ nicht. Die Nachricht ist eine 'Wildcard No Data Response'.
- Es sind NSEC/3 RR in der Authority Section vorhanden, aber die zuvor genannten Bedingungen sind nicht erfüllt. Somit ist es eine 'Name Error Response'.
- Die Answer Section listet den oder die angeforderten RR auf. Es gab also einen Treffer. Dies wird 'Answer Response' genannte.

In Sonderfällen kann es vorkommen, dass die Antwort leer ist. Sie beinhaltet also keine RR in der Answer Section und keine NSEC/3 RR in der Authority Section. Diese Möglichkeit sollte abgefangen werden und wird in dieser Arbeit als 'Empty Response' bezeichnet.

3.2.7.2 Abgleich der Response Types

Nachdem der Response Type der Antwort ermittelt wurde, kann er mit dem zuvor vom simulierten aggressiven Resolver bestimmten Response Type verglichen werden. Bei einer Übereinstimmung ist die erhaltene Response durch den aggressiv genutzten Cache auf jeden Fall korrekt. Andernfalls muss genauer differenziert werden.

Sofern der simulierte aggressive Resolver eine 'No Response' als Antwort geliefert hat, liegt kein Fehlerfall vor. Es ist schließlich möglich, dass die Informationen aus dem Cache nicht ausreichend waren, um eine konkrete Antwort zu bilden. In jedem anderen Fall muss der simulierte aggressive Resolver einen Fehler gemacht haben, da er eine positive oder negative Antwort generieren konnte, die nicht mit der Antwort des realen Resolvers übereinstimmt. Eine Ausnahm kann es aber geben, denn es kann, wie bereits erwähnt, daran liegen, dass sich die Daten auf dem NS verändert haben.

3.3 Erfassung von Statistiken

Damit die Effizienz der aggressiven Nutzung von NSEC/NSEC3 Caching beurteilt werden kann, ist es erforderlich gewisse Statistiken zu erfassen und anschließend auszuwerten. In diesem Abschnitt wird begründet dargelegt, an welchen Stellen, welche Statistiken erfasst und nicht erfasst werden können. Die erste Unterteilung erfolgt dadurch, welche relevant und welche vernachlässigbar sind.

3.3.1 Relevante Statistiken

Zu den relevanten Statistiken gehört das Zählen aller vorgekommenen Pakete je nach Verbindungsstrang, aber vor allem von Anfragen und Antworten zwischen den Clients und dem Resolver, da darauf der Fokus liegt. Diese Pakete können direkt nach der Anwendung des Filters gezählt werden.

Ebenso gehört die Anzahl der aufgetretenen Fehler dazu. Hier kann zwischen Implementierungsfehlern und fehlerhaften DNS Nachrichten unterschieden werden. Eine genaue Lokalisation für die Erhebung dieser Statistik gibt es nicht, da sie von der Implementierung abhängt. Fehlerhafte DNS Nachrichten können beim Einlesen der DNS Nachricht erfasst werden.

Genauso wichtig ist es zu protokollieren, wie viele NSEC3 Records Opt-Out nutzen und dadurch die aggressive Nutzung verhindern. Dies trifft eine Aussage darüber, wie

bedeutsam die gemessene Effizienzsteigerung ist, wenn denn eine zu verzeichnen ist. Zusätzlich wird die Anzahl aller in den Simulation Cache aufgenommenen Wildcard, NSEC und NSEC3 Records festgehalten.

Des Weiteren können *false Positives* in Bezug auf das Cache Verhalten durch die Verifizierung der Antwort festgestellt werden. False Positive bedeutet, dass der simulierte aggressive Resolver eine positive Antwort, also alles außer 'No Response', geliefert hat und der Response Type dieser Antwort nicht mit dem Response Type aus der realen Antwort übereinstimmt. Dabei ist allerdings zu beachten, dass die Verifikation nicht immer korrekt sein muss, da eine Veränderung der Zonendaten innerhalb der TTL zu einem falschen Ergebnis führen kann. Dennoch werden die 'false Positives' gezählt.

Die wohl wichtigsten und interessantesten Werte sind jedoch die Latenzzeiten. Die Latenz ergibt sich aus der Differenz der beiden Zeitstempel von Query und Response. Wenn eine Response durch den simulierten Cache erzeugt werden kann, wird künstlich eine Latenz von Null angenommen, da die Latenz für einen im Cache vorliegenden RR für gewöhnlich unter einer Millisekunde liegt. Bei der Erfassung der Latenzzeiten kann noch dahingehend differenziert werden, ob der ermittelte Response Type mit dem wirklichen Response Type übereinstimmt. Eine genaue Beschreibung und die Bedeutung der verschiedenen, gemessenen Latenzzeiten und deren Einordnung ist Kapitel 5 zu entnehmen.

3.3.2 Vernachlässigte Statistiken

Statistiken, die nicht relevant sind oder nicht aufgezeichnet werden können, werden vernachlässigt. Dazu gehören die Verbindungsstränge aus Abschnitt 3.2.5, die als unerwartet deklariert wurden. So ist bspw. die Anzahl der Anfragen, die von außerhalb stammen und an den Resolver gesendet werden, uninteressant. Sie kann dokumentiert werden, wird aber nicht genauer betrachtet.

Nicht erfassbar sind false Negatives in Bezug auf das simulierte Cache Verhalten, also wie oft der simulierte aggressive Resolver bei der Antwort 'No Response' nicht richtig lag und evtl. doch eine Antwort hätte ermitteln können. Diese Art der Bewertung der Antwort kann nicht getroffen werden, da es so einen Resolver noch nicht gibt und keine Vergleichswerte vorhanden sind.

4 Implementierung

Dieses Kapitel beschreibt grob, wie das Konzept implementiert worden ist. Es beginnt mit allgemeinen Informationen, wie die genutzte Programmiersprache und Programmbibliotheken sowie angewendete Frameworks oder Tools. Danach folgt eine Übersicht der wichtigsten Klassen. Ihr Aufbau und wie sie zusammenarbeiten wird erläutert.

4.1 Allgemeines

Dieser Abschnitt behandelt allgemeine Details zur Implementierung. Außerdem wird in ihm dargelegt, wie der Netzwerkmitschnitt des Resolvers im Szenario dieser Arbeit verarbeitet bzw. gelesen wurde. Das Kapitel Konzept beinhaltet nicht, wie die Daten eingelesen werden sollen, da dies vom eingesetzten Dateiformat, evtl. besonderen Vorverarbeitungsschritten und auch der Implementierung abhängt.

4.1.1 Implementierungsdetails

Auf Grund dessen, dass die einzulesenden Daten auf einem anderen Gerät zu finden sind als auf dem die Entwicklung der Software stattgefunden hat[1] und die Performance des Programms nicht komplett unerheblich gewesen ist, wurde die Implementierung in Java geschrieben, da Java durch seinen JIT-Compiler eine schnelle Datenauswertung erwarten lässt und man plattformunabhängigen Programmcode erhält. Auf beiden Systemen wurde JDK in der Version 1.8.0_162 verwendet.

Zu den genutzten Bibliotheken gehören java.io und java.nio zum Einlesen der Daten, java.util für Objekte und Funktionen, die jegliche Art von Liste betreffen (Arrays, Collections, Maps usw.), java.security um Hashfunktionen anwenden zu können und javax.xml.bind zur Umwandlung von Byte-Arrays in Hexadezimalstrings, aber vor allem die zusätzlich eingebundene Library org.xbill.DNS[2].

[1] siehe Abschnitt 5.1 Ursprung der Daten
[2] mehr Informationen unter http://www.dnsjava.org/

Diese Library ermöglicht neben dem Einlesen von kompletten DNS Nachrichten auch die Konvertierung aus dem Base32 Format in einen String und die Datenverwaltung von allen notwendigen DNS Komponenten im Sinne der objektorientierten Programmierung. Denn sie enthält jeweils eine Klasse für jeden RR Type. Ebenfalls ist eine Klasse für den Header einer DNS Nachricht, inklusive der Funktionen zum Auslesen aller relevanten Felder, vorhanden. Gleichermaßen nützlich ist die Klasse 'Name', die für die Namen der RR zuständig ist und einen Vergleich von Namen oder die Umwandlung eines Namens in seine kleingeschriebene Form ermöglicht. Ein Name-Objekt kann durch einen String, ein Byte-Array oder die Zusammensetzung zweier Name-Objekte konstruiert werden.

Durch die Verwendung dieser Programmbibliothek ist kein eigenständiges, byteweise Einlesen der DNS Nachricht notwendig. Die Klasse Message erwartet die Daten der DNS Nachricht lediglich als Byte-Array.

4.1.2 Datenverarbeitung

Der Netzwerkmitschnitt liegt im Dateiformat PCAP vor. Dafür gibt es zwar andere Programmbibliotheken, die das Einlesen ermöglichen, allerdings sind diese sehr groß und mit der Installation anderer Software verbunden. Außerdem müssten drei Layer durch einen Parser verarbeitet, IP-Fragmentierungen behoben und TCP-Streams rekonstruiert werden, weil es sich bei PCAP um einen rohen Netzwerkmitschnitt handelt. Dadurch würde das Verhältnis von Aufwand zu Nutzen zu groß werden.

Stattdessen wurde ein sogenanntes 'Reassemble DNS' Programm, geschrieben in Python, zur Vorverarbeitung der Daten ausgeführt. Es sorgt dafür, dass, unabhängig von UDP und TCP, fragmentierte Pakete zusammengeführt werden. Als Produkt erhält man eine Datei im eigens spezifizierten DNS Format. In diesem Format beseht jedes Paket aus Zeitstempel, IP-Version, IP-Adresse und Port des Ursprungs sowie des Ziels, Transport-Typ (UDP oder TCP) und Länge der DNS Nachricht, schlussendlich gefolgt von der DNS Nachricht.

