INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit in Informatik

Konzeption und Entwicklung eines Detektors zur Erkennung von Verletzungen der Netzneutralität

Bastian Asam

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit in Informatik

Konzeption und Entwicklung eines Detektors zur Erkennung von Verletzungen der Netzneutralität

Bastian Asam

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Vitalian Danciu Martin Metzker Dr. Nils gentschen Felde
Abgabetermin:	17. Juni 2011

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 17. Juni 2011

. . . . . .
(Unterschrift des Kandidaten)

Kurzfassung

In dieser Arbeit wird ein Detektor zur Erkennung von Verletzungen der Netzneutralität konzipiert, entwickelt und in Form eines Prototypen realisiert. Eine ausführliche Anforderungsanalyse zusammen mit einer umfassenden Recherche verwandter Themengebiete und alternativer Werkzeuge bilden die Voraussetzung für das Konzept.

Der Detektor wird als verteiltes System, bestehend aus einem Server und beliebig vielen Agenten, konzipiert. Der Server koordiniert über ein eigens entwickeltes Protokoll Agenten, die dann mit Hilfe von Werkzeugen Verkehr generieren und Messungen durchführen. Zusammen mit Komponenten zur Eingabeverarbeitung, zur Auswertung und zur zeitlichen Steuerung bildet dies die Architektur des Detektors.

Um sowohl eine zeitliche als auch eine technische Trennung von Testspezifikation und Implementierung zu gewährleisten, wird eine Eingabesprache entworfen, über die in einer deskriptiven Form verschiedenste Tests spezifiziert werden können. Ein Test in dieser Sprache besteht aus jeweils beliebig vielen Agenten (Messpunkten), Probes (Verkehrstypen), Trigger (ereignisbasierten Aktionen) und zeitlichen Rahmenbedingungen.

Eine solche Testspezifikation enthält alle für einen Test notwendigen Informationen und steuert den Detektor, ohne auf eine weitere Interaktion mit einem Benutzer angewiesen zu sein. Durch ausgewählte und mit dem Prototypen durchgeführte Feldversuche wird gezeigt, dass der hier konzipierte Detektor als Werkzeug zur Erkennung von Netzneutralitätsverstößen eingesetzt werden kann.

Abstract

In this thesis, a detector to recognize violations of net neutrality is designed, developed and implemented in the form of a prototype. A detailed requirements analysis together with a comprehensive research of related topics and alternative tools are a prerequisite for the concept.

The detector is designed as a distributed system consisting of a server and many agents. Through a specially developed protocol the server coordinates agents that generate traffic and perform measurements with the help of tools. Together with components for input processing, evaluation and timing, this forms the architecture of the detector.

To ensure both, a temporal and a technical separation of test specification and implementation, a command language is designed that can specify various tests in a descriptive form. A test in that language consists of any number of agents (measurement points), probes (traffic types), trigger (event-based actions) and time constraints.

Such a test specification includes all for a test necessary information and controls the detector, without the need to further interact with a user. Selected and with the prototype performed field tests showed that this designed detector can be used as a tool to recognize net neutrality violations.

Inhaltsverzeichnis

1 Einleitung

Das Thema Netzneutralität wird derzeit vor allem politisch heftig diskutiert. Verletzungen der Netzneutralität sind heute technisch relativ einfach durchzuführen aber mangels geeigneter Werkzeuge schwer nachzuweisen.

In dieser Arbeit wird deshalb eine Architektur für einen flexiblen Detektor für Verletzungen der Netzneutralität entwickelt. Zuerst wird untersucht auf welche Weise Verletzungen durchgeführt werden können, wie sich deren Auswirkungen bemerkbar machen und wie sie durch Messungen des Netzverkehrs nachgewiesen werden können. Da die Bandbreite an Manipulationsmöglichkeiten sehr groß ist, wird der Detektor als Framework konzipiert, das leicht erweitert und angepasst werden kann. Der technische Fortschritt wird zu weiteren, jetzt noch nicht bekannten Verfahren führen, die dann in den Detektor integriert werden können.

Als geeigneter Ansatz zur Erkennung von Verstößen gegen die Netzneutralität wird das active Probing aus dem Bereich des Fehlermanagements als grundsätzliches Testverfahren mit vielseitigen Testmöglichkeiten gewählt. Dieser generische Ansatz unterscheidet sich daher auch gravierend von bereits existierenden Werkzeugen, die meist ein paar spezifische Arten von Netzverkehr auf Manipulationen untersuchen können.

Zur Ermittlung der für das Erkennen von Netzneutralitätsverletzungen notwendigen Fähigkeiten des Detektors wird in Kapitel 2 eine Anforderungsanalyse erstellt. Neben einem politischen Exkurs über das Thema Netzneutralität in Kapitel 3, wird daraufhin nach einer genauen Betrachtung relevanter wissenschaftlicher Arbeiten und bereits nutzbarer Werkzeuge ein geeignetes Konzept in Kapitel 4 entworfen.

Die Steuerung des Detektors erfolgt in einer extra konzipierten Eingabesprache, die Spezifikationen verschiedenster auch äusserst komplexer Tests unterstützt und für eine eindeutige Art der Eingabe sorgt. Die Eingabesprache selbst wird über eine formale Sprache ähnlich der Erweiterten Backus-Naur-Form (EBNF) definiert, über die auch die Erzeugung eines Parsers erfolgt.

Als Architektur wird ein verteiltes System verwendet, bei dem ein Server viele über das Internet verteilte Agenten für aktive Messungen mit künstlich erzeugtem Verkehr einsetzt und deren Ergebnisse zentral verwaltet und auswertet.

Alle nach den Anforderungen notwendigen Architekturkomponenten werden entworfen und schließlich am Ende in Form eines Prototyps als Proof of Concept implementiert (Kapitel 5) und anhand von Feldversuchen in Kapitel 6 evaluiert.

Kapitel 7 fasst die Arbeit zusammen, erörtert die erreichten Ziele und gibt einen Ausblick auf das weitere Vorgehen.

Im Folgenden gibt dieses Kapitel eine für diese Arbeit gültige Definition von Netzneutralität und führt eine spezifische Betrachtung des Problembereichs durch.

1.1 Definition Netzneutralität

Netzneutralität wird aus verschiedenen Perspektiven definiert, eine allgemein akzeptierte Definition existiert nicht.

Eine gebräuchliche universelle Definition fordert pauschal die Gleichbehandlung von Internetverkehr [Hon08], das heißt es darf keinerlei Differenzierung (im Kontext der Netzneutralität üblicherweise als Diskriminierung bezeichnet) von Nachrichten im Netz statt finden. Es gibt jedoch einige weitere Definitionen, die sich in ihrer Striktheit bezüglich der Auslegung von Diskriminierung unterscheiden.

1.1.1 Limitierte Diskriminierung und Kategorisierung

Der Erfinder des World Wide Web, Tim Berners Lee, beschrieb in seinem Blog 2006 Netzneutralität folgendermaßen:

> *„If I pay to connect to the Net with a certain quality of service, and you pay to connect with that or greater quality of service, then we can communicate at that level.“* [Lee06a]

Mit anderen Worten: Es darf kein Netzbetreiber die Konnektivität zugunsten oder zulasten bestimmter Endkunden, Inhalte oder Dienste beeinflussen [Sie11]. Es ist seiner Meinung nach entscheidend, dass zwei über das Internet verbundene Kommunikationspartner jede beliebige Internetanwendung nutzen können, ohne aufgrund ihrer Identität oder ihrer Art der Nutzung diskriminiert zu werden. Der Kunde bezahlt im Allgemeinen für die Verbindung zum Netz, unabhängig von der vorgesehenen Nutzung dieser Verbindung. Für exklusiven Zugang zu einem Netzteilnehmer darf aber nicht bezahlt werden müssen [Lee06b].

Weiter hebt er hervor, dass Netzneutralität nicht einen kostenlosen Internetzugang impliziert oder Aufpreise für höhere Service Qualität verbietet.

1.1.2 Limitierte Diskriminierung ohne QoS Kategorisierung

Diese Definition stammt aus der US-amerikanischen Gesetzgebung und erlaubt nur die Priorisierung von Inhalt (Content), Anwendungen oder Diensten, wenn dafür keine unterschiedlichen Preise erhoben werden [DSK+07].

Davon abgesehen darf ein Breitband Service Provider die Möglichkeit einer beliebigen Person den Breitband Zugang für beliebige Inhalte, Anwendungen oder Dienste zu nutzen weder blocken, diskriminieren, beeinflussen, verschlechtern oder sonst irgendwie störend darauf einwirken [DSK+07].

1.1.3 Absolut keine Diskriminierung

Folgt der vorher beschriebenen allgemeinsten Form der Definition und wird von Tim Wu folgenderweise erklärt:

> „Network neutrality is best defined as a network design principle. The idea is that a maximally useful public information network aspires to treat all content, sites, and platforms equally.“ [Wu]

Imprint Magazine zitiert in diesem Zusammenhang Susan P. Crawford, Professorin an der Juristischen Fakultät Cardozo der Yeshiva Universität, New York City [Uhl07]. Demnach muss ein neutrales Internet Pakete nach der first-come, first-serve Strategie weiterleiten, ohne Beachtung möglicher Quality of Service Überlegungen. Auf Wikipedia werden diese beiden Standpunkte als unterschiedliche Definitionen aufgeführt [Wik]. Im Ergebnis sind diese Definitionen allerdings äquivalent und erlauben keinerlei Diskriminierung oder Manipulation von Netzverkehr.

Für diese Arbeit ist ein neutrales Internet als absolut diskriminierungsfrei definiert. Folglich wird im Weiteren jede differenzierte Behandlung von Netzverkehr als Verstoß gegen die Netzneutralität betrachtet.

1.2 Problembeschreibung

Es gibt eine Vielzahl an Kriterien nach welchen der Verkehr differenziert werden kann. Eine Gruppierungsmöglichkeit bietet das OSI-Schichtenmodell mit Blick auf die im Internet eingesetzten Protokolle und deren Header Informationen:

- Schicht 3: nach Protokoll, Quell- und Zieladresse, Verschlüsselung, Fragmentierung und Service Typ
- Schicht 4: nach Protokoll, Port und Fenstergrößen, Sequenznummern und den Umfang transportierter Nutzdaten
- Schicht 7: nach Nutzdaten, Session und Verschlüsselung
- Schicht unabhängig: nach dem gesamten Datenvolumen und der Senderichtung

Eine Manipulation kann ebenfalls unterschiedliche Ausprägungen aufweisen. Es können sämtliche Pakete verworfen werden wie zum Beispiel bei einer Portsperre, verfälschte Fenstergrößen für die Flusssteuerung propagiert werden oder durch das Verwerfen einiger Pakete die Überlastkontrolle mißbraucht werden.

Eine Untersuchung der Nutzdaten erfolgt durch eine sogenannte Deep Packet Inspection (DPI), die für einen unbeeinflussten Betrieb bei hohen Bandbreiten schnelle und spezielle Hardware voraussetzt.

Weitere Charakteristiken, die Netzbetreiber zum Anlass für Manipulationen nehmen könnten, sind über fortgeschrittene Betrachtungen des Kommunikationsvorgangs zu gewinnen oder aus betriebswirtschaftlichen Abhängigkeiten der Betreiber zu schließen:

- Verkehr kann anhand des eingeschlagenen Weges charakterisiert werden.
- Eine bestimmte Richtung, Übertragungsrate und das Quell- und Zielnetz beschreiben einen Datenfluss.
- Aus Sicht eines Netzbetreibers fremder oder externer Datenverkehr wird abhängig von der Belastung nur zu bestimmten Zeiten stärker vernachlässigt.
- Peering Verträge regeln Zeit und Volumen des Datenaustausches zwischen autonomen Systemen.

Diese Listen erheben keinesfalls Anspruch auf Vollständigkeit, geben aber einen ausreichenden Einblick in die Vielzahl an Manipulationsmöglichkeiten und deren Hintergründe, die, abhängig vom Einfallsreichtum der Netzbetreiber, noch zunehmen werden und eine erschöpfende Betrachtung sehr schwierig machen.

Als Resultat solcher Manipulationen verhält sich das Netz ähnlich zu einem mit defekten Komponenten. Deshalb liegt es nahe, Verfahren zur Fehlererkennung zu betrachten, welche bereits seit vielen Jahren Gegenstand intensiver Forschungen sind. Einige im Umfeld des Fehler Managements publizierte Verfahren lassen sich prinzipiell auch in dem hier betrachtetem Gebiet der Netzneutralitätsuntersuchungen anwenden. Davon besonders die aktiven Verfahren, die über generierte Nachrichten, so genannte Probes, Fehler zu lokalisieren versuchen.

Hier wiederum können Verfahren zur Topologie Erkennung mit in Betracht gezogen werden, um z.B. verdächtige einzelne Systeme besser zuordnen und damit vollständiger testen zu können.

Allerdings müssen die Verfahren auf ihre Eignung für das Auffinden von Netzneutralitätsverstößen untersucht werden und gegebenenfalls durch den Einsatz neuer Testnachrichten sinnvoll erweitert werden, da insbesondere die Unterscheidung von gezielten Manipulationen gegenüber durch Fehler hervorgerufenen Irregularitäten in diesen Arbeiten nicht adressiert wird.

Aus diesen Beobachtungen lässt sich die Erkenntnis ziehen, dass ein Detektor für Netzneutralitätsverstöße nicht auf einmal erschöpfend entwickelbar ist, sondern viel mehr adaptiv für die Erkennung aller Arten von Manipulationen erweiterbar sein muss.

1.3 Ergebnisse der Arbeit

Die Arbeit legt das Fundament eines Detektors zur Erkennung von Verletzungen der Netzneutralität. Zu diesem Zweck wurde eine Server Agenten Architektur entwickelt, die, über die Möglichkeit sukzessiver Erweiterungen, frei wählbare und aktive Probe Messungen zwischen beliebig vielen im Internet verteilten Agenten durchführen kann.

Alle Arten von Messungen, insbesondere auch komplexe zusammengesetzte Messungen z.B. zum Zweck der Differenzierung zwischen unerwünschten Fehlern und beabsichtigter Manipulationen, werden als Tests in einer wohldefinierten Eingabesprache vollständig spezifiziert. Auf diese Art wird die Testspezifikation zeitlich und technisch von ihrer Umsetzung getrennt und Einschränkungen, wie bei anderen Arbeiten, vermieden.

Die Eingabesprache wird mit Hilfe eines Parser Generators neben weiteren Komponenten wie z.B. einem Timer in der Architektur so integriert, dass der Detektor sowohl bezüglich der Messungen, als auch bezüglich des Agentensystems leicht erweiterbar ist. Zur Kommunikation zwischen Server und beliebig vielen Agenten kommt darüber hinaus ein eigens für diesen Zweck konzipiertes Protokoll zum Einsatz.

Als Evaluation der Konzepttauglichkeit wurde ein Prototyp mit rudimentären Testmöglichkeiten implementiert und darüber ausgewählte Testfälle im Einsatz dokumentiert. Sie zeigen, dass der Prototyp in der Lage ist interessante Verkehrsbeobachtungen durchzuführen, eine Auswertung der Ergebnisse allerdings noch über den Nutzer erfolgen muss.

Eine Weiterentwicklung des Detektors ist an vielen Stellen sinnvoll und erforderlich und gerade für eine verlässliche und fundierte Erkennung von Verletzungen der Netzneutralität notwendig.

Im nächsten Kapitel wird für ein ausgewähltes Unternehmen eine durch Netzneutralitätsverletzungen ausgehende reelle Bedrohung als Szenario dargestellt und aus Sicht des Unternehmens entscheidende Fragen zu solchen Manipulationen gestellt. Darunter ist auch die Frage, welche Teile der Problembeschreibung aus Abschnitt 1.2 tatsächlich von Netzbetreibern zur Verkehrsdifferenzierung genutzt werden. An einen unter anderem diese Fragen beantwortenden Detektor von Netzneutralitätsverstößen werden in der Folge ganz bestimmte Anforderungen gestellt und in einem Katalog als Basis für das weitere Vorgehen aufgelistet.

2 Anforderungsanalyse

Potenziell bergen Verletzungen der Netzneutralität auch für Unternehmen eine große Gefahr. Bieten diese Dienste über das Internet an, sind sie von einem funktionierenden und verlässlichen Internet abhängig. Häufig werden dabei bestimmte Arten von Dienstgüten (Quality of Service, QoS) garantiert, wohl wissend, dass diese nicht vom Netz unterstützt werden und daher auf Erfahrungswerten und großzügig ausgelegten Infrastrukturen basieren.

Wäre das Netz nicht neutral, könnten Unternehmen direkten Einfluss auf die Priorität ihres Verkehres ausüben. Da allerdings derzeit noch heftig über die Zukunft der Neutralität im Internet debattiert wird (vgl. Abschnitt 3.1), gibt es noch keine klaren Festlegungen darüber, wie Netzbetreiber mit ihrem Verkehr umgehen dürfen. Für Unternehmen ist dies ein unbefriedigender Zustand, weil das Netz heute einerseits nicht mehr frei von Manipulationen ist, andererseits aber auch noch keine Tarife für priorisierten Verkehr angeboten werden.

Im Folgenden wird anhand eines Szenarios die derzeitige Problematik für ein global agierendes Unternehmen am Beispiel der Amadeus IT Group SA betrachtet und daraus Anforderungen für einen Detektor zur Erkennung von Verletzungen der Netzneutralität abgeleitet. Eine verlässliche Erkennung ist Grundvoraussetzung um Amadeus eine sinnvolle Reaktion auf Manipulationen zu ermöglichen.

2.1 Szenario

Die Amadeus IT Group vertreibt das gleichnamige Computer Reservation System (CRS) in Verbindung mit technischen Lösungen zu dessen Betrieb [SA]. Damit gehört Amadeus zu den führenden Anbietern von Buchungssystemen für die Reise- und Tourismusindustrie weltweit. Sowohl Reiseanbieter wie z.B. die Bahn, Airlines, Reedereien, Hotels und Autoverleiher als auch Reiseagenturen verwenden das Amadeus CRS, dessen Transaktionen, immerhin mehr als 850 Millionen allein im Jahr 2009, zentral in Europas größtem Rechenzentrum in Erding bei München (Deutschland) bedient werden.

2.1.1 Ausgangssituation

Amadeus sieht Manipulationen im Sinne von Verletzungen der Netzneutralität als reale Bedrohung ihres Geschäfts an und sucht nach Möglichkeiten diese nachzuweisen. Bisher sind lediglich vage Vermutungen auf Basis von Beobachtungen des Netzverkehrs möglich. Ein fundierter Nachweis soll sowohl effektive Gegenmaßnahmen ermöglichen, als auch eine verbesserte Gesprächsgrundlage in Verhandlungen mit Netzanbietern schaffen. Dieses Ziel soll in Kooperation mit der Ludwig-Maximilians-Universität München erreicht werden.

Jede einzelne Transaktion im Netz von Amadeus muss letztlich über das Internet zum Rechenzentrum in Erding gesendet werden. Dazu befinden sich sämtliche Kundensysteme in einem weltumspannenden virtuellem privatem Netz-Verbund (Virtual Private Network, VPN), der über die Netze vieler verschiedener Netzbetreiber in unterschiedlichen Ländern realisiert wird.

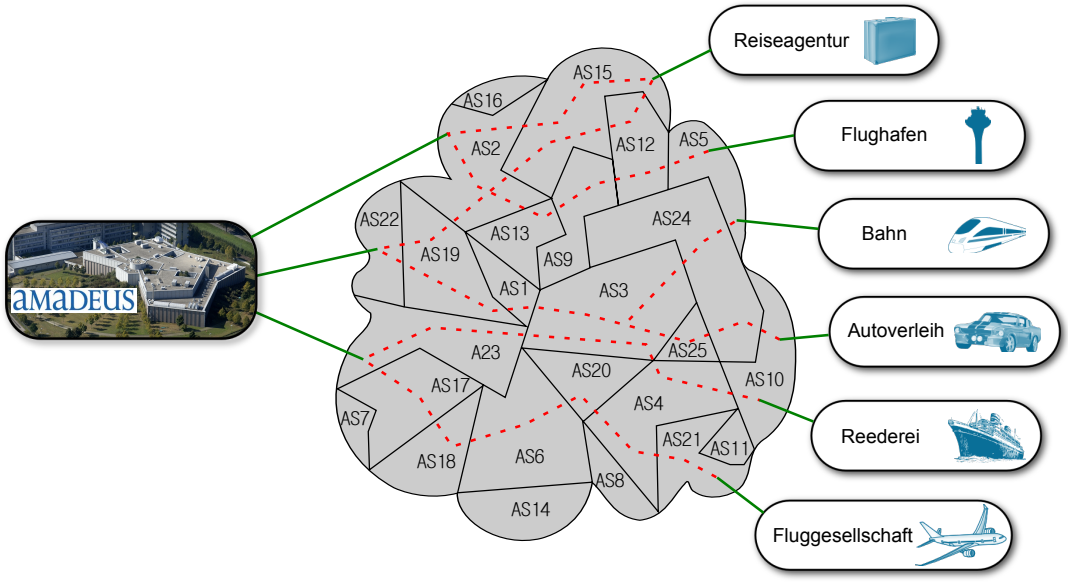


Abbildung 2.1: Amadeus Netz

Abbildung 2.1 zeigt schematisch den Aufbau des Amadeus Netzes. Bekannt sind jeweils nur die autonomen Systeme (AS), die für Amadeus Kunden und Amadeus selbst den Netzzugang realisieren und in der Zeichnung über grüne durchgezogene Linien angebunden sind. Welche Wege aber für die Verbindung dieser Netzzugänge im Internet eingeschlagen werden, hängt an den jeweiligen Routing Entscheidungen der durchlaufenen AS, d.h. die roten gestrichelten Linien bilden Wege durch das Internet, die Amadeus unbekannt sind, laufend Änderungen unterliegen und an vielen Stellen in verschiedenen AS Manipulationen erfahren können.

Wird nun beispielsweise bei der Anbindung des Flughafens eine Manipulation beobachtet, verläuft die Route nach Abbildung 2.1 durch die autonomen Systeme 2,13,9,12 und 5. Dies kann Amadeus allerdings nur durch eine zeitnahe Routenverfolgung ermitteln, da die Routen keinesfalls statisch sind. Jedes dieser AS kann nun für eine Manipulation verantwortlich sein, insbesondere auch mehrere gleichzeitig. Eine Identifizierung der verantwortlichen AS ist daher eine herausfordernde Aufgabe.

Die mit einem VPN verbundene Verschlüsselung des produktiven Datenverkehrs sorgt einerseits für Sicherheit, macht es aber andererseits den Netzanbietern unmöglich die Daten im Rahmen einer Deep Packet Inspection zu identifizieren, da der Dateninhalt nicht entschlüsselt werden kann. Dadurch kann das Amadeus Netz definitiv nicht unter Manipulationen auf Grund bestimmter Dateninhalte leiden. Möglich wäre aber eine generelle differenzierte Behandlung von verschlüsseltem Verkehr.

Eine Netzneutralitätsverletzung zeigt sich im Amadeus VPN nur durch Datenverlust, Verzögerungen, Umleitungen und Verbindungsabbrüche. Da die Attraktivität des Produktes CRS wesentlich von dessen Bedienkomfort und damit u.a. von der Reaktionszeit und Ausfallsicherheit des Systems abhängt, welche wiederum durch die Qualität der Internetverbindung beeinflusst werden, sorgt dies für einen außerordentlich hohen Stellenwert einwandfreier Konnektivität bei Amadeus.

Beobachtet wurden schon einige verdächtige Vorgehensweisen wie z.B. offensichtlich zeit-

gesteuerte Verbindungsabbrüche mit unter Umständen so kurzen Intervallen, dass eine verschlüsselte Verbindung praktisch nicht nutzbar ist. Mit Abstand am häufigsten wahrgenommen wurde das sogenannte Hot Potato Routing, anhand welchem autonome Systeme Pakete über den kürzesten Weg raus aus deren Netz zum nächsten, im Border Gateway Protocol (BGP) Pfad verzeichneten, AS weiterleiten. Dies führt zu global suboptimalen Pfaden mit höheren Verzögerungen [SS10], stellt aber bei Gleichbehandlung des Verkehrs keine Verletzung der Netzneutralität dar. Für einen fundierten Nachweis von gezielten und beabsichtigten Manipulationen fehlt Amadeus bisher ein geeignetes Werkzeug. Und solange Netzanbieter Manipulationen dementieren, entbehrt ein Vorgehen dagegen jeglicher Grundlage.

2.1.2 Fragestellungen und Herausforderungen

Derzeitiger Wunsch von Amadeus ist vor allem das Verhalten der Netzbetreiber besser analysieren zu können, deren Möglichkeiten besser zu verstehen und die aktuelle Praxis im Umgang mit Netzneutralität zu erkunden. Im Speziellen sollen folgende Fragen geklärt werden:

- **Wird überhaupt manipuliert?**

  Liegt bei einer bestimmten Unregelmäßigkeit überhaupt ein gezielter Manipulationsversuch eines Providers vor oder handelt es sich um normales Routing-Verhalten oder um einen ungewollten Fehler?

- **Was wird manipuliert?**

  Welche Art von Verkehr ist betroffen? Betrifft es nur bestimmte Kommunikationsteilnehmer? Anhand welcher Kriterien wird manipuliert? Sind nur bestimmte Ports betroffen oder werden die Pakete nach deren Inhalt mittels DPI bewertet? Können Manipulationen eventuell umgangen werden?

- **Wie werden Manipulationen realisiert?**

  Über welche Technik verfügen die Netzbetreiber zur Durchführung von Manipulationen? Können nur einzelne Pakete betrachtet werden oder ganze Verkehrsströme?

- **Wie zeigen sich Manipulationen?**

  Welche Auswirkungen hat die Manipulation? Werden Verbindungen getrennt oder nur langsamer? Wächst die Round Trip Time (RTT) oder verringert sich der Durchsatz?

- **Wann wird manipuliert?**

  Wird erst nach bestimmten Datenmengen oder nach einer gewissen Zeit manipuliert? Wie lange hält eine solche Manipulation an? Gibt es Tageszeiten mit verstärkter Manipulation?

- **Wer manipuliert?**

  Wo geschieht die Manipulation, bzw. welcher der auf dem Weg durchlaufenen Provider ist dafür verantwortlich?

Die Beantwortung dieser Fragen impliziert die Möglichkeit, Netzneutralitätsverstöße fundiert nachzuweisen.

2.2 Anforderungskatalog

Das Szenario mit seinen Fragestellungen ist Ausgangspunkt für die Identifizierung von Anforderungen, die an einen Detektor zur Erkennung von Verletzungen der Netzneutralität zu stellen sind, und entscheidend für die Konzeption und Entwicklung eines solchen Werkzeuges.

1. **Variable Testmöglichkeiten**

   Eine Verletzung der Netzneutralität kann auf vielfältige Art geschehen. Die Bandbreite der möglichen Manipulationen reicht von simplen Port Blockierungen und zeitlich abhängigen Drosselungen bis zu hoch speziellen Eingriffen auf Basis genauer Untersuchungen der Nutzdaten durch DPI. Um Untersuchungen in allen Richtungen vornehmen zu können, muss der Detektor uneingeschränkt alle denkbaren Testmöglichkeiten unterstützen.

