Formal Methods: Header Space Analysis

Benedikt Jaeger
Betreuer: Cornelius Diekmann
Hauptseminar - Innovative Internettechnologien und Mobilkommunikation SS2014
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: bene.jaeger@tum.de

KURZFASSUNG

Computernetzwerke sind heutzutage sehr groß und komplex, was es immer schwieriger macht, einen genauen Überblick über die eigentliche Funktionalität zu haben. Außerdem erschweren eine große Zahl von verschiedenen Netzwerkprotokollen oder andere Software- und Hardware-Komponenten, wie Firewalls und Network Address Translators (NATs), die Netzwerkanalyse. Zum Beispiel ist es immer schwieriger bestimmte Eigenschaften zu überprüfen, wie Reachability (Erreichbarkeit) und Slice Isolation (Isolation von Teilnetzen voneinander). Es gibt lediglich einige wenige Tools wie Ping oder Traceroute, die Netzwerkanalysten Hilfestellung leisten. Allerdings ist das oft nicht genug. Deshalb stellt Kazemian [1] ein neues Framework vor, das eine statische Netzwerkanalyse ermöglicht. Die sogenannte Header Space Analysis ermöglicht es ein Netzwerk nach häufigen Fehlerquellen zu durchsuchen. Zhang [3] zeigt allerdings, dass es sich dabei nicht um eine Innovation sondern lediglich um eine etwas umständlichen Anwendung von Finite State Machines (endliche Automaten) handelt. Dieses wird im Folgenden noch einmal aufgegriffen und die zwei Herangehensweisen werden miteinander verglichen.

Schlüsselworte

Header, Router, Netzwerkanalyse, Finite State Machine

1. EINLEITUNG

Auf praktische Weise kann man leicht in Computernetzwerken prüfen ob zwei Rechner miteinander kommunizieren können. Es wird lediglich ein Paket von A nach B geschickt. Solange das funktioniert, ist alles in Ordnung. Was aber wenn das Paket verloren geht oder möglicherweise sogar beim falschen Empfänger ankommt. Schon wird es schwierig den genauen Fehler zu finden. Die Größe von heutigen Netzen erschwert zunehmend die Netzwerkanalyse. Auf dynamischen Wegen ist diesem Problem oft fast nicht beizukommen. Daher beschäftigt sich diese Arbeit mit der statischen Analyse von Netzwerken. Man betrachtet nur ein unveränderliches Modell des Netzwerkes und versucht darin Fehler in den einzelnen Bausteinen zu finden.
Bei dieser Ausarbeitung wird zunächst in Kapitel 2 das Konzept der **Header Space Analysis** (HSA) vorgestellt und ein bestimmter Anwendungsbereich, der Überprüfung von Erreichbarkeit in Netzwerken, genauer betrachtet. Dabei wird auch die Komplexität des Algorithmus abgeschätzt und damit der Erfolg der HSA bei einem Test am Netz der Stanford University begründet. Anschließend wird in Kapitel 3 das gleiche Verfahren mit Hilfe von **Finite State Machines** (FSM) angewandt und ein Vergleich zur Header Space Analysis gezogen.

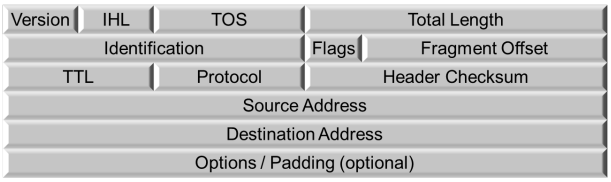

Abbildung 1: Aufbau eines IPv4-Headers. Darin sind alle wichtigen Informationen enthalten, um das Paket richtig zu bearbeiten. Zum Beispiel der Typ des Headers, seine Länge, der Typ des folgenden Headers, das Time-To-Live-Feld und natürlich Start- und Zieladresse.

2. HEADER SPACE ANALYSIS

Um verschiedene, teils komplizierte Abläufe in einem Netzwerk zu verstehen und mögliche Fehler aufzudecken, werden im Folgenden nur bestimme Netzwerke betrachtet. Diese verhalten sich unabhängig von der Zeit. Dabei bleiben sämtliche Verbindungen zwischen den Netzwerkknoten unverändert und die Funktionalität dieser ändert sich nicht. Dies nennt man statische Netzwerkanalyse.
Eine Methode dazu ist die Header Space Analysis. Dabei handelt es sich folglich um ein theoretisches Konzept, mit dessen Hilfe man versucht die Abläufe und Wege in einem Netzwerk zu analysieren. Anhand der Ergebnisse lassen sich anschließend mögliche Fehlerquellen finden. Der Name setzt sich zusammen aus der Analyse des mathematischen Raumes (Space), der von den Headerbits erzeugt wird.
Dabei wird versucht das reale Netzwerk so gut wie möglich in einem Model zu simulieren. Manche Informationen kann man dabei außen vor lassen, wie zum Beispiel die Checksumme oder, wenn man vor allem an IP-Routing interessiert ist, die Hardwareadressen. Andere sollte man auf jeden Fall beibehalten, da sie oft Ursache oder zumindest Indikator für häufige Probleme sind. Zum Beispiel das Time-to-Live Feld eines IP-Headers (vgl. Abbildung 1).
Pakete bestehen grundsätzlich aus zwei Teilen. Zum einen aus dem Header, in dem unter anderem alle relevanten Informationen des Senders und Empfängers enthalten sind, und zum anderen aus der Payload, dem Teil, der die eigentliche Nachricht enthält. Für den Weg des Pakets durch das Netzwerk ist die Payload irrelevant,daher wird im Folgenden nicht zwischen Header und Paket unterschieden.

doi: 10.2313/NET-2014-08-1_12

2.1 Definition des Frameworks

Die Header Space Analysis ist ein Model in dem folgende drei wichtigen Bestandteile eines realen Netzwerkes modelliert werden.