Für das DNS-Dateiformat ist ein Parser entwickelt worden. Die Datei wird byteweise eingelesen und interpretiert. Der Bytebereich der DNS Nachricht kann einfach als Byte-Array an org.xbill.DNS.Message übergeben werden.

4.2 Aufbau der Klassen

Um den Aufbau der Implementierung grob zu skizzieren, enthält dieser Abschnitt die Klassendiagramme der wichtigsten Bestandteile. Komplexere Stellen im Programmablauf und die Zusammenhänge der Klassen werden erläutert.

Alle in den Darstellungen verwendeten Typen, die nicht auch als Klasse enthalten sind oder nicht zu den Standardtypen gehören, sind der Library org.xbill.DNS entnommen. Ein Ausnahme bildet `Statistics`. Dies ist eine Klasse, die für die Erfassung der Statistiken zuständig ist. Da sie aber keinen besonderen Bezug zu den anderen Komponenten besitzt, ist sie nicht in den Diagrammen festgehalten worden.

Die Klassendiagramme geben nicht jedes Detail der Implementierung wieder, da sie nur eine ungefähre Vorstellung geben sollen. Sie folgen dabei den allgemein gültigen Darstellungsformen von UML[3]:

- '+' repräsentiert public, '#' protected und '-' private
- Ein Block für eine Klasse besteht aus drei Bereichen: Klassenname, Variablen und Methoden.
- Abstrakte Klassen und Methoden sind kursiv gedruckt.

Constants Da einige wiederkehrende Werte verwendet werden, sind sie als Konstanten in der Klasse Constants implementiert. Hier eine Liste der im Verlauf des Kapitels genannten Konstanten:

`RESOLVER_ADDR` :	Die IP-Adresse des Resolvers
`INTERNAL_ADDR_OFFSET` :	Der erste Block der IP-Adressen aus dem internen Netzwerk
`MILLI_SECOND` =	0.001

4.2.1 Resource Records

Resource Records spielen eine essenzielle Rolle. Ein Überblick ihrer Implementierung ist in Abbildung 4.1 dargestellt. Die DNS Library stellt schon einige Methoden zur Verfügung, aber nicht alle die notwendig sind. Deswegen sind die beiden Klassen `EnhancedNSECRecord` und `EnhancedNSEC3Record` entwickelt worden, die die entsprechenden Klassen der Library erweitern. In beiden Fällen ergänzen sie die Basisklasse um die Methoden `getClosestEncloser`, `matches` und `inbetween`.

Alle Methoden arbeiten basierend auf dem NSEC/3 Record, über den sie aufgerufen worden. `getClosestEncloser`: Man erhält den Closest Encloser zu einem gegebenen

[3]UML steht für Unified Modeling Language

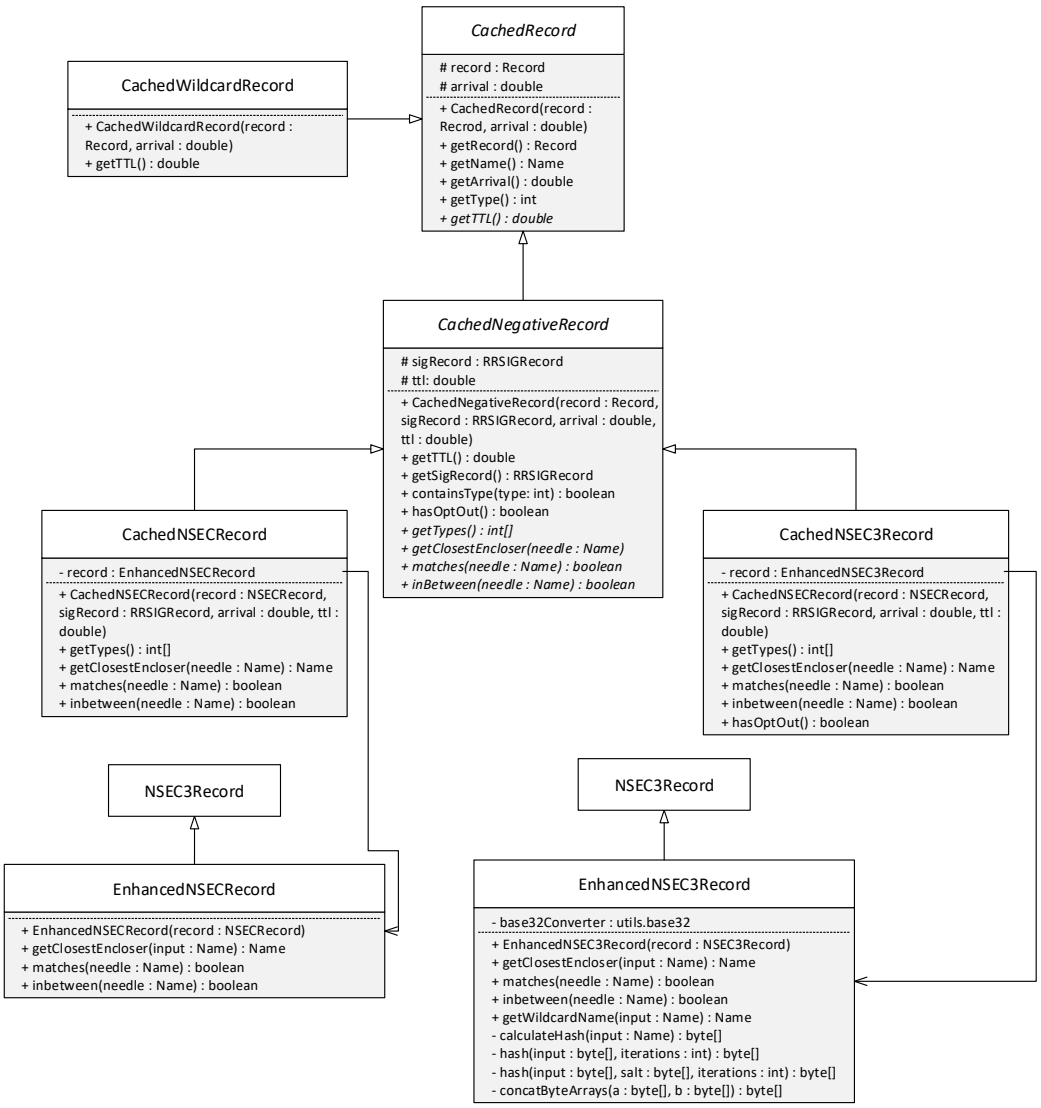

Abbildung 4.1: Vererbungshierarchie der Record Klassen

Namen. `matches`: Dieser Record ist ein direkter Treffer zu einem gegebenen Namen. `inbetween`: Dieser Record ist ein abgedeckter Treffer zu einem gegebenen Namen.

Für NSEC3 RR ist eine Erweiterung um die Hashfunktionalität obligatorisch gewesen. So kann der `EnhancedNSEC3Record` anhand seiner Hashparameter den Hashwert für einen gegebenen Namen berechnen. Dies ist sowohl für `matches`, als auch für `inbetween` ein Muss.

Neben den eben genannten Vererbungen gibt es auch unterschiedliche Implementierungen des `CachedRecord`, jeweils eine für jeden Cache, positiv und negativ, bzw. für jeden Record Type. Eine direkte wichtige Spezialisierung des `CachedRecord` ist der `CachedNegativeRecord`. Dieser ist zwar auch eine abstrakte Klasse, sorgt aber dafür, dass später die Methoden unabhängig von der konkreten Implementierung aufgerufen werden können. Diese Implementierungen sind `CachedNSECRecord` und `CachedNSEC3Record`, abhängig davon, ob ein NSEC oder ein NSEC3 Record im Cache gespeichert bzw. abgerufen werden soll. Das heißt über den Methodenaufruf `cachedNegativeRecord.matches(input)` kann ein direkter Treffer ermittelt werden ohne vorher überprüfen zu müssen, ob es sich bei dem aktuellen RR um einen NSEC oder NSEC3 RR handelt.

Beide Implementierungen des `CachedNegativeRecord` reichen die Aufrufe der Methoden an die erweiterten Recordklassen weiter und implementieren sie somit nicht selbst. Dieser Ansatz wurde so gewählt, damit die Methoden für die Bestimmung eines direkten bzw. abgedeckten Treffers unabhängig davon, ob sie im Cache vorhanden sind, angewendet werden können. Sie werden nämlich auch für die Verifizierung der Antwort benötigt.

4.2.2 Sliding Window

Der Aufbau des Sliding Window ist Abbildung 4.2 zu entnehmen. Der Schlüssel wird einmalig von außerhalb über `slidingWindow.setKey(ip, port)` gesetzt. Die übergebenen Parameter hängen davon ab, ob das aktuelle Paket eine Query von Intern an den Resolver oder eine Response vom Resolver an das interne Netzwerk ist. Auf diese Weise muss der Schlüssel für alle nachfolgenden Operationen nicht immer wieder übergeben oder ermittelt und lediglich in einer lokalen Variable gespeichert werden. Davon profitieren die Methoden `addEntry`, `getEntry` und `removeEntry`.

Das Sliding Window speichert die Anfragen in einer Hash Map als QueryEntry. Diese Klasse speichert die Question, den Ankunftszeitpunkt und den ermittelten Response Type des Simulation Cache.