2. **Vergleichbarkeit**

   Ergebnisse, die aus Messungen des Detektors stammen, müssen für sinnvolle Auswertungen vergleichbar sein. Es muss sichergestellt werden, dass bei unveränderter Konfiguration des Detektors und keinerlei Veränderungen im Netz, ein wiederholter Test zu ähnlichen Ergebnissen führt. Ist dies nicht der Fall, kann damit auf Veränderungen im Netz geschlossen werden.

3. **Erweiterbarkeit**

   Die Möglichkeiten einer Manipulation sind entsprechend der Problembeschreibung in Abschnitt 1.2 vielfältigst und abhängig von der technischen Entwicklung. So waren DPIs bei Leitungen mit 10 GBit/s überhaupt erst seit 2007 möglich und damit innerhalb autonomer Systeme einsetzbar [JM07]. Um den Detektor für die Vielzahl an Testmöglichkeiten sowohl in der Gegenwart als auch in der Zukunft auszulegen und im Hinblick auf eine sukzessive Weiterentwicklung des Detektors durch folgende Arbeiten müssen Leistungsmerkmale einfach erweitert werden können. Zu diesen gehören nicht nur zusätzliche Messmethoden und Messpunkte, sondern auch Module zur Analyse der Messungen bis hin zu Möglichkeiten einer automatisierten Reaktion.

4. **Skalierbarkeit**

   Das einfache Einbinden neuer Messpunkte in den Detektor ist notwendig, um Netze weiterer autonomer Systeme untersuchen zu können. Eine effiziente Unterstützung vieler Messpunkte ermöglicht eventuelle Unterschiede zwischen Ländern, z.B. aufgrund unterschiedlicher Gesetzgebungen, festzustellen sowie Netzneutralitätsverletzer zu lokalisieren.

5. **Zeitliche Steuerung**

   Das typische Internet Nutzungsverhalten mit deutlichen Unterschieden zwischen Tag und Nacht [Mon08] oder zu besonderen Ereignissen wie der Fussball WM 2010 [Gmb10] lässt vermuten, dass es zu Spitzenzeiten eher zu regulierenden Maßnahmen von Seiten der Netzbetreiber kommt, als zu vergleichsweise ruhigen Zeiten. Für eine möglichst flexible und vollständige zeitliche Steuerung zur Untersuchung genau solcher Vorkommnisse müssen Tests sowohl zu bestimmten Zeitpunkten und über definierte Zeitspannen als auch in Intervallen durchgeführt werden können.

6. **Ereignisbasierte Reaktion**

   Um solche im vorherigen Punkt angesprochenen zeitlichen Unterschiede zu bemerken, ist eine definierbare Reaktion auf bestimmte Ereignisse hilfreich. Damit lassen sich dann beispielsweise weitere Messmethoden starten oder zusätzliche Messpunkte beteiligen. Gerade auch zur Lokalisierung eines Netzneutralitätsverletzers ist es sinnvoll, diese erst nach einem konkreten Verdacht zu starten.

7. **Integration externer Programme**

   Netzbetreiber regulieren nach einschlägigen Verkehrstypen wie z.B. Voice over IP [Dud06] oder Peer to Peer Verkehr [Con09]. Für realistische Untersuchungen bedarf es korrekter Verkehrsmuster solcher Dienste. Diese künstlich zu generieren ist allerdings oft äusserst aufwendig und für den hier gewünschten Zweck unnötig. Es reicht aus, diese Dienste über einschlägig bekannte Software zu nutzen und deren Verhalten zu evaluieren.

8. **Einfache Bedienung**

   Eine einfache unmissverständliche Bedienung mit klar definierbarem Verhalten ermöglicht auch aussenstehenden Personen, die an der Entwicklung nicht beteiligt waren, die Nutzung des Detektors. Trotz vieler weit verstreuter Messpunkte sollte die Bedienung von lediglich einer Person aus über beispielsweise eine einzige Benutzeroberfläche erfolgen können. Damit lässt sich gleichzeitig auch die Möglichkeit schaffen, den Nutzer bei der Auswertung der Daten mit Grafiken zu unterstützen.

9. **Sparsamer Umgang mit Ressourcen**

   Die Tatsache, dass möglichst viele Messpunkte an möglichst vielen Orten der Erde im Allgemeinen zu besseren Ergebnissen beitragen, führt zu dem ökonomischen Wunsch, diese eventuell auf kleinen embedded Geräten zu platzieren und möglichst nach dem Plug-and-Play Prinzip zu konfigurieren. Dies senkt die Kosten pro Messpunkt sowohl in der Anschaffung als auch im Betrieb deutlich und vereinfacht zudem deren Verteilung. Hier gibt z.B. das in Kapitel 3.3.2 vorgestellte Atlas Projekt von RIPE einen guten Eindruck, wie so etwas aussehen kann.

Im Weiteren wird diese Arbeit das Ziel verfolgen ein Konzept für einen Detektor zur Erkennung von Verletzungen der Netzneutralität zu entwickeln, das diese Anforderungen erfüllt.

Für die Entwicklung eines geeigneten Konzeptes werden im nächsten Kapitel andere Arbeiten zum Nachweis von Netzneutralitätsverstößen diskutiert, verwandte Themenbereiche betrachtet und bereits entwickelte Werkzeuge besprochen und von dem hier entwickelten Detektor abgegrenzt. Ein Exkurs zur politischen Diskussion unterstreicht ausserdem die Relevanz des Themas.

3 Stand von Politik, Wissenschaft und Technik

In der Politik wird das Thema Netzneutralität bereits weitreichend diskutiert. Einführend zeigt Kapitel 3.1 eine Übersicht über die aktuelle Diskussion und belegt anhand von einigen Beispielen, dass Netzneutralitätsverstöße heute schon längst Realität sind. Weitere Untersuchungen zur Netzneutralität sind Voraussetzung für qualifizierte politische Entscheidungen und mehr Transparenz in der Telekommunikationsbranche.

Anschließend wird der Stand der aktuellen Forschung diskutiert und mangels expliziter Literatur zum Thema Netzneutralität verwandte Themengebiete identifiziert und erörtert. So gleicht das Erkennen von Verstößen z.B. in vielerlei Hinsicht dem Erkennen von Fehlern. Auch die Methoden, die solche Manipulationen erst ermöglichen und im Umfeld des Traffic Engineerings eingesetzt werden, kommen aus der Forschung für Netz Management und Quality of Service (QoS) Management. Verfahren zur Topologie Erkennung bieten darüber hinaus interessante Ansätze zur Ermittlung manipulierter Verbindungspfade.

Abschließend werden noch bereits existierende Werkzeuge, die in der Lage sein sollen die Netzneutralität zu testen, vorgestellt und von dieser Arbeit abgegrenzt.

3.1 Aktuelle Diskussion

Die Diskussion über Netzneutralität entwickelte sich bereits Ende des 20. Jahrhunderts aus möglichen, aber damals noch theoretischen Bedrohungen gegen die Ende zu Ende Charakteristik des Internets. Diese rät dazu, die Intelligenz in einem Netz an die Spitze eines geschichteten Systems zu setzen, an dessen Ende Nutzer mittels Anwendungen über das Netz mit Hilfe möglichst einfacher und allgemeiner Protokolle kommunizieren. Im Speziellen ging diese Bedrohung von der Integration von Internet Service Providern (ISP) in Kabelnetz Betreibern aus, mit der Folge, dass Kunden gezwungenermaßen den ISP ihrer Kabelbetreiber nutzen müssen und der damit fehlende Wettbewerb zu höheren Preisen führt [LL00].

Der Begriff 'Net Neutrality' wurde erst später von Tim Wu geprägt [Wu11], der 2002 einen Artikel über Breitband Diskriminierung schrieb und dort Network Neutrality als Konzept einführte [Bil10]. Welche Tragweite das Thema mittlerweile erreicht hat, zeigt auch Tim Wu's aktuelle Stelle als Senior Advisor für die Federal Trade Commission (FTC), der US-amerikanischen Bundeshandelskommission, im 'Office of Policy Planning' [Bil11]. Dies assistiert der Kommission bei der Entwicklung und Durchsetzung von weitreichenden Wettbewerbs- und Verbraucherschutz Initiativen und berät in Fällen von neuen oder komplexen politischen und rechtlichen Belangen [DeS11].

Der Ruf nach einer Regulierung mit dem Ziel die Netzneutralität gesetzlich zu sichern ist heftig umstritten. Dabei wird meist übersehen, dass es durchaus historische Erfahrungen auf dem Gebiet gibt, wenn man die interessanten Parallelitäten zwischen dem Internet- und dem gewöhnlichen Transportgeschäft (Waren- und Personentransport) sieht.

In den USA gibt es beispielsweise mit dem Communications Act bereits seit 1934 verbindliche Pflichten für Transportunternehmen. Darin wird unter anderem eine Versorgungspflicht, ein Diskriminierungsverbot und eine Verbindungspflicht zwischen den einzelnen Unternehmen geregelt. Begründet wurden die Reglementierungen durch das öffentliche Interesse an gerechten Transportmöglichkeiten für jedermann zu vernünftigen Preisen und der häufigen Monopolstellung von Transportunternehmen [Spe02]. Sogar die Telefongesellschaften wurden unter diese Pflichten gestellt.

Auch wenn die US-amerikanische Telekommunikationsaufsicht (Federal Communications Commission, FCC) erklärt, dass Computer Services nicht unter den Communications Act fallen, könnten daraus wertvolle Ansätze zum Umgang mit Netzbetreibern gewonnen werden und die dort gesammelten historischen Erfahrungen Hilfestellungen bei der Lösung heutiger Streitpunkte beim Thema Netzneutralität geben.

3.1.1 Bekannt gewordene Beispiele von Verletzungen der Netzneutralität

Waren es anfangs eher theoretische Überlegungen, zeigen jüngere Ereignisse vor allem in den USA, dass Verstöße gegen die Netzneutralität bereits Realität sind. Die folgenden Beispiele gewähren zugleich auch einen Einblick in bereits eingesetzte Methoden:

- Der zweitgrößte Internet Service Provider der USA, Comcast, drosselte 2007 gezielt BitTorrent Verkehr. Über Deep Packet Inspection wurden BitTorrent Sessions erkannt und mittels gefälschter TCP Reset Pakete das Flusskontroll-Fenster minimiert. Dies hat einen drastisch verringerten Durchsatz für diesen Peer-to-Peer Dienst zur Folge. Erst nach Einschreiten der FCC wurde diese Praxis aufgegeben [Pas10].

- T-Mobile kündigte Anfang 2009 an, die beliebte VoIP Anwendung Skype für das iPhone in Deutschland zu blockieren und beruft sich dabei darauf, dass die Nutzung von VoIP und Instant Messaging nicht Gegenstand der Verträge sei [Bar09]. Kein viertel Jahr später lenkt T-Mobile teilweise ein und bietet nun für zwei Tarife eine VoIP-Option gegen Aufpreis an [Tel09].

- Beide großen Telekommunikationsanbieter in den USA, Verizon und AT&T, priorisieren ihre jeweiligen IP gestützten Video-on-demand Dienste [Pas10]. Während dies für die Nutzer dieser Dienste durchaus sinnvoll erscheint, liegt die Problematik in der Benachteiligung von Wettbewerbern. Nur die jeweils eigenen Video-on-demand Dienste werden bevorzugt, während beispielsweise Netflix Kunden auf ausreichende Bandbreite hoffen müssen.

3.1.2 Pro Netzneutralität

Wie die Beispiele zeigen ist die Zeit der selbstverständlichen Netzneutralität vorüber. Gefordert werden daher Regeln zum Schutz der Netzneutralität. Ohne denen bestünde die Gefahr der Diskriminierung, Innovationsbremse und Wettbewerbsverzerrung.

Die Deutsche Telekom möchte von datenintensiven Inhalteanbietern, wie z.B. Google und Youtube, eine Gebühr verlangen, da diese die Infrastruktur der Telekom nutzen, um ihre gewinnbringenden Inhalte zu verbreiten. Bisher war die einzige Einnahmequelle der ISPs der Endkunde, der allgemein für seine gewählte Art der Konnektivität (z.B. DSL 6000) zahlt. Mit diesem Geschäftsfeld lassen sich allerdings nicht mehr genug Gewinne erzielen,

weshalb versucht wird, neue Geschäftsfelder zu erschließen. Das Problem an den Plänen ist jedoch, dass die Auswahl an für den Endkunden verfügbaren Inhalten durch die Telekom beschränkt würde und dies eine Form der Zensur darstellt [Bie10].

Die Politik, zumindest in Deutschland und den USA, vertritt derzeit eher die Ansicht, die Netzneutralität schützen zu müssen. Um diese Thematik genauer untersuchen zu lassen, hat die Bundesregierung in Deutschland Mitte 2010 als Teil der Enquete-Kommission 'Internet und digitale Gesellschaft' u.a. die Projektgruppe Netzneutralität gegründet [udG10]. Deren Vorsitzender Peter Tauber sieht in der Netzneutralität eine Vorraussetzung für Innovationen [Tau10]. Zur Sicherstellung der Netzneutralität vertraut die derzeitige Koalition den Kräften des Marktes und will zu diesem Zeitpunkt noch keine regulatorischen oder gesetzlichen Maßnahmen ergreifen, übersieht dabei jedoch die fehlende Transparenz in der Telekommunikationsbranche. Kunden haben gegenwärtig keine ausreichenden Möglichkeiten das Verhalten ihrer Netzbetreiber zu beurteilen.

Anders stellt sich die Situation in den USA dar. Dort hat die FCC Ende 2010 Regeln zum Schutz des offenen Internets beschlossen, die die folgenden sechs Prinzipien umfassen [Kre10a]:

1. ***Transparency***. *Consumers and innovators have a right to know the basic performance characteristics of their Internet access and how their network is being managed.*
2. ***No Blocking***. *A right to send and receive lawful traffic. This prohibits blocking of lawful content, apps, services, and the connection of non-harmful devices to the network.*
3. ***Level Playing Field***. *A right to a level playing field. A ban on unreasonable discrimination. No approval for so-called „pay for priority“ arrangements involving fast lanes for some companies but not others.*
4. ***Network Management***. *An allowance for broadband providers to engage in reasonable network management. These rules don't forbid providers from offering subscribers tiers of service or charging based on bandwidth consumed.*
5. ***Mobile***. *Broadly applicable rules requiring transparency for mobile broadband providers, and prohibiting them from blocking websites and certain competitive applications.*
6. ***Vigilance***. *Creation of an Open Internet Advisory Committee to assist the Commission in monitoring the state of Internet openness and the effects of our rules.*

Trotz der Tatsache, dass diese Regeln gegenüber früheren Entwürfen [Pas10] bereits weniger strikt sind und damit Bürgerrechtsorganisationen nicht weit genug reichen [Kre10b], brachte Verizon kurze Zeit später eine Klage wegen Zweifel an der Autorität der FCC auf den Weg [Kan11].

3.1.3 Contra Netzneutralität

Generell lässt sich beobachten, dass verständlicherweise vor allem die Netzbetreiber als Eigentümer der Netze versuchen, Regulierungen zu verhindern oder wenigstens aufzuweichen.

Ein effektives Verkehrsmanagement (Traffic Engineering) ermöglicht den Betreibern einen besseren Investitionsschutz, da sich vorhandene Infrastruktur dann besser auslasten und damit länger betreiben lässt. Die Deutsche Telekom erreichte für den Ausbau ihrer VDSL

Infrastruktur daher sogar für mehrere Jahre einen extra Paragraphen („Neue Märkte“) im deutschen Telekommunikationsgesetz, der allerdings auf Druck der EU Kommission im April 2011 wieder aufgehoben wurde [Kes11].

Die Netzbetreiber müssen nach neuen Einnahmequellen suchen, um ihre immensen Kosten des nötigen Netzausbaus refinanzieren zu können [FTD10]. Das Problem, dass Inhalteanbieter wie YouTube von schnellen Netzen kostenlos profitieren, kann z.B. umgangen werden, indem solche Anbieter mit den Providern eine bevorzugte Behandlung vereinbaren. Dienste die keine Verzögerungen vertragen, wie beispielsweise Telefon oder Videostreaming, müssten sich dann nicht mehr die vorhandene Kapazität mit den vielen herkömmlichen Diensten teilen, bei denen eine gewisse Verzögerung kein Problem darstellt, wie z.B. bei Mail, Nachrichten und Blogs. Wenn dadurch der Verbraucher störungsfrei über das Internet telefonieren, Videos abrufen oder Computer spielen kann, ist ein Aufpreis für diesen Service gerechtfertigt. Wichtig ist allerdings, und das sollen die FCC Regeln garantieren, den im Internet vorherrschenden Wettbewerb der Ideen und Produkte zu bewahren und jegliche Zensur zu verhindern.

Neben solchen pauschalen Ansätzen sind auch selektivere denkbar, bei dem die Netzbetreiber pro transportiertem Gut entschädigt werden. Wenn ein Kunde beispielsweise einen Film über die mittlerweile zahlreichen Video-on-Demand Anbieter bezieht, benötigt er dafür viel Bandbreite, die der Netzbetreiber teuer zur Verfügung stellen muss [Sin10]. Für den Film entrichtet der Endkunde bereits eine Gebühr an den Video Anbieter, der wiederum einen Teil an die Netzbetreiber abführt. Besonders in diesem Fall kommt es zusätzlich noch zu starken zeitabhängigen Häufungen in den Abendstunden, wodurch für die Sicherstellung eines reibungslosen Betriebs hohe Überkapazitäten erforderlich sind. Die Kosten für deren Bereitstellung müssen daher gerechtfertigterweise auf die Nutzer und Anbieter dieser Dienste umgelegt werden.

Allgemein eignen sich differenzierte Verbindungen aufgrund der Heterogenität der existierenden Dienste im Bezug auf deren Anforderungen an die Internetverbindung [Eul05] deutlich besser dazu, jeden Dienst optimal zu bedienen und für eine bessere Erfahrung für den Nutzer zu sorgen. Ein eingeschränkter Wettbewerb muss daraus nicht zwangsweise folgen.

3.2 Forschung

Trotz der umfangreichen politischen Diskussion trägt noch relativ wenig explizite Literatur zu einer fundierten wissenschaftlichen Auseinandersetzung bei.

Einen guten Einblick in die technische und ökonomische Seite gibt Jon Crowcroft von der Cambridge Universität in seiner Arbeit „Net Neutrality: The Technical Side of the Debate ∼ A White Paper“ [Cro06]. Er gibt zu bedenken, dass alle Argumente für oder gegen Neutralität schon vor dem Internet existierten und die jetzige Diskussion nur das Ergebnis der Fokussierung auf eine Infrastruktur ist. Da das Internet heute Dienste erbringt, die noch vor kurzen dedizierten Infrastrukturen überlassen waren, ergeben sich Konflikte - oft sogar innerhalb eines Unternehmens. So haben Mobilfunkunternehmen ein wirtschaftliches Interesse daran, ihre für Sprachkommunikation entwickelten verbindungsorientierten Dienste wie GSM über freien paketorientierten Diensten wie Skype zu priorisieren. Sie versuchen damit u.a. eine Degradierung zu einfachen Daten Transportern zu verhindern.

Die Integration von früher über dedizierter Infrastruktur angebotener Dienste ins globale

Netz schließt ehemals lukrative Einnahmequellen. Es war selbstverständlich, für Telefonate oder Fernsehen Gebühren zu zahlen, womit die korrespondierenden Infrastrukturen finanziert wurden. Diese Dienste können heute teilweise kostenlos über das Internet genutzt werden oder aber die Vergütung geht an den Betreibern der kostenintensiven Infrastruktur vorbei und ausschließlich zu den Inhalteanbietern.

Crowcroft kommt nach Betrachtung der eingesetzten Netzelemente mit Proxies, Caches, Firewalls usw., den Beziehungen zwischen ISPs, der Evolution der IP Dienste von einfachen textbasierten Nachrichten zu Video- und Sprachdiensten und der Zugangsart der Endkunden sogar zu dem Schluss, dass es noch nie Neutralität im Netz gab, aber aktives Unterscheiden von Verkehrstypen eine Möglichkeit für die Zukunft bietet.

David Passmore von der Burton Group hingegen schlägt bereits mögliche politische und ökonomische Gegenmaßnahmen bei ignorierter Netzneutralität vor [Pas10].

Die Möglichkeit neben dem normalen Verkehr im Internet einen bevorzugt behandelten „Premium“ Verkehr zu schaffen, wird häufig gefordert und käme einem direkten Eingriff in die Netzneutralität gleich. Mit Hilfe von Simulationen kann gezeigt werden, dass die benötigte zusätzliche Kapazität selbst bei geringen Mengen Premium Verkehr einen starken Einfluss auf das Netz haben kann [YRK+08].

Da Verstöße gegen die Netzneutralität in verschiedenster Weise getätigt werden können und deren Auswirkungen mannigfaltig ausfallen können, ist eine dedizierte Untersuchung darauf nur möglich, wenn von verwandten Problemfällen unterschieden werden kann.

Die zu erwartenden Auswirkungen, wie u.a. schlechter Durchsatz, hohe Verzögerung, Verbindungsabbrüche und unerreichbare Dienste, sind weitestgehend identisch zu denen, die durch Fehler versucht werden. Fehler Management, Topologie Erkennung und Traffic Engineering ist dabei bereits seit Jahrzehnten Gegenstand ausführlicher Forschung und bietet auch gute Konzepte für Netzneutralitätsuntersuchungen.

3.2.1 Fehler Management

Seit der Einführung moderner Kommunikationssysteme sind auch Methoden zum Umgang mit Fehlern im Fokus wissenschaftlicher Recherchen. Untersuchungen auf Fehler haben mit der Erforschung von Netzneutralitätsverstößen den Versuch gemein, Anomalien im Netz trotz eingeschränkter Informationen zu erkennen und zu lokalisieren. Der Prozess der Fehler Diagnose kann grob in drei Stufen eingeteilt werden: Fehler Erkennung, Fehler Lokalisierung und Testen [SS04].

Fehler Erkennung

Heutige Netzkomponenten informieren im Fehlerfall selbstständig über sogenannte Alarme einen zentralen Netz Manager. Ein Fehler wird daher in einem so überwachten Netz aktiv von betroffenen Komponenten gemeldet [BCF94]. Dieses Verfahren ist im Umfeld der Netzneutralität nicht anwendbar, da im Sinne der Netzbetreiber kein Fehler, über den es zu berichten gäbe, vorliegt und das zentrale Konzept das Informieren vieler Nutzer nicht vorsieht.

Fehler Lokalisierung

Anders sieht es bei active Probing aus, das neben der Auswertung von Alarmen zur Fehler Lokalisierung zum Einsatz kommt und anhand selbst erzeugter Nachrichten, sogenannter

Probes, Fehler zu lokalisieren versucht. Auf diesem Ansatz basiert auch die grundsätzliche Idee, wie ein Nachweis von Netzneutralitätsverstößen gelingen kann.

Fehler können im Prinzip in allen Schichten des Internet Modells auftreten und für jede Schicht existieren geeignete Probes ([SS02] und [Gop00]). Da nicht jede Schicht für eine Manipulation geeignet ist, lassen sich mit den Probes auch alle für Manipulationen anfällige Schichten untersuchen.

Ermitteln lassen sich durch unterschiedliche Probes eine Menge für diese Arbeit relevante Daten, die durch Manipulationen beeinflusst werden, wie z.B. Messwerte für die Bandbreite, die Verzögerung und den Paketverlust [NS06]. Dabei sind im Weiteren bereits entwickelte Verfahren zur Auswahl geeigneter Probes und deren Sendestationen interessant und eventuell geeignet, in dem hier entwickelten Detektor eingesetzt zu werden. Maitreya Natu und Adarshpal S. Sethi von der Universität Delaware, USA, entwickelten über mehrere Arbeiten hinweg ein komplettes Fehler Diagnose Tool, dessen Systemarchitektur aus drei Hauptkomponenten besteht und in Abbildung 3.1 gezeigt wird.

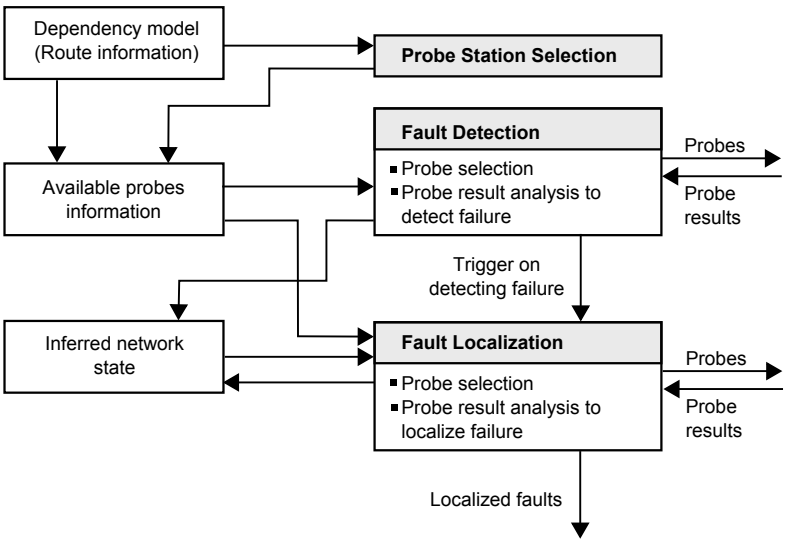


Abbildung 3.1: System Architektur für Fehler Diagnose aus [NS06]

Das Probe Station Selection Modul wählt die besten Standorte für das Aufstellen von Probe Stationen (siehe dazu [NS02]). Das Fault Detection Modul wählt und sendet periodisch die kleinste zum Auffinden eines Fehlers mögliche Menge an Probes ([NS06]) und das Fault Localization Modul leitet aus den Ergebnissen der Probes den Zustand des Netzes ab und sendet bei Bedarf automatische zusätzliche Probes, um weitere Informationen für eine Lokalisierung des Fehlers zu gewinnen ([NS07]). Grundsätzlich verfolgte auch IBM schon 2002 diesen Ansatz mit eigenen Algorithmen (siehe dazu [BRM+02] und [BRM01]), die Natu und Sethi in Simulationen mit ihren vergleichen.

Nachdem Probes wie gewöhnlicher Netzverkehr behandelt werden, müssen sie explizit für das Messen bestimmter Metriken konstruiert werden. Pathchar beispielsweise ist ein Werkzeug, dass mittels Modifikationen des Time-to-live (TTL) Feldes im IP Header Messungen durchführt und so versucht Internet Verbindungscharakteristiken einzuschätzen [Dow99]. Packet Quartets nennt sich eine ähnliche Methode, die den Fokus auch auf unsichtbare, d.h. den TTL Wert nicht verringernde, Knoten legt [PV02].

Testen

Die Methoden zur Fehler Lokalisierung liefern Hypothesen über die wahren Fehlerquellen. Die Stufe des Testens versucht die Thesen zu verifizieren, indem für die Diagnose optimale Testfälle gewählt werden und so zwischen maximaler Information und minimaler Anzahl an Probes ein Kompromiss versucht wird [BF06].

3.2.2 Topologie Erkennung

Die Topologie des Internets unterliegt heftigen Änderungen und teilweise auch dem Geschäftsgeheimnis von Netzbetreibern, so dass sie im Allgemeinen als unbekannt gelten muss. Traditionell wird zur Ermittlung von Verbindungspfaden traceroute eingesetzt.