1. Als erstes werden sämtliche Pakete, die das Netzwerk passieren nur auf ihren Header reduziert. Diese werden unabhängig von der Art des Headers als Bitfolge einer bestimmten Länge, die man anfangs festlegen muss, dargestellt.

2. Zweitens wird der Ort eines Paketes, also die Position innerhalb des Netzes, als Port dargestellt. Ports sind eindeutig identifizierbar innerhalb des Netzwerkes und sind vergleichbar mit den Interfaces eines Routers. Zudem werden die Ports miteinander verbunden um die eigentliche Topologie des Netzwerkes zu simulieren.

3. Und drittens werden alle Netzwerkkomponenten, wie Router oder Switches, unabhängig ob es sich um Hardware- oder Softwarekomponenten handelt, als Middleboxes modelliert. Dabei ist es vollkommen egal was für eine Funktion sie haben. Es sind von außen gesehen nur Backboxes, die auf ankommende Pakte bestimmte Funktionen anwenden und sie dann weiterleiten. Der Verständlichkeit halber werden diese Middleboxes im Folgenden trotzdem mit Router oder Switches bezeichnet.

2.1.1 Header Space H

Der **Header Space** ist ein grundlegender Baustein des Frameworks. Alle möglichen Header einer festgelegten Länge werden als Vektoren, oder Punkte, in diesem Raum dargestellt. Die maximale Länge eines Headers muss anfangs festgelegt werden. Da Pakete oft unterschiedlich viele und unterschiedlich lange Header besitzen, je nach verwendeten Protokollen, sollte hier die maximale Länge gewählt werden. Kleinere Header können beispielsweise mit Padding-Bits aufgefüllt werden, da die meisten Header genaue Informationen enthalten, wie lang sie sind und welcher Header in der nächsten Schicht verwendet wird. Der Header Space ist mathematisch wie folgt definiert

$$H \subseteq \{0,1\}^L$$

wobei L die maximale Header-Länge ist. Man kann klar erkennen, dass in keiner Weise zwischen verschiedenen Headern oder Header-Feldern unterschieden wird, was eine Analyse unabhängig von sämtlichen Protokollen ermöglicht. Außerdem wird noch ein drittes Zeichen x, ein sogenanntes Wildcard-Zeichen, hinzugenommen, welches signalisiert, dass ein bestimmtes Bit beliebig ist, also sowohl 1 als auch 0 sein kann. Ein Header, der Wildcardbits enthält wird als Wildcardheader definiert. Man hat zwei Möglichkeiten einen Wildcardheader darzustellen. Entweder man erweitert den Header Space um dieses Zeichen.

$$H' \subseteq \{0,1,x\}^L$$

Oder man stellt sie als Vereinigung von normalen Headern dar, dabei wächst die Anzahl der Vereinigungen aber exponentiell zur Zahl der Wildcarzeichen. Ein Punkt im Header Space wird im Folgenden mit h bezeichnet.

2.1.2 Network Space N

Der **Network Space** setzt sich aus dem kartesischen Produkt des Header Space und der Menge aller Ports zusammen. Ports sind eindeutig identifizierbar in dem Netzwerk und immer mit einer Middlebox verbunden. Punkte im Network Space werden als Tupel (h, p) aus dem Raum

$$N \subseteq \{0,1\}^L \times \{0, ..., P\}$$

dargestellt, wobei $\{0, ..., P\}$ die Menge aller Ports innerhalb des Netzwerks ist. Dadurch lassen sich Pakete modellieren, die sich an einem bestimmten Ort (Port) im Netzwerk befinden. Der Network Space ist notwendig um zum Beispiel das Durchqueren eines Pakets durch ein Netzwerk zu modellieren, wobei es schrittweise von Router zu Router geschickt wird.
Die Identifikationsnummer des Ports auch binär kodiert und einfach an den Header angehängt. Somit handelt es sich bei einem Punkt im Network Space, im Vergleich mit einem Punkt im Header Space, um eine längere Bitfolge. Hier werden die Ports, aufgrund der besseren Lesbarkeit, entweder gemäß der Middlebox benannt, an der sie anliegen, zum Beispiel besitzt ein Router R die Ports R_0, R_1 und R_2. Oder sie werden einfach mit $p_1, p_2, ...$ durchnummeriert.