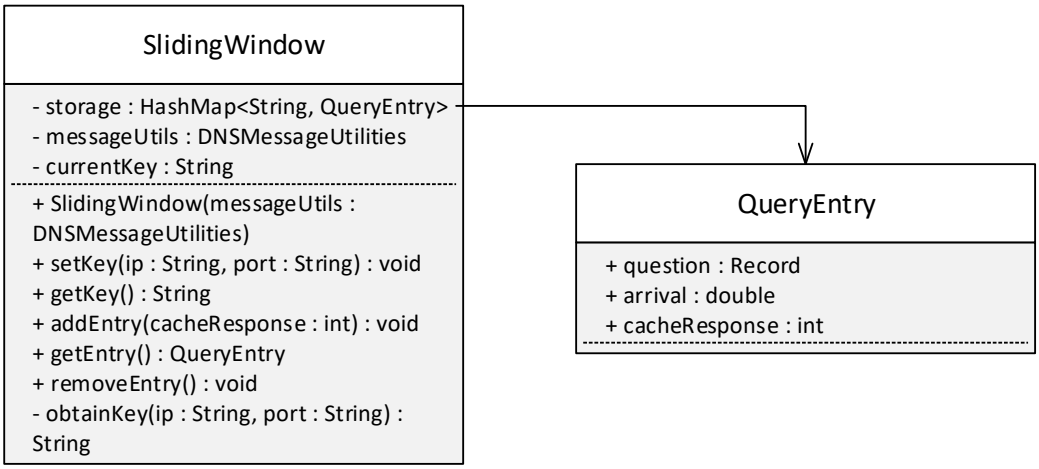

Abbildung 4.2: Aufbau des Sliding Window

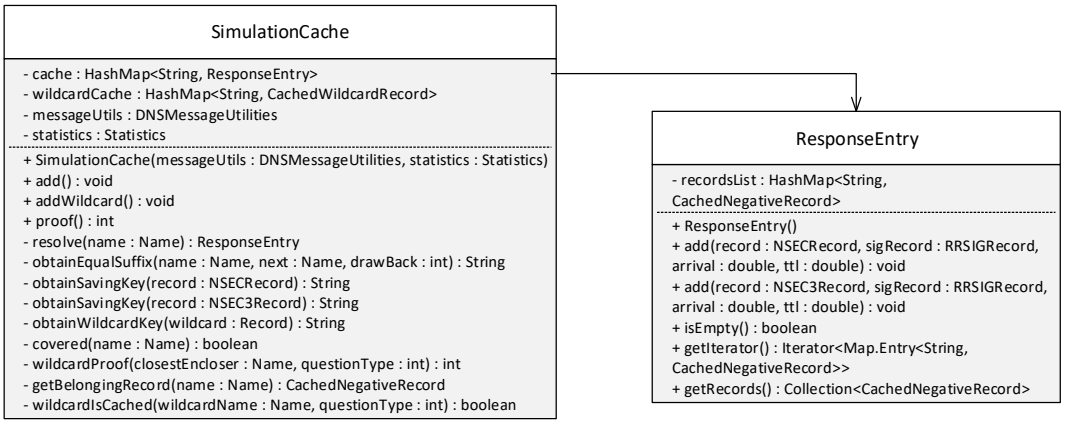

Abbildung 4.3: Aufbau des Simulation Cache

4.2.3 Simulation Cache

Der Simulation Cache ist ähnlich wie das Sliding Window aufgebaut. Abbildung 4.3 zeigt, dass auch hier eine Hash Map eingesetzt wird und äquivalent zum QueryEntry eine Klasse ResponseEntry existiert. Strukturell unterscheiden sich jedoch die beiden Klassen, da, wie in Abschnitt 3.2.4 erwähnt, ein Eintrag im Simulation Cache aus einer Liste von Records besteht. Dementsprechend besitzt der ResponseEntry Methoden, die meistens für Listen benötigt werden, wie `add`, `isEmpty` und `getIterator`.

Diese Hash Map beinhaltet aber nur NSEC und NSEC3 Records. Wildcards werden in einer separaten Hash Map als CachedWildcardRecord abgelegt.

Die wichtigsten Methoden des Simulation Caches sind folgende: `resolve` findet im negativen Cache den ResponseEntry für einen bestimmten Namen. Das Herzstück bildet `proof`, da es den Response Type ermittelt. `covered` funktioniert vom Ablauf sehr ähnlich, dabei wird allerdings ausschließlich überprüft, ob der gegebene Name von einem NSEC/3 RR abgedeckt wird oder nicht. Der in Abschnitt 3.2.2.2 beschriebene Wildcard Proof ist in der gleichnamigen Methode wiederzufinden.

Die meisten anderen Methoden sollten ebenfalls durch ihren Namen den in Kapitel 3 erläuterten Verfahren einfach zuzuordnen sein.

4.2.4 Verarbeitung der DNS Nachrichten

Nachdem die maßgeblichen Komponenten veranschaulicht wurden, geht es um den zentralen Verarbeitungspunkt. Alle Komponenten kommen zusammen und, wie in den Schritten in Abschnitt 3.2.6 aufgelistet, wird die DNS Nachricht eingelesen, nach den Filterkriterien eingeordnet und abgehandelt.

4.2.4.1 Einlesen, parsen und verfügbar machen

Die Simulation beginnt in der Klasse 'Simulator'. Im Konstruktor werden je eine Instanz von SimulationCache, SlidingWindow, DNSMessageUtilities und ein paar weiteren Klassen, die zum Schreiben von Logfiles notwendig sind, erstellt. In der einzigen öffentlichen Methode `readFile` erhält der Simulator den Pfad zur Datei und liest sie Byte für Byte über einen BufferedInputStream ein. Die notwendigen Informationen, wie bspw. IP-Adressen und Zeitstempel, werden verarbeitet und vorübergehend in lokalen Variablen gespeichert.

Der eigentliche Kernpunkt ist die Klasse 'DNSMessageUtilities'. Sie erhält zur kurzweiligen Speicherung den Zeitstempel des aktuellen Pakets und die durch die DNS Library verarbeitete DNS Nachricht. Die gleiche Instanz der DNSMessageUtilities wird an
SimulationCache und SlidingWindow weiter gereicht. Ziel war es einen überall verfügbaren Speicher der aktuellen Nachricht zu erhalten.

Das Prinzip der Datenhaltung ist dabei simpel gehalten: Die Instanz der DNSMessageUtilities wird durch erneutes setzen der DNS Nachricht zurückgesetzt, alle lokalen Klassenvariablen werden auf 'null' gesetzt. Beim Aufruf eines Getters wird der angefragte Wert gegen 'null' überprüft und ggf. durch den Setter gesetzt. Hierbei handelt es sich nicht um einen 'richtigen' Setter, da der Wert nicht von Außerhalb gesetzt werden

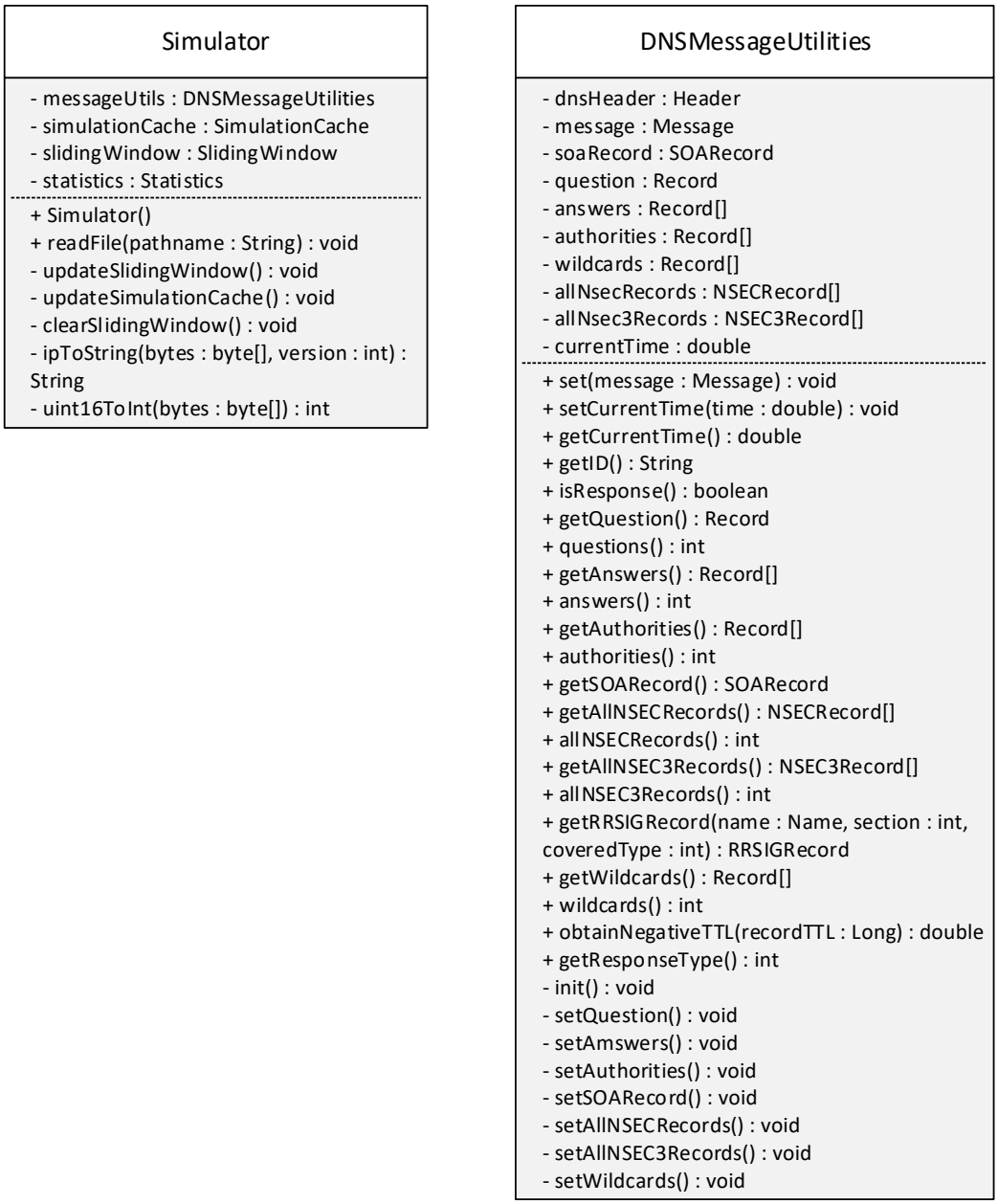

Abbildung 4.4: Simulator und DNSMessageUtilities als zentraler Dreh- und Angelpunkt

kann, sondern vielmehr um ein Setzen der lokalen Klassenvariable auf Basis der gegebenen DNS Nachrichten. Außerdem gibt es zu jeder Getter-Methode, die ein Array zurück gibt, eine weitere Methode, um die Anzahl der Elemente des Arrays zu erhalten.