Vor allem durch den Einsatz von Routern zum Lastausgleich (Load Balancer) leidet traceroute unter einigen ernsthaften Problemen. Insbesondere Paris traceroute versucht solche Anomalien durch gezielte Kontrolle der Paket Header zu vermeiden [ACO+06] und so ein präziseres Bild der tatsächlichen Route der Pakete zeichnen zu können. Der Preis dafür ist allerdings eine sehr hohe Anzahl nötiger Probes, was einem größeren Einsatz im Internet entgegen steht.

Für den großen Einsatz zur Erkennung der Topologie ganzer Internet Service Provider wurde Rocketfuel entwickelt [SMWA04]. Durch das Einbeziehen von Routing Tabellen des Border Gateway Protocols (BGP) und mit Informationen des Domain Name Systems (DNS) können, ohne signifikanten Verlust an Präzision, die Messungen besser fokussiert und die Anzahl nötiger Probes deutlich reduziert werden.

Weitere Ansätze basieren ebenfalls auf dem Einsatz von active Probing [HPMc02] oder diskutieren explizit Algorithmen zum effektiven Zusammenführen verteilter Topologieinformationen [DFC05]. Andere nutzen zusätzlich zu Probes Heuristiken zur Erkundung der Internet Topologie [GT00].

Verfahren zur Topologie Erkennung helfen auch bei Untersuchungen zur Netzneutralität, beispielsweise bei der Lokation von Messpunkten oder bei der fundierten Erhebung manipulierter Pfade. Besonders bei der Lokalisierung von Manipulationen zur Ermittlung des verantwortlichen autonomen Systems sind präzise Verfahren zur Routenbestimmung entscheidend und Paris Traceroute und Rocketfuel interessante Werkzeuge.

3.2.3 Traffic Engineering

„Internet Traffic Engineering ist definiert als der Aspekt der Internet Entwicklung, der sich mit der Frage der Leistungsbewertung und Performance Optimierung operativer IP Netze beschäftigt“ (übersetzt aus [ACE+02] Kapitel 1.1). Zu unterscheiden gilt inter-domain und intra-domain Traffic Engineering (TE). Inter-domain bezieht sich auf das Routing von Verkehr zwischen autonomen Systemen, während intra-domain TE innerhalb eines autonomen Systems den Verkehr kontrolliert. Beide können signifikanten Einfluss auf die Verbindungsqualität haben und stehen daher auch im Fokus von Netzneutralität Betrachtungen. Aus Netzbetreiber Sicht behindert das Einhalten der Netzneutralität die Möglichkeiten des TE, da Verkehr grundsätzlich nicht differenziert behandelt werden darf und damit beispielsweise unter Umständen sinnvolle Priorisierung von verzögerungskritischen Diensten ausgeschlossen wird.

Der Verkehrsaustausch zwischen autonomen Systemen wird derzeit fast ausschließlich über das BGP abgewickelt und durch sogenannte Peering und Transit Abkommen vertraglich ge-

regelt. Daher spielen hier überwiegend ökonomische Aspekte eine Rolle bei der Wegewahl. Zur Entlastung des eigenen Netzes ist die häufigste Strategie, den Verkehr am nächstmöglichen Ausgangs Punkt abzugeben (vgl. Hot Potato Routing) [ACE+02]. Solange dies für jede Art des Verkehrs gleich erfolgt, ist dies nicht als Verletzung der Netzneutralität zu werten, da sie nicht eine optimale Wegewahl fordert.

Wird jedoch innerhalb eines AS eigener Verkehr bevorzugt, diskriminiert das AS externen Verkehr auf Basis der Herkunft. Im Intra-domain TE liegt die Motivation mehr in der optimalen Auslastung der vorhandenen Ressourcen und dem Einhalten von QoS Versprechen. Eingesetzte Techniken sind Overlay Netze (z.B. über QRON [LM04] oder mit DaVinci [HZSL+08]), Integrated Services (IntServ [Wro97]) zur Priorisierung von IP-Paketen und Differentiated Services (DiffServ [NBBB98]) zur Klassifizierung von IP-Paketen, sowie zu QoS Zwecken in Verbindung mit Multiprotocol Label Switching (MPLS) [AAC+03].

Nachdem auch das eingesetzte TE von den AS geheim gehalten wird, gibt es bereits eine Reihe von Werkzeugen zum Testen von Verbindungen. Einige davon werden im folgenden Kapitel 3.3 beschrieben.

3.3 Netzneutralität Werkzeuge

Besonders interessant im Hinblick auf diese Arbeit sind bereits existierende Werkzeuge für Netz Messungen. Einige adressieren bereits das Thema Netzneutralität (z.B. Netalyzr und NANO), andere dienen ursprünglich anderen Zwecken, bieten aber interessante Vorgehensweisen.

3.3.1 Netalyzr

Das International Computer Science Institute (ICSI) versucht über den öffentlich zugänglichen Service Netalyzr Daten über die Konnektivität möglichst vieler Internet Nutzer zu sammeln. Netalyzr ist als kleines browserfähiges Java Applet von jedem Internet Nutzer ausführbar und dient dabei zwei Zwecken. Zum einen erfährt der Nutzer detaillierte Informationen über seine Internetverbindung, zum anderen verschaffen die zahlreichen gespeicherten Testergebnisse einen sehr guten Überblick über die weltweit vorherrschenden und zeitlich variierenden Zustände von Internetverbindungen [KWNP10].

Durchgeführt werden Tests auf der Vermittlungsschicht zur Adressierung, IP Fragmentierung, Maximum Transmittion Unit (MTU), Verzögerung, Bandbreite, Pufferung und IPv6 Fähigkeit. Auf der Anwendungsschicht werden ausgewählte Protokolle wie HTTP, FTP, POP3, SMB, SSH und so weiter auf Verfügbarkeit geprüft, sowie ausführlich das DNS Verhalten untersucht. Zusätzlich wird versucht noch eventuelle HTTP Proxies und Caches zu identifizieren und über optionale Nutzer Rückmeldungen Informationen über deren lokale Gegebenheiten (z.B. drahtlose oder Kabel gebundene Internetverbindung) erhoben. Ein Beispiel für ein Untersuchungsergebnis ist auf der Projekt Webseite unter `http://netalyzr.icsi.berkeley.edu/restore/id=example-session` einzusehen.

Die Architektur des Systems und den Ablauf eines Tests zeigt Abbildung 3.2. Nachdem der Nutzer die Website besucht hat (Punkt 1) wird die Session beim Start des Tests (Punkt 2) vom Front-End zu einem zufällig gewähltem Back-End Netzknoten umgeleitet. Der Browser lädt und startet das Applet (Punkt 3), welches anschließend Testverbindungen zu verschiedenen Netalyzr Servern im Back-End und im Fall von DNS Tests unter Umständen auch im Front-End ausführt (Punkt 4). Die Resultate und der Original-Netzverkehr werden für

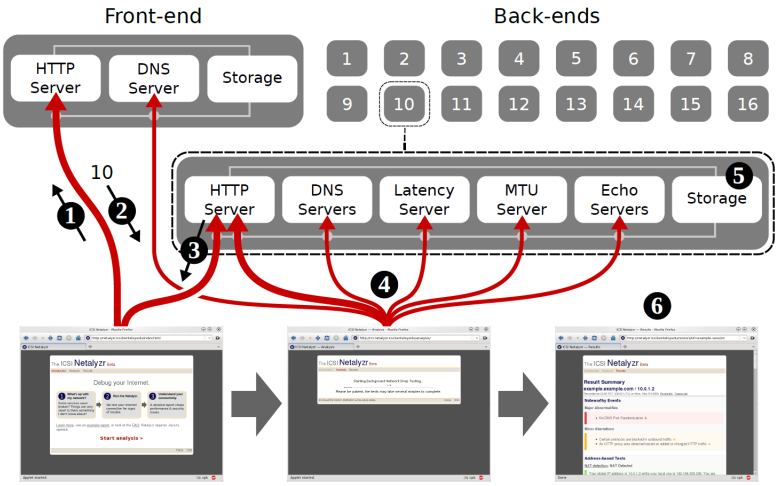


Abbildung 3.2: Architektur von Netalyzr aus [KWNP10] (Kapitel 2)

spätere Analysen (Punkt 5) gespeichert und der Nutzer bekommt eine Zusammenfassung seiner Testergebnisse angezeigt (Punkt 6).

Der Ansatz umgeht das Problem vieler Probestationen, testet allerdings immer die geschilderten fixen Varianten in Richtung der Netalyzr Server und erlaubt keine Tests zwischen zwei beliebigen Endpunkten.

3.3.2 RIPE Atlas Projekt

Das Réseaux IP Européens Network Coordination Centre (RIPE NCC), die Regionale Internet Registry (RIR) für Europa, den Nahen Osten und Zentralasien, ist dabei, mit dem Atlas Projekt das größte Netzwerk für Internet Messungen aufzubauen ([NCC]). Dafür werden sehr kleine, speziell für diesen Zweck entwickelte Geräte (in diesem Umfeld werden diese Geräte selbst als Probes bezeichnet) weltweit an Freiwillige verteilt, die sie nach dem Plug and Play Prinzip in ihre Netzinfrastruktur integrieren. Vorteilhaft an diesem Verfahren ist der isolierte Betrieb frei von Einflüssen durch andere Anwendungen und die Eignung für einen dauerhaften Betrieb (24/7) dank minimalen Stromverbrauchs [Kar10]. Die begrenzte Hardwareleistung allerdings schränkt die möglichen Tests stark ein, welche derzeit nach den FAQs nur RTT und Betriebszeit Messungen umfassen.

Für Messungen zur Netzneutralität müssten, sofern die Hardware dies zu lässt, die Probes noch deutlich in ihren Fähigkeiten ausgebaut werden.

3.3.3 NetPolice

Stark fokussiert auf die Messung unterschiedlicher Verkehrsbehandlungen in Backbone ISPs ist NetPolice. Von Endsystemen aus versucht es ausschließlich über Messungen von Paketverlusten Differenzen, verursacht entweder aufgrund des Dateninhalts oder aufgrund von Routing Verfahren, zu erkennen [ZMZ09]. Mit Unterschieden von bis zu 5% in der Verlustrate, häufig gekoppelt an Type-of-Service Werte, ließen sich einige ISPs der Verkehrsdifferenzierung überführen. Für ausführliche Neutralitätsuntersuchung ist NetPolice allerdings zu eingeschränkt und nicht erweiterbar.

3.3.4 NANO

Das Georgia Institute of Technology in Atlanta, USA, entwickelte mit NANO (Network Access Neutrality Observation) ein System zur Erkennung von Diskriminierungen von Seiten der ISPs gegen spezielle Klassen von Anwendungen, Nutzern oder Zieladressen [TMFA09]. Mit anderen Worten ein System zur Erkennung von Verletzungen der Netzneutralität wie in dieser Arbeit. Es nutzt einen passiven Ansatz, bei dem über das Internet verteilte NANO Clients Leistungsdaten sammeln und an einen zentralen Server übermitteln. Zur Unterscheidung von anderen Formen des Verlustes an Servicequalität werden kausale Beziehungen zwischen einem ISP und der beobachteten Leistung unter Berücksichtigung von Störfaktoren hergestellt. Damit wird dann aus Performance Vergleichen verschiedener ISPs auf Missbrauch geschlossen.

Der Ansatz setzt ebenfalls, wie Netalyzr in Abschnitt 3.3.1, auf freiwillige Testteilnehmer, die ihren Verkehr untersuchen lassen und ermöglicht generelle Einschätzungen des Verhaltens von ISPs. Er ist hingegen nicht für spezifische Untersuchungen, beispielsweise bestimmter Verbindungen über mehrere ISPs hinweg und auf ausgewählte Verkehrsarten hin, geeignet.

3.3.5 Glasnost

Das Glasnost Projekt des Max Planck Institutes für Software Systeme versucht mehr Transparenz im Internet zu erreichen. Ähnlich wie beim Netalyzr (vgl. Kapitel 3.3.1) werden über Browser basierte Messungen entweder speziell die Behandlung von BitTorrent Verkehr untersucht oder Breitband Netze für private Endkunden beispielsweise über Bandbreite oder Verzögerung charakterisiert[GDH+08].

3.3.6 Zusammenfassende Betrachtung

Jedes dieser Werkzeuge analysiert nur ausgewählte Aspekte des Internets. Entweder wird nur bestimmter Verkehr untersucht (Glasnost), nur ein Messwert betrachtet (NetPolice) oder allein mit einem zentralen Messpunkt getestet (Netalyzr). NANO ist zwar in der Lage den gesamten Verkehr zu betrachten, versucht aber über Vergleiche des allgemeinen Verhaltens von ISPs nur die Existenz von Manipulationen zu klären. Damit lässt sich lediglich die erste Frage aus Abschnitt 2.1.2 beantworten (Wird überhaupt manipuliert?). Für spezifischere Betrachtungen ist es jedoch nicht geeignet. RIPE hingegen zeigt vor allem, wie die Hardware für viele Messstationen im Internet erfolgreich verteilt werden kann. Die Software erlaubt indes noch keine für Neutralitätsuntersuchungen sinnvolle Tests. Zu einer hinreichenden Klärung der Fragestellungen aus Abschnitt 2.1.2, wie von dem hier entwickelten Detektor gefordert, ist daher keines dieser Werkzeuge in der Lage.

Nach dieser ausführlichen Betrachtung bereits verfolgter Ansätze, lässt sich nun ein geeignetes Konzept für den Detektor erstellen. Der Ansatz des active Probings aus der Fehler Lokalisierung wird als Grundprinzip für die Messungen verfolgt und in einer erweiterbaren Architektur eingesetzt. Zur Vermeidung von Einschränkungen, unter denen die hier vorgestellten Werkzeuge leiden, wird die Testspezifikation über eine wohldefinierte Eingabesprache von deren Umsetzung getrennt. Während damit alle Arten von Tests spezifizierbar sind, wird die Implementierung schrittweise über weitere Arbeiten ausgebaut.

4 Konzeption und Entwurf

Die in Kapitel 2.2 definierten Anforderungen bilden neben den topologischen Eigenschaften des Internets die Grundlage der folgenden konzeptionellen Entscheidungen für den Entwurf eines Detektors zur Erkennung von Verletzungen der Netzneutralität. Dazu wird eine geeignete Architektur für den Detektor entworfen, sowie erforderliche Komponenten anhand eines Anwendungsbeispiels aus dem Szenario identifiziert.

Die Basisarchitektur bildet ein Server Agenten System, das mit vielen geografisch weit verteilten Agenten zentral über einen Server steuerbar ist und das Gesamtsystem des Detektors in zwei Hauptkomponenten teilt. Der Server bildet die Interaktionsschnittstelle mit dem Benutzer, der durch die eigens dazu konzipierte Sprache wohldefinierte Eingaben tätigt, und liefert Ergebnisse der Untersuchungen. Für die Konzeption der Eingabesprache dient eine formale Metasprache, anhand der die Eingabedaten dann, wie bei der prototypischen Umsetzung (Kapitel 5) beschrieben, in ein geeignetes internes Format konvertiert werden können.

Die Agenten wiederum werden von dem Server nach Bedarf ausgewählt und je nach gewünschten Tests spezifisch angewiesen aktiv Verkehr zu erzeugen, zu protokollieren und Messergebnisse zurückzuliefern. Weiterhin wird in der Architektur ein Kommunikationsprotokoll zur Interaktion zwischen Server und Agenten, eine Timer Funktionalität für die Möglichkeit Tests zeitlich zu steuern sowie das Einbinden verschiedener Werkzeuge für Messungen in Netzen spezifiziert.

4.1 Architektur

Um eine geeignete Architektur für einen Detektor zu finden, wird zuerst eine flexible Möglichkeit gesucht, wie nicht neutrales Verhalten im Internet untersucht werden kann. Dabei sind die Charakteristiken des Internets ausschlaggebend. Anschließend werden die benötigten Teilkomponenten identifiziert und beschrieben, Entwurfsentscheidungen für die Architektur getroffen und das Kommunikationsprotokoll entwickelt.

4.1.1 Ausgangssituation

Das Internet ist, kurz gesagt, ein „Netz von Netzen“, welche auch als autonome Systeme (AS) bezeichnet werden [Tan02]. Ein AS gehört dabei einem Netzbetreiber, welcher nicht notwendigerweise auch ein Netzanbieter sein muss, sondern beispielsweise wie im Fall Amadeus auch ein großes Unternehmen sein kann.

Jedes AS unterliegt, wie der Name schon sagt, eigenständiger Verwaltung und Kontrolle, was zur Folge hat, dass im Normalfall keine genauen Informationen über technische Ausstattung und Topologie erhältlich sind. Die einzige Möglichkeit ein fremdes AS auf Netzneutralitätsverstößen hin zu testen, ist demnach von benachbarten mit Messpunkten ausgestatteten autonomen Systemen aus. Für einen generischen Detektor ergibt sich daraus die Notwendigkeit, beliebig viele unabhängige Messpunkte in möglichst vielen AS integrieren zu können.

Als nächstes wird ein in diesem Umfeld für Untersuchungen der Netzneutralität geeignetes Testverfahren gewählt.

4.1.2 Prinzipielles Messverfahren

Messpunkte im Allgemeinen können passiv operativen Verkehr beobachten (passive Monitoring) oder aktiv Verkehr generieren (Active Probing, vgl. Kapitel 3.2.1). Während Monitoring im laufenden Betrieb für konkret auftretende Probleme speziell an den beobachteten Orten Unterstützung leistet, ist active Probing unabhängig von existierenden produktiv Verkehr und erlaubt Ende zu Ende Tests [Cot01]. Jede gewünschte Verkehrsart muss allerdings beim aktiven Messen extra erzeugt werden und verbraucht entsprechend zusätzliche Bandbreite. Ausserdem können Netzbetreiber künstliche Nachrichten unter Umständen entdecken und mit gezielten Gegenmaßnahmen die Messungen kompromittieren [TMFA09].

Für das Aufspüren von Netzneutralitätsverstößen mit all seinen mannigfaltigen Manipulationsmöglichkeiten ist der Ansatz des aktiven Testens geeigneter, da flexibel jede Art von Verkehr zu jeder beliebigen Zeit erzeugt werden kann. So sind spezifische Untersuchungen bestimmter Verkehrstypen möglich und es kann beispielsweise leicht untersucht werden, ob gleicher Verkehr über unterschiedliche Ports verschieden behandelt wird oder ob ein Protokoll deutlich langsamer Daten transferiert als ein anderes.

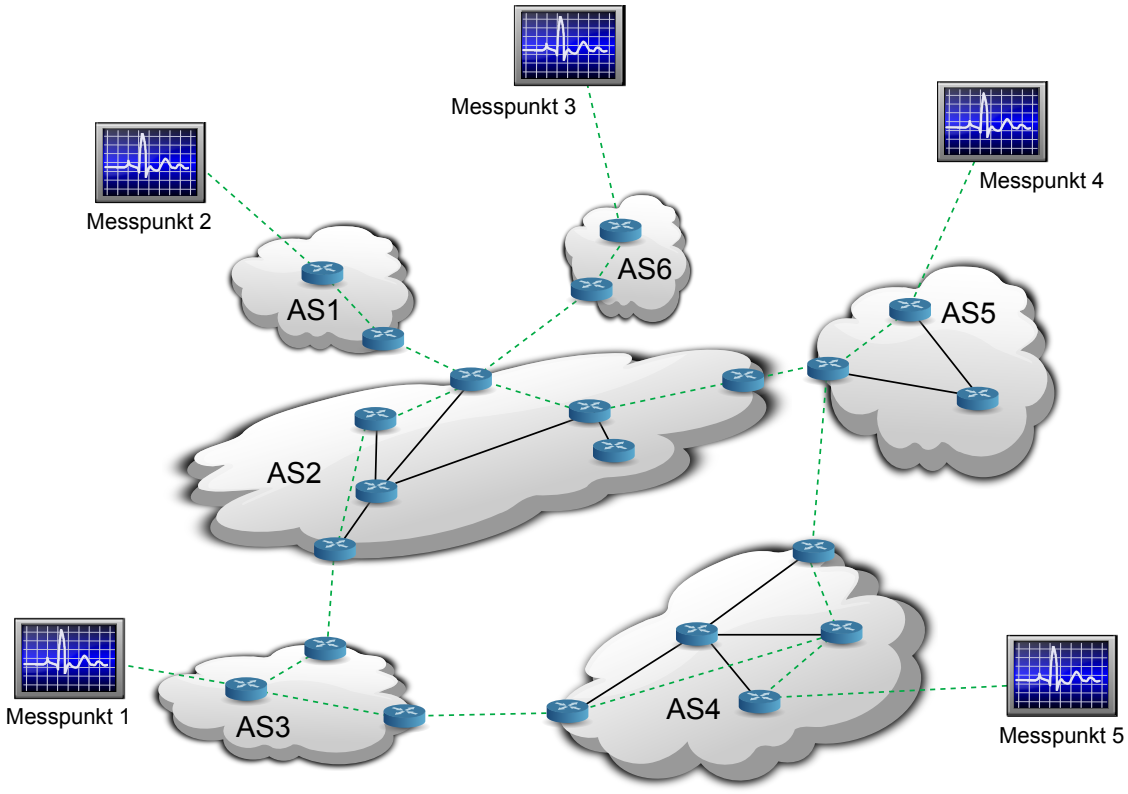


Abbildung 4.1: Autonome Systeme im Internet mit aktiven Messpunkten

Die Messpunkte des hier entwickelten Detektors müssen also über das bloße Messen hinaus in der Lage sein, selbstständig verschiedenste Arten von Verkehr zu erzeugen und zu

verarbeiten.

Exemplarisch zeigt Abbildung 4.1 sechs autonome Systeme mit fünf aktiven Messpunkten im Internet. Tests zwischen den Messpunkten sollen über die grün gestrichelten Pfade geroutet werden und führen dabei durch jedes der sechs AS. Wird beispielsweise ein Verstoß zwischen Messpunkt 4 und 5 festgestellt, können dafür fünf Router in zwei AS verantwortlich sein. Zwischen Messpunkt 2 und 5 sind es schon, je nach Route elf oder neun Router in vier AS. Es ist zu erwarten, dass generell die Aussagekraft von Tests gegen Verstöße der Netzneutralität mit der Anzahl der verfügbaren Messpunkte steigt, weshalb eine hohe Anzahl von Messpunkten von der Architektur zu unterstützen ist.

4.1.3 Server und Agenten

Eine hohe Anzahl von Messpunkten komfortabel bedienen zu können, zählt mit zu den wichtigsten Eigenschaften eines guten Detektors. Um dies zu ermöglichen wird der Detektor in einen zentralen Server zur Bedienung und Auswertung sowie in Clients, sogenannten Agenten, die nach den Anweisungen des Servers Tests durchführen, aufgeteilt.

Der Server ermöglicht es einer Person von einem beliebigen Arbeitsplatz aus viele, weit verteilte Agenten komfortabel zu bedienen und unterstützt damit u.a. Anforderung 8. Agenten können in der Folge kompakt und damit auch Ressourcen schonend (nach Anforderung 9) konzipiert werden, da sie auf direkte Interaktion mit einem Benutzer verzichten können und in der Folge einige Komponenten zentral platziert werden können. Dadurch werden nicht nur die Agenten entlastet, sondern auch z.B. das Problem der Synchronisierung eingeschränkt und komplexere Auswertungen über die Ergebnisse aller beteiligten Agenten möglich. Der Server kommuniziert mit seinen Agenten über ein Overlay Netz, dessen Verbindungen ebenfalls durch das Internet führen und damit zumindest teilweise gleiche Pfade aufweisen können wie die Testpfade, über welchen die Agenten ihre Tests abwickeln.

Der Verwaltung der Agenten erfolgt implizit aus der Testspezifikation, die in Abschnitt 4.2 ausführlich besprochen wird und Informationen zu den jeweiligen Agenten enthält. Die Agenten selbst kennen ihren Server nicht und können theoretisch auch von mehreren Servern benutzt werden. Eine genaue Beschreibung der Kommunikation zwischen dem Server und den Agenten erfolgt durch das Protokoll in Abschnitt 4.1.7.

Im Folgenden hilft ein Anwendungsbeispiel bei der Ermittlung von notwendigen Komponenten des Detektors. Es beschreibt ein aktives Vorgehen zur Bestimmung eines Kommunikationspfades zwischen zwei Messpunkten und hilft, sie im Rahmen der Server Agenten Architektur sinnvoll einzubinden.

4.1.4 Anwendungsbeispiel

Da die Topologie des Internets als unbekannt angesehen werden muss, ist es erforderlich den Pfad zwischen zwei Messpunkten extra zu ermittelt. Gängigstes Werkzeug hierfür ist traceroute, das über manipulierte Time-to-live (TTL) Felder in ICMP Anfragen von jedem Transitsystem Time-out Antworten provoziert und darüber die Route ermittelt. Interessant im Umfeld von Netzneutralität ist die Ermittlung solcher Routen nach auffälligem Verkehrsverhalten, um die eventuell verantwortlichen Router auszumachen.

Als einfaches Beispiel kann hier ein Anstieg der Übertragungszeit zwischen zwei Messpunkten dienen. Um diese Round Trip Time (RTT) zu messen, wird üblicherweise ping eingesetzt, welches die RTT ebenfalls über ICMP Nachrichten misst.

Eine Kombination dieser beiden Werkzeuge in der Art, dass beispielsweise erst nach der Überschreitung einer gewissen RTT ein traceroute ausgeführt wird, erlaubt gezielt erst nach problematischen Verzögerungen die Route zu untersuchen beziehungsweise nach einem anfänglich erfolgten traceroute eine Änderung nachzuweisen. Der Mechanismus erlaubt also auf eine bestimmte Beobachtung zu reagieren und dann automatisiert die dazugehörige Route zu ermitteln. Diese Funktionalität ist ausserdem Voraussetzung für eine Lokalisierungsfunktion, die sinnvollerweise ebenfalls erst nach einem, auf eine Netzneutralitätsverletzung hinweisendem Ereignis ausgeführt wird.

4.1.5 Ermittlung von Komponenten

Um dem Detektor seine Aufgabe zu vermitteln, werden Tests von einem Nutzer deklarativ beschrieben. Dazu ist eine wohldefinierte Art der Eingabe gerade aufgrund der zu unterstützenden vielfältigen und komplizierten Tests erforderlich. Die Eingabe erfolgt idealerweise zentral an dem Server, der diese verarbeiten kann und worauf sein weiteres Vorgehen gründet.

Nach Anforderung 5 soll auch eine zeitliche Steuerung möglich sein, wozu eine Timer Komponente benötigt wird.

Erst zu den festgelegten Zeiten wird dann mit allen in der Eingabe spezifizierten Agenten eine Verbindung aufgebaut und alle für den Test nötigen Informationen (z.B. Art und Ziel der Messungen) über ein eigens entwickeltes Protokoll zwischen den beiden Kommunikationskomponenten übermittelt. Die Messpunkte leiten schließlich aus diesen Informationen ihre Aufgabe ab.