2.1.3 Transfer Funktion

Jeder Router, Switch, Firewall, Nat und alle anderen Netzwerkkomponenten werden vollkommen gleich als Middleboxes behandelt. Sie unterscheiden sich lediglich durch ihre Funktion für das Netzwerk. So unterscheidet sich zum Beispiel eine Firewall, die nur Pakete von bestimmten Absendern weiterleitet, von einem Router, der die Pakete anhand ihrer IP-Adresse in die richtige Richtung leitet, nur durch ihre individuelle Funktion. Diese werden in der Header Space Analysis als **Transfer Funktion** modelliert. Diese Funktionen lassen sich noch weiter untergliedern. Die grundlegendste davon ist die **Box Transfer Funktion**, welche die Funktion einer einzelnen Middlebox beschreibt. Mathematisch lässt sich die Funktion wie folgt beschreiben:

$$T(h,p) : (h,p) \to \{(h_1,p_1),(h_2,p_2),\dots\}$$

Bei dieser Definition fallen in erster Linie zwei Merkmale auf. Zum einen ist es möglich, dass sich der ankommende Header in seiner Struktur verändern kann. Dies geschieht zum Beispiel bei dem Time to Live (TTL) Feld in einem IPv4-Header oder mit der Source- und Destination-IP bei einer NAT-Middlebox. Zum anderen handelt es sich bei der Ausgabe der Funktion um eine Menge von Paketen, was in der Realität auch bei Broadcasts oder Multicasts geschieht. Zudem ist zu jeder Box Transfer Funktion auch eine Umkehrfunktion T^{-1} definiert. Allerdings nicht im mathematischen Sinne, da sie nicht injektiv ist. Festzuhalten ist allerdings, dass sich zu jeder Output-Menge gibt auch alle Header der Input-Menge bestimmen lassen. Die Signatur der Umkehrfunktion sieht also folgendermaßen aus

$$T^{-1} : \mathcal{P}(N) \to \mathcal{P}(N).$$

Mit $\mathcal{P}(N)$ ist die Potenzmenge von N bezeichnet, also der Menge, die alle verschiedenen Teilmengen aus dem Netzwerk Space enthält. Insbesondere enthält sie auch die leere Menge.

Um die Transfer Funktion besser zu verstehen folgt ein kurzes Beispiel. Ein Router, der ankommende Pakete anhand

doi: 10.2313/NET-2014-08-1_12

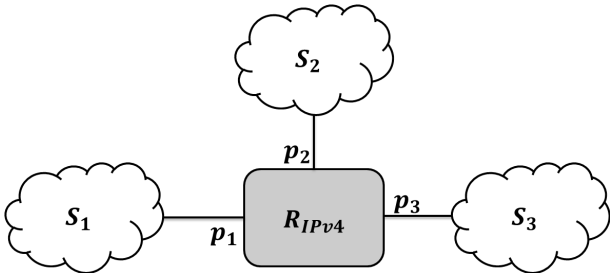

Abbildung 2: Ein IPv4-Router, der mit drei verschiedenen Netzen an den jeweiligen Ports verbunden ist

ihrer Ziel-IP-Adresse in die jeweiligen Subnetze weiterleitet, wie in Abbildung 2 zu sehen, kann mit folgender Transferfunktion modelliert werden. Dabei ist die Funktion $ip\_dest(h)$ eine Hilfsfunktion, die die Ziel-IP-Adresse des Headers h zurückliefert.

$$T_{IPv4}(h,p) = \begin{cases} \{(h,p_1)\} & \text{if } ip\_dest(h) \in S_1 \\ \{(h,p_2)\} & \text{if } ip\_dest(h) \in S_2 \\ \{(h,p_3)\} & \text{if } ip\_dest(h) \in S_3 \\ \{\} & \text{otherwise} \end{cases}$$

Da sich Netzwerke in der Regel aus vielen Routern, Switches, usw. zusammensetzen, wird eine Hilfsfunktion eingeführt, die alle Middleboxes innerhalb des Netzes simuliert. Sie hat die gleichen Ein- und Ausgaben wie die normale Box Transfer Funktion und dient lediglich als Hilfe um die Anwendungen übersichtlicher zu gestalten. Seien $T_1, ..., T_n$ alle Box Transfer Funktionen der jeweiligen Switches $switch_1$, ..., $switch_n$ innerhalb des Netzwerkes, dann ist die sogenannte **Network Transfer Funktion** Ψ demnach folgendermaßen definiert:

$$\Psi(h,p) = \begin{cases} T_1(h,p) & \text{if } p \in switch_1 \\ ... & ... \\ T_n(h,p) & \text{if } p \in switch_n \end{cases}$$

Diese Funktion ruft die zu dem aktuellen Port passende Box Transfer Funktion auf. So genügt ein einzelner Aufruf der Ψ-Funktion, um die jeweilige Transferfunktion aufzurufen, unabhängig von der Position des Pakets im Netzwerk.
Bisher wurde noch in keinem Schritt die tatsächliche Topologie des Netzwerkes modelliert. Das ist die eigentliche Struktur, die festgelegt welche Middleboxes miteinander verbunden sind. Dabei werden meistens einfache, ungerichtete Graphen verwendet, auf denen man die üblichen Graphenalgorithmen ausführen kann, wie das Finden von Pfaden. Dazu wird neben der Liste der Ports auch noch eine Menge von Tupeln von der Form (p_i, p_j) benötigt, die bedeuten, dass Port i mit Port j verbunden ist. Im Weiteren wird einfachheitshalber davon ausgegangen, dass alle Verbindungen bidirektional sind, also in beide Richtungen gehen. Die **Topology Transfer Funktion** Γ ist so definiert, dass sie das Eingabepaket gemäß dem Netzwerkgraphen an den nächsten Port weiterleitet.

$$\Gamma(h,p) = \begin{cases} \{(h,p^*)\} & \text{if p is connected to } p^* \\ \{\} & \text{if p is not connected} \end{cases}$$

Sie modelliert das Verhalten sämtlicher Links oder Verbindungen in dem Netz. Dabei ist zu beachten, dass sich nur der Port ändert und die Header unverändert bleiben. Außerdem ist zu dieser Funktion die Umkehrfunktion analog definiert, da Netzwerke hier als ungerichtete Graphen betrachtet werden und die Verbindungen bidirektional sind. Es gilt also immer $\Gamma^{-1} = \Gamma$.