4.2.4.2 Filtern und abhandeln

Der Simulator hat alle Bytes des aktuellen Pakets eingelesen, die DNS Nachricht der Library übergeben und die Message in den DNSMessageUtilities gespeichert. Nun beginnt die Filterung und Abhandlung der Pakete. Dazu einige Auszüge aus dem Programmcode.

Sliding Window aufbauen Das aktuelle Paket ist eine Query von einem Client an den Resolver. Dementsprechend muss das Sliding Window aufgebaut werden.

```
if ( !this.messageUtils.isResponse()
  && ipSrcString.startsWith(Constants.INTERNAL_ADDR_OFFSET)
  && ipDstString.equals(Constants.RESOLVER_ADDR) )
{
  this.statistics.countInternalQuery();
  this.slidingWindow.setKey(ipSrcString, portSrcString);
  updateSlidingWindow();
}
```

Im Code geschieht folgendes: Es wird für die Statistik gezählt. Anschließend wird der Schlüssel des Sliding Window gesetzt. Die Zuordnung von Query und Response im Sliding Window erfolgt über IP-Adressen, Ports und die Transaction ID. Der Schlüssel wird beim Hinzufügen aus Ursprungsadresse und -port plus Transaction ID zusammengesetzt. Danach wird die Methode `updateSlidingWindow` aufgerufen, welche wiederum nur Statistiken für Cache Hit und Miss dokumentiert und folgenden Befehl aufruft:

```
this.slidingWindow.addEntry(this.simulationCache.proof());
```

Sliding Window abbauen Im Fall einer Response vom Resolver an einen Client sieht es sehr ähnlich aus.

```
else if ( this.messageUtils.isResponse()
  && ipSrcString.equals(Constants.RESOLVER_ADDR)
  && ipDstString.startsWith(Constants.INTERNAL_ADDR_OFFSET) )
{
  this.statistics.countInternalResponse();
  this.slidingWindow.setKey(ipDstString, portDstString);
```

```
  clearSlidingWindow();
}
```

Der Code besagt: Für die Statistik zählen, Schlüssel setzen, dieses Mal mit Zieladresse und -port, und `clearSlidingWindow()` aufrufen. Diese Methode ist aber im Vergleich zu `updateSlidingWindow()` etwas umfangreicher. Sie holt sich das Element aus dem Sliding Window, prüft es gegen 'null' und führt die Verifizierung der Antwort[4] durch. Außerdem werden hier die Statistiken zu den Latenzzeiten erfasst.

Simulation Cache aufbauen Der letzte Punkt ist der Aufbau des Simulation Cache.

```
else if ( this.messageUtils.isResponse()
  && !ipSrcString.startsWith(Constants.INTERNAL_ADDR_OFFSET)
  && ipDstString.equals(Constants.RESOLVER_ADDR) )
{
  this.statistics.countExternalResponse();
  updateSimulationCache();
}
```

Wieder wird für die Statistik inkrementiert und diesmal `updateSimulationCache()` aufgerufen. In der Methode werden NSEC oder NSEC3 Records, wenn sie gegeben sind, über `simulationCache.add()` dem Cache hinzugefügt. Das gleiche gilt für Wildcards und `simulationCache.addWildcard()`.

4.2.5 Statistiken

Generelle Statistiken, wie Zähler zu den verschiedenen Paketen, werden in der Klasse 'Statistics' festgehalten. Zum Zählen und Protokollieren von Fehlern gibt es eine Klasse namens 'ExceptionCounter'. Diese Klassen sind trivial und werden nicht weiter erläutert. Relevant hingegen ist die Klasse 'LatencyStatistics', die in Statistics instanziiert und genutzt wird, um die berechneten Latenzen in Kategorien abzuspeichern.

Eine Latenzkategorie beschreibt einen bestimmten Zahlenbereich für die berechneten Latenzwerte. So werden alle Werte einer bestimmten Kategorie zugeordnet. Je nach Konfiguration wird bspw. jeder Wert ab Null und unter zwei Millisekunden der ersten Kategorie und jeder Wert ab zwei und unter vier Millisekunden der zweiten Kategorie zugewiesen usw.

[4] siehe Abschnitt 3.2.7 Verifizierung der Antwort

Konstruktor Der Konstruktor erwartet eine `timeRange`, die angibt wie groß der Messbereich für die Latenzkategorien maximal sein soll, und einen `timeSpread`, der die Weite der einzelnen Kategorien angibt. Daraus wird dann mit folgender Rechnung die Anzahl der Kategorien berechnet:

```
this.latencyCategories = new int[(timeRange/timeSpread)+1];
```

Es wird eine Kategorie mehr für alle Werte, für die $x \geq timeRange$ gilt, benötigt.

Für spätere Berechnungen sind beide Werte in Millisekunden zu verstehen:

```
this.latencyTimeRange = timeRange * Constants.MILLI_SECOND;
this.latencyTimeSpread = timeSpread * Constants.MILLI_SECOND;
```

Latenzwert speichern Nach Anlegen der Kategorien, kann über die Methode `add` die Latenz berechnet und ihrer Kategorie zugeordnet werden.

```
public void add(double current, double past) {
  double latency = current - past;

  if (latency < this.latencyTimeRange) {
    int i = (int) (latency / this.latencyTimeSpread);
    this.latencyCategories[i]++;
  } else {
    this.latencyCategories[this.latencyCategories.length - 1]++;
  }
}
```

Für die Zuweisung wird erst einmal die Differenz der beiden Zeitwerte gebildet, um die Latenz zu erhalten. Wenn die berechnete Latenz kleiner als der gesetzte Maximalwert `latencyTimeSpread` ist, wird der Schlüssel für das Array einfach aus der Division der Latenz und der Weite der Kategorien berechnet. Durch die Umformung zu Integer wird der Wert nicht gerundet, sondern die Nachkommastellen werden abgeschnitten. Die entsprechende Kategorie wird inkrementiert.
Im anderen Fall, also dass die Latenz größer oder gleich dem Maximalwert ist, wird einfach in der letzten Kategorie hochgezählt.

5 Auswertung

Das Auswertungskapitel klärt zu Beginn, wie die Daten für die Simulation erfasst worden sind. Der Hauptteil ist die Präsentation der Ergebnisse. Dabei wird zwischen den Werten von allgemeinen Zählern, verschiedenen Latenzmessungen und dem Zählen von Fehlern unterschieden. Zunächst wird für jeden dieser Punkte erläutert, wie die Werte entstanden sind. Danach werden die Ergebnisse beschrieben und ggf. grafisch dargestellt. Nach dem Hauptteil erfolgt eine Interpretation der Ergebnisse, indem Bezüge hergestellt und gedeutet werden.

5.1 Aufbau der Datenerfassung

In diesem Abschnitt werden Ursprung und Speicherort der analysierten Daten beleuchtet. Es wird beschrieben, woher die Daten stammen, wie sie erfasst und dokumentiert sowie wo und wie sie abgespeichert wurden.

Ursprung der Daten Im September 2013 hat das Zentrum für Informations- und Mediendienste der Universität Duisburg-Essen einen Mitschnitt des Netzwerkverkehrs des Resolvers der Universität für einen Zeitraum von ungefähr 16 Tagen erstellt. Die Daten wurden anonymisiert, indem die IP-Adressen geändert wurden ohne Informationen zu verfälschen. So sind die aus dem internen Netzwerk immer noch von denen von außerhalb zu unterscheiden. Ebenso ist die Adresse des Resolvers zur Wiedererkennung statisch geblieben.

Speicherort der Daten Trotz Anonymisierung sind die Daten aus Gründen des Datenschutzes nicht direkt ausgehändigt worden. Sie sind auf einem Server der Universität abgespeichert und durch einen Benutzernamen und ein Passwort geschützt.
Um also die Simulation durchzuführen und die gewünschten Werte für die Auswertung zu erfassen, ist ein Upload des Programms mit anschließender Ausführung auf dem Server notwendig gewesen. Die Ergebnisse sind in Textdateien dokumentiert und hinterher runtergeladen und ausgewertet worden.

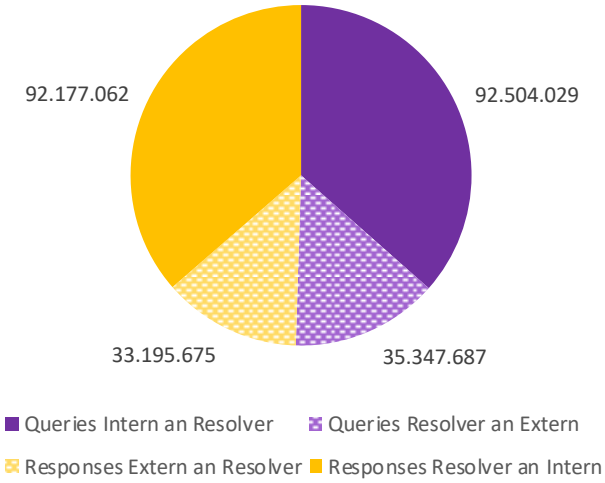

Abbildung 5.1: Verteilung der DNS Nachrichten auf die Verbindungsstränge im Netzwerk

5.2 Ergebnisse

Der Vorgang der Simulation sowie die Erfassung der Daten und Statistiken stehen fest. In diesem Abschnitt erfolgt eine Beschreibung und Auswertung der Ergebnisse. Fast alle im Text genannten Zahlenwerte sind als Rundungen zu verstehen.