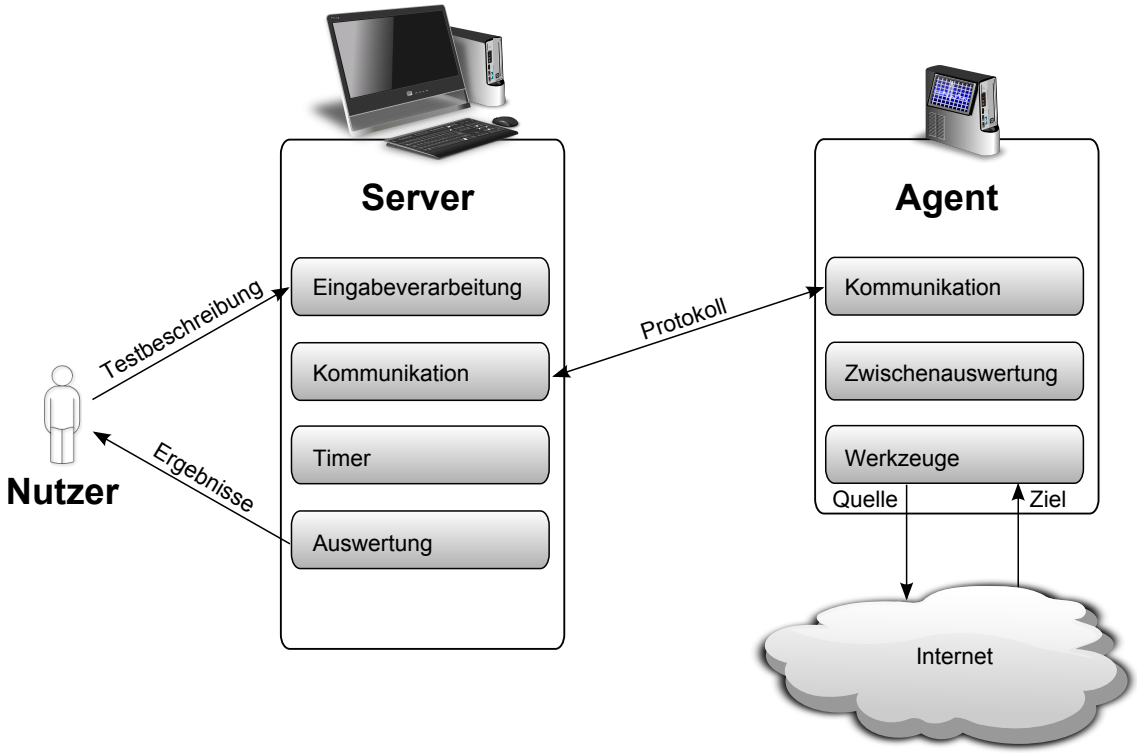


Abbildung 4.2: Konzeptionelle Komponenten des Detektors

Der Agent muss nun in der Lage sein, verschiedene Werkzeuge als Quell- oder Zieldienste für die Messungen auszuführen und deren Ausgaben gleichzeitig für eventuelle Reaktionen auszuwerten. Im Anwendungsbeispiel aus Abschnitt 4.1.4 muss z.B. als Quelldienst auf einem Agenten traceroute und ping ausgeführt werden und die RTT der ping Ausgabe gegen einen spezifizierten Wert verglichen werden, um gegebenenfalls einen weiteren Traceroute starten zu können. Der Zielmesspunkt muss die ping und traceroute Anfragen korrekt beantworten und kann z.B. über ein weiteres Werkzeug eine zusätzliche Protokollierung zur Aufdeckung eventueller Datenverluste ausführen.

Alle Ausgaben der Dienste sind zu sammeln und ebenfalls wieder über die Kommunikationskomponenten am Ende an den Server als Ergebnisse zu übertragen.

Der Server wartet bis er von allen Agenten die Ergebnisse bezogen hat und verfügt letztendlich über alle Messergebnisse. Diese können nun beliebig ausgewertet und dem Nutzer in geeigneter Form präsentiert werden. Denkbar wäre bei dem Anwendungsbeispiel eine grafische Ausgabe in Form einer Karte mit den ermittelten Wegen.

Eine Übersicht über alle konzeptionellen Komponenten der Detektor Architektur zeigt Abbildung 4.2

Nachdem nun das Konzept der Architektur steht, können die Komponenten in den folgenden Entwurfsentscheidungen konkretisiert werden.

4.1.6 Entwurfsentscheidungen

Die Komponenten wurden in Abschnitt 4.1.5 bereits identifiziert. Nachfolgend werden die einzelnen Komponenten und ihre Aufgaben beschrieben.

- **Eingabesprache**

  Eine wohldefinierte und ausdrucksmächtige Art der Eingabe erfordert der angestrebte Funktionsumfang. Dieser setzt eine exakte Spezifikation aller möglichen, auch Zeit gebundenen Tests mit beliebig vielen Agenten, verschiedenen Protokollen mit spezifischen Parametern und definierbaren Reaktionen voraus. Zu diesem Zweck und zur Trennung von Testspezifikation und Implementierung wird eine Domänen spezifische formale Sprache als Eingabesprache definiert, anhand der die Eingabe überprüft und in ein internes Format umgewandelt werden kann. Der Benutzer übergibt seine deklarativ in dieser Eingabesprache beschrieben Tests dem Server, der daraufhin die Tests wie spezifiziert durchführt. Eine weitere Interaktion mit dem Benutzer während der Tests ist nicht vorgesehen. Die ausführliche Beschreibung der Eingabesprache folgt in Kapitel 4.2.

- **Kommunikation**

  Zu jedem, an einem Test beteiligtem Agenten muss eine separate, bidirektionale Verbindung aufgebaut werden, über die nach einem Protokoll kommuniziert werden kann. Die Agenten und deren Adressen werden aus der Eingabe gewonnen und den Tests zugeordnet, um nur mit den in dem Test benötigten Agenten zu kommunizieren. Das Protokoll bestimmt im weiteren den genauen Ablauf der Kommunikation, d.h. die Art der Übermittlung der Test spezifischen Informationen (sogenannte TestInfos) am Anfang und die Übertragung der Ergebnisse am Ende. Während der eigentlichen Testdurchführung bleiben zwar die Verbindungen offen, es werden aber zur Vermeidung von Einflüssen auf die Messergebnisse keine Daten ausgetauscht.

- **Timer**

  Für die zeitliche Steuerung aus Anforderung 5 wird eine Timer Komponente benötigt, die verschiedene Arten von zeitlichen Definitionen versteht und vom Server aus die zeitliche Steuerung der Tests übernimmt.

- **Werkzeuge**

  Zur Generierung des beim aktiven Testen notwendigen Verkehrs für die Messungen werden in den Agenten, wie von Anforderung 7 gefordert, bewährte Werkzeuge eingesetzt. Dadurch lassen sich zum einen auch komplizierte Verkehrsarten wie z.B. BitTorrent Verkehr exakt nachbilden, zum anderen wird der Aufwand für die Unterstützung solcher Testfälle durch die Integration passender Werkzeuge drastisch reduziert. Ausserdem wird damit die Weiterentwicklung und Fehlerbeseitigung den Entwicklern der Werkzeuge überlassen.

  Des weiteren lassen sich Monitoring Werkzeuge zur Ermittlung weiterer Messwerte parallel zu den eigentlichen Testwerkzeugen ausführen.

- **Auswertung**

  Eine Auswertung ist an zwei Stellen möglich und zugleich notwendig. Das für eine eventuelle Reaktion zu untersuchende Messergebnis kann schon im Agenten vorliegen oder erst im Server aus den Ergebnissen mehrerer Agenten gewonnen werden. Eine abschließende Aufbereitung der Ergebnisse für den Nutzer ist hingegen auf dem Server ausreichend.

Nun lassen sich alle Komponenten in der vollständigen Architektur des Detektors einbinden. Zuvor wird allerdings noch das Kommunikationsprotokoll genauer entworfen, um die Funktionsweise der Architektur zu verdeutlichen.

4.1.7 Kommunikationsprotokoll

Die Kommunikation zwischen dem Server und seinen Agenten dient im Allgemeinen genau zwei zentralen Aufgaben:

1. **Steuerung der Agenten**

   Die Agenten benötigen Informationen zu ihren Aufgaben und ihrer Rolle, welche sie in Form einer TestInfo von dem Server zu gegebener Zeit erhalten.

   Über die Rolle erfährt der Agent, ob er Quelle oder Ziel des Tests ist. Eine dritte Rolle ist nötig für Tests, die eine zentrale Instanz benötigen, da sich der Server selbst nicht an den Messungen beteiligt.

   Anhand der Aufgabe muss der Agent die für den Test passenden Werkzeuge ermitteln und nach den enthaltenen Anweisungen ausführen.

2. **Sammeln der Ergebnisse**

   Entweder direkt durch die Ausgabe der Testwerkzeuge oder durch zusätzliche Monitoring Werkzeuge ermitteln die Agenten beliebig viele Messergebnisse und übertragen diese nach den Tests an den Server.

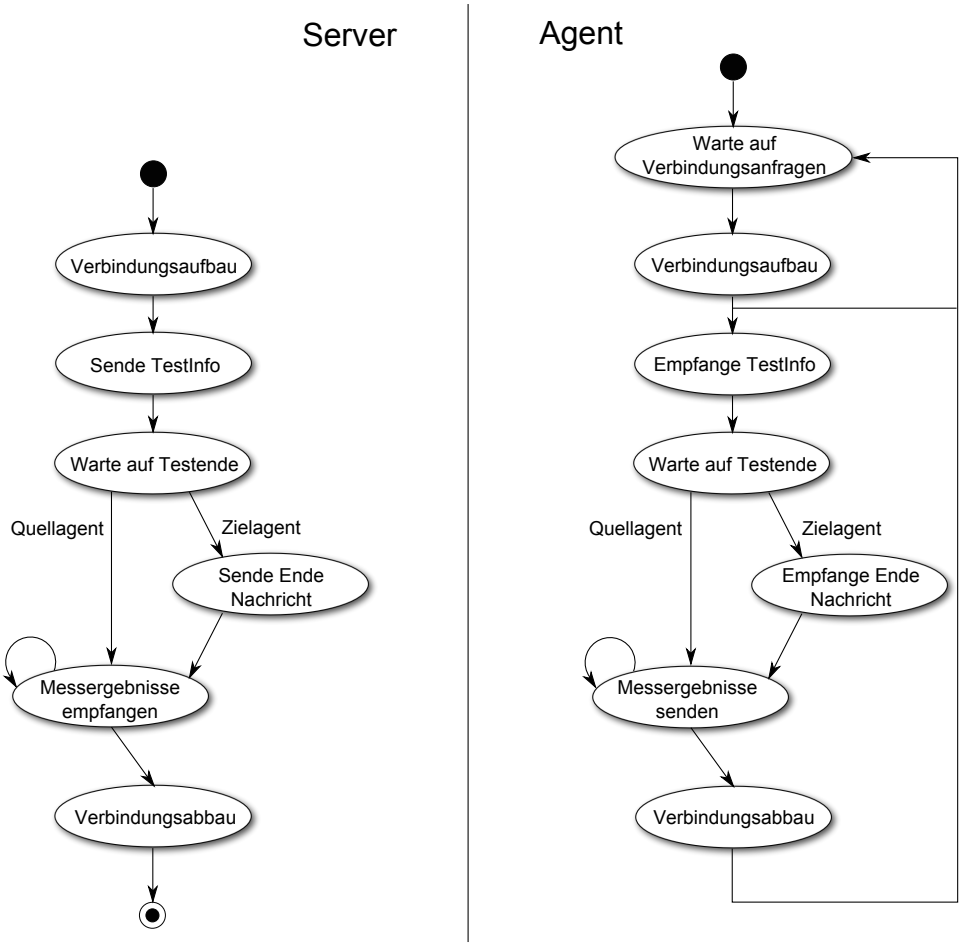


Abbildung 4.3: Protokoll zur Kommunikation zwischen Server und Agenten

Den genauen Ablauf des Protokolls zeigt Abbildung 4.3 in Form eines Zustandsdiagramms, links für den Server, rechts für einen Agenten. Der Server baut mit jedem Agenten eine Verbindung auf und sendet die entsprechende TestInfo. Der Agent befindet sich stets im Wartezustand. Auch nachdem eine Verbindung aufgebaut wurde, wartet er parallel dazu auf weitere, wodurch er beispielsweise gleichzeitig Quelle und Ziel eines Tests werden kann. Die Verbindung bleibt während der Tests bestehen, aber es erfolgt, um die Tests nicht zu beeinflussen, kein Datenaustausch bis die Tests beendet sind.

Nur wenn der verbundene Agenten als Ziel fungiert, erwartet dieser eine Ende Nachricht, um den Empfangsdienst beenden zu können. Anders als die Quellagenten, kann er nicht entscheiden, wann die Tests vollständig abgeschlossen sind. Die dritte Rolle hat für das Protokoll keine spezielle Bedeutung und wird der Rolle als Quelle gleichgesetzt.

Anschließend werden alle beim Test erstellten Dateien mit Messergebnissen zum Server übertragen. Explizit ist dieser Schritt nicht mehr von der Rolle abhängig, da auch beim Zielagenten Messungen durchführbar sein sollen. Ist die Übertragung der Dateien abgeschlossen, wird die Verbindung beendet und der Agent befindet sich wieder ausschließlich im Wartezustand.

Der Ablauf eines Tests kann im Folgenden skizziert werden, da die Architektur nun vollständig konzipiert ist.

4.1.8 Ablauf einer Messung

Die gesamte Architektur des Detektors ist in Abbildung 4.4 dargestellt. Das Bild greift die Darstellung des Internets aus Abbildung 4.1 wieder auf und platziert Agenten an die Stellen der Messpunkte. Der Server ist dabei alleinige Interaktionsstelle mit dem Benutzer, d.h. hier werden die Tests durch die Eingabe vom Benutzer spezifiziert und deren Ergebnisse gespeichert und ausgewertet.

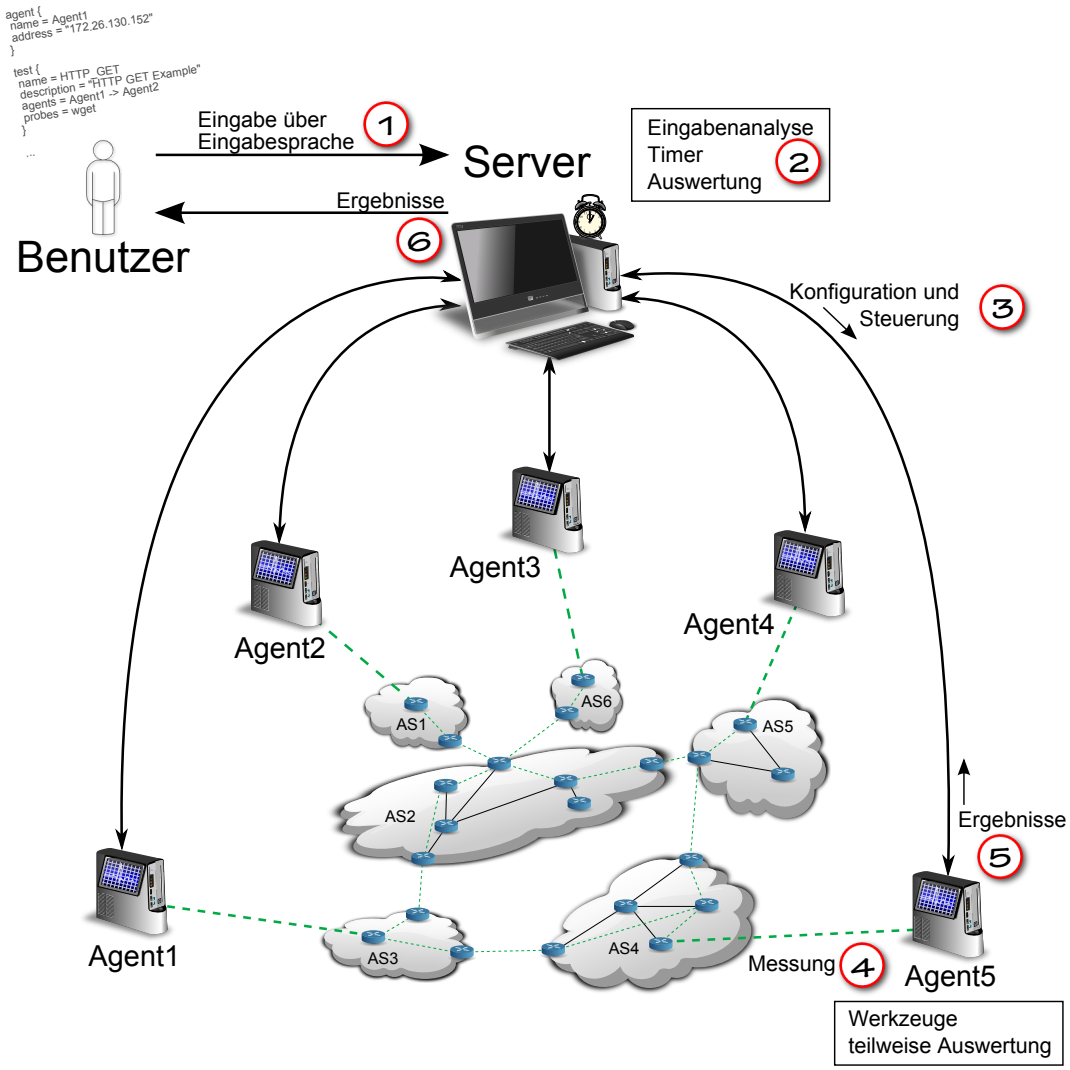


Abbildung 4.4: Architektur des Detektors

Der Ablauf des Detektors verläuft wie folgt: Anfangs spezifiziert ein Benutzer Tests deklarativ in der Eingabesprache und übergibt sie dem Server (Punkt 1). Aus der Eingabe, nach erfolgreicher Validitätsprüfung, gewinnt der Server alle zur Durchführung der Tests nötigen

Informationen (Punkt 2). Nachdem eine Timerkomponente einen korrekten, für diesen Test festgelegten Zeitpunkt meldet, konfiguriert und steuert der Server über TestInfos alle für den aktuellen Test einzusetzenden Agenten gemäß dem Protokoll und wartet auf Ergebnisse (Punkt 3).

Die Agenten wählen daraufhin anhand der empfangenen TestInfos geeignete Werkzeuge und führen mit deren Hilfe Messungen durch (Punkt 4). Bei entsprechend spezifizierten Tests können die Messungen für sofortige Reaktionen zumindest teilweise noch innerhalb der Agenten ausgewertet werden. Alle Messergebnisse werden protokolliert und im Anschluss an den Server zurückgesendet (Punkt 5).

Anhand der gesammelten Messergebnisse kann der Server dann weitere Auswertungen durchführen, gegebenenfalls Reaktionen einleiten und die Ergebnisse in ein für die Ausgabe an den Benutzer optimiertes Format bringen (Punkt 6).

Im nächsten Abschnitt wird nun die Eingabesprache zur Spezifikation der Tests konzipiert.

4.2 Formale Sprache

Die nötige Variabilität an Testmöglichkeiten nach Anforderung 1 diktiert eine wohldefinierte und ausdrucksmächtige Form der Eingabe, die über die Definition einer Eingabesprache exakt charakterisiert wird. Die bloße Steuerung über Parameter beim Programmaufruf erweist sich hier als zu eingeschränkt in ihrer Ausdruckskraft.

Die Eingabesprache selbst wird formal über eine, an der Erweiterten Backus-Naur-Form (EBNF) angelehnte, Metasprache beschrieben. Die Metasprache spezifiziert also die Elemente der Eingabesprache, in der die Eingabe erfolgen muss.

Eine valide Eingabe wird anschließend für die weitere Verwendung in ein geeignetes internes Format konvertiert. Dieses Vorgehen sorgt nicht nur für eine eindeutige Art der Eingabe, sondern auch für eine Trennung der Abhängigkeit zwischen Eingabe und Implementierung. Sowohl die Eingabesprache als auch die Implementierung lässt sich so ohne die Notwendigkeit einer Anpassung der Gegenseite modifizieren oder sogar austauschen.

Die Eingabesprache wird mit dem Ziel entwickelt, alle denkbaren Tests für eine Erkennung von Netzneutralitätsverstößen unabhängig von der zur Verfügung stehenden Implementierung spezifizieren zu können.

Sprachelemente

Prinzipiell besteht eine formale Sprache aus einer Menge von Symbolen und Formatierungsregeln, die beschreiben wie diese Symbole kombiniert werden können [Lin01]. Abschnitt 4.2.1 beschreibt die eingesetzte Metasprache im Detail.

Jede formale Sprache beginnt mit einer einzelnen Startregel, von der aus sich alle weiteren Sprachkonstrukte konstruieren lassen. Die hier beschriebene Sprache erlaubt von der Startregel aus die freie Anordnung folgender Ausgangsdefinitionen (Objekte) in beliebiger Anzahl: Kommentar, Test, Agent, Agentenset, Probe, Probeset, Period und Trigger.

Ausdrucksmächtigkeit

Wie zu Beginn dieses Abschnittes bereits beschrieben, Bedarf die Sprache einer hohen Ausdrucksmächtigkeit und weist zu diesem Zweck einige entscheidende Konzepte und Funktionen auf.

Über Referenzen werden Redundanzen vermieden, alternative Schreibweisen und syntaktische Optimierungen ermöglicht und zusätzlich mit Hilfe des 'all' und 'except' Konzepts gemäß der Mengenlehre intuitiv alle oder Teile der verfügbaren Referenzen unterstützt. Literale und Klammerungen sorgen für klare Anweisungen mit einer gewissen Selbsterklärung und Struktur. Sets gruppieren Definitionen für einfache Wiederverwendung sinnvoller Kombinationen von Agenten und Probes zu Mengen.

Die Spezifikation der Eingabesprache erfolgt durch eine Metasprache, die im Folgenden eingeführt wird.

4.2.1 Metasprache

Als Metasprache zur formalen Definition der Eingabesprache wird eine eigene, an die Erweiterte Backus-Naur-Form (EBNF) angelehnte Sprache eingesetzt, die auch von dem in Kapitel 5 gewählten Parser Generator interpretiert werden kann. Anhand von Listing 4.1 wird die Grammatik der Metasprache im Folgenden erläutert.

Listing 4.1: Beispiel der Metasprache

```
1  start        : expr+ ;
2  expr         : NAME DEF string ;
3  string       : '"'(CHAR | NUMBER | WS | SPECIALS)* '"' ;
4
5
6  NAME         : 'name' ;
7  DEF          : '=' ;
8  CHAR         : 'a'..'z' | 'A'..'Z' | '_' | '<' | '>' | '-' | '+' ;
9  NUMBER       : '0'..'9' ;
10 WS           : ' ' | '\t' | '\r' | '\n'  ;
11 SPECIALS     : '.' | '/' | ':' | ',' | '!' | '@' | '#' | '\\' | '|' | '\%' |
       '\'' | '{' | '}' | '\$' | '=' | '&' ;
```

Ein Programm in dieser Metasprache besteht aus einer Serie von Produktionsregeln und Token. Produktionsregeln werden im Gegensatz zu Token mit kleinen Anfangsbuchstaben eindeutig markiert. In diesem Beispiel sind die Produktionsregeln 'start', 'expr' und 'string' und die Token 'NAME', 'DEF', 'CHAR', 'NUMBER', 'WS' und 'SPECIAL'. Alle Symbole in einfachen Anführungszeichen sind Literale und müssen genau so in der Eingabesprache vorkommen. Aus der EBNF werden die Zeichen '?', '+', '*' und '|' mit ihren jeweiligen Bedeutungen übernommen: '?' für optionales (0 oder 1), '+' für mindestens einmaliges (1 bis x), '*' für beliebiges Vorkommen (0 bis x) und '|' für Alternativen. Klammerung weist ebenfalls die übliche Funktionalität zur Gruppierung und Reihenfolgenfestlegung auf.

Das Beispiel lässt sich wie folgt interpretieren: 'start' ist eine Liste von 'expr' Regeln, 'expr' besteht exakt aus 'NAME DEF string', wobei 'string' einer in Anführungszeichen stehenden Zeichenkette entspricht (Zeile 3). Durch Interpretation der Token und Literale zeigt Listing 4.2 einen möglichen validen Ausdruck.

Listing 4.2: Beispiel eines validen Ausdrucks gemäß der Metasprache

```
1  name = "Agent0815"
```

Für die hier entwickelte Eingabesprache sind eine Menge definierter Token als Literale nötig, die in Listing 4.3 aufgeführt sind und im weiteren Verlauf der Beschreibung der formalen Sprache zum Einsatz kommen. Diese Token dürfen nie an anderen als an den im Nachfolgenden beschriebenen Orten auftreten.

Listing 4.3: Literale der Eingabesprache

```
1  DEF                          : '=' ;
2  RIGHT                        : '->';
3  LEFT                         : '<-';
4
5  TEST                         : 'test' ;
6  TESTS                        : 'tests' ;
7  AGENT                        : 'agent' ;
8  NAME                         : 'name' ;
9  ADDRESS                      : 'address' ;
10 PORT                         : 'port' ;
11 TYPE                         : 'type' ;
12 LOCATION                     : 'location' ;
13 COMPANY                      : 'company' ;
14 STREET                       : 'street' ;
15 ZIP                          : 'zip' ;
16 CITY                         : 'city' ;
17 COUNTRY                      : 'country' ;
18 AS                           : 'AS' ;
19 CC                           : 'CC' ;
20 GRID                         : 'grid' ;
21
22 AGENTS                       : 'agents' ;
23 AGENTSET                     : 'agentset' ;
24 AGENTSETS                    : 'agentsets' ;
25 NET                          : 'net' ;
26 SOURCE                       : 'source' ;
27 DESTINATION                  : 'destination' ;
28 ALL                          : 'all' ;
29 EXCEPT                       : 'except' ;
30
31 PROBE                        : 'probe' ;
32 PROBES                       : 'probes' ;
33 PROBESET                     : 'probeset' ;
34 DESCRIPTION                  : 'description' ;
35 REPEAT                       : 'repeat' ;
36 GENERATED                    : 'generated' ;
37 RECORDED                     : 'recorded' ;
38
39 IPv4                         : 'IPv4' ;
40 IPv6                         : 'IPv6' ;
41 ICMPv4                       : 'ICMPv4' ;
42 ICMPv6                       : 'ICMPv6' ;
43 PING                         : 'PING' ;
44 TRACEROUTE                   : 'TRACEROUTE' ;
45
46 TCP                          : 'TCP' ;
47 UDP                          : 'UDP' ;
48
49 HTTP                         : 'HTTP' ;
50 GET                          : 'GET' ;
51 POST                         : 'POST' ;
52 FTP                          : 'FTP' ;
53 TORRENT                      : 'TORRENT' ;
54
55 RTP                          : 'RTP' ;
56
```

```
57 START                        : 'start' ;
58 END                          : 'end' ;
59 DURATION                     : 'duration' ;
60 INTERVAL                     : 'interval' ;
61 PERIOD                       : 'period' ;
62 NOW                          : 'now' ;
63
64 TRIGGER                      : 'trigger' ;
65 METRIC                       : 'metric' ;
66 BEHAVIOR                     : 'behavior' ;
67 VALUE                        : 'value' ;
68 TRIGGER_VALUES               : 'RTT' | 'TROUGHPUT' | 'TIME' ;
69 ACTION                       : 'action' ;
70 TRUE                         : 'true' ;
71
72 ID                           : (CHAR | NUMBER)+ ;
```

4.2.2 Test, Probe und Trigger

Um die Eingabesprache auch exakt definieren zu können, sind vorab ein paar zusätzliche terminologische Festlegungen zu treffen.

Da der Detektor durch Anforderung 3 eine Vielzahl von verschiedenen Testmöglichkeiten unterstützen soll, ist es sinnvoll unterschiedliche Tests zu unterscheiden. Ein einzelner **Test** kann mehrere **Testdurchläufe** benötigen und besteht dabei aus **Agenten**, **Probes**, **zeitlichen Bestimmungen** und **Triggern**.