Die Network Transfer Funktion und die Topology Transfer Funktion werden immer nacheinander aufgerufen, um einen Hop eines Paketes zu simulieren. Daher wird die Konkatenation dieser beiden Funktionen in einer eigenen Funktion zusammengefasst, nämlich $\Phi(.) = \Psi(\Gamma(.))$. Der konsekutive Aufruf dieser Funktion ermöglicht mehrere Hops oder entsprechende Schritte in einem Netzwerk zu gehen. Die folgende Funktion ermittelt alle Tupel (h', p'), die erzeugt werden, wenn man mit (h, p) startet und k Schritte berechnet.

$$\Phi^k(h,p) = \Psi(\Gamma(...\Psi(\Gamma(h,p)...)$$

Damit sind die wichtigsten Bestandteile des Header Space Frameworks definiert und man kann damit beginnen, erste Netzwerkprobleme zu analysieren.

2.2 Reachability

Eine der grundlegendsten Fragen in der Netzwerkanalyse ist „Kann Host a Pakete zu Host b senden?“, oder „Welche Pakete erreichen Host b von Host a?“. Diese Frage lässt sich eindeutig beantworten.
Der dafür verwendete Algorithmus lässt sich in zwei Hauptbestandteile zerlegen. Zunächst berechnet man durch mehrmaliges Anwenden der Φ-Funktion auf einen Wildcardheader die Menge der Header, die b erreichen um anschließend durch Anwenden der Umkehrfunktionen die ursprünglichen Header, die von Host a losgeschickt werden, zu erhalten.

2.2.1 Algorithmus

Diese erste Idee bedarf aber noch einiger Feinheiten, um die Komplexität zu reduzieren. Zum Beispiel würden durch Anwenden der Φ-Funktion auch Pakete in Netzwerkbereiche weitergeleitet werden, von denen man den Ziel-Host gar nicht mehr erreichen kann. Deshalb werden anfangs alle Pfade von a nach b aufgelistet. Allerdings stellt sich dabei die Frage, wie mit Kreisen im Netzwerkgraph verfahren werden soll. Ein mögliches Vorgehen wäre es, eine maximale Anzahl von Schleifen festzusetzen und somit alle Pfade aufzulisten. Dabei genügt logischerweise eine geringe Zahl, da Netzwerke auf Effizienz ausgerichtet sind und es nicht passieren sollte, dass sich Pakete im Kreis bewegen.
Nun schickt Host a einen kompletten Wildcardheader, also einen Header, in dem jedes Bit beliebig ist, los und man wendet darauf alle Transferfunktionen entlang des aktuellen Pfades an. Anschließend wird dies für jeden Pfad gemacht und die jeweiligen Mengen vereinigt. Sei $R_{a\to b}$ die Menge aller Header, die b von a erhält und sind die Pfade von der Form

$$a \to S_1 \to ... \to S_{n-1} \to S_n \to b$$

so erhält man für die Menge der bei b ankommenden Header

$$R_{a\to b} = \bigcup_{a\to b \text{ paths}} T_n(\Gamma(T_{n-1}(...(\Gamma(T_1(h,p)...)).$$

Nach diesem Schritt muss man nun den ganzen Weg wieder zurück gehen, da sich diese Header auf dem Weg geändert

doi: 10.2313/NET-2014-08-1_12

haben können. Demnach ist es notwendig durch Anwenden der Umkehrfunktionen die ursprüngliche Header-Menge h_a zu berechnen.

$$h_a = T_1^{-1}(\Gamma(...(T_{n-1}^{-1}(\Gamma(T_n^{-1}(h,b)...))$$

Damit ist die Frage der Erreichbarkeit beantwortet. Allerdings liefert der Algorithmus zunächst nur die Information, dass Pakete ankommen oder nicht. Er liefert nicht die möglichen Fehlerursachen. Will man diese aufspüren, kann man zum Beispiel das Netz nach dem Divide-and-Conquer Prinzip in kleinere Teilnetze zerteilen und den Algorithmus auf diese ausführen.

Zum besseren Verständnis findet man in Abbildung 3 ein Beispiel. Dabei werden Header der Länge acht bit verwendet. Zur Visualisierung der Header an den jeweiligen Positionen werden sie in ein Koordinatensystem eingetragen, wobei jeweils die ersten vier Bits, als natürliche Zahl interpretiert, in x-Richtung und die letzten vier in y-Richtung abgetragen werden. Das Ziel ist wiederum alle Header zu finden, die b von a erreichen. Alle Header die in den einzelnen Schritten des Algorithmus entstehen sind über bzw. unter die jeweiligen Router als Graph dargestellt. Die Transferfunktionen der Middleboxes lauten

$$T_A(h,p) = \begin{cases} \{(h, A_0)\} & \text{if } h = 1001xxxx \\ \{(h, A_1)\} & \text{if } h = 1010xxxx \\ \{(h, A_2)\} & \text{if } h = 1100xxxx \end{cases}$$

$$T_B(h,p) = \begin{cases} \{(h, B_0)\} & \text{if } h = 10xxxx01 \\ \{(h, B_1)\} & \text{if } h = 10xxxx00 \end{cases}$$

$$T_C(h,p) = \begin{cases} \{((h\&00001111)|11000000, C_0)\} & \text{if } h = 0011xxxx \\ \{((h\&00001111)|00110000, C_1)\} & \text{if } h = 1100xxxx \end{cases}$$

$$T_D(h,p) = \begin{cases} \{(h, D_0)\} & \text{if } h = xxxx1xxx \\ \{(h, D_1)\} & \text{if } h = xxxx0xxx \end{cases}.$$