5.2.1 Zähler

Während der Simulation wurden verschiedene Aspekte gezählt: Vom Parser gelesene DNS Nachrichten, vom Simulation Cache aufgenommene Resource Record sowie mögliche und auch richtige Antwortermittlungen.

DNS Nachrichten Jede Nachricht, die über die erwarteten und auch unerwarteten Verbindungsstränge versendet worden ist, die in Abbildung 3.3 als Pfeile zu sehen sind, ist gezählt worden. Die erwarteten Nachrichten sind in Abbildung 5.1 grafisch dargestellt. So hat es 92,5 Millionen Anfragen aus dem internen Netzwerk der Universität an den Resolver und umgekehrt 92,2 Millionen Antworten gegeben. Zwischen Resolver und externem Netzwerk sind 35,3 Millionen vom Resolver ausgehende Anfragen und 33,2 Millionen Antworten zum Resolver hin eingelesen worden. Zusammen macht das einen Gesamtverkehr von 253,2 Millionen DNS Nachrichten aus.

Bei den unerwarteten DNS Nachrichten sind 46 Anfragen zwischen Resolver und internem Netzwerk und 899 Antworten aus dem internen Netzwerk zum Resolver erfasst

worden. Es hat knapp 600.000 Anfragen von Extern an den Resolver sowie Antworten vom Resolver ins externe Netzwerk gegeben. Insgesamt sind es 1,2 Millionen unerwartete Nachrichten.

Aufbau des Caches Im Simulation Cache ist die Anzahl der hinzugefügten Resource Records, unabhängig davon, ob ein Record schon mal vorkam, gezählt worden. Während des kompletten Programmablaufs sind 432.000 NSEC und 7 Millionen NSEC3 Resource Records dem Cache hinzugefügt worden. Bei Wildcard Records sind es lediglich 737 Stück gewesen. Das bedeutet, dass 94% aller dem negativen Cache hinzugefügten Resource Records vom Typ NSEC3 gewesen sind. Nebenbei ist auch erfasst worden, dass 6,9 Millionen der NSEC3 Resource Records ein gesetztes Opt-Out Flag hatten. Dies entspricht einem Anteil von 99%.

Nutzung des Caches In Bezug auf die Nutzung des Caches ist gezählt worden, wie oft ein Zugriff erfolgt und ob er positiv (Hit) oder negativ (Miss) gewesen ist. Es hat insgesamt 92,5 Millionen Zugriffe gegeben. 80% aller Zugriffe auf den Cache haben einen Hit ergeben und somit ist jeder fünfte ein Miss gewesen.

Verifikation des Response Types Bei der Verifizierung des Response Types, beschrieben in Abschnitt 3.2.7 *Verifizierung der Antwort*, ist zwischen einer positiven Verifikation, also einer Übereinstimmung des Response Types, und false Positives unterschieden worden, was im Prinzip einer negativen Verifikation gleichkommt.
Die Anzahl der positiv verifizierten DNS Nachrichten des simulierten aggressiven Resolvers beträgt 5,1 Millionen. Den größeren Anteil von 93% machen jedoch die false Positives mit 69,1 Millionen dokumentierten, negativ verifizierten Nachrichten aus.

5.2.2 Latenzmessung

Die Erfassung der Latenz erfolgt in Kategorien von Null bis 2000 Millisekunden. Jede Kategorie ist eine Millisekunde groß. Das bedeutet es gibt 2001 Kategorien denen die berechneten Latenzwerte zugeordnet werden. Die letzte Kategorie ist für alle Latenzwerte ab 2000 ms vorgesehen. Ein Wert von z. B. 5,3 ms würde der sechsten Kategorie, die für 5 bis 5,9 ms zuständig ist, zugeordnet werden. Nach der Berechnung der Latenz einer jeden DNS Nachricht, die vom Resolver an einen Client gesendet wurde und der eine Anfrage zugeordnet werden konnte, wird einfach der Zähler in der entsprechenden Kategorie inkrementiert. So muss nicht jeder Latenzwert exakt bis zur letzten Nachkommastelle dokumentiert werden.

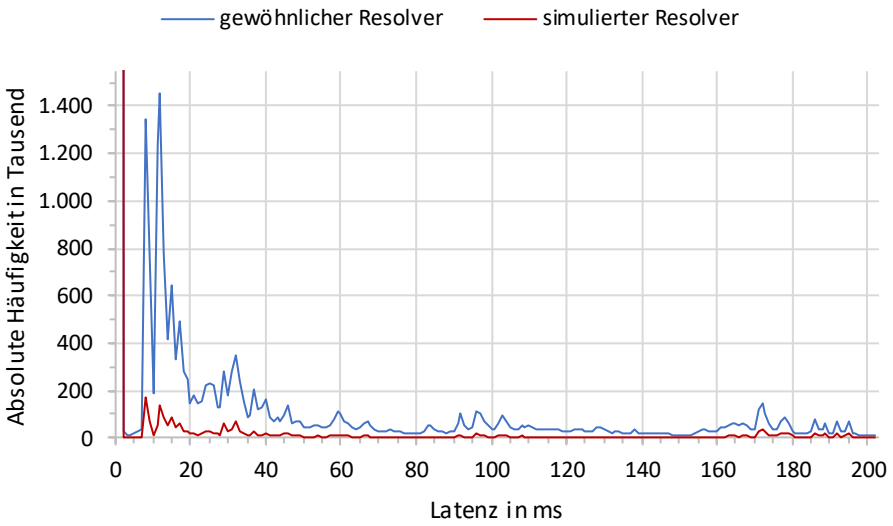

Abbildung 5.2: Ausschnitt aus der absoluten Häufigkeitsverteilung der Latenzkategorien des gewöhnlichen und des simulierten Resolvers im Vergleich

Unter drei verschiedenen Aspekten erfolgt die Dokumentation und Darstellung der Latenzverteilung. Als erstes ist die Latenz aller DNS Nachrichten zwischen dem Resolver und den Clients, denen eine Anfrage zugeordnet werden konnte, erfasst worden. Dies entspricht der Latenzverteilung eines gewöhnlichen Resolvers. Darauffolgend wird die Latenz des simulierten Resolvers beschrieben. Im Fall eines Cache Hit wird die erste Kategorie, die für 0 bis 0,9 ms zuständig ist, inkrementiert. Im gegenteiligen Fall eines Cache Miss wird wie zuvor verfahren. Die Latenz wird berechnet und dokumentiert. So wird die Latenzverteilung eines Resolvers mit aggressiver Nutzung von NSEC/NSEC3 Caching nachempfunden. Am Schluss steht die Latenzverteilung für die DNS Nachrichten, bei denen der simulierte aggressive Resolver den gleichen Response Type wie der gewöhnliche Resolver ermittelt hat. In anderen Worten: Die Verifizierung der Antwort war positiv. Automatisch bedeutet dies auch, dass es einen Cache Hit gegeben hat, denn die Antwort des gewöhnlichen Resolvers kann niemals 'No Response' sein.

Bei allen Darstellungen der Latenzverteilungen handelt es sich entweder um ein Liniendiagramm auf den direkten Daten oder um eine empirische Verteilungsfunktion. Letzteres bedeutet, dass jeder Kategorie aus der Latenzberechnung eine summierte Häufigkeit des Auftretens zugewiesen wird. Auf der x-Achse steht somit die Latenz und auf der y-Achse der relative Anteil aller betrachteten Latenzen, die diesem oder einem kleineren Latenzwert entsprechen.

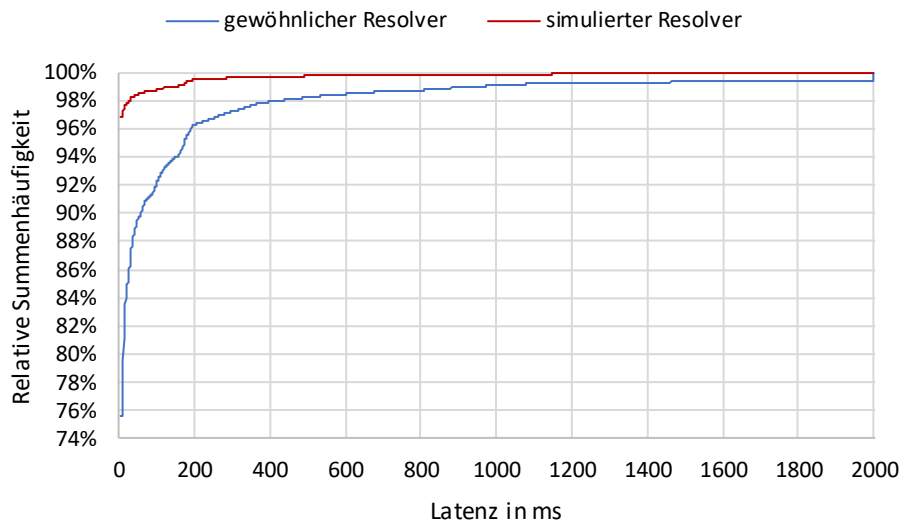

Abbildung 5.3: Relative aufsummierte Häufigkeitsverteilung der Latenzkategorien des gewöhnlichen und des simulierten Resolvers im Vergleich

5.2.2.1 Latenzen des gewöhnlichen Resolvers

Abbildung 5.2 zeigt die Latenzverteilung des gewöhnlichen Resolvers in absoluter Häufigkeit der Kategorien als blauen Graphen. In der Darstellung ist die Kategorie für alle Latenzen unter einer Millisekunde nach oben hin abgeschnitten, da sonst der restliche Verlauf beider Graphen nicht mehr zu erkennen wäre. Außerdem ist es ein Ausschnitt bis 200 ms, denn danach bleiben die Graphen fast konstant auf einer Höhe ohne große Schwankungen. Lediglich am Ende in der letzten Kategorie hätte es einen Anstieg gegeben, was daran liegt, dass alle Ergebnisse, die größer als 2000 ms sind, in ihr gesammelt werden.