Agenten repräsentieren die Messpunkte des Detektors und definieren die Endpunkte eines Tests.

Eine Probe ist ein bestimmter Typ von Netzverkehr, der zum Testen zwischen Agenten eingesetzt werden soll. Dies kann ein beliebiger Verkehrstyp sein, also z.B. sowohl eine einfache ICMP Anfrage, als auch ein Video Stream oder ein HTTP Download, der einen Aspekt der Problembeschreibung in Abschnitt 1.2 testet. Die Testmöglichkeiten werden also überwiegend durch die unterstützten Probes bestimmt.

Die zeitlichen Bedingungen sollen Anforderung 5 genügen und eine zeitliche Steuerung des Detektors gewährleisten. Sie werden entweder über einen Anfangs- und Endzeitpunkt mit definierbarem Intervall oder über einen Anfangszeitpunkt, einer Dauer und einem Intervall spezifiziert.

Trigger bezeichnen einen Mechanismus, der es ermöglicht auf bestimmte Ereignisse während eines Tests zu reagieren und damit Anforderung 6 zu erfüllen. Da Trigger nicht für jeden Test notwendig sind, werden sie als optionale Erweiterung gesehen.

Auf dieser Terminologie aufbauend wird anschließend die Eingabesprache, ausgehend von der Definition eines Tests, mit Hilfe der entwickelten und in Abschnitt 4.2.1 gezeigten Metasprache vorgestellt.

4.3 Formale Spezifikation von Messungen

Die formale Sprache spezifiziert einerseits eine Reihe von referenzierbaren Elementen, auf die Testdefinitionen direkt verweisen, erlaubt aber auch eine Mengenbildung und alternative Schreibweisen. Im Folgenden werden zuerst die Test Elemente beschrieben und anschließend Referenzen und das Mengen Konzept erläutert.

4.3.1 Test Elemente

Die logischen Bestandteile eines Tests bestehen entsprechend Abschnitt 4.2.2 aus den Elementen Agent, Probe, Trigger und Zeit.

Test

Listing 4.4 zeigt die Definition eines Tests in der formalen Metasprache.

Listing 4.4: Formale Definition eines Tests

```
1  test: TEST '{' name description agent_refs
2              (probe_refs | trigger_refs) time* '}' ;
3          name :          NAME DEF ID ;
4          description :   DESCRIPTION DEF STRING ;
5          agent_refs :    AGENTS DEF idrefs ;
6              idrefs :        ID (',' ID)* ;
7          probe_refs :    PROBES DEF idrefs ;
8          trigger_refs :  TRIGGER DEF  id '{' value action '}'
9                          (',' id '{' value action '}')* ;
10             value :         VALUE DEF (STRING | TRUE) ;
11             action :        ACTION ( ( '{'
12                             (trigger_refs | test_refs | probe_refs)+ '}' )
13                             | ( DEF ( 'delete' | 'restart' | 'stop' ) ) ) ;
14         time :          begin ((end interval) | period_ref) ;
15             begin :         START DEF ( DATETIME | NOW) ;
16             end :           END DEF DATETIME ;
17             interval :      INTERVAL DEF DURATION_TYPE ;
```

Demnach enthält ein Test einen Namen, eine Beschreibung, Referenzen auf Agenten, Referenzen auf Probes oder Trigger und optional beliebig viele Zeitaspekte. Referenzen werden anschließend in Kapitel 4.3.2 definiert, die Spezifikation der referenzierten Einheiten (Agenten, Probes, Trigger und Time) werden im Folgenden ausgeführt.

Das Token 'ID', wie in Zeile 72 in Listing 4.3 zeigt, besteht nur aus Buchstaben und Zahlen. Leerzeichen und Sonderzeichen sind nicht erlaubt. Da hier auf umschließende Anführungszeichen verzichtet wird, dürfen IDs nie einem Tokennamen entsprechen, d.h. beispielsweise 'name = test' ist nicht valide. Die Grammatik der formalen Metasprache erlaubt auch eine schöne grafische Darstellung in Form eines Syntaxdiagramms, wie Abbildung 4.5 ebenfalls für die Test Definition beweist. Insbesondere die Klammerung und die Art der erlaubten Produktionen nach den EBNF Regeln '?', '+', '*' und '|' werden anschaulicher dargestellt. Im Folgenden wird zugunsten der Diagramme auf die textbasierte Form verzichtet.

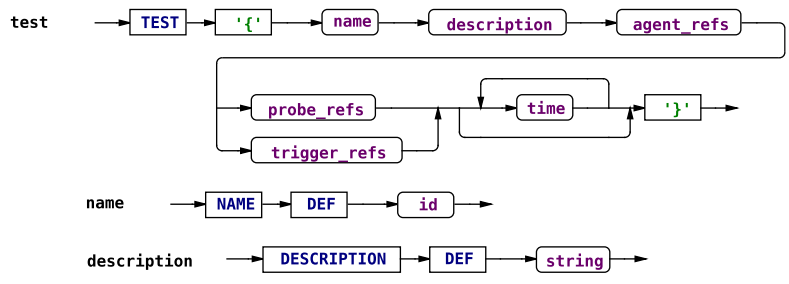


Abbildung 4.5: Syntaxdiagramm eines Tests

Während der Name in Form einer eindeutigen ID und die Beschreibung als gewöhnliche von Anführungszeichen umschlossene Zeichenkette weitgehend selbsterklärend sind, bedürfen die Referenzen und die Zeitoption noch weiteren Erklärungen. Referenzen zeigen über die eindeutigen Namen auf Definitionen von Agenten (Abbildung 4.6), Probes (Abbildung 4.7) und Trigger (Abbildung 4.8). Dies ermöglicht die einfache Wiederverwendung und Eindeutigkeit definierter Typen. Ein Beispiel einer Test Definition zeigt Listing 4.5.

Listing 4.5: Beispiel einer Test Definition

```
1 test {
2     name = BITTORRENT
3     description = "BitTorrent Example"
4     agents = Agent1 -> Agent2
5     probes = BitTorrent
6     start = 2011-06-18T12:00:00+01:00
7     end = 2011-06-18T13:00:00+01:00
8     interval = 00:00:00
9 }
```

Hier wird beginnend am 18.06.2011 um 12 Uhr mitteleuropäischer Zeit (+01:00) (Zeile 6) von Agent1 nach Agent2 die Probe BitTorrent ausgeführt. Solange 13 Uhr nicht erreicht ist, wird der Test wiederholt, da als Intervall '00:00:00' angegeben ist, was als sofortige Wiederholung interpretiert wird (Zeile 8). Damit ist keine klare Aussage getroffen wie oft der Test durchgeführt wird, allerdings ist sichergestellt, dass während dieser einen Stunde ständig ein Test läuft. Eine genaue Funktionsbeschreibung der Referenzen folgt in Abschnitt 4.3.2 und eine Beschreibung der genauen Syntax der Zeit erfolgt bei der Definition der time Produktionsregel zu Abbildung 4.9.

Agenten

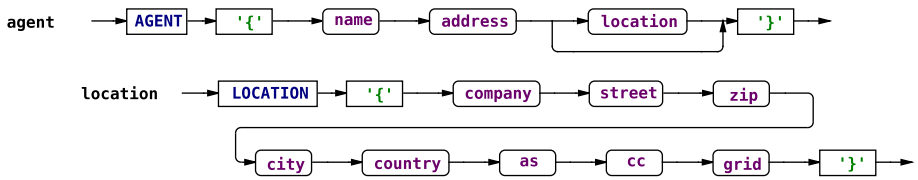


Abbildung 4.6: Syntaxdiagramm eines Agenten

Die Agenten Definition zeigt Abbildung 4.6 und eine mögliche Ausprägung zeigt Listing 4.6. Dabei werden Agenten primär durch ihren eindeutigen Namen und ihre Adresse definiert. Die Adresse kann dabei eine IPv4 oder IPv6 Adresse sowie ein Fully Qualified Domain Name sein (Zeile 4). Optional erlauben die Felder unter Location das Speichern des Firmennames, der Straße, der Postleitzahl, der Stadt, des Landes, der autonomen System Nummer ('AS'), der Rechenzentrumsnummer ('CC') und der Grid Koordinaten des jeweiligen Agenten (Zeilen 7 bis 14). Die jeweiligen Datentypen, Zeichenketten oder Zahlen, sind selbsterklärend und daher nicht extra aufgeführt. Der Location Teil kann weggelassen werden, folglich würden nur noch die Attribute 'name' und 'address' gesetzt sein. Zu Beachten ist die Unterscheidung von Gross- und Kleinschreibung.

Listing 4.6: Beispiel einer Agent Definition mit optionalem Location Teil

```
agent
{
    name = Agent1
    address = "172.26.130.152"
    location
    {
        company = "Amadeus Data Processing GmbH"
        street = "Berghamer Str. 6"
        zip = 85435
        city = "Erding"
        country = "DE"
        AS = 12888
        CC = 1
        grid = "AA34"
    }
}
```

Probes

Eine Probe, wie in Abbildung 4.7 dargestellt, wird wieder über einen referenzierbaren Namen und einer Beschreibung definiert. Zusätzlich erlaubt der optionale 'repeat' Eintrag beliebig häufige Wiederholungen. Der letzte Eintrag klassifiziert die Probe nach 'recorded' oder 'generated'.

Im Fall von 'recorded' wird anschließend über 'file =' eine Datei referenziert, die beispielsweise von Wireshark gespeicherten Datenverkehr enthält. Dies ermöglicht komplexe Szenarios oder produktiv Verkehr aufzuzeichnen und in wiederholbaren Tests zu untersuchen.

Im zweiten Fall mit 'generated' wird eine genaue Definition von künstlich zu erzeugendem Verkehr nach dem OSI-Referenzmodell [Tan02] ermöglicht. Demnach werden sukzessive die im Internet relevanten Schichten 3, 4 und 7 spezifiziert. Die fehlenden Schichten 1 und 2 des OSI-Modells sind infrastrukturabhängig und 5 und 6 sind im Internet nicht implementiert. Mit Blick auf die anstehende Umstellung auf IPv6 gibt es im Internet in der Vermittlungsschicht derzeit vier in diesem Zusammenhang relevante Protokolle: IPv4, IPv6, ICMPv4 und ICMPv6. Im Fall ICMP bieten sich hier die bekannten Funktionalitäten des Pings oder des Traceroutes für Testzwecke an.

IP ist Basis für die Protokolle TCP und UDP der Transportschicht und benötigen jeweils einen zu definierenden Port.

Auf TCP und UDP setzen schließlich eine ganze Reihe für den Detektor interessante Protokolle auf, welche hier nur durch HTTP, FTP, BitTorrent ('TORRENT') und RTP beispielhaft repräsentiert sind. Bezüglich weiterer Protokolle kann hier die formale Sprache spezifisch erweitert werden. Da es Anwendungsschicht Protokolle gibt, die sowohl auf TCP als auch auf UDP aufsetzen können (z.B. das RealTime Streaming Protocol (RTSP) [SRL98]) kann dies durch die Spezifikation des Transportschicht Protokolls ebenfalls ausgedrückt werden.

Ein Beispiel einer Probe Definition wird in Listing 4.7 gegeben. Schön ist hier die stufenweise Zusammenstellung der Schichten zu erkennen. Die TCP Schicht benötigt die Angabe eines Ports, die HTTP Schicht die Angabe eines HTTP Befehls, der in diesem Fall wiederum eine Pfadangabe benötigt. Das zweite Attribut "-v" ermöglicht optional die direkte Übergabe von Parametern an das in der Implementierung verwendete Werkzeug.

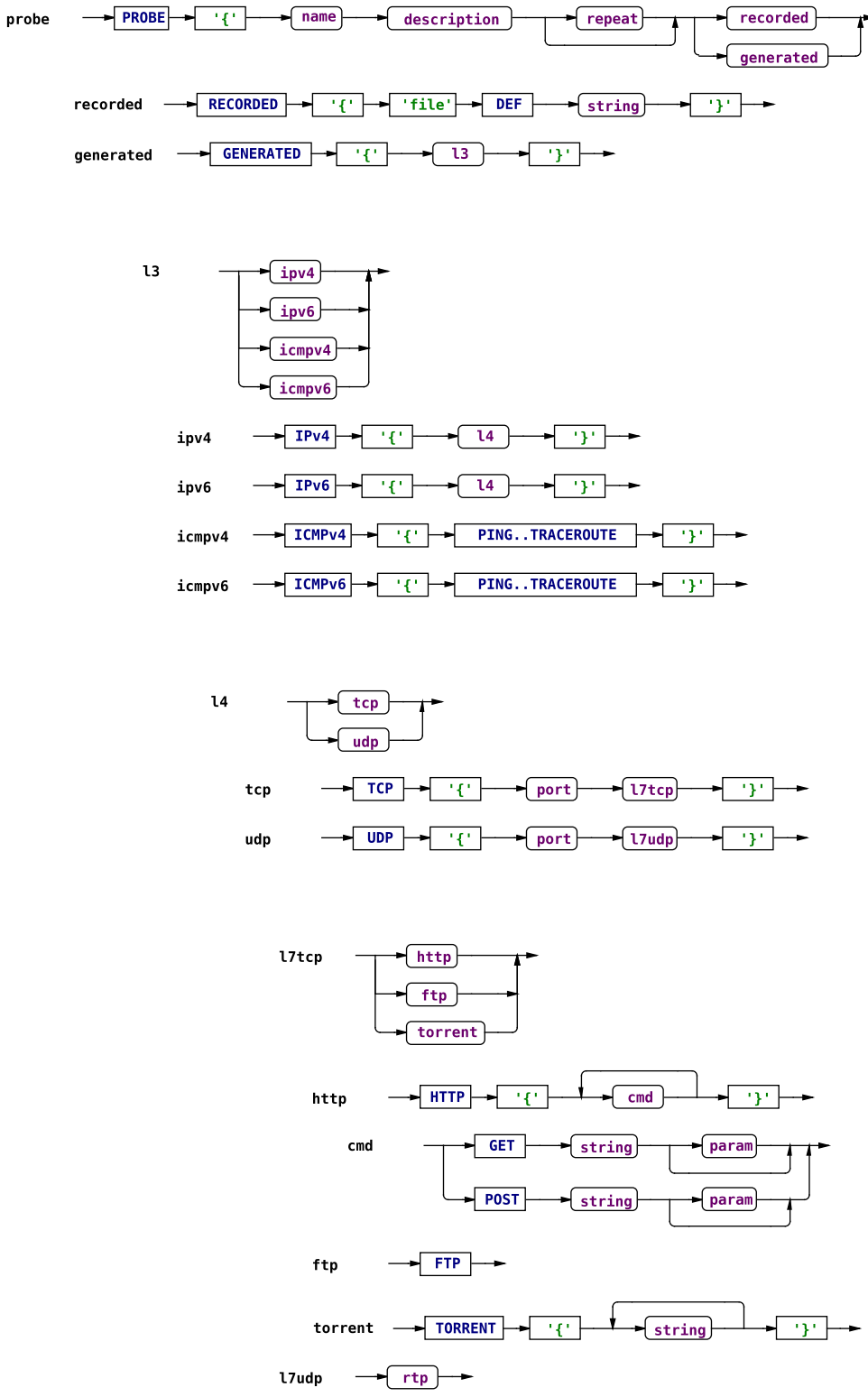


Abbildung 4.7: Syntaxdiagramm einer Probe

Listing 4.7: Beispiel einer Probe Definition für das HTTP Protokoll

```
probe {
    name = HTTP_GET
    description = "Download der Datei test per HTTP GET"
    repeat = 1
    generated {
        IPv4 {
            TCP {
                port = 8080
                HTTP {
                    GET "/data/test" "-v"
                }
            }
        }
    }
}
```

Trigger

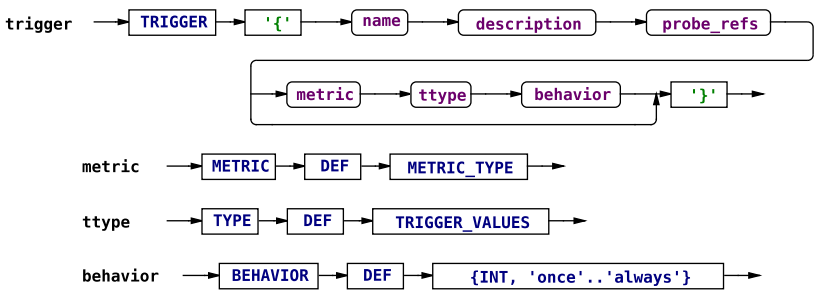


Abbildung 4.8: Syntaxdiagramm eines Triggers

Auch ein Trigger nach Abbildung 4.8 erhält einen eindeutigen Namen und eine Beschreibung, ausserdem Referenzen auf Probes und optional eine Metrik, einen Trigger-Type (’type’ mit dem Token ’ttype’) und ein bestimmtes Verhalten (’behavior’). Die Syntax von Referenzen wird unter 4.3.2 erläutert, die restlichen drei optionalen Felder bestimmen, wodurch der Trigger ausgelöst wird. Gemäß der Metrik, die nur aus den Vergleichsoperatoren ’==’, ’>’, ’<’, ’<=’, ’>=’ besteht, wird ein Vergleich definiert, der das Triggerkriterium bildet. Der erste Operand repräsentiert das Messergebnis des durch den Trigger Typ definierten Wertes und der zweite stellt einen Schwellenwert dar, der innerhalb der in Abbildung 4.12 definierten Trigger Referenz angegeben wird. Über das Verhalten (’behavior’) lässt sich ausserdem festlegen, wie oft dieser Trigger auslösen soll.

Listing 4.8: Beispiel einer Trigger Definition

```
trigger {
    name = max_RTT
    description = "Triggers when RTT gets to high"
    probes = ping
    metric = >
    type = RTT
    behavior = 3
}
```

Listing 4.8 zeigt ein Beispiel für einen Trigger, der maximal drei mal auslöst. Er referenziert die Probe mit dem Namen 'ping' und beobachtet den durch die Probe ermittelten Wert für die RTT. Das Metrik Attribut legt als Vergleichsoperator ein '>' fest und zusammen mit dem Schwellenwert, der erst bei der Referenzierung auf diesen Trigger angegeben wird und hier noch unbekannt ist, entspricht dies dem Ausdruck 'RTT > Schwellenwert'.

Zeit

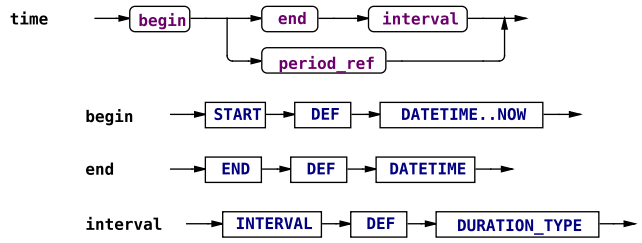


Abbildung 4.9: Syntaxdiagramm der Zeiteinstellungen

Ein Test wird mit den optionalen Zeit Feldern vervollständigt. Größtmögliche Variabilität bietet die Syntax nach Abbildung 4.9 durch einen obligatorischen Anfangszeitpunkt ('begin') in Verbindung mit entweder einem Endzeitpunkt mit Intervallspezifikation oder einer Referenz auf eine definierte Periode (Abbildung 4.10). Während 'begin' zur Einfachheit auch 'now' erlaubt, müssen ansonsten Zeitangaben über 'DATETIME' exakt nach der W3C Note vom 15. September 1997 definiert werden [WW97]. Dessen Syntax für die hier verwendete Zeit mit vollständigem Datum und Stunden, Minuten und Sekunden beschreibt Listing 4.9.

Listing 4.9: Syntax für Datum und Zeit nach W3C Note [WW97]

```
1  YYYY-MM-DDThh:mm:ssTZD (z.B. 1997-07-16T19:20:30+01:00)
2
3  mit
4
5  YYYY = vierstelliges Jahr
6  MM   = zweistelliger Monat (z.B. 01=Januar)
7  DD   = zweistelliger Monat (01 bis 31)
8  hh   = zweistellige Stunde (00 bis 23)
9  mm   = zweistellige Minute (00 bis 59)
10 ss   = zweistellige Sekunde (00 bis 59)
11 TZD  = Zeitzone (Z oder +hh:mm oder -hh:mm)
```

Das Intervall wird einfach über eine Zeitspanne nach dem gleichen Muster wie die Zeit in 'DATETIME' definiert. Optional sind auch Tage mit 'd:' als Präfix vor der Zeitangabe erlaubt (z.B. 1d:01:00:00 bedeutet alle 25 Stunden). Auch eine leere Eingabe ist erlaubt, was zu sofortiger Wiederholung des Tests führt.

Eine Periode (Abbildung 4.10) wird definiert durch eine Gesamtdauer ('duration') und einem Intervall in der eben beschriebenen Syntax. Daneben gibt es wiederum einen Namen für die Referenzierung in 'time' und eine Beschreibungsmöglichkeit.

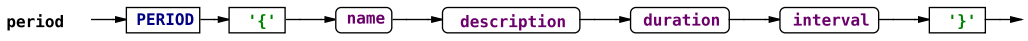


Abbildung 4.10: Syntaxdiagramm einer Period

Ein Beispiel für eine Definition einer Period gibt Listing 4.10. Dies definiert eine insgesamte Testdauer von einer Woche (7 Tage), mit stündlichen Testdurchläufen.

Listing 4.10: Beispiel einer Period Definition

```
period {
    name = WeekEveryHour
    description = "Test every Hour for one Week"
    duration = 7d:00:00:00
    interval = 01:00:00
}
```

4.3.2 Referenzen

Zur Vermeidung von Redundanzen werden Referenzen nach dem Prinzip der ID und IDREF des XML Standards herangezogen [BPSM+08]. Während der eigentliche Referenzmechanismus nicht in EBNF beschrieben werden kann, werden hier nur die dafür notwendigen Konstrukte vorgestellt. Die Realisierung der Referenzen erfolgt erst in der prototypischen Umsetzung (Kapitel 5). Ausserdem wird das Konzept durch einige Erweiterungen für eine benutzerfreundlichere Eingabe optimiert.

Agentenreferenzen

Dies betrifft vor allem die Referenzierung von Agenten nach Abbildung 4.11.

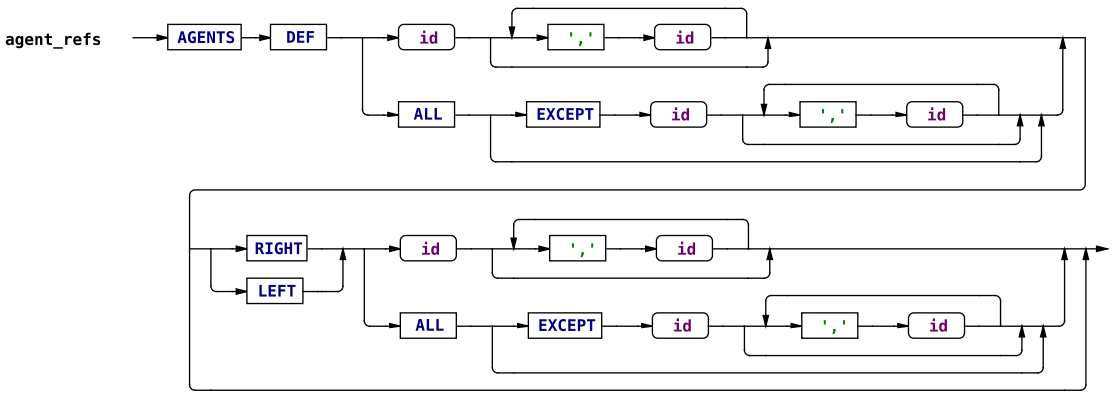


Abbildung 4.11: Syntaxdiagramm der Referenzen auf Agenten

Einfachste Art der Referenzierung besteht aus der Komma separierten Auflistung von Agenten Namen, wonach jeder Agent mit allen anderen den Test durchführt. Da Daten im Internet generell in zwei Richtungen gesendet werden können, impliziert diese Schreibweise einen Duplex Test, d.h. der Test wird einmal für jede Richtung ausgeführt. Alternativ besteht die Möglichkeit über den Begriff 'all' alle innerhalb dieser Eingabe spezifizierten Tests zu

verwenden. Zusätzlich lassen sich mit 'all except' Agenten von dem Test wieder ausschließen. Für gerichtete Tests in nur eine bestimmte Richtung kann zwischen zwei Komma separierten Listen '->' oder '<-' stehen. Tests werden dann je nach Pfeilrichtung von Agenten der einen Liste zu Agenten der anderen Liste ausgeführt. Beide Listen unterstützen ebenfalls 'all' und 'all except'.

Listing 4.11 zeigt ein paar Beispiele zu möglichen Referenzierungen.

Listing 4.11: Beispiele von Referenzierungen auf Agenten

```
1  agents = Agent1,Agent2,Agent3
2
3  agents = all
4
5  agents = Agent1 -> Agent2,Agent3
6
7  agents = Agent1 -> all except Agent1
8
9  agents = Agent1 <- Agent2,Agent3
10
11 agents = all except Agent2,Agent3 <- all except Agent1
```

Listing 4.11: Beispiele von Referenzierungen auf Agenten

Das erste Beispiel in Zeile 1 führt einen Test von allen drei Agenten zu allen drei Agenten aus, inklusive der unter Umständen ungewollten lokalen Selbsttests wie z.B. von Agent1 zu Agent1. Angenommen es sind nur die drei Agenten Agent1, Agent2 und Agent3 definiert, dann entspricht das zweite Beispiel (Zeile 3) dem ersten.

Das dritte Beispiel ((Zeile 5) unterscheidet nun über die verkürzte Schreibweise mit '->' Quellagenten (Agent1) und Zielagenten (Agent2 und Agent3). Im vierten Beispiel wird das all except Konstrukt eingesetzt, was unter der selben Annahme wie oben wiederum eine alternative Schreibweise zum dritten Beispiel darstellt (Zeile 7).

Beispiel fünf (Zeile 9) zeigt die Möglichkeit über '<-' Ziel- vor Quellagenten zu spezifizieren und das letzte Beispiel (Zeile 11) wiederum unter bekannter Annahme eine Alternative Schreibweise mit beidseitiger Verwendung von 'all except'. Wichtig zu bemerken ist jedoch, dass sich alle Spezifikationen über 'all' durch veränderte Agentendefinitionen ebenfalls verändern können. Während bei einer zusätzlichen Definition eines Agent4 die referenzierten Agenten aus Beispiel eins, drei und fünf identisch bleiben und Agent4 hier nicht zum Einsatz kommt, wird er bei den übrigen Beispielen verwendet.

Da dies jedoch immer noch nicht alle denkbar sinnvollen Möglichkeiten der Agentenzuordnung bietet, können die Agenten noch Netzen zugeordnet werden. Dies ist aber nur durch die Definition eines in Kapitel 4.3.3 beschriebenen Sets möglich.

Triggerreferenzen

Die Referenzen auf Trigger, wie in Abbildung 4.12 dargestellt, sind insofern besonders, da sie jeder Referenz zusätzliche Informationen zuordnen. Unter 'value' wird ein Wert angegeben, der mit der im Trigger spezifizierten Metrik verglichen wird und dessen Einheit sich aus dem Trigger Typ ableitet. Ist der Trigger Typ beispielsweise 'RTT' wird der 'value' Wert als Millisekunden interpretiert. Soll der Trigger immer auslösen ist 'value' mit 'true' anzugeben.