Die Funktionen T_A, T_B und T_D gehören zu einfachen Middleboxes, die in Abhängigkeit von bestimmten Header-Feldern Entscheidungen treffen, wie zum Beispiel Router oder Firewalls. Dagegen wirkt T_C auf den ersten Blick komplizierter. Grundsätzlich erkennt man, dass sie nicht nur Pakete weiterleitet sondern auch ihre Header verändert. Sie ändert die ersten vier Bits von 1100 zu 0011 und umgekehrt. Dies ist durch ein binäres AND mit einer Maske und anschließendem binärem OR implementiert. Die Funktionsweise von T_C ähnelt der eines Network Address Translators (NAT).
Die Menge der Header, die in b ankommen, wird zunächst durch Anwenden der jeweiligen Transferfunktionen berechnet, wie in Abbildung 3 visualisiert. Die Anzahl der einfachen Pfade in diesem Netzwerk ist Zwei und es wird auf das Hinzunehmen von Schleifen verzichtet. Man erhält also

$$R_{a\to b} = \{10100x01, 00110xxx\}.$$

Anschließend wendet man die Umkehrfunktionen an und erhält

$$h_a = \{10100x01, 11000xxx\}.$$

Offensichtlich ist h_a zum Teil identisch mit $R_{a\to b}$. Je nachdem welche Header-Felder man modelliert, ist dies unterschiedlich. Zum Beispiel das TTL Feld garantiert, dass sie nicht identisch sind. Das Anwenden der Umkehrfunktion ist also auf jeden Fall notwendig.

2.2.2 Komplexität

Als Erstes wird versucht die Worst-Case Laufzeit abzuschätzen. Dabei kann jeder Router $O(2^L)$ neue Pakete erzeugen. Diese kann er jeweils an jeden seiner Ports schicken. Wenn das Netzwerk einen Durchmesser d hat und die genannten Fälle an jedem Router eintreten erhält man

$$O((2^L * P)^d) \text{ mit } P = |ports|.$$

Zudem kann d noch weiter wachsen indem Schleifen beliebiger Länge auftreten.

Allerdings überschätzt diese Schranke die tatsächliche Laufzeit deutlich, da die Header Space Analysis in der Praxis durchaus gute Ergebnisse liefert, wie das nächste Kapitel zeigt. Unter Hinzunahme einer Annahme über die Struktur von Netzwerken, kann deshalb die Komplexität auf durchschnittlich $O(dR^2)$ reduziert werden.
Es wird angenommen, dass Computernetze immer auf Effizienz ausgerichtet sind. Pakete sollen möglichst schnell und direkt übertragen werden. Demnach sollen auf ein eingehendes Paket nur sehr wenige Transferregeln zutreffen. Sei R_1 die Anzahl der Pakete die an einem Router ankommen und R_2 die Zahl der Transferregeln des Routers. Man geht davon aus, dass anstatt R_1R_2 nur cR_1, mit $c << R_2$, neue Header erzeugt werden. Diese Annahme wird als **Linear Fragmentation Assumption** bezeichnet. An einem Router werden $O(R)$ ankommende Header mit $O(R)$ Transferregeln verglichen und generieren wiederum $O(R)$ Header. Demnach liegt die Komplexität, wie bereits erwähnt, in

$$O(R^2 + R^2 + ... + R^2) = O(dR^2).$$

Diese Annahme findet in der Praxis Bestätigung, da das Framework an einem größeren Netzwerk, nämlich dem Netz der Stanford University, erfolgreich getestet wurde.

2.3 Hassel und ein Test am Stanford Netz

Auch wenn die Header Space Analysis in der Theorie zu funktionieren scheint, ist das noch keine große Hilfe zur praktischen Analyse von Netzwerken. Daher wurde, neben der theoretischen Grundlage, auch ein Tool namens **Header Space Library** oder **Hassel** implementiert. Dieses in Python geschriebene Programm kann unter anderem aus Routern die entsprechenden Transferfunktionen auslesen und sie in eine für die *Header Space Analysis* kompatible Form bringen. Allerdings ist das bis jetzt nur bei Cisco-Routern möglich.
Anschließend stellt Hassel mehrere Funktionen zur Netzwerkanalyse bereit, wie die schon genannte Überprüfung auf Erreichbarkeit und zum Beispiel das Suchen nach Schleifen im Netzwerk, oder das Überprüfen ob ein Teilnetz von einem anderen isoliert ist, was im Sinne der IT-Sicherheit interessant ist. Schließlich wurde Hassel am Netz der Stanford University getestet, einem Netz bestehend aus unter anderem fünf /16 IPv4 Subnetzen und über 757,000 Forwarding-Regeln. Der Test fand unter anderem 12 Endlosschleifen, wobei die TTL außer acht gelassen wurde. Genaue Ergebnisse sind in Kazemias Arbeit [1] nachzulesen. Festzuhalten ist, dass man trotz großer Datenmenge in relativ kurzer Zeit