Auch ohne die erste Kategorie, in der es für den gewöhnlichen Resolver 69,6 Millionen gezählte DNS Nachrichten gegeben hat, ist zu erkennen, dass die Anzahl der verarbeiteten DNS Nachrichten mit steigender Latenz stark abnimmt. Zu Beginn schwankt die Anzahl etwas. Zwischen 7 und 7,9 ms sowie 10 und 11,9 ms befinden sich die beiden ersten Spitzen, die zu sehen sind. Danach sinkt der Graph schnell, mit einer kleinen Spitze von ungefähr 350.000 im Bereich zwischen 30 und 32,9 ms.

Die Latenzverteilung als empirische Verteilungsfunktion ist in Abbildung 5.3 ebenfalls als blauer Graph dargestellt. Auch hier ist besonders auffällig, dass der Graph seinen Ursprung bei 75,54% hat. Das bedeutet, dass 75,54% aller Anfragen, denen eine Antwort zugeordnet werden konnte, in unter einer Millisekunde beantwortet wurden. Danach steigt der Graph nur noch mäßig. Die Latenz von 90% aller Antworten liegt unter 58 ms und 97% liegen unter 273 ms.

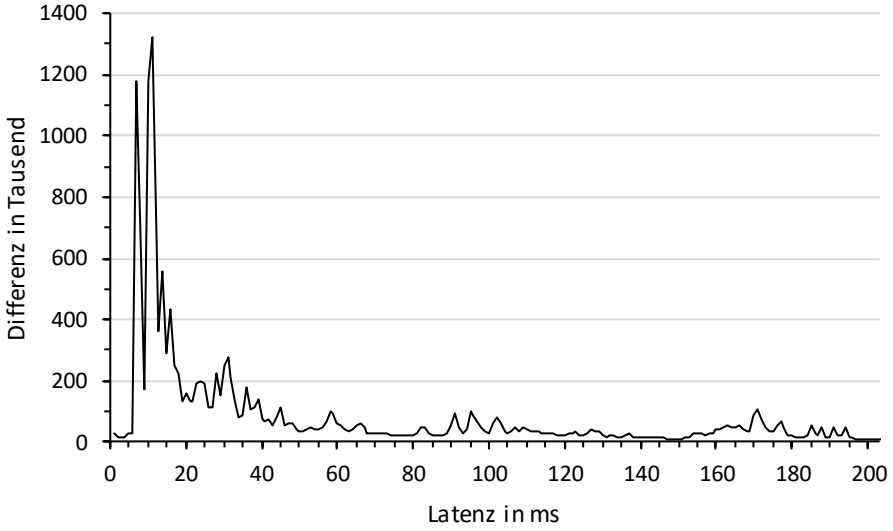

Abbildung 5.4: Die Differenz der Kategorien 1 bis 200 ms zwischen gewöhnlichem Resolver und simuliertem Resolver stellt die Umverteilung respektive Ersparnis von Latenzzeiten dar.

5.2.2.2 Latenzen des simulierten Resolvers

Die Erfassung dieser Statistik setzt sich aus der Latenz des simulierten aggressiven Resolvers und der des gewöhnlichen Resolvers zusammen. Bei einem Treffer im Cache ist in der ersten Kategorie inkrementiert worden, ansonsten erfolgt die Latenzberechnung wie beim gewöhnlichen Resolver. Auch bei der Darstellung dieses Graphen ist auf die erste Kategorie verzichtet worden. Die Verteilungsfunktion ist ebenfalls in Abbildung 5.2 enthalten, allerdings in Rot. Diesmal sind 89,2 Millionen DNS Nachrichten mit einer Latenz von unter einer Millisekunde gezählt worden. Das ist eine Steigerung von knapp 20 Millionen. Ferner ähnelt die Verteilung der vorherigen, wobei die Spitzen sehr viel niedriger oder vollkommen abgeflacht sind.

Die angesprochene Steigerung ist auch in Abbildung 5.3 zu sehen, denn der Graph hat seinen Ursprung bei 96,86%. Es ist also eine Differenz von über 20% im Vergleich zur empirischen Verteilungsfunktion des gewöhnlichen Resolvers zu verzeichnen.

In Abbildung 5.4 ist die Differenz der absoluten Verteilungsfunktionen der beiden Resolver für die Latenzwerte 1 bis 200 ms zu sehen. Dies ist die Anzahl der DNS Nachrichten, die ursprünglich der zugeordneten Latenzkategorie entstammen, aber nun durch den simulierten aggressiven Resolver schneller beantwortet werden können. Bildlich gesprochen werden die Werte des roten Graphen denen des blauen Graphen in Abbildung 5.2 abgezogen. Dadurch wird die Umverteilung der Latenzwerte und somit die

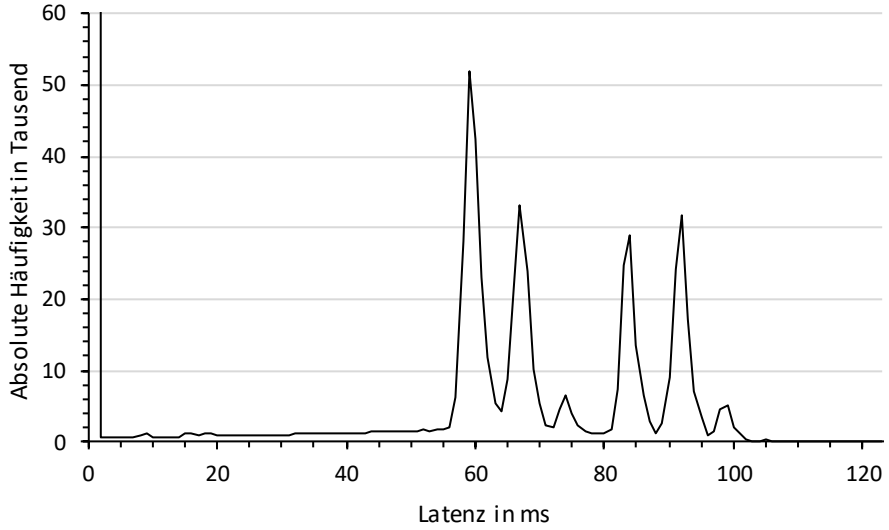

Abbildung 5.5: Absolute Häufigkeitsverteilung der Latenzkategorien bei Übereinstimmung des Response Types

eingesparte Latenz deutlich. Die Grafik endet bei 120 ms, da danach keine Veränderung mehr zu erkennen ist.

Der Verlauf des roten Graphen in Abbildung 5.3 ist ansonsten sehr ähnlich zu dem Verlauf des Graphen des gewöhnlichen Resolvers, was auch Abbildung 5.4 unterstützt. Da vor allem im Latenzbereich von 55 bis 100 ms Umverteilungen stattgefunden haben, verändert sich auch der Verlauf des Graphen der empirischen Latenzverteilung des simulierten Resolvers stärker in diesem Bereich und weniger im Bereich der höheren Latenzwerte jenseits von 100 ms.

5.2.2.3 Latenzen bei Übereinstimmung des Response Types

Die letzte Statistik der berechneten Latenzwerte gibt die Latenzverteilung aller DNS Nachrichten wieder, bei denen der Response Type des simulierten aggressiven Resolvers mit dem des gewöhnlichen Resolvers übereinstimmte. Die Summe aller Werte der absoluten Häufigkeitsverteilung entspricht der Anzahl von 5,1 Millionen korrekt synthetisierter DNS Nachrichten in Abschnitt 5.2.1.

Ein Ausschnitt der ersten 200 Kategorien der absoluten Latenzverteilung für DNS Nachrichten mit einem verifiziertem Response Type ist in Abbildung 5.5 zu sehen. Der Wert der ersten Kategorie, die wieder nicht vollständig dargestellt ist, beträgt 4,4 Millionen. Generell sind die Spitzen dieses Graphen nicht so hoch, wie bei den anderen zuvor.