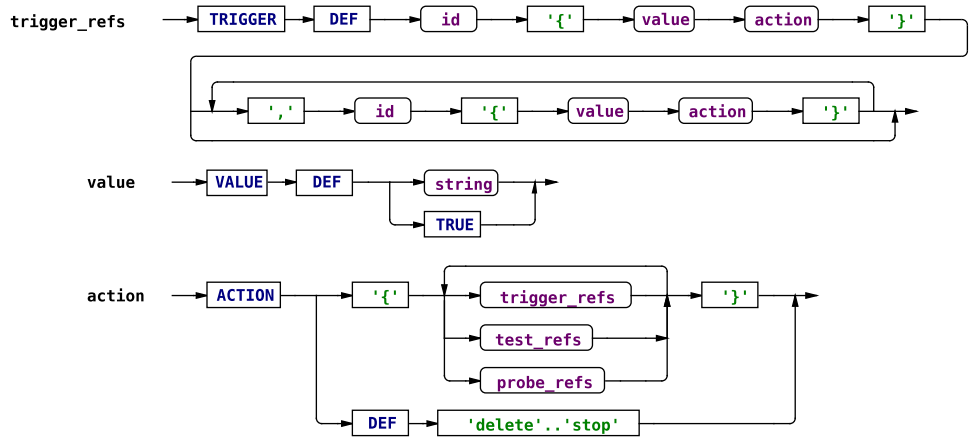


Abbildung 4.12: Syntaxdiagramm der Referenzen auf Trigger

Mit 'action' wird schließlich die Reaktion auf das Auslösen des Triggers entweder über die Anweisungen 'delete', 'restart' und 'stop' definiert oder es werden weitere Probes, Trigger und komplette zusätzliche Tests referenziert.

Ein Beispiel mit einer Referenz auf den in Listing 4.8 definierten Trigger 'max_RTT' zeigt Listing 4.12.

Listing 4.12: Beispiel einer Referenzierung auf Trigger

```
1 trigger = max_RTT {
2     value = "0.020"
3     action {
4         probes = traceroute
5     }
6 }
```

Zusammengenommen ergibt sich nun als Triggerkriterium 'RTT $>$ 0.020', d.h. der Trigger wird ausgelöst, wenn die gemessene RTT den Wert '0.020' übersteigt. Als Reaktion wird dann die Probe 'traceroute' ausgeführt. Per Komma können weitere Trigger definiert werden, ebenso auch innerhalb des action Attributes für aufeinander folgende Trigger.

Weitere Referenzen

Die übrigen Referenzen funktionieren nach dem einfachen XML Konzept, inklusive der Unterscheidung von REF für Referenzen auf genau ein Objekt, wie im Fall der Period Referenzen (Abbildung 4.13), und REFS für Referenzen auf beliebig viele Objekte, wie bei Referenzen auf Tests (Abbildung 4.14) und Probes (Abbildung 4.15).

Abbildung 4.13: Syntaxdiagramm der Referenz auf Period

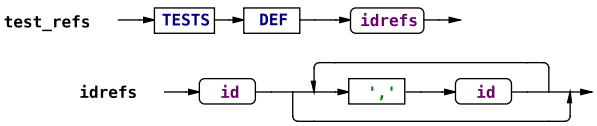


Abbildung 4.14: Syntaxdiagramm der Referenz auf Tests

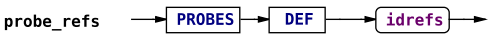


Abbildung 4.15: Syntaxdiagramm der Referenzen auf Probes und ProbeSets

Als Beispiel genügt hier Listing 4.13.

Listing 4.13: Beispiel einer Referenzierung auf Probes oder Probesets

```
1    probes = traceroute,ping,HTTPset
```

Nachdem Referenzen keinerlei Namenskonventionen besitzen und daher keine Typen Unterscheidung möglich ist, müssen alle Namen global eindeutig sein. Eine Referenzierung eines falschen Typs kann von der formalen Sprache nicht unterbunden werden. Dafür ist es so möglich, über die Referenzen für Probes nach Abbildung 4.15 gleichzeitig auch die in Abschnitt 4.3.3 eingeführten ProbeSets zu referenzieren. In dem Beispiel ist sprachlich nicht unterscheidbar, welche der Referenzen ein Set und welche eine Probe referenzieren. Unterstützung kann hier nur eine intelligente Benennung bieten.

4.3.3 Set Konzept

Zur weiteren Vereinfachung und im Fall der Agenten auch zum weiteren Ausbau der Ausdrucksmächtigkeit dient das sogenannte Set Konzept, welches im allgemeinen Gruppierungen ermöglicht. Solche Mengen gibt es nur für Agenten und Probes.

Probeset

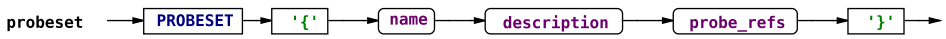


Abbildung 4.16: Syntaxdiagramm eines Probesets

Abbildung 4.16 zeigt die einfache Syntax einer Probeset Definition, die nur aus einem referenzierbaren Namen, einer Beschreibung und Referenzen auf Probes besteht. Dies erlaubt häufig zusammen eingesetzte Probes zu verbinden und über nur eine Referenz auszuwählen.

Listing 4.14: Beispiel einer Probeset Definition

```
1 probeset {
2     name = HTTPvsHTTPS
3     description = "HTTP GET und HTTPS GET"
4     probes = HTTP_GET,HTTPS_GET
5 }
```

Listing 4.14 zeigt eine solche Set Definition. Hier werden die beiden Probes mit den Namen 'HTTP_GET' und 'HTTPS_GET' in einem Set zusammengefasst. In diesem Fall wird ein Vergleich zwischen unverschlüsseltem und verschlüsseltem HTTP GET Downloads ermöglicht. Hierbei reicht nun eine Referenz in der Testdefinition auf 'HTTPvsHTTPS' aus.

Agentenset

Agentensets bieten noch eine zusätzliche Gruppierungsfunktion nach Netzen (Abbildung 4.17). Ausserdem wird hier eine andere Schreibweise zur Unterscheidung von Quell- und Zielagenten über 'source' und 'destination' ermöglicht. Auch hier ist 'all' und 'all except' erlaubt, allerdings besteht dann auch hier das Agentenset nicht mehr garantiert aus immer denselben Agenten.

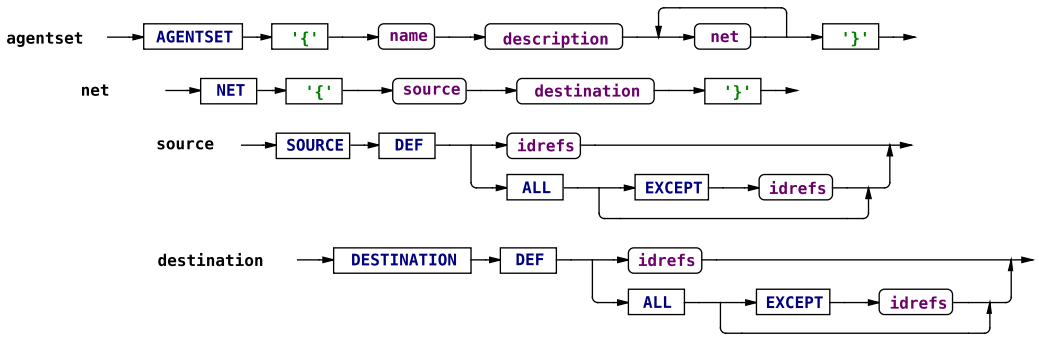


Abbildung 4.17: Syntaxdiagramm eines Agentensets

Einige Beispiele zur besseren Erklärung der Ausdruckskraft zeigt Listing 4.15. Die ersten sechs Beispiele entsprechen den Beispielen aus Listing 4.11 und zeigen damit eine alternative Schreibweise. Deutlich fällt auf, wie stark die Schreibweise mit '->' und '<-' aus Abschnitt 4.3.2 die Definition bei gleicher Ausdruckskraft verkürzt.

Das siebte Beispiel in Zeile 56 zeigt die hier eingeführte Erweiterung mit mehreren 'Net' Konstrukten, die in der verkürzten Schreibweise nicht ausgedrückt werden kann. Dazu lassen sich zur einfachen Durchführung eines Tests beliebige 'Net' Gruppen aus Quell- und Zielagenten erstellen.

Listing 4.15: Beispiele von Agentenset Definitionen

```
1  agentset {
2      name = Example1
3      description = "Setdefinition for Example One"
4      net {
5          source = Agent1,Agent2,Agent3
6          destination = Agent1,Agent2,Agent3
7      }
8  }
9
10 agentset {
11     name = Example2
12     description = "Setdefinition for Example Two"
13     net {
14         source = all
15         destination = all
16     }
17 }
18
19 agentset {
20     name = Example3
21     description = "Setdefinition for Example Three"
22     net {
23         source = Agent1
24         destination = Agent2,Agent3
25     }
26 }
27
28 agentset {
29     name = Example4
30     description = "Setdefinition for Example Four"
31     net {
32         source = Agent1
33         destination = all except Agent1
34     }
35 }
36
37 agentset {
38     name = Example5
39     description = "Setdefinition for Example Five"
40     net {
41         source = Agent2,Agent3
42         destination = Agent1
43     }
44 }
45
46
```

```
47 agentset {
48     name = Example6
49     description = "Setdefinition for Example Six"
50     net {
51         source = all except Agent1
52         destination = all except Agent2,Agent3
53     }
54 }
55
56 agentset {
57     name = Example7
58     description = "Setdefinition for Example Six"
59     net {
60         source = Agent1
61         destination = all except Agent1
62     }
63     net {
64         source = Agent2
65         destination = Agent3
66     }
67     net {
68         source = Agent2
69         destination = Agent1,Agent3
70     }
71 }
```

Listing 4.15: Beispiele von Agentenset Definitionen

4.3.4 Beispiele gültiger Testeingaben

Ein einfaches vollständiges Beispiel für eine valide Eingabe eines Tests mit einer Probe ist in Listing 4.16 zu sehen.

Listing 4.16: Beispiel für Eingabesprache: HTTPdownload Test

```
1  agent {
2      name = Agent1
3      address = "172.26.130.152"
4  }
5
6  agent {
7      name = Agent2
8      address = "172.26.130.168"
9  }
10
11 probe {
12     name = HTTP_GET
13     description = "HTTP default test"
14     repeat = 1
15     generated {
16         IPv4 {
17             TCP {
18                 port = 8080
19                 HTTP {
20                     GET "/data/test" "-v"
21                 }
22             }
```

```
23          }
24      }
25  }
26
27  test {
28      name = HTTPdownload
29      description = "HTTP GET Example"
30      agents = Agent1 -> Agent2
31      probes = HTTP_GET
32  }
```

Hier wird der Download Anfrage der Datei '/data/test' von Agent1 nach Agent2 mittels HTTP GET beschrieben. Die Agenten werden in den Zeilen 1 bis 9 minimal spezifiziert. Anschließend wird die Probe definiert: Einmalig soll eine generierte Testnachricht über das HTTP Protokoll, dass auf IPv4 und TCP aufsetzt, mit den in Zeile 20 spezifizierten Attributen gesendet werden. In den Attributen ist der HTTP Befehl GET gesetzt, zusammen mit einer Pfadangabe auf die Datei 'test' sowie mit "-v" die optionale Möglichkeit genutzt, dem zum Einsatz kommendem Werkzeug extra Parameter zu übergeben.

Schließlich folgt ab Zeile 27 die eigentliche Definition des Tests. Die vorher definierten Bestandteile des Tests werden von hier aus referenziert. So wird Agent1 zur Quelle und Agent2 zum Ziel der Probe. Durch die Anfrage wird der Download der Datei von Agent2 nach Agent1 ausgelöst und durchgeführt. Bemerkenswert ist auch, dass in diesem Beispiel keine Zeitangaben gegeben werden, was als Wunsch zu sofortiger einmaliger Durchführung des Tests zu interpretieren ist.

Listing 4.17 ist dem Anwendungsbeispiel aus Abschnitt 4.1.4 nachempfunden und zeigt die Spezifikation eines Triggermechanismus.

Listing 4.17: Beispiel für Eingabesprache: pathChange Test

```
1  trigger {
2      name = oneTrace
3      description = "do one traceroute"
4      probes = traceroute
5  }
6
7  trigger {
8      name = max_RTT
9      description = "Triggers when RTT gets to high"
10     probes = ping
11     metric = >
12     type = RTT
13     behavior = 3
14 }
15
16
17 agent {
18     name = Agent1
19     address = "172.26.130.152"
20 }
21
22 agent {
23     name = Agent2
24     address = "172.26.130.168"
25 }
26
```

```
27 probe {
28     name = traceroute
29     description = "Traceroute"
30     repeat = 1
31     generated {
32         ICMPv4 {
33             TRACEROUTE
34         }
35     }
36 }
37
38 probe {
39     name = ping
40     description = "Round Trip Time"
41     repeat = 50
42     generated {
43         ICMPv4 {
44             PING
45         }
46     }
47 }
48
49
50 test {
51     name = pathChange
52     description = "traceroute after ping TTL change"
53     agents = Agent1 -> Agent2
54     trigger = oneTrace {
55         value = true
56         action {
57             trigger = max_RTT {
58                 value = "100"
59                 action {
60                     probes = traceroute
61                 }
62             }
63         }
64     }
65     start = 2011-06-18T12:00:00+01:00
66     period = testonce
67 }
68
69 period {
70     name = testonce
71     description = "One Test only"
72     duration = 00:00:00
73     interval = 00:00:00
74 }
```

Die beiden Agenten werden wie schon im vorangegangenen Beispiel (Listing 4.16) definiert. Nun werden aber zwei Probes definiert. Die erste spezifiziert ab Zeile 27 einen einmaligen traceroute auf Basis von ICMP Version 4 und die zweite veranlasst 50 pings ebenfalls über ICMP Version 4.

Die Trigger werden gleich zu Beginn spezifiziert und erfordern nähere Betrachtung. Der erste Trigger mit Namen 'oneTrace' wird ab Zeile 1 minimal definiert, d.h. ohne Angaben zu Metrik, Typ und Verhalten, und verweist auf die Probe traceroute. Der zweite ab Zeile

7 referenziert auf die Probe 'ping' und spezifiziert als Metrik '>', als Typ 'RTT' und als Verhalten '3'. Entscheidend für das spätere Testverhalten ist nun deren Referenzierung in der Test Spezifikation, die in den Zeilen 50 bis 67 erfolgt. Die Triggerreferenzen sind ineinander geschachtelt definiert, d.h. der erste Trigger löst einen weiteren aus.

Zuerst wird der Trigger 'oneTrace' zusammen mit einer 'value' gleich 'true' und einer weiteren Triggerreferenz in 'action' referenziert (Zeile 54). Der spezielle Wert in 'value' veranlasst ein garantiertes, nicht auf einer Auswertung basiertes Auslösen, nachdem die Probe 'traceroute' durchgeführt wurde, die in der bereits beschriebenen Trigger Definition in Zeile 1 und folgende referenziert wird. Ausgelöst wird der Trigger 'max_RTT' aus Zeile 7 mit einer 'value' von '100' und einer 'action' mit einer erneuten Referenz auf die Probe 'traceroute'. Zusammen mit der Definition des Triggers bedeutet dies, dass der Trigger bei einer durch einen der 50 pings ermittelten RTT größer als 100 ms (die Einheit ergibt sich aus der durch ping gelieferten Einheit) auslöst und einen weiteren traceroute startet. Durch das im Trigger spezifizierte Verhalten von '3' wird dieser getriggerte traceroute allerdings nur maximal drei mal durchgeführt.

4.4 Beispiel der Durchführung eines Tests

Der Ablauf des Tests aus Listing 4.17 wird ausführlich in Abbildung 4.18 in Form eines Sequenzdiagrams dargestellt. Noch einmal kurz zusammengefasst, wird die Verbindung nach einem vorangegangenen traceroute über ping Nachrichten ständig auf ihre RTT untersucht und falls diese 100 ms überschreitet bestimmt einer von maximal drei weiteren traceroutes den aktuellen Pfad erneut.

Ausserdem wird in dem Test, wie schon im ersten Beispiel (Listing 4.16) Agent1 zur Quelle und Agent2 zum Ziel des Tests erklärt. Zusätzlich wird hier noch ein Startzeitpunkt in Verbindung mit einer ab Zeile 69 definierten 'period' gewählt. Konkret bedeutet dies den Start des Tests am 18.06.2011 um 13 Uhr mitteleuropäischer Zeit (MEZ; + 1 Stunde) mit einer einmaligen Durchführung.

Abbildung 4.18 zeigt zu dem Beispiel ein Sequenzdiagram zum konkreten konzeptionellen Ablauf. Zu Gunsten der Übersichtlichkeit wird hier allerdings ein 'repeat' Wert von 5 statt 50 für die Probe 'ping' angesetzt.

Durch Ausdrücke der in Abschnitt 4.2 spezifizierten Eingabesprache teilt der Nutzer dem Server in Punkt 1 den gewünschten Test mit. Der Server parsed die Eingabe, ermittelt daraus die jeweils spezifizierten und am Test beteiligten Agenten und generiert zur Konfiguration TestInfos, die er zu dem definierten Zeitpunkt (18.06.2011, 13 Uhr MEZ) an die Agenten verteilt (Punkt 2).

In Punkt 3 entscheiden die Agenten auf Grundlage der TestInfos, welche Werkzeuge zum Einsatz kommen. Da bei ICMP Nachrichten auf der Zielseite kein extra Werkzeug benötigt wird und in diesem Fall auch keine weiteren Beobachtungen angestellt werden, kann Agent2 diesen Test bereits als beendet erklären und ein leeres Ergebnis an den Server zurückschicken (Punkt 4).

Agent1 hingegen beginnt nach dem ersten Traceroute (Punkt 5) ping Anfragen an Agent2 zu richten und die gemessenen RTT mit 100 ms zu vergleichen. Bereits beim zweiten mal wird dieser Wert überschritten und der Trigger mit neuerlichem traceroute ausgelöst (Punkt 6). Danach werden wieder weiter pings durchgeführt, bis erneut ein Wert über 100 ms für die RTT gemessen wird. Dies ist in diesem Beispiel sofort wieder der Fall. Intern muss der

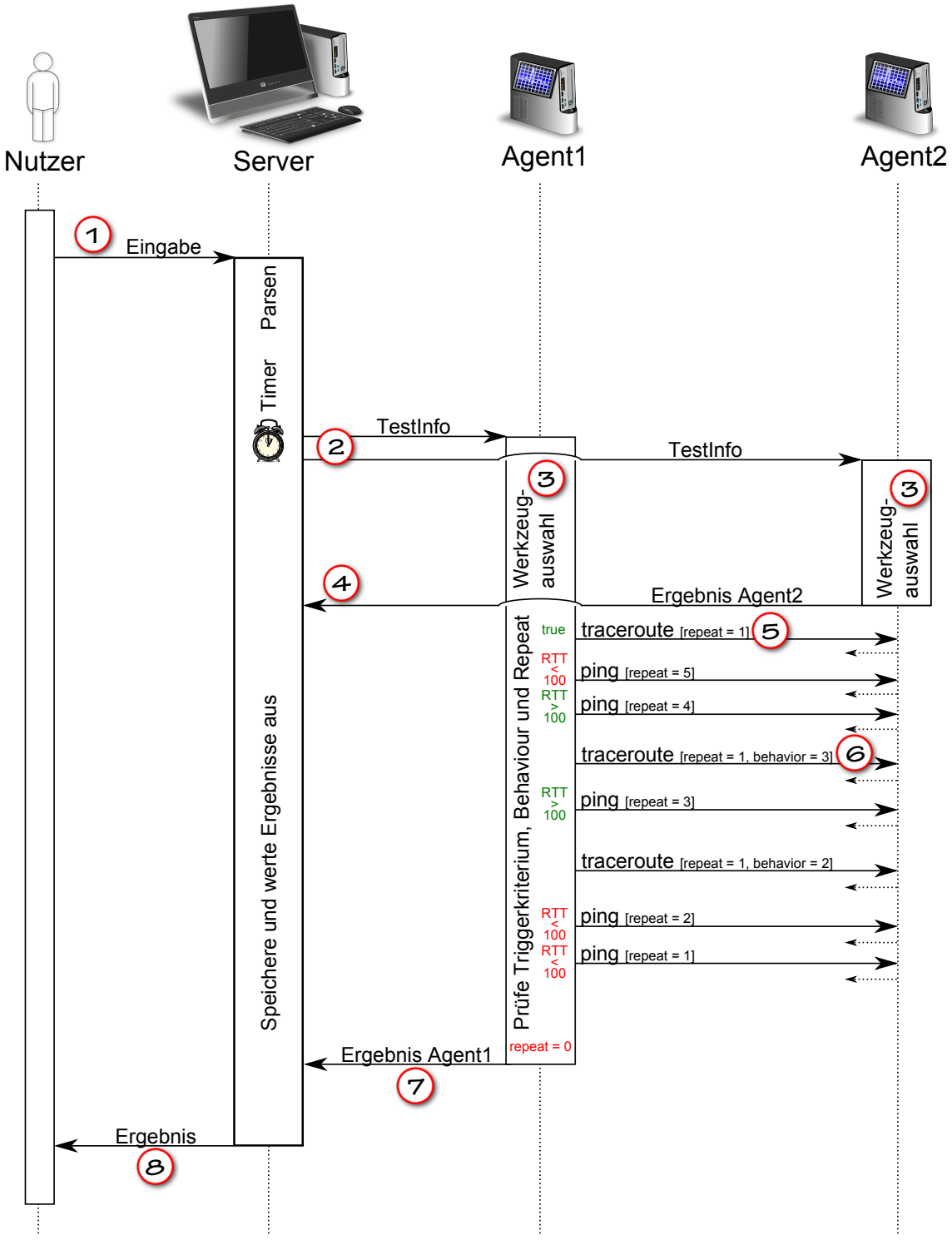


Abbildung 4.18: Sequenzdiagram des Anwendungsbeispiels

Agent dabei stets die aktuellen Werte für repeat und behavior verwalten, um zum korrekten Zeitpunkt den Test zu beenden, beziehungsweise trotz Auslösen des Triggers traceroute nicht erneut zu starten.

Nach dem fünften Ping ist der Test schließlich beendet und die protokollierten Ausgaben aller Probes (fünf Pings und drei Traceroutes) werden an den Server übertragen (Punkt 7). Theoretisch hätte noch ein Traceroute durchgeführt werden können, da am Ende der Wert von 'behavior' noch bei 2 liegt. Bei Punkt 8 speichert der Server nun alle Ergebnisse und kann einen Vergleich der protokollierten Pfade durchführen und dem Nutzer ausgeben.

Nachdem nun ein Konzept für den Detektor entwickelt wurde, wird im nächsten Kapitel der Prototyp umgesetzt. Die Formale Sprache wird zur Erzeugung einer generischen internen Datenstruktur eingesetzt und Komponenten der Architektur mit ihren Aktivitäten beschrieben und realisiert.

5 Prototypische Umsetzung

In diesem Kapitel wird die prototypische Implementierung vorgestellt. Wichtiger Teil der Umsetzung ist der praktische Einsatz der formalen Sprache mit der Verwaltung der Eingabedaten, die Komponenten des Servers und der Agenten mit ihren Aktivitäten und deren Kommunikation über das Protokoll.

Ausserdem werden, basierend auf der technischen Ausstattung in Verbindung mit den Anforderungen aus Kapitel 2.2, noch abschließende Entscheidungen bezüglich der verwendeten Plattform, der eingesetzten Programmiersprache, der benutzten Bibliotheken und der passenden Werkzeuge getroffen.

5.1 Handhabung der Eingabe

Die in Kapitel 4.2 konzipierte formale Sprache beschreibt genau eine gültige Eingabesprache für den Detektor, der jede Eingabe genügen muss. Dieser Vorgang der Spracherkennung wird auch parsen genannt und mit Hilfe des Tools ANTLR in der Version 3 (ANTLRv3) umgesetzt.

5.1.1 ANTLR

ANother Tool for Language Recognition (ANTLR) ist ein Tool zur automatischen Konstruktion von Programmen zur Spracherkennung oder kurz ein Parser Generator [Par07]. Von der Beschreibung einer formalen Sprache, auch Grammatik genannt, ausgehend generiert ANTLR ein Programm das bestimmt, ob eine Eingabe zu dieser Sprache konform ist.

Während es viele verschiedene Parser Generatoren gibt, bietet ANTLR für diesen Fall eine Reihe von Vorteilen. Es bietet eine plattformunabhängige IDE (ANTLRWorks) auf JAVA Basis mit Debug Unterstützung, kann für Menschen lesbaren Code in sehr vielen verschiedene Zielsprachen erzeugen, u.a. Java und C. Dank des integrierten Lexers und der Möglichkeit auch Tree Parser zu erstellen ist kein Einsatz weiterer Software erforderlich.

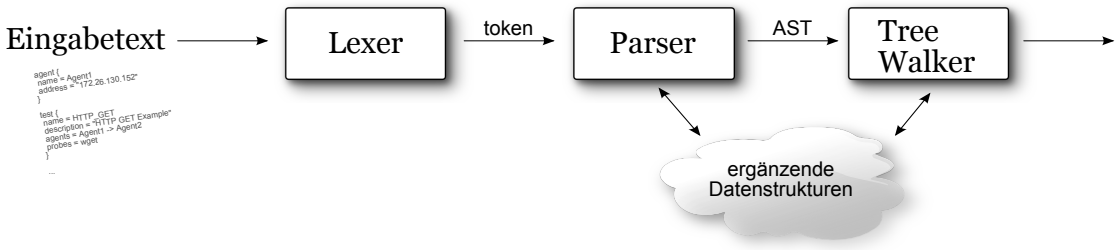


Abbildung 5.1: Verarbeitung mit ANTLR nach [Par07] (Seite 24)

Die Funktionsweise wird im Folgenden entsprechend der Verarbeitungsreihenfolge wie in

Abbildung 5.1 erläutert. Die Schritte dienen der Erzeugung eines sogenannten Abstract Syntax Trees (AST), also einer spezifischen und hochgradig komprimierten Datenstruktur, für die weitere Verwendung in der Implementierung.

Die von ANTLR benötigte Grammatik hat dabei weitgehend die Form der in Abschnitt 4.2.1 definierten Metasprache und wird hauptsächlich zur Baumerzeugung um ANTLR spezifische Ausdrücke erweitert.

5.1.2 Lexer

Die lexikalische Analyse ordnet die Zeichen des Eingabedatenstroms bestimmten Token zu. Die Token werden von ANTLR automatisch aus der Grammatik gewonnen. Die Token Namen tauchen daher auch bereits in Kapitel 4.2 auf und müssen in ANTLR immer mit einem Großbuchstaben beginnen. Aus 'test' in der Eingabe wird das Token 'TEST', aus 'name' wird 'NAME' und so weiter. Sowohl die Eingabesprache als auch die formale Sprache in ANTLR ist case-sensitiv, weshalb Lexer und Parser Regeln innerhalb einer Textdatei definierbar sind.

Neben den trivialen Token mit einfacher Stringersetzung können auch komplexere Strukturen wie im Beispiel 'DATETIME' aus Listing 5.1 durch Kombinationen von Token erzeugt werden. Dies ist die Entsprechung der W3C Definition aus Listing 4.9 in der formalen Sprachedefinition von ANTLR. Hier kommen auch sogenannte Hilfsregeln zum Einsatz, die nur Teil anderer Token Regeln sein dürfen und durch das Merkmal 'fragment' gekennzeichnet sind.