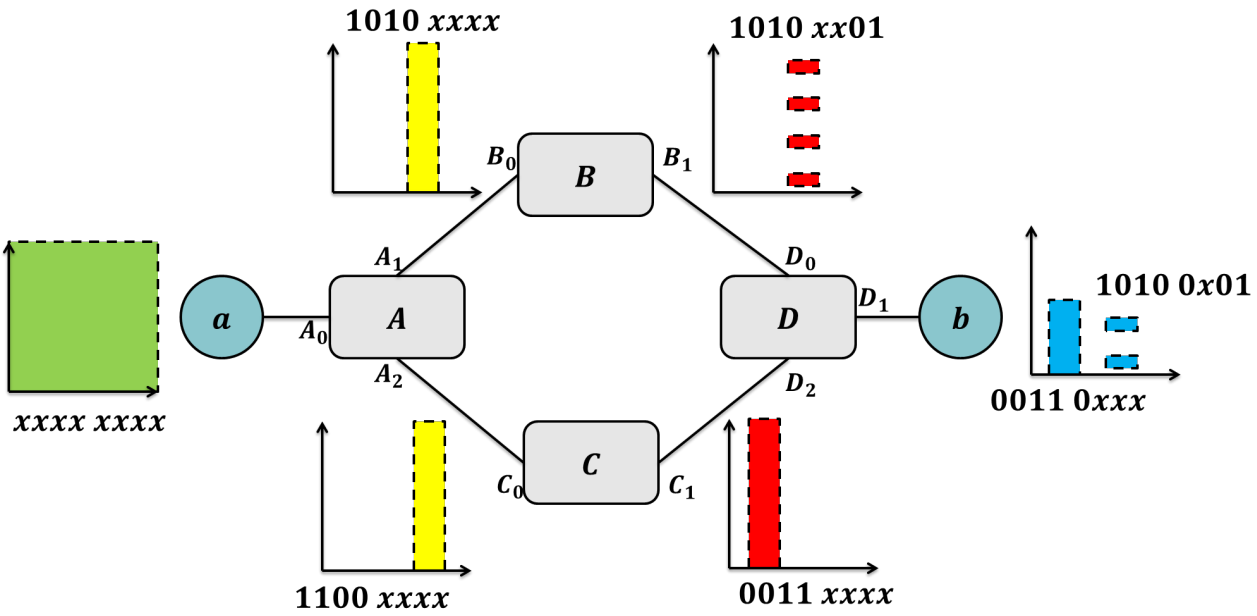

Abbildung 3: Überprüfen der Erreichbarkeit zwischen den Hosts a und b mithilfe der HSA. Hier ist visualisiert, wie die Header ihre Form verändern auf ihrem Weg von a nach b.

vorzeigbare Ergebnisse erhalten hat.
Da nun die Header Space Analysis definiert und ihre Funktion und Effizienz aufgezeigt wurde, kann nun der Vergleich mit den Finite State Machines gezogen werden.

3. FINITE STATE MACHINES

Der Ansatz der Header Space Analysis wirkt auf den ersten Blick wie eine Innovation, bei genauerem Betrachten ist er allerdings einem Verfahren mit **Finite State Machines** (FSM), auf deutsch Endlichen Automaten, sehr ähnlich. Endliche Automaten finden vor allem in der Theoretischen Informatik Anwendung. Zhang bezeichnet den Ansatz der Header Space Analysis als isomorph zu dem Ansatz mit Finite State Machines [3].

3.1 Definition Finite State Machine

Bei einer Finite State Machine handelt es sich grundsätzlich um ein 5-Tupel $(\Sigma, S, s_0, \delta, F)$. Dabei ist Σ eine endliche, nicht leere Menge von Zeichen, genannt das Eingabealphabet. S ist eine endliche, nicht leere Menge von Zuständen und s_0 ein Element aus S, das der Startzustand ist. Je nach Anwendungsgebiet ist es auch möglich, eine Teilmenge von S als Startzustand festzulegen. Der wichtigste Bestandteil des Automaten ist die Übergangsfunktion $\delta : \Sigma \times S \to P(S)$. Sie erzeugt eine Menge von Zuständen in Abhängigkeit des aktuellen Zustandes und der Eingabe. Schließlich gibt es auch noch eine Menge F von Endzuständen, den Zuständen in denen der Automat stoppt.
In Abbildung 4 sieht man eine graphische Darstellung eines endlichen Automaten. Dabei werden Zustände als Kreise, die Übergangsfunktion als Pfeile zwischen den Kreisen, das Eingabealphabet als Zeichen über den Pfeilen und die Endzustände als Doppelkreis dargestellt.

3.2 Reachability mit Finite State Machines

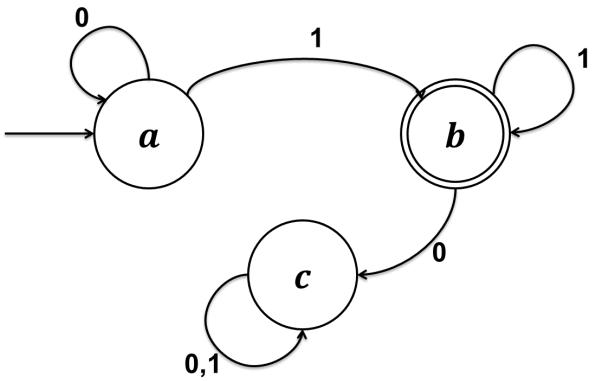

Abbildung 4: Graphische Darstellung eines endlichen Automaten mit Zuständen {a,b,c}. Die Übergangsfunktion ist mit Pfeilen visualisiert, an denen das jeweilige Zeichen aus dem Eingabealphabet steht. Dieser einfache Automat kann zum Beispiel zum erkennen der Sprache 0^*11^* verwendet werden