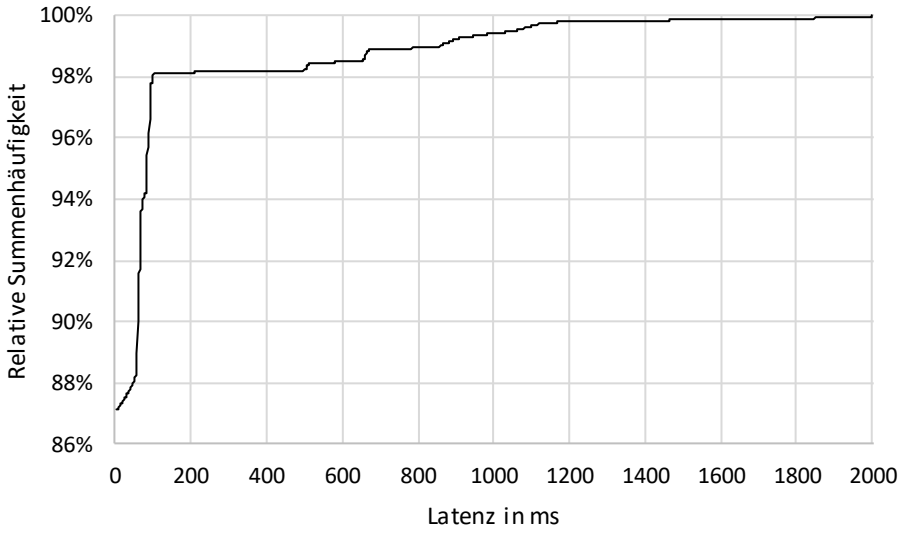

Abbildung 5.6: Relative aufsummierte Häufigkeitsverteilung der Latenzkategorien bei Übereinstimmung des Response Types

Auffällig ist, dass sie hier bei höheren Latenzwerten auftreten und nicht einzig in dem Bereich unter 20 ms.

Die eben beschriebene Verlagerung in den höheren Latenzbereich ist auch in Abbildung 5.6 zu vermerken. Der große Sprung ist nicht zu Beginn des Graphen, sondern etwas später im Verlauf sichtbar. Das liegt daran, dass gerade Antworten mit höheren Latenzzeiten eingespart wurden, da dies jene Antworten sind, bei denen ein gewöhnlicher Resolver erst die Nameserver hätte kontaktieren müssen. Des Weiteren ist der Verlauf des Graphen nicht wie bisher stetig steigend, stattdessen sind kleine Sprünge zu erkennen.

5.2.3 Fehler

Während der gesamten Simulation kann es an mehreren Punkten zu Fehlern kommen. Sie sind separiert gezählt und protokolliert worden. Nachfolgend werden die verschiedenen Fehlertypen beschrieben und ihre Häufigkeit beziffert.

QueryEntry In Kapitel 4 ist beschrieben, dass ein QueryEntry einem Eintrag im Sliding Window entspricht. Beim Hinzufügen, Auslesen oder Entfernen dieser Einträge kann es zu Fehlern kommen. Wenn versucht wird einen Eintrag zu lesen, der nicht existiert, wird der Fehler 'QueryEntry not found' geworfen. Falls es beim Hinzufügen eines

Eintrags unter dessen Schlüssel schon einen Eintrag gibt, wird er zwar überschrieben, aber trotzdem der Fehler 'QueryEntry already exists' geworfen.

Insgesamt gab es ungefähr 90.000 Fehler in Bezug auf einen QueryEntry. 'QueryEntry not found' macht dabei mit 89.000 den größeren Teil aus. 'QueryEntry already exists' trat nur 481 Mal auf. Dieser Fehler ist vermutlich darauf zurückzuführen, dass der Resolver für eine bestimmte Zeit überlastet gewesen ist, Anfragen nicht beantwortet hat und diese somit erneut gestellt wurden.

RRSIG Resource Records RRSIG Resource Records spielen bei der Untersuchung der Wildcards eine bedeutsame Rolle. Daher werden sie zusammen mit den NSEC und NSEC3 Records gespeichert. Beim Erfassen der zugeordneten RRSIG RR kann es vorkommen, dass der RR nicht gefunden werden kann oder es mehrere Treffer gibt. Dabei sind alles in allem 2,2 Millionen Fehler erfasst worden.

Der häufigste Fehler mit knapp 1,8 Millionen Mal war, dass kein entsprechender RRSIG RR für einen NSEC3 RR gefunden wurde. Gefolgt davon ist der gleiche Fehler in Bezug auf NSEC RR mit einer Häufigkeit von knapp 370.000. In Summe sind es 2,15 Millionen RRSIG RR, die nicht gefunden wurden. Allerdings ist auch gezählt worden, wie oft dieser Fehler bei einer DNS Nachricht mit Truncated Flag auftrat und das war in 84,3% der Fälle gegeben. Bei einem sicherheitsbewussten Resolver würde dies zur einer Neuanfrage über TCP führen. Wenn der Resolver, dessen Netzwerkverkehr hier betrachtet wurde, so verfahren ist, wird die Antwort der Neuanfrage eingelesen und dem Cache hinzugefügt. Ansonsten wird der entsprechende NSEC/3 RR nicht gespeichert, um eine Verfälschung der Ergebnisse zu verhindern.

Zu einer mehrdeutigen Ermittlung des RRSIG RR ist es in knapp 42.000 Fällen gekommen. Dabei wurden die zugehörigen NSEC/3 RR ebenfalls nicht im Cache abgelegt. Bei einer mehrdeutigen Ermittlung liegen bspw. folgende Resource Records vor.

193.190.in-addr.arpa.	10800	IN	NSEC	194.190.in-addr.arpa.	NS RRSIG NSEC
193.190.in-addr.arpa.	10800	IN	RRSIG	NSEC	5 4 10800 ...
193.190.in-addr.arpa.	10800	IN	RRSIG	NSEC	5 4 10800 ...

Die RRSIG RR sind gekürzt dargestellt, besitzen aber eine unterschiedliche Signatur.

Einlesen der DNS Nachricht Die Anzahl des Auftretens dieses Fehlers ist zwar mit 18 Stück verschwindend gering, dennoch sollte er erwähnt werden. Beim Lesen und Verarbeiten der DNS Nachricht durch die DNS Library kann es zur sogenannten WireParseException kommen, wenn die Nachricht fehlerhaft oder unvollständig ist.

5.3 Interpretation der Ergebnisse

Nach der Präsentation und Beschreibung der Ergebnisse erfolgt eine Interpretation dieser. Es geht darum, wie die verschiedenen Statistiken zusammenhängen, welche signifikanten Unterschiede es zwischen den Resolvern gibt, was die Verifikation des Response Types für die Ergebnisse bedeutet und welche Aussagen durch die Fehler über die Ergebnisse getroffen werden können. Am Schluss steht eine Gesamtauswertung, also eine zusammenfassende Beurteilung der Ergebnisse mit Blick auf Beantwortung der ursprünglichen Frage, ob ein aggressiver Resolver Latenzzeiten einsparen kann im Vergleich zu einem gewöhnlichen Resolver.

5.3.1 Simulierter Resolver vs. gewöhnlicher Resolver

In der Differenzfunktion in Abbildung 5.4 ist gut zu sehen, wie viele DNS Nachrichten und damit verbundene Latenzzeiten eingespart werden konnten. Abbildung 5.2 verdeutlicht den Sprung von 75,54% auf 96,86% in der ersten Kategorie. Außerdem sieht man, dass der rote Graph des simulierten Resolvers schneller in die Nähe von 100% kommt als der blaue Graph des gewöhnlichen Resolvers, denn dieser hat am Ende in der letzten Kategorie einen Sprung. Das bedeutet, dass auch gerade die besonders hohen Latenzwerte von über zwei Sekunden durch das aggressive Cache Verhalten des simulierten Resolvers in unter einer Millisekunde beantwortet wurden.

5.3.2 Korrekter Response Type

Bei allen Zugriffen auf den Cache gab es in 74,6 Millionen Fällen einen Hit, jedoch entsprach der ermittelte Response Type nur 5,1 Millionen Mal dem der realen Antwort. Das ist ein sehr geringer Anteil von 6,8%. Es kann vorkommen, dass sich während der Zeit, in der sich der Resource Record im Cache befindet, Veränderungen in der Zone stattfinden. Eine solche Änderung ist theoretisch möglich, aber im Vorfeld nicht so oft erwartet worden. Dazu kommt, dass einige Antworten des realen Resolvers eine Empty Response gewesen sind. Answer Section und Authority Section enthalten bis auf evtl. einen SOA RR keine Resource Records. Das deutet auf Fehlverhalten des Resolvers hin.

Darüber hinaus erfolgt kein genauer Abgleich der Response Types. D. h. bei einer simulierten Name Error Response und einer realen No Data Response würde der Response Type nicht übereinstimmen. Das Endergebnis, dass es den Namen in der Form nicht gibt, ist jedoch beinahe identisch. Daher ist die Aussage, dass 92,2% der ermittelten

Antworten falsch wären, mit Vorsicht zu betrachten. Die Verifikation der Antwort gibt bloß Auskunft darüber, welche Antworten auf jeden Fall richtig gewesen sind, aber nicht umgekehrt.

Eine Aussage, die daher getroffen werden kann, ist, dass die dargestellte Latenzverteilung in Abbildung 5.6 diejenigen Latenzwerte beinhaltet, die ohne Zweifel eingespart werden können, da in diesen Fällen der Response Type korrekt gewesen ist.

5.3.3 Bedeutsamkeit der Fehler

Im Verhältnis zur Anzahl der verarbeitet DNS Nachrichten ist die Menge der aufgetretenen Fehler insgesamt vertretbar. Allein in Bezug auf RRSIG RR sind relativ viele Fehler aufgetreten, wobei ein Fehler in der Implementierung ausgeschlossen werden kann, da 84,3% der fehlerhaften Antworten von NS an den Resolver tatsächlich keinen RRSIG RR enthielten. Das wirf die Frage auf, inwiefern die Steigerung der Effizienz dadurch beeinflusst wird.

5.3.4 Gesamtauswertung

Trotz des hohen Anteils von 93,5% der NSEC3 Resource Records mit gesetztem Opt-Out Flag, gemessen an der Anzahl aller im Cache gespeicherten NSEC/3 RR, ist insgesamt eine wesentliche Einsparung von Latenzzeiten zu verzeichnen. Wenn man den Latenzwert aller Kategorien, multipliziert mit der absoluten Häufigkeit des Auftretens in dieser Kategorie, aufsummiert, erhält man das Minimum der Gesamtlatenz. Es handelt sich um ein Minimum, da die letzte Kategorie alle Latenzwerte von mehr als zwei Sekunden beinhaltet und damit auch Werte von über drei Sekunden und mehr. Die Gesamtlatenz wird im Format *Stunden:Minuten:Sekunden* angegeben.