Listing 5.1: Formale Definition von DATETIME in ANTLR

```
1  DATETIME
2      :   DATE 'T' TIME TZD ;
3
4  fragment
5  DATE    :   (NUMBER NUMBER NUMBER NUMBER) '-' (('0' NUMBER) | ('1' '0'..'2'))
       '-' (('0'..'2' NUMBER) | ('3' '0'..'1')) ;
6  // YYYY-MM-DD
7
8  fragment
9  TIME    :   (('0'..'1' NUMBER)|('2' '0'..'3'))':'('0'..'5' NUMBER)
       ':'('0'..'5' NUMBER) ;
10 // hh:mm:ss
11
12 fragment
13 TZD :   (('Z') | (      // Time Zone Designator
14               ('-'|'+')
15                  (
16                      (
17                          (('0' NUMBER)|('1' '0'..'1'))
18                          ':'('0'..'5' NUMBER)
19
20                      )
21                   | '12:00'
22                   )
23              )
24          )
25      ;
26 // +hh:mm or -hh:mm
```

Zusätzlich dazu lassen sich auch imaginäre Token erstellen, die in der Eingabesprache nicht präsent sind, und sogenannte Alias Token wie DEF, RIGHT und LEFT, die für mehr Klarheit in der formalen Sprache sorgen. Da gerade die imaginären Token später eine wichtige Aufgabe erfüllen, werden sie hier kurz aufgezählt: TestTree, AgentTree, AgentSetTree, ProbeTree, ProbeSetTree, PeriodTree und TriggerTree.

Eine Lexer Grammatik hat im Gegensatz zu einer Parser Grammatik keine Startregel, d.h. jede Tokendefinition kann zu jeder Zeit einem Wort in der Eingabe entsprechen.

Abbildung 5.2 verdeutlicht noch einmal den durch den Lexer durchgeführten Prozess der Tokenbildung.

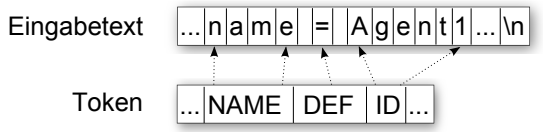


Abbildung 5.2: ANTLR Lexer nach [Par07] (Seite 46)

In der NNTester genannten Grammatik für den Detektor werden Leerzeichen, Kommentare und Zeilentrenner durch den Lexer vor dem Parser versteckt.

5.1.3 Parser

Der Parser bildet nun aus den Token einen sogenannten Parse Tree, der die Token entsprechend der Eingabereihenfolge in einen Baum einfügt. Aus Gründen der besseren Übersicht zeigt Abbildung 5.3 den Parse Tree zu dem Beispiel aus Listing 4.16 mit nur den zum test Attribut gehörigem Teil.

Der vollständige Parse Tree findet sich im Anhang unter Abschnitt 4.

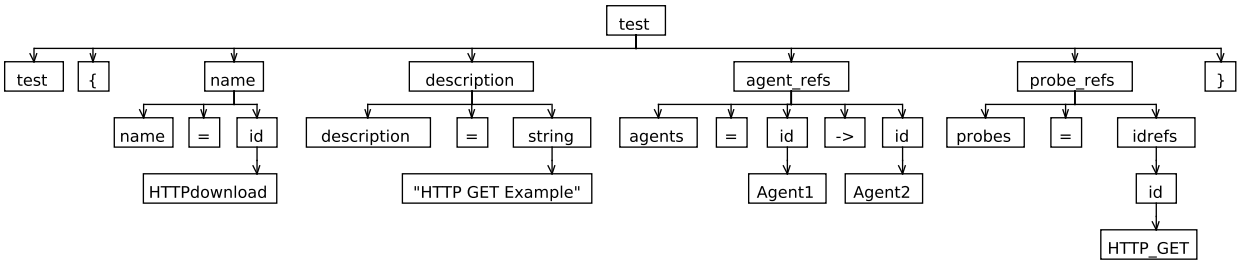


Abbildung 5.3: Ausschnitt aus dem Parse Tree des HTTPdownload Tests

Dies ist bereits eine gut strukturierte Datenstruktur, deren Aufbau allerdings exakt der Struktur der Eingabesprache entspricht und eine Menge überflüssiger Token enthält. Überflüssige Token sind beispielsweise Klammern, die die Eingabesprache strukturieren, aber nach dem Parsen keine Information mehr liefern. Der ANTLR Parser bietet deshalb Möglichkeiten an, den Baum anzupassen und einen sogenannten Abstract Syntax Tree (AST) auszugeben.

Der AST enthält dann keine überflüssigen Token mehr und wird optimiert für die spätere Verarbeitung konstruiert. Den zu Abbildung 5.3 korrespondierenden AST zeigt Abbildung 5.4.

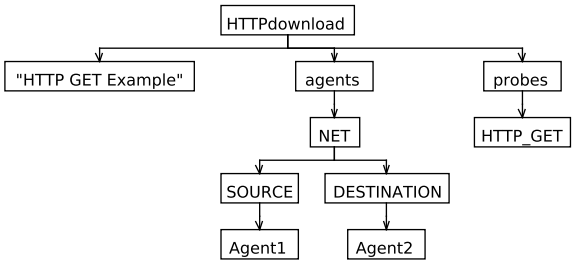


Abbildung 5.4: Ausschnitt aus dem AST des HTTPdownload Tests

Hier ist schön, neben der deutlichen Verschlankung, das Einfügen zusätzlicher Token zu sehen. Aus 'Agent1 -> Agent2' wurde Agent1 'SOURCE', Agent2 'DESTINATION' und beiden ein 'NET' zugeordnet. Damit wird die verkürzende Schreibweise innerhalb des 'test' Attributes der ausdrucksstärkeren Schreibweise der Agentensets angepasst. Dadurch ist später im Programm keine Unterscheidung der zwei Möglichkeiten mehr nötig.

Dieser Schritt ist deshalb so wichtig, da er die verschiedenen Schreibweisen für eine Testspezifikation aus der Konzeption in Abschnitt 4.3.1 abstrahiert und zu einer allgemein gültigen überführt. Somit wird die Mengenbildung über das Set Konzept aus Abschnitt 4.3.3 und die verkürzte Schreibweise für Agentenreferenzen vor der Implementierung verborgen.

Entscheidend in diesem Schritt ist ausserdem das Sammeln aller gleichen Objekte unter jeweils einem Objektknoten im AST. Der 'AgentTree' sammelt also beispielsweise alle Agenten unter sich. Hierfür werden die in Kapitel 5.1.2 bereits aufgezählten imaginären Token als Objektknoten verwendet. Am Ende besteht der AST also nur aus einer Wurzel und den sieben Objektknoten mit den entsprechenden Teilbäumen der jeweiligen Objekte als Kinder. Für das vollständige HTTPdownload Beispiel zeigt Abbildung 5.5 den erzeugten AST.

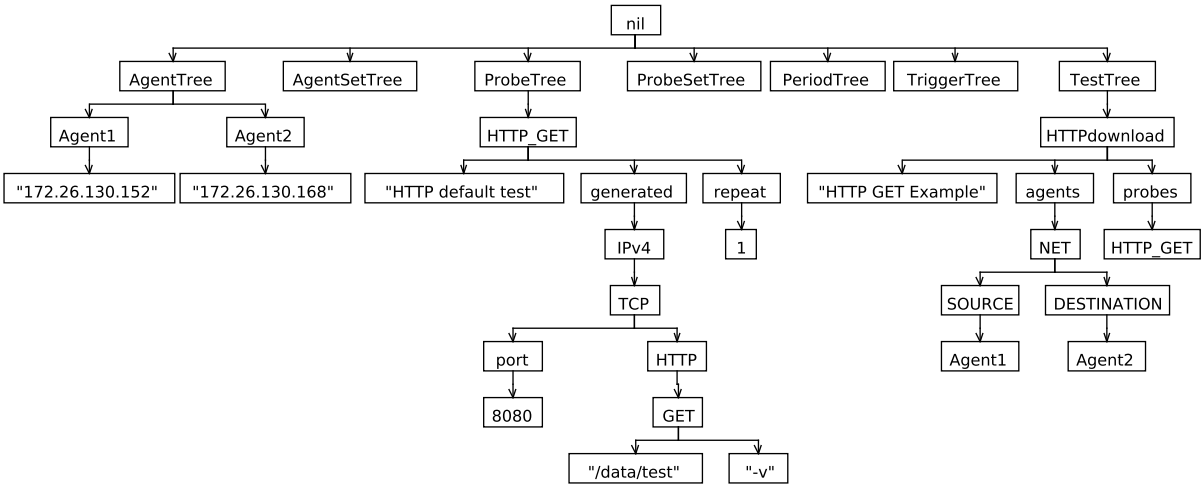


Abbildung 5.5: Vollständiger AST zum HTTPdownload Test

Die vollständige NNTester Grammatik für den Lexer und Parser zeigt Abschnitt 1 im Anhang.

So ist der AST schon sehr gut für eine Nutzung im Detektor vorbereitet. Trotzdem spe-

zialisiert ein sogenannten Tree Walker den AST noch weiter indem alle Informationen in den 'TestTree' Teilbaum integriert werden.

5.1.4 Tree Walker

Der Tree Walker arbeitet mit dem durch den Parser erstellten AST und konstruiert wieder einen AST. In dem Fall des Detektors wird der Tree Walker benutzt, um die Referenzen aufzulösen und zu integrieren. Abbildung 5.6 zeigt den fertigen Teilbaum für den HTTPdownload Test. Verändert werden durch den Tree Walker ausschließlich Teilbäume des 'TestTree' Objektknotens, die übrigen sechs Objektteilbäume bleiben unverändert und sind daher nicht wieder auf der Abbildung zu sehen.

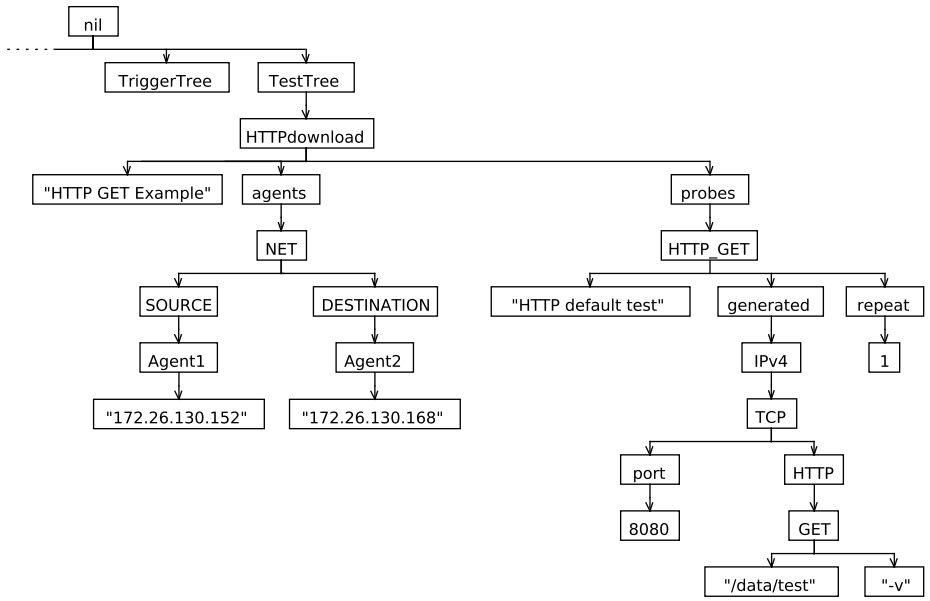


Abbildung 5.6: Endgültiger AST nach Tree Walker zum HTTPdownload Test

Als Ergebnis finden sich nun alle Tests vollständig, d.h. inklusive aller referenzierten Objekte, als Teilbäume im 'TestTree' und der Testname im Wurzelknoten. Verweisen mehrere Tests auf dieselben Objekte, werden diese kopiert. Die Detektor Software kann nun einfach für jeden Test den jeweiligen Baum verwenden, der, trotz der Möglichkeit Eingaben unterschiedlich zu formulieren, jetzt immer eine normalisierte Form aufweist.

Die vollständige NNTesterTreeWalker Grammatik für den Tree Walker findet sich im Anhang unter Abschnitt 2.

5.2 Komponenten

Der Parser ist allerdings nur eine von vielen für den Detektor notwendigen Komponenten, die in dem Server und den Agenten die Funktionalitäten der Anforderungen gewährleisten.

5.2.1 Protokoll

Die Funktionsweise des Protokolls wurde bereits in der Konzeption in Abschnitt 4.1.7 ausführlich beschrieben. Wichtig für den Entwurf ist nun der Aufbau der TestInfos und der Ergebnisse.

Für einen möglichst generischen Umgang mit Messergebnissen werden Ergebnisse in Textdokumente geschrieben und übertragen. Dadurch kann die Ausgabe jedes beliebigen Werkzeugs und umfangreiche tabellarische Aufstellungen von Monitoring Werkzeugen in Komma separierten Listen protokolliert werden. die Anzahl der bei einem Test entstehenden Dokumente kann dabei unbegrenzt sein. Für eine eindeutige Zuordnung aller Dokumente zu einem Test werden diese nach einer bestimmten Namenskonvention u.a. mit einer eindeutigen Test ID benannt.

Als TestInfo erhalten die Agenten fünf separate Informationen: eine Test ID, ihre Rolle, die Nummer ihres Netzes, ihren Namen und den Test AST als geklammerte Zeichenkette.

1. Die Test ID ist eine zufällig generierte Zahl, die der Server jedem Test zuweist, und ermöglicht die Identifizierung einzelner Tests und ihrer Ergebnisse.

2. Die drei Rollen sind aus dem Konzept (Abschnitt 4.1.7) bekannt und entscheiden, neben der im Test AST enthaltenen Probe Spezifikation, über die einzusetzenden Werkzeuge. Nur so kann ein Agent entscheiden, ob er für den 'HTTPdownload' Test das Werkzeug wget ausführt oder einen Webserver startet.

3. Die Nummer des Netzes teilt dem Agenten mit, zu welchem Netz ('Net') in der Testspezifikation (vgl. Abschnitt 4.3.3) er gehört und erlaubt ihm die Bestimmung seiner Testpartner, d.h. seiner Ziel bzw. Quellagenten.

4. Der in der Spezifikation vergebene Agentenname wird zusätzlich zur Test ID für eine eindeutigere Benennung der Ergebnisdateien genutzt.

5. Der Test AST repräsentiert den Test Teilbaum, der diesen Test vollständig beschreibt und enthält folglich alle spezifizierten Informationen zu diesem Test. Im Weiteren wird dazu näher eingegangen.

Weil der von ANTLR erzeugte AST keine zum Versenden geeignete Datenstruktur ist, wird er als sogenannter String Tree über eine entsprechend geklammerte Zeichenkette abgebildet und übertragen. Um eine gleichartige Implementierung zu ermöglichen, wird diese Zeichenkette ebenfalls mit ANTLR über eine weitere Grammatik geparsed und in die vom Server gewohnte AST Datenstruktur überführt. Da die String Tree Repräsentation gegenüber dem original AST Metainformationen zu den einzelnen Tokentypen einbüßt, ist ein gewisser Informationsverlust unvermeidbar. Entscheidend ist allerdings, dass die Informationen zum Test vollständig erhalten bleiben.

Listing 5.2 zeigt für den HTTPdownload Test die String Tree Repräsentation. Die Grammatik für die Überführung des String Trees in einen AST findet sich im Anhang unter Abschnitt 3.

Listing 5.2: HTTPdownload Testspezifikation als String Tree

```
(HTTPdownload "HTTP GET Example"
    (agents
        (NET
            (SOURCE
                (Agent1 "172.26.130.152")
            )
            (DESTINATION
                (Agent2 "172.26.130.168")
            )
        )
    )
    (probes
    (HTTP_GET "HTTP default test"
        (generated
            (IPv4
                (TCP
                    (port 8080)
                        (HTTP
                            (GET "/data/test" "-v")
                        )
                    )
                )
            )
            (repeat 1)
        )
    )
)
```

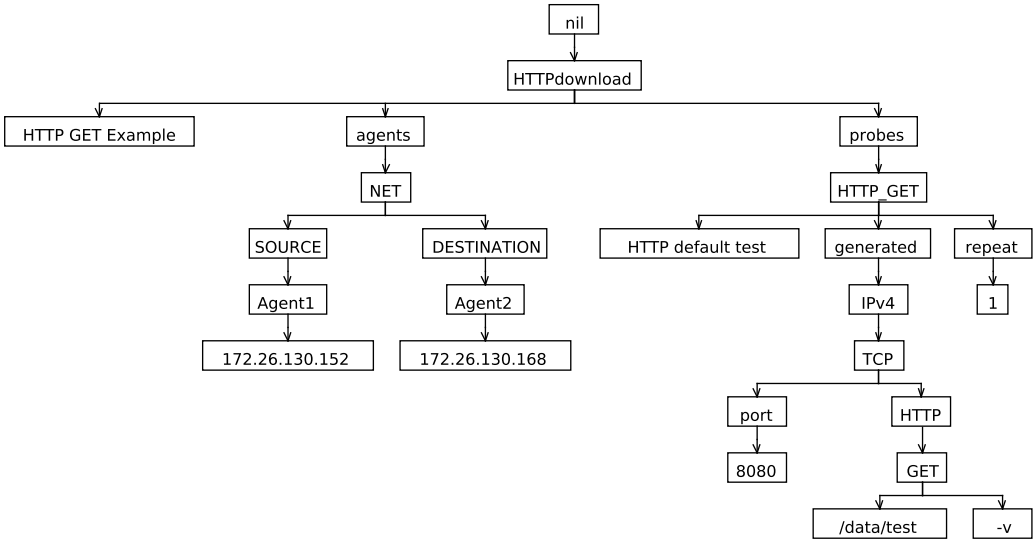


Abbildung 5.7: HTTPdownload AST nach dem Parsen im Agent

Bei der Umsetzung in den AST werden noch die nun unnötigen Anführungszeichen um die Zeichenketten, die keine IDs sind, entfernt. Das Resultat zeigt Abbildung 5.7. Damit verfügt nun auch der Agent über den zu diesem Test gehörigen AST (vgl. den AST des Servers aus Abbildung 5.6). Während der Server neben den sechs Objektteilbäumen auch mehrere

Test AST Teilbäume verwalten kann (eben für jeden spezifizierten Tests einen), erhalten die Agenten ausschließlich den für sie relevanten Teilbaum.

Ein weiterer Schritt über einen Tree Walker ist hier nicht mehr erforderlich.

5.2.2 Server

Die Komponenten für den Server werden in Abbildung 5.8 grafisch grau hinterlegt dargestellt. Nachdem durch den ANTLR Lexer, Parser und TreeWalker aus der Eingabe ein AST erzeugt wird, wird für jeden Test seine Startzeit und eventuell sein Intervall in den Timer eingetragen. Dieser löst zu den definierten Zeiten ein Signal aus und der entsprechende Test wird durchlaufen. Hier muss zwischen Test und Testdurchlauf unterschieden werden: Ein Test kann bei entsprechend spezifizierter Dauer und definiertem Intervall aus mehreren Testdurchläufen bestehen.

Um eine gegenseitige Beeinflussung der Tests zu vermeiden und Vergleichbarkeit nach Anforderung 2 zu gewährleisten, wird immer nur ein Testdurchlauf gleichzeitig ausgeführt. Eventuell überlappende Testdurchläufe werden sequentiell durchgeführt. Dank dieser Vorgehensweise lassen sich mehrere Tests gleichzeitig und, abgesehen von nicht immer exakten Startzeiten, ohne Behinderung einzelner Testdurchläufe durchführen.

Ein jeder Testdurchlauf folgt im Server dem gleichem Ablauf. Zuerst wird eine sogenannte TestInfo in der Komponente Testdurchlauf erzeugt, in der die für den Agenten nötigen Informationen zusammengefasst werden. Diese wird dann in der Komponente für die Kommunikation serialisiert und an jeden an diesem Test beteiligten Agenten geschickt. Dazu wird mit jedem Agenten für jede Rolle eine Kommunikationsbeziehung über die Komponenten Quellagent und Zielagent hergestellt, d.h. insbesondere auch mehrere Kommunikationsbeziehungen mit ein und demselben Agenten. Ist ein Agent zugleich Source und Destination werden zwei Verbindungen benötigt, ist er innerhalb mehrerer Net Konstrukte können auch mehr als zwei nötig sein.

Die Aktivitäten (nicht grau hinterlegt) für den Quellagent bestehen aus dem Senden der TestInfo und dem Empfangen der Ergebnisse. Ist der verbundene Agent in einer Ziel Rolle, wird nach dem Ende aller Quellagenten eine Ende Nachricht geschickt und erst danach die Ergebnisse empfangen.

Die Ergebnisse können nun beliebig ausgewertet werden. Auch manche Trigger lassen sich erst hier auf die gesammelten Ergebnisse anwenden, genauso wie unter 'action' spezifizierte Aktionen hier eventuell noch ausgelöst werden müssen. Anschließend ist dieser Testdurchlauf beendet und ein neuer kann beginnen.

Das zentrale Server Konzept unterstützt bereits die Realisierung einer einfachen Bedienung, wie in Anforderung 8 gefordert. Eine GUI Komponente kann auf die wohldefinierte Sprache zurückgreifen und würde vor dem Parser platziert werden. In der Auswertungskomponente können Grafiken und Diagramme zur Unterstützung generiert werden. Da alle Ergebnisse aller jemals durchgeführten Tests auf dem einen Server liegen und Vergleichbarkeit gewährleistet ist, lassen sich auch Untersuchungen über viele Tests und über verschieden lange Zeitspannen anstellen.

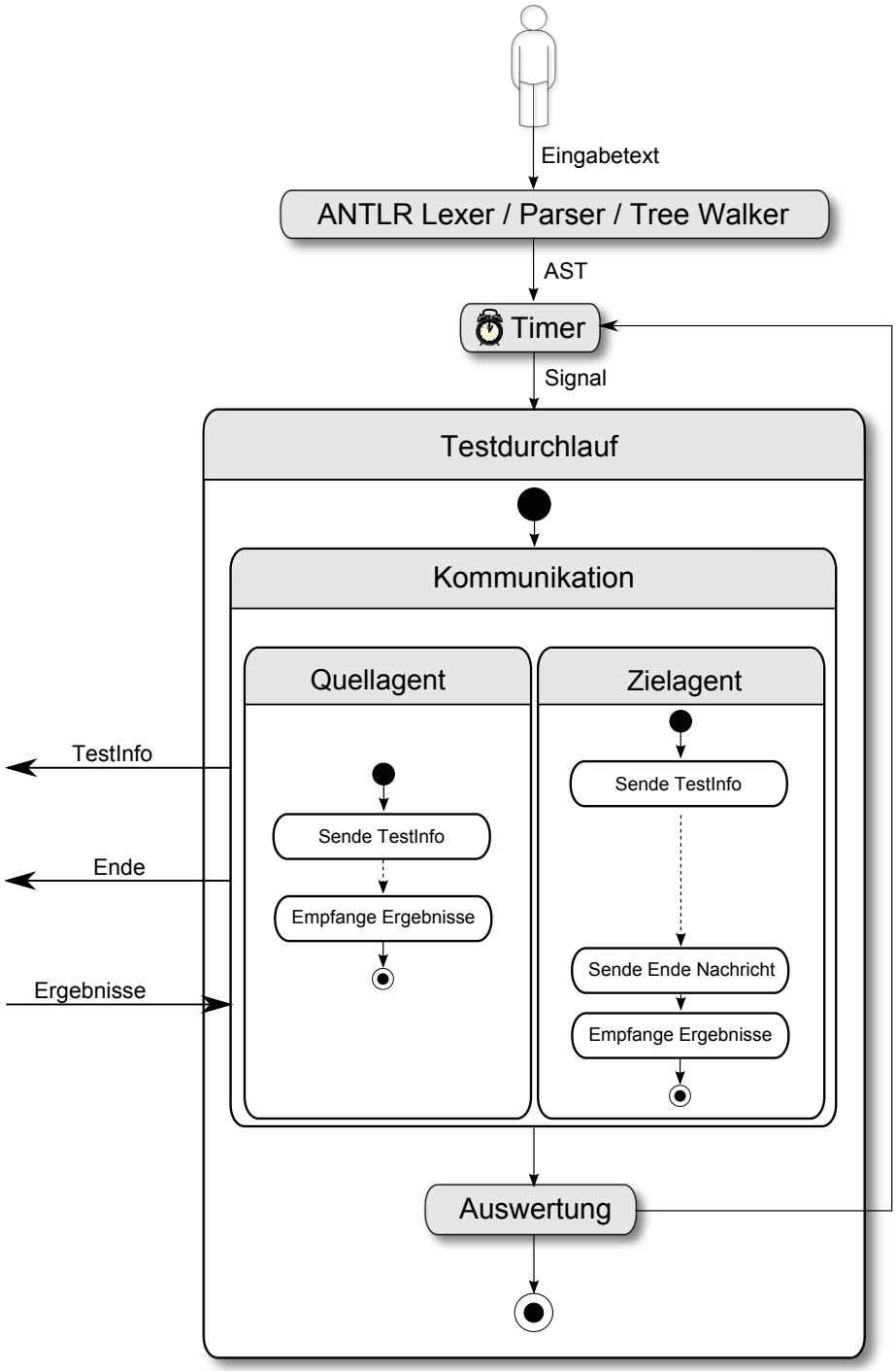


Abbildung 5.8: Server Komponenten und Aktivitäten

5.2.3 Agenten

Jeder Agent akzeptiert zu jeder Zeit Verbindungsanfragen des Servers. Der Agent akzeptiert explizit, um beispielsweise zugleich Quelle und Ziel zu sein, mehrere Verbindungen gleichzeitig und behandelt jede Kommunikationsbeziehung gleichberechtigt. Abbildung 5.9 zeigt die Komponenten und deren Aktivitäten.

Nach Empfang der TestInfo wird der empfangene String Tree wie in Abschnitt 5.2.1 beschrieben geparsed, um auch im Agent einen AST zu erhalten. Anschließend wird in der Testkomponente nach Rolle unterschieden.

Als Ziel muss der Agent jeden innerhalb dieses Tests theoretisch benötigten Dienst starten. Dazu muss zu jeder definierten Probe ein Empfangsdienst spezifiziert sein. Manche Dienste brauchen allerdings nicht extra initiiert werden, da sie, wie beispielsweise ICMP Requests, automatisch vom Betriebssystem beantwortet werden. Im Fall des 'HTTPdownload' Beispiels wird auf der Empfangsseite ein Webserver benötigt, der die Anfragen auch beantworten kann. Insbesondere müssen auch Dienste gestartet werden, die nur bei Auslösen eines Triggers benötigt werden, da der Agent keine Möglichkeit hat, dies zu erfahren. Beendet werden die Dienste durch eine entsprechende Nachricht vom Server.

Falls der Agent als Quelle arbeitet, werden die durch den Probe Teilbaum des AST beschriebenen Dienste, neben eventuell zusätzlichen Monitoring Diensten, der Reihe nach ausgeführt, deren Ergebnisse protokolliert und an den Server zurückgeschickt. Ist ein Trigger definiert, müssen noch während der Ausführung des Quelldienstes die Ergebnisse nach den Triggerkriterien, sofern sie denn lokal zu bestimmen sind, untersucht werden. Löst ein Trigger aus, wird der ursprüngliche Dienst gestoppt und nach der spezifizierten Aktion weiter verfahren.