Ein Automat, der die gleiche Funktionalität wie die *Header Space Analysis* aufweist, lässt sich folgendermaßen konstruieren. Die Menge der Zustände ist die Menge aller Punkte aus dem Network Space $\{0,1\}^L \times \{0,...,P\}$. Auf den ersten Blick wirkt das nicht sehr effizient, da allein die Zustandsmenge Ausmaße der in der Größenordnung $O(2^{L+P})$ annimmt. Das ist ein allgemeines Problem bei Automaten mit großer Zustandsmenge. Daher wird die Zustandsmenge meistens nicht explizit angegeben und gehofft bzw. vorhergesagt, dass die Menge der aktuellen Zustände nicht explodiert, also rapide ansteigt. Als Eingabealphabet kann man je nach Anwendung verschieden Mengen wählen. Für das

doi: 10.2313/NET-2014-08-1_12

Überprüfen der Erreichbarkeit genügt irgendeine einelementige Menge, deshalb wird für diesen Algorithmus das Eingabealphabet außer Acht gelassen.
Als Übergangsfunktion wird die aus der HSA schon bekannte Φ-Funktion verwendet. Diese verbindet in der State Machine alle Zustände, die in dem Netzwerk in einem Hop erreicht werden können. Im Bezug auf die Erreichbarkeit ist auch klar welche Mengen man für den Start- und Endzustand wählen muss, nämlich zum einen die Menge aller Zustände der Form $(..., a)$ und zum anderen alle Zustände der Form $(..., b)$. Also alle Pakete, die bei Port a beziehungsweise bei Port b liegen.

3.3 FSM Algorithmen

Die meisten Algorithmen, die Erreichbarkeit auf endlichen Automaten überprüfen, arbeiten nach einem einfachen Prinzip. Es werden rekursiv Zustände hinzugefügt, die von den aktuellen Zuständen erreicht werden können. Diese Iteration lässt sich folgendermaßen formulieren. Sei R_i die Menge von Zuständen, die in bis zu i Schritten erreicht werden können, dann ist

$$R_n = R_{n-1} \cup \bigcup_{y \in R_{n-1}} T(y).$$

Die Menge R_n setzt sich also aus zwei Teilen zusammen. Zum einen allen Zuständen, die in $n-1$ Schritten erreicht werden, und zum anderen allen Zuständen, die von der aktuellen Menge durch Anwenden der Transferfunktion erreicht werden können. Dieser Schritt wird anschließend solange wiederholt bis ein Fixpunkt R^* erreicht wird. Also sich die Menge der erreichten Zustände nicht mehr ändert. Dies ist der Fall, wenn gilt

$$R^*(x) = R_n = R_{n-1}.$$

Diese Fixpunktiteration wird als **Structural FSM Traversal** bezeichnet [2]. Auf den ersten Blick scheint dieser Algorithmus ein sicherer Weg zu allen erreichbaren Zuständen zu sein, allerdings wird dabei ein großes Problem außer Acht gelassen. Es kann nämlich passieren, dass die Zustandsmenge explodiert und somit eine effiziente Berechnung unmöglich macht. Dies ist zum Beispiel dann der Fall wenn das Netzwerk unendliche Kreise enthält und man diese nicht vorher herausfiltert. Dennoch liegt in der Regel die Anzahl der Zustände bei diesen Automaten in Bereichen, mit denen man nicht mehr effizient arbeiten kann. Zum Beispiel erhält man bei einer Headergröße von 32 bit und einem Netzwerk mit nur acht Ports bereits bis zu $2^{40} \approx 1.0 \cdot 10^{12}$ verschiedene Zustände.
Man ist schließlich darauf angewiesen, dass die Zustandsmenge in geordneten Bahnen wächst. Außerdem ist es wichtig die erreichten Zustände in geeigneten Datenstrukturen abzulegen, um ein effizienteres Arbeiten zu ermöglichen. Dabei werden häufig **Binary Decision Diagramms** (BDDs) oder **Disjunktive Normalformen** (DNFs) verwendet. Es ist immer wichtig die passende Datenstruktur für das jeweilige Problem auszuwählen.
Unabhängig davon weist diese Iteration starke Ähnlichkeiten zur HSA auf. Tatsächlich handelt es sich bei der Header Space Analysis um einen Ansatz der auf Finite State Machines basiert.

3.4 Vergleich FSM mit HSA

Wie bereits erwähnt lässt es sich zeigen, dass es sich bei der Header Space Analysis um ein Verfahren handelt, das sehr