Für den gewöhnlichen Resolver ergibt sich so eine Gesamtlatenz von 1023:49:10. Durch den Einsatz eines Resolvers mit aggressivem NSEC/NSEC3 Caching können mindestens 3,5% dieser Zeit eingespart werden, da dies die Gesamtlatenz aller DNS Nachrichten mit übereinstimmendem Response Type ist. Wenn man davon ausgeht, dass alle false Positives korrekt sind, ist sogar eine Verringerung der Gesamtlatenz auf 173:07:01, also um 83,1%, möglich.

6 Fazit

Zum Abschluss der Arbeit werden der entwickelte Lösungsansatz zur Evaluation eines Resolvers mit aggressivem NSEC/NSEC3 Caching und die daraus resultierenden Ergebnisse in Bezug auf die gesteigerte Effizienz zusammengefasst wiedergegeben und bewertet. Des Weiteren wird ein Ausblick auf zukünftige Arbeiten gegeben, indem offen gelassene Aspekte erörtert werden.

6.1 Zusammenfassung der Umsetzung

Der Lösungsansatz wurde entwickelt, um eine bestmögliche Bewertung der Frage, ob eine Effizienzsteigerung zu erkennen ist, beantworten zu können. Dafür wurden die nach RFC 8198 [6] relevanten Aspekte eines aggressiven Resolvers basierend auf Verfahren des DNS, die in einer Vielzahl von RFC definiert sind, simuliert.

Das Ziel ist durch eine ereignisorientierte Simulation auf Stream-Basis umgesetzt worden. Eine entsprechenden Filterung der DNS Nachrichten zur Zuordnung von Absender und Empfänger sowie von Query und Response hat stattgefunden. Der Cache ist in Bezug auf Speicherung und Zugriff einem gewöhnlichen Resolver nachempfunden und um das Verhalten eines aggressiven Resolvers ergänzt worden. Alle daraus resultierenden Verarbeitungsschritte, wie Parsen, Filterung, Aufbau des Caches, Antwortermittlung und -verifizierung, sind durchgeführt worden. Statistische Daten zum Umfeld und Verhalten des Resolvers sind an entsprechenden Stellen zur Auswertung erfasst worden.

Insgesamt bildet diese Umsetzung eine solide Lösung, da eine strukturierte Abarbeitung der DNS Nachrichten, ebenso wie eine differenzierte Erfassung von Fehlern und potenziellen Ursachen für eine Störung der Effizienzsteigerung, erfolgt sind. Sie ermöglicht so eine ausführliche Betrachtung und Bewertung eines Resolvers mit aggressiver Nutzung des NSEC/NSEC3 Caches.

6.2 Schlussbetrachtung der Resultate

Auch in Anbetracht der hohen Anzahl fehlender RRSIG RR und NSEC3 RR mit gesetztem Opt-Out Flag ist deutlich zu erkennen, dass eine signifikante Einsparung von Latenzzeiten durch die Nutzung von aggressivem NSEC/NSEC3 Caching möglich ist. Ausgehend davon, dass die Zuverlässigkeit in Bezug auf Erhalt der entsprechenden RRSIG RR gesteigert werden kann, könnte dieses Verfahren für eine noch höhere Einsparung von Latenzen sorgen. Durch eine verringerte Nutzung des Opt-Out Flags könnte vermutlich noch mehr eingespart werden. Außerdem besteht die Möglichkeit, dass in einem aktuelleren Datensatz weniger NSEC3 Resource Records mit Opt-Out und mehr Wildcards zum Einsatz kommen. Möglicherweise und vorübergehend falsche Antworten aufgrund von Änderungen in den Zonendaten sollten toleriert werden, denn eine Einsparungsmöglichkeit von bis zu 83% der Gesamtlatenz ist nicht außer Acht zu lassen.

6.3 Ausblick für zukünftige Arbeiten

Da Umfang und Dauer dieser Arbeit begrenzt sind und nicht auf alle Aspekte und Vorteile der aggressiven Nutzung von NSEC/NSEC3 Caching eingegangen werden kann, gibt es noch potenzielle Themen für weitere Arbeiten.

Verbesserungen seitens der Nameserver Im Rahmen dieser Arbeit ist die Betrachtung der Anfragen von Resolver an Nameserver nicht genauer untersucht worden. Man könnte die Simulation dahingehend erweitern oder abändern, um diesen Verbindungsstrang zu beachten. Dadurch könnte man die eingesparte Last ausgehend von einem Resolver bestimmen. Zusammen mit einem Netzwerkmitschnitt aus Sicht des Nameserver wäre es möglich die Einsparung von Bandbreite und Belastung des Nameserver zu evaluieren. Man könnte bei der Analyse gezielt auf die angesprochenen Random QNAME Attacks achten und überprüfen, ob und wie weit die Nameserver durch ein aggressive Nutzung des Caches vor diesen Attacken geschützt werden würden.

Evaluation anhand neuer Daten Der hier analysierte Datensatz stammt aus dem Jahr 2013 und ist somit nicht mehr aktuell. Es könnte eine erneute Analyse und Bewertung des aggressiven Cache Verhalten mit Verwendung eines neuen Datensatzes durchgeführt werden. Beruhend auf der Annahme, dass zum heutigen Zeitpunkt eine größere Verbreitung von DNSSEC mit einhergehendem erhöhtem Einsatz von NSEC/3 Resource Records existiert, könnte vergleichsweise zu diesem Datensatz eine erhöhte Effizienzsteigerung gemessen werden.

Literaturverzeichnis

[1] Andrews, M.: Negative Caching of DNS Queries (DNS NCACHE) / RFC Editor. Version: March 1998. https://www.rfc-editor.org/rfc/rfc2308.txt. RFC Editor, March 1998 (2308). – RFC. – ISSN 2070–1721

[2] Arends, R. ; Austein, R. ; Larson, M. ; Massey, D. ; Rose, S.: DNS Security Introduction and Requirements / RFC Editor. Version: March 2005. http://www.rfc-editor.org/rfc/rfc4033.txt. RFC Editor, March 2005 (4033). – RFC. – ISSN 2070–1721

[3] Arends, R. ; Austein, R. ; Larson, M. ; Massey, D. ; Rose, S.: Protocol Modifications for the DNS Security Extensions / RFC Editor. Version: March 2005. http://www.rfc-editor.org/rfc/rfc4035.txt. RFC Editor, March 2005 (4035). – RFC. – ISSN 2070–1721

[4] Arends, R. ; Austein, R. ; Larson, M. ; Massey, D. ; Rose, S.: Resource Records for the DNS Security Extensions / RFC Editor. Version: March 2005. http://www.rfc-editor.org/rfc/rfc4034.txt. RFC Editor, March 2005 (4034). – RFC. – ISSN 2070–1721

[5] Braden, R.: Requirements for Internet Hosts - Application and Support / RFC Editor. Version: October 1989. https://www.rfc-editor.org/rfc/rfc1123.txt. RFC Editor, October 1989 (1123). – STD. – ISSN 2070–1721

[6] Fujiwara, K. ; Kato, A. ; Kumari, W.: Aggressive Use of DNSSEC-Validated Cache / RFC Editor. Version: July 2017. https://tools.ietf.org/html/rfc8198. RFC Editor, July 2017 (8198). – RFC. – ISSN 2070–1721

[7] Gieben, R. ; Mekking, W.: Authenticated Denial of Existence in the DNS / RFC Editor. Version: February 2014. https://www.rfc-editor.org/rfc/rfc7129.txt. RFC Editor, February 2014 (7129). – RFC. – ISSN 2070–1721

[8] Laurie, B. ; Sisson, G. ; Arends, R. ; Blacka, D.: DNS Security (DNSSEC) Hashed Authenticated Denial of Existence / RFC Editor. Version: March 2008.

http://www.rfc-editor.org/rfc/rfc5155.txt. RFC Editor, March 2008 (5155). – RFC. – ISSN 2070–1721

[9] Lewis, E.: The Role of Wildcards in the Domain Name System / RFC Editor. Version: July 2006. http://www.rfc-editor.org/rfc/rfc4592.txt. RFC Editor, July 2006 (4592). – RFC. – ISSN 2070–1721

[10] Lottor, M.: Domain administrators operations guide / RFC Editor. Version: November 1987. http://www.rfc-editor.org/rfc/rfc1033.txt. RFC Editor, November 1987 (1033). – RFC. – ISSN 2070–1721

[11] Mockapetris, P.: Domain names - concepts and facilities / RFC Editor. Version: November 1987. http://www.rfc-editor.org/rfc/rfc1034.txt. RFC Editor, November 1987 (1034). – STD. – ISSN 2070–1721

[12] Mockapetris, P.: Domain names - implementation and specification / RFC Editor. Version: November 1987. http://www.rfc-editor.org/rfc/rfc1035.txt. RFC Editor, November 1987 (1035). – STD. – ISSN 2070–1721

[13] Postel, J. ; Reynolds, J.K.: Domain requirements / RFC Editor. Version: October 1984. https://www.rfc-editor.org/rfc/rfc920.txt. RFC Editor, October 1984 (920). – RFC. – ISSN 2070–1721

[14] Thomson, S. ; Huitema, C. ; Ksinant, V. ; Souissi, M.: DNS Extensions to Support IP Version 6 / RFC Editor. Version: October 2003. https://www.rfc-editor.org/rfc/rfc3596.txt. RFC Editor, October 2003 (3596). – RFC. – ISSN 2070–1721