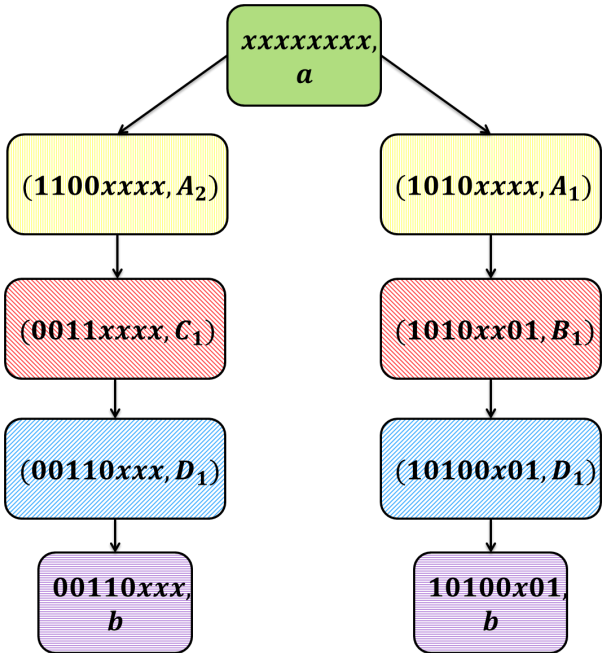

Abbildung 5: Anwendung der Fixpunktiteration auf das vorherige Beispiel. Die jeweilig erreichten Zustände erzeugen einen Baum, wobei die Tiefe maximal der Anzahl der Schritte entspricht. Gleiche Knoten können entfernt werden. Jeweils gleiche Farben werden bei gleichem Iterationsschritt erreicht.

stark an Finite State Machines angelehnt ist. [3] Dabei werden die jeweiligen Headerbits in disjunktiver Normalform angegeben. Zum Beispiel der Header $1x01$ wird als der boolesche Ausdruck

$$x_0 \wedge \neg x_2 \wedge x_3$$

repräsentiert. Wendet man die Fixpunktiteration auf das Beispiel aus Abbildung 3 an, erhält man schrittweise den Baum aus Abbildung 5, ähnlich wie bei der Breitensuche in normalen Graphen. Die Bezeichnung an den Kanten wurde weggelassen, da das Eingabealphabet hier nur aus einem Zeichen besteht. Es ist gut erkennbar was passieren würde, wenn jeder Knoten viele weitere erzeugen würde. Die Zahl würde exponentiell wachsen und eine weitere Berechnung sehr verlangsamen beziehungsweise unmöglich machen. Außerdem kann man wie schon bei der Header Space Analysis unwichtige Zustände ausblenden, indem man zunächst die Pfade im Netzwerk durchnummeriert und anschließend nur diesen Pfaden folgt.

Das wurde in Abbildung 5 bereits so gemacht, da ansonsten der Baum bereits bei diesem sehr kleinen Netzwerk komplizierter werden würde. Wenn alle erreichten Zustände nach ihren jeweiligen Ports zusammengefasst werden, also unter anderem alle Zustände, die einen Port enthalten, der zu D gehört, erhält genau die gleiche Aufstellung, die man auch mit der eader Space Analysis erhalten hat. Außerdem gelten für die Finite State Machines auch die gleichen Annahmen, wie zum Beispiel die Linear Fragmentation Assumption. So kann man in der Regel bei einem Computernetzwerk davon ausgehen, dass die Zustandsmenge nicht übermäßig wächst.

doi: 10.2313/NET-2014-08-1_12

Neben dem Erreichbarkeitsproblem lassen sich auch andere Probleme, die in der Header Space Analysis gelöst wurden, genauso mithilfe von Finite State Machines bewältigen. Das Problem der Loop-Erkennung wurde in der HSA [1] unnötig verkompliziert, und kann mithilfe allgemeiner Algorithmen auf Finite State Machines [3] gelöst werden.
Die Simulation von FSMs ist bei wachsender Komplexität immer schwieriger, da die Zustandsmenge zu schnell explodiert. Allerdings ist der unerwartete Erfolg der Header Space Analysis unumstritten und der gezeigte Isomorphismus zu Finite State Machines führt dazu, das man mit ihnen ähnlich gute Ergebnisse erzielen kann, und wahrscheinlich sogar auf effizienteren Wegen. Da es für die Simulation von Automaten optimierte Software gibt, die über die Jahre immer weiter optimiert wurde.

4. SCHLUSS

Nachdem die Header Space Analysis bei den Tests am Stanford Netz gute Resultate lieferte, kann man davon ausgehen, dass die getroffenen Annahmen zumindest zum Teil der Wirklichkeit entsprechen, unter denen die Komplexität wesentlich geringer ist als die gezeigte Worst-Case Laufzeit war. Aber wenn schon die, umständlichen und schwächeren Techniken zu ansehnlichen Resultaten führen [3], dann ist es wahrscheinlich möglich mit Finite State Machine Algorithmen bessere Resultate zu erzielen. Nichtsdestotrotz funktioniert der Ansatz der Header Space Analysis, auch wenn es sich dabei im Großen und Ganzen um eine FSM für Computernetzwerke handelt. Zudem sollte man die Methoden noch gegen größere Netzwerke testen, um die Stabilität des Verfahrens genauer zu prüfen, da die Effizienz bislang nur mit Annahmen über die Netzwerke und nicht über die Algorithmen begründet wurde. Alles in allem lässt der unerwartete Erfolg der Header Space Analysis darauf hoffen, dass man sich wieder mehr mit der formalen Verifikation von Finite State Machines für Computernetzwerken beschäftigt.

5. LITERATUR

[1] Peyman Kazemian, George Varghese, and Nick McKeow. Header space analysis: Static checking for networks. *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012.

[2] Dominik Stoffel, Markus Wedler, Peter Warkentin, and Wolfgang Kunz. Structural fsm traversal. *IEEE Transactions on Computer-Aided Design of Integrated Circuits ad Systems*, 23(5), 2004.

[3] Shuyuan Zhang, Sharad Malik, and Rick McGeer. Verification of computer switching networks: An overview. *Automated Technology For Verification and Analysis*, 2012.

doi: 10.2313/NET-2014-08-1_12