Wie bereits in Kapitel 4.3.1 beschrieben, kann dies einerseits das einfache Ausführen weiterer Probes oder Trigger sein, andererseits das Ausführen eines völlig neuen Tests, sowie die Optionen Löschen, Neustart oder Stop, die nur über den Server vollständig ausgeführt werden können und als Ergebnis an den Server gemeldet werden. Sind hingegen weitere Probes oder Trigger als Aktion spezifiziert, wird ein neuer Quelldienst mit zugehörigen Monitoring Diensten und eventueller Trigger Überprüfung gestartet. Nach dessen Ende werden eventuell verbleibende Dienstwiederholungen des ursprünglichen Dienstes ausgeführt, welche wieder die jeweilige Untersuchung auf das Triggerkriterium beinhalten.

Während der eigentlichen Durchführung der Tests findet zur Vermeidung von Einflüssen keine Kommunikation mit dem Server statt, die Verbindung bleibt allerdings offen. Als Dienste können alle denkbaren Netzdienste herangezogen werden, insbesondere kann auch nach Anforderung 7 Fremdsoftware ausgeführt werden.

Die Test Komponente ist beendet, wenn alle dafür gestarteten Dienste ebenfalls wieder beendet sind. Danach werden entweder die Ergebnisse aus den Protokolldateien der Dienste und des Monitorings oder eine Information bezüglich Triggeraktionen an den Server gesendet.

Nun ist das System des Detektors vollständig entworfen und kann in einem Prototypen realisiert werden. Dazu wird im Folgenden die für die Implementierung verwendete Hardware mit Ubuntu als Betriebssystem und die einzelnen Komponenten des Detektor Prototypen beschrieben.

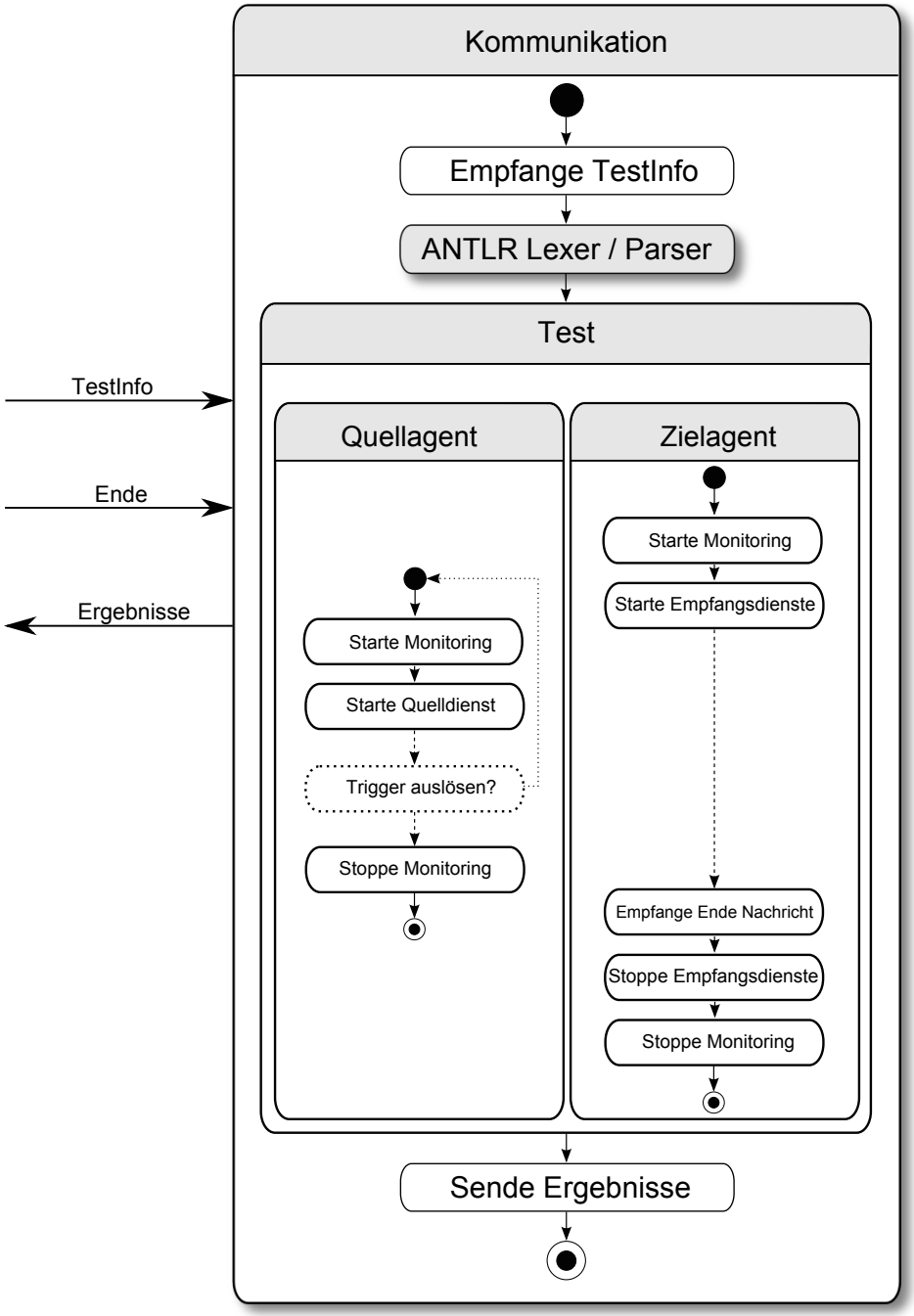


Abbildung 5.9: Agent Komponenten und Aktivitäten

5.3 Ausstattung der Systeme

Für die Überprüfung der Tauglichkeit des Konzepts reichen die zwei nachfolgend beschriebenen Systeme aus. Als Server fungiert das leistungsstärkere System, das zugleich auch als Agent agiert.

5.3.1 Hardware

Zum Einsatz kommt als Agent ein älterer Fujitsu Siemens Laptop der S Serie mit 2 GB RAM, 32-Bit Architektur und Fast Ethernet aus den ausrangierten Beständen von Amadeus, der für den Zweck über ausreichend Performance verfügt. Als Server agiert ein neuer Lenovo Laptop der ThinkPad Edge Reihe mit 4 GB RAM, 64-Bit Architektur, Gigabit Ethernet und integriertem UMTS Modul. Durch diese beiden Systeme wird der Detektor sowohl für 32-Bit als auch für 64-Bit Architekturen entwickelt. Weitere Fujitsu Siemens Laptops sollen das System in Zukunft ausbauen. Dabei stellen deren Netzschnittstellen mit maximal 100 MBit/s eine gewisse Beeinträchtigung dar, da sie nicht in der Lage sind breitbandige Internetverbindungen voll auszulasten.

5.3.2 Software

Als Betriebssystem kommt Ubuntu in der Long Term Support Version 10.04.2 (Lucid Lynx) vom 18. Februar 2011 zum Einsatz. Dafür sprechen eine Reihe von Gründen: Die Anforderungen verlangen einen sparsamen Umgang mit Ressourcen nach Anforderung 9 mit der Möglichkeit einer eventuellen späteren Integration des Systems auf ein embedded Device. Ubuntu hat relativ wenig Software vorinstalliert und gilt daher als vergleichsweise schlanke Linux Distribution, die sich aber gezielt um die benötigten Funktionalitäten erweitern lässt. Ausserdem funktionieren die meisten embedded Devices zumindest mit einer Art Linux.

Auf Linux laufen auch bei weitem die meisten Werkzeuge für Netzuntersuchungen, die laut Anforderung 7 in den Detektor integriert werden sollen. Einige davon werden auch von dem Prototyp eingesetzt.

Ebenfalls ein Grund für Linux sind die Kosten und Lizenzen. In Zukunft soll das System aus möglichst vielen weit verteilten Agenten bestehen, welche alle ein Betriebssystem benötigen. Einerseits ist Ubuntu kostenlos, andererseits ist keine Lizenzierung notwendig, die das Verteilen zu Fremdunternehmen erschweren würde.

Als Entwicklungsumgebung ist Eclipse auf dem Server eingerichtet, der auch über alle verwendeten Bibliotheken verfügt. Wegen der unterschiedlichen Architekturen der beiden Systeme muss die Agentensoftware für jedes System extra kompiliert werden. Die verwendeten Bibliotheken werden dabei jedoch mit eingebunden und brauchen nicht als Shared Libraries auf den Agenten zur Verfügung stehen.

Damit ist die Basis für die Entwicklung gelegt, die eigentliche Prototypsoftware wird anschließend vorgestellt.

5.4 Prototyp des Detektors

Die Wahl der Programmiersprache für den Detektor fiel vor allem aufgrund der sehr guten Integrationsmöglichkeiten durch zahlreiche freie Bibliotheken auf C. Die meisten Werkzeuge für Netze sind in C geschrieben und bieten unter Umständen sogar die Möglichkeit einer

direkten Integration über eine Bibliothek. Weiteres Argument dafür war die vollständige Unterstützung von C als Ausgabesprache des ANTLR Generators. Abgesehen von zwei Skriptsprachen sind derzeit nur C und Java ohne bekannte Fehler und synchron zur ANTLR3 Entwicklung [IN11]. Die einzelnen Module werden im Folgenden getrennt nach Server und Agent beschrieben.

5.4.1 Server

Abbildung 5.10 zeigt die Komponenten der NNTester genannten Server Software, welche der Ausführungsreihenfolge nach beschrieben werden.

NNTesterLexer, NNTesterParser und NNTesterHelper

Die vom Benutzer übergebene Textdatei wird im ANTLR Lexer zu einem Token Stream umgewandelt und anschließend wird durch den ANTLR Parser der erste AST, wie in Abschnitt 5.1.2 und 5.1.3 schon ausführlich beschrieben, modelliert. Diese beiden Module erzeugt ANTLR aus der NNTester Grammatik (Anhand Abschnitt 1).

Für erweiterte Funktionalität während des Parsens sorgen innerhalb der Grammatik definierte Funktionsaufrufe zu Funktionen eines Helper Moduls. Im Fall von NNTesterParserHelper wird hier jede spezifizierte Portnummer auf einen gültigen Wert zwischen 0 und 65535 geprüft.

NNTesterTreeWalker und NNTesterTreeWalkerHelper

Anschließend wird der AST erneut durch die von ANTLR auf Basis der NNTesterTreeWalker Grammatik aus Abschnitt 5.1.4 erzeugten NNTesterTreeWalker Komponente geparsed. In diesem Schritt werden dank des erneuten Helper Moduls für den Tree Walker die Referenzen aufgelöst und die damit verlinkten Teilbäume in den Test Teilbaum kopiert. Diese Art der Baum Manipulation kann über die ANTLR Grammatik nicht ausgedrückt werden und wird deshalb in dem Helper Modul selbst implementiert. Nur so ist es möglich den endgültigen AST in genau eine normalisierte Struktur zu überführen, wie im Entwurf in Abschnitt 5.1.4 beschrieben, und unabhängig von der Art der Eingaben zu machen. Somit muss in der folgenden Implementierung nicht mehr auf Set Definitionen oder verkürzte Schreibweisen geachtet werden.

Timer

Aus dem nach dem Tree Walker fertig modellierten AST extrahiert die Timer Komponente die für die korrekte zeitliche Ausführung aller spezifizierten Tests notwendigen Informationen und konfiguriert über die Standard C Bibliothek (time.h und signal.h) einen System Timer. Dieser löst zu den spezifizierten Zeitpunkten Signale aus, welche von der Signalverwaltung den korrespondierenden Tests zugewiesen werden und deren Durchführung dann eingeleitet wird. Signalisiert das System den Start mehrerer Tests gleichzeitig, werden diese in eine Warteschleife (sys/queue.h) eingereiht und sequenziell ausgeführt.

Test

Die Test Komponente erstellt für jeden Agenten und seine Rollen eine TestInfo gemäß dem Protokollentwurf in Abschnitt 5.2.1 und startet über jeweils einen eigenen Thread die Kom-

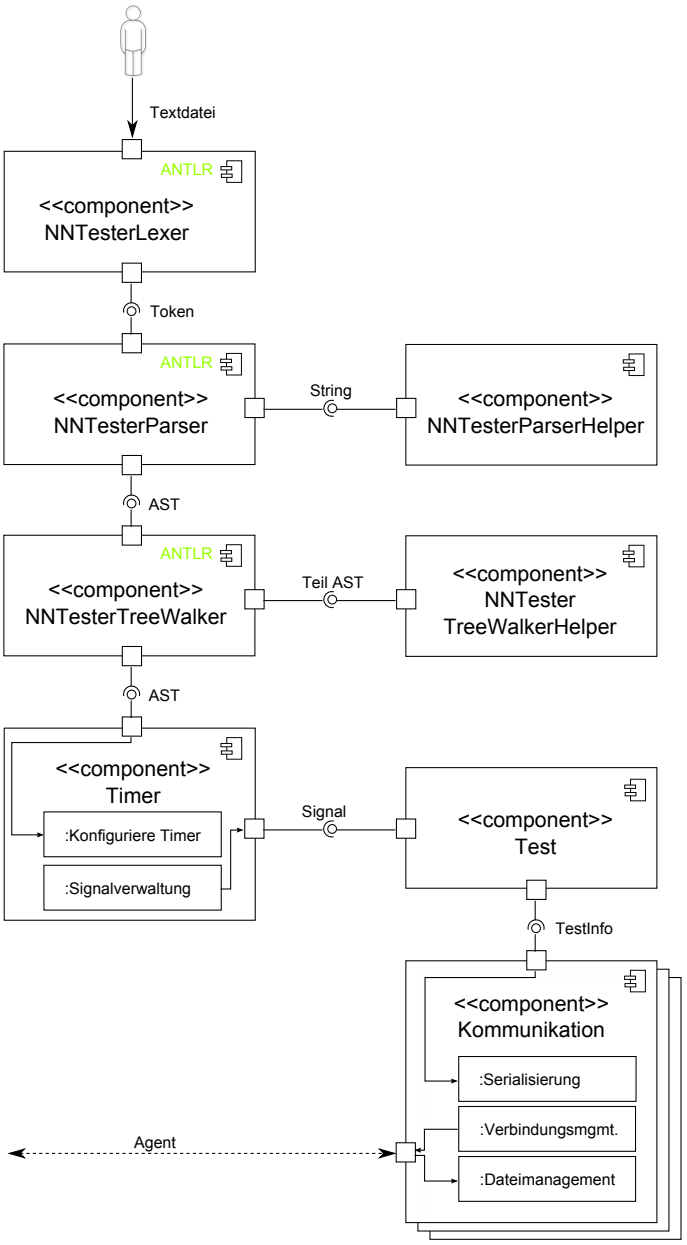


Abbildung 5.10: NNTester Komponentendiagramm (Server)

munikationskomponente. Die TestInfo ist in Form eines Structs mit einem Integer als Test ID, einem String für die Rolle ('src', 'src1' oder 'dst'), einem Integer für die Netznummer, einem String für seinen Namen und wieder einem String für den AST in Form eines String Trees realisiert.

Kommunikation

In der Kommunikationskomponente wird die TestInfo mit Hilfe der tpl Bibliothek ([Han]) serialisiert, eine TCP Verbindung über Port 2195 zu dem Agenten aufgebaut und die TestInfo

übertragen.

Je nach Rolle des Agenten wird nun entweder, im Falle einer Kommunikation mit einem Quell Agenten, auf die Übertragung der Testergebnisse in Form von beliebig vielen Textdateien gewartet oder, im Fall mit einem Ziel Agenten, dem Agenten dieser Kommunikationsbeziehung vor dem Empfang von Ergebnissen und nach dem Ende der Tests bei allen Quell Agenten über eine spezielle Ende Nachricht ('\0\0\0SHUTDOWN\0\0\0') das Ende dieses Tests mitgeteilt.

Das Warten auf das Ende der Quell Agenten wird durch eine von wechselseitigem Ausschluss geschützte (über pthread mutex) Variable realisiert, die das Ende aller Kommunikationsthreads mit Quell Agenten signalisiert.

Für die Übertragung der Ergebnisse wird in festgelegter Reihenfolge zuerst der Dateinamen übertragen, dann Zeile für Zeile der Inhalt der Textdateien bis ein Dateiende Symbol ('\0\0\0') das Ende dieser Datei anzeigt. Es können auf diese Weise beliebig viele Dateien hintereinander übertragen werden, bis der Agent den Server durch das Beenden der Verbindung über das Ende informiert.

Der Server speichert alle empfangenen Dateien mit deren Namen in dem Ordner 'measurements'. Eine Auswertung der Dateien erfolgt in diesem Prototyp noch nicht.

5.4.2 Agent

Für das NNTesterAgent Programm zeigt Abbildung 5.11 die Komponenten eines Agenten, die im Folgenden einzeln erläutert werden.

Kommunikation

Die Kommunikationskomponente ist der Ausgangspunkt eines Agenten. Die Agenten Software wartet auf dem TCP Port 2195 auf Verbindungsanfragen eines Servers. Sobald eine Verbindung zustande kommt, wird ein eigener Thread für diese Verbindung erzeugt, während der Vaterprozess wieder auf weitere Verbindungen wartet. Dadurch ist es möglich, den Agenten gleichzeitig als Ziel und Quell Agent oder sogar gleichzeitig innerhalb mehrerer spezifizierter Netz Mengen einzusetzen. Die Zahl der möglichen Verbindungen wird durch die Software selbst nicht begrenzt, allerdings können üblicherweise nur begrenzt viele Threads erzeugt werden.

Nachdem also ein Thread für diese Verbindung erzeugt wurde, wird die TestInfo empfangen, mit tpl deserialisiert und der Testkomponente, die den eigentlichen Test startet, übergeben. Die TestInfo enthält, entsprechend dem Entwurf (vgl. Abschnitt 5.2.1), für diesen einen Test spezifische Daten. Der Agent hat keine Kenntnis über die anderen nur dem Server bekannten Testspezifikationen. Der Zeitpunkt der Testdurchführung wird ebenfalls nur durch das Eintreffen der TestInfo bestimmt und eine Überprüfung der Zeit findet nicht statt. Dies hat zwar einen etwas ungenauen Startzeitpunkt zur Folge, erspart aber eine zeitliche Synchronisation des gesamten Testsystems und eine Betrachtung der jeweiligen Zeitzonen. Wenn Ergebnisse allerdings über einen Zeitstempel verfügen sollen, ist zumindest eine zeitliche Synchronisation über NTP (Network Time Protocol) zu empfehlen.

Während die Testkomponente die Tests durchführt, erfolgt kein weiterer Datenaustausch, mit Ausnahme einer möglichen Ende Nachricht bei einem Einsatz als Zielagent. Diese erfolgt allerdings erst nachdem alle Quellagenten ihre Tests beendet haben und führt zu keiner Beeinträchtigung von Tests.

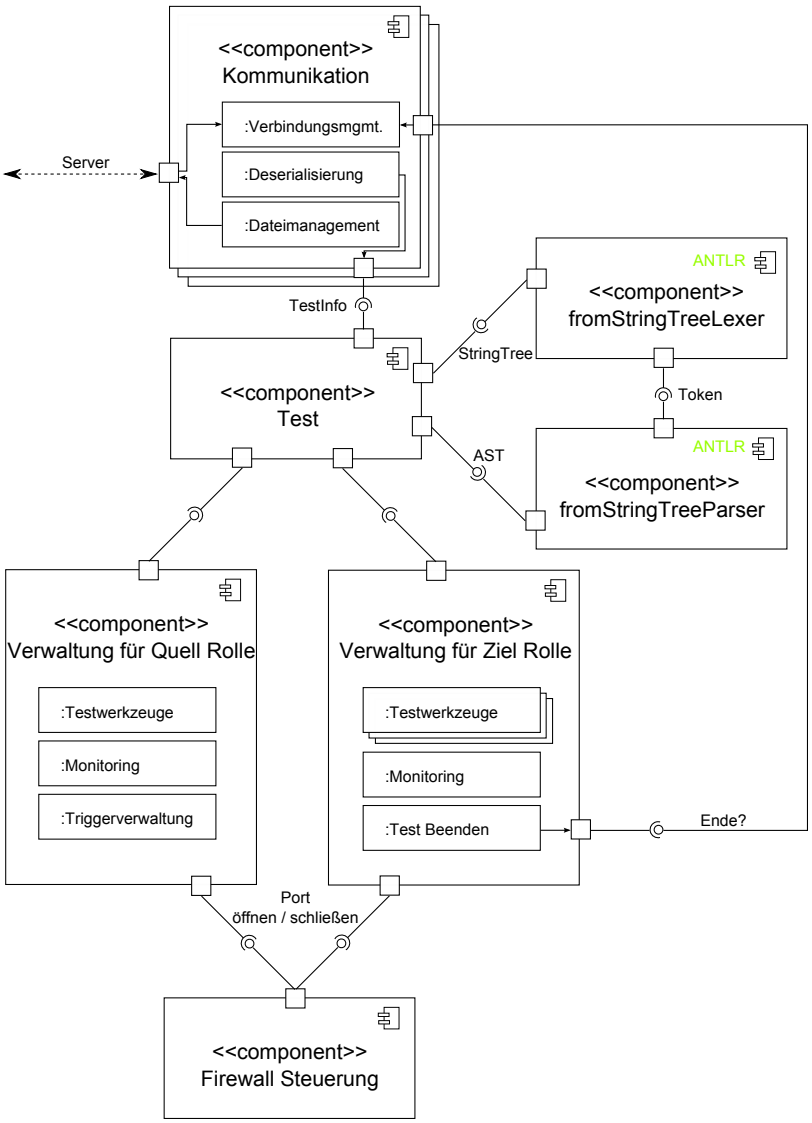


Abbildung 5.11: NNTesterAgent Komponentendiagramm

Abschließend, nach dem eigentlichen Testvorgang, werden alle zu diesem Test gehörenden Ergebnis Dateien nach dem in Abschnitt 5.4.1 beschriebenem Vorgehen zum Server übertragen. Welche Dateien zu welchem Test gehören wird über deren Namen bestimmt, der am Anfang immer die jeweilige Test ID enthält.

Test

Die Testkomponente agiert als Zwischenstation vor den eigentlichen Tests und entscheidet anhand der Rolle welche Verwaltungskomponente einzusetzen ist. Für eine einheitliche Datenrepräsentation im Server und im Agenten wird der in der TestInfo enthaltene String Tree durch den fromStringTree Lexer und Parser wieder in einen AST überführt.

FromStringTreeLexer und fromStringTreeParser

Diese beiden Module erzeugen aus dem String Tree wieder einen AST, der, wie in Abschnitt 5.2.1 gezeigt, dem Teilbaum entspricht, der den durchzuführenden Test spezifiziert. Nebenbei werden die Anführungszeichen bei Zeichenketten entfernt, was mit ANTLR internen Funktionen zu realisieren ist und deshalb keine Helper Komponente benötigt.

Verwaltung für Quell Rolle

In der Verwaltung für den Einsatz als Quelle werden nun durch die im AST spezifizierten Probes die Werkzeuge ausgewählt und eingesetzt. Nebenbei können noch zusätzliche Monitoring Werkzeuge für weitere Messdaten aktiviert werden. Falls ein Trigger spezifiziert ist, erfolgt hier die Auswertung der Triggerkriterien mit der entsprechenden Reaktion. Im Prototyp wird als Trigger Reaktion bisher nur das Auslösen weiterer Probes oder Trigger auf RTT Messungen unterstützt. Die Testwerkzeuge für eine Funktion als Quelle werden bisher ausschließlich über popen() aufgerufen, sind damit nicht in der Software integriert und müssen auf den Agentensystemen installiert sein. In diesem Prototypen eingesetzte Werkzeuge sind ping, traceroute, wget, transmission (BitTorrent Client in Version 2.05, da die vorinstallierte 1.95 einen kritischen Fehler enthielt [Pro]) und opentracker (BitTorrent Tracker [End]).

Als zusätzliches Monitoring Werkzeug kommt dstat ([Wie]) für die Messung der Übertragungsgeschwindigkeit zum Einsatz, das die bekannten Monitoring Werkzeuge vmstat, iostat, netstat und ifstat kombiniert und noch um weitere Funktionen erweitert.

Vor der Nutzung einiger Werkzeuge wird jeweils der betroffene Port über die Firewall Steuerung in der Ubuntu UFW (Uncomplicated Firewall) geöffnet und nach dem Test wieder geschlossen. Für diese Steuerung sind im Moment allerdings Root Rechte erforderlich, d.h. die NNTesterAgent Software muss auch mit diesen ausgeführt werden.

Verwaltung für Ziel Rolle

Entscheidend für die Verwaltung eines Ziel Agenten ist, dass dieser keine Informationen über das Auslösen von Triggern oder die Dauer einzelner Probes hat und deshalb alle eventuell benötigten Empfangswerkzeuge starten muss. Während in einem Quell Agenten immer nur ein Sendewerkzeug aktiv ist (Monitoring Werkzeuge sind explizit davon ausgenommen), muss ein Ziel Agent in der Lage sein, mehrere Empfangswerkzeuge gleichzeitig auszuführen.

Ausserdem hat er keine Möglichkeit zu prüfen, wann ein Test abgeschlossen ist und benötigt dazu eine Ende Nachricht des Servers.

Monitoring und Firewall Steuerung verläuft wie in der Quell Rolle. Als Empfangswerkzeuge kommen im Prototypen nur ein Webserver und transmission zum Einsatz. Der Webserver, der hier nur als einfacher Fileserver fungiert, ist über die GNU libmicrohttpd Bibliothek ([Groa]) direkt in die Agenten Software integriert und lässt sich in einem eigenen Thread auslagern.

Mit dem Prototypen wurden, für ausgewählte Tests, die im nächsten Kapitel beschriebenen Feldversuche durchgeführt.

6 Feldversuche und Evaluation

In diesem Kapitel werden mit dem implementierten Detektor Prototyp ausgewählte Feldversuche durchgeführt. Dazu werden jeweils die Testspezifikation und die Ergebnisse besprochen. Am Ende wird der Prototyp evaluiert und Anreize für Weiterentwicklungen gegeben. Als Testsystem kommen zwei Laptops zum Einsatz, die über verschiedene Zugangsarten mit dem Internet verbunden sind:

- Agent1 ist per UMTS drahtlos mit dem Netz der deutschen Telekom verbunden und agiert gleichzeitig auch als Server.
- Agent2 ist über einen privaten Kabelanschluss in dem Netz von Kabel Deutschland.

Zusätzlich dazu wird für eine Unterstützung der Interpretation ein Test in einem lokalen Netz (LAN) wiederholt.

6.1 Datentransfer über HTTP

Der erste Test folgt der Spezifikation aus Listing 4.16 und veranlasst Agent1 die Datei 'Route66.mp4' von Agent2 über das HTTP Protokoll über den Befehl 'GET' herunterzuladen. Zu beachten ist die unter Umständen etwas verwirrende Schreibweise der Richtung des Downloads. Spezifiziert wird 'Agent1 -> Agent2', da Agent1 den HTTP GET Befehl an Agent2 schickt. Das eigentliche Herunterladen der Datei erfolgt dann aber von Agent2 nach Agent1. Die Datei muss dabei vorab auf dem Quellagenten in dem spezifizierten Ordner hinterlegt werden. Für Testzwecke sollte gewöhnlicherweise eine bestimmte Auswahl an unterschiedlichen Dateien ausreichen.

Die Testspezifikation zeigt Listing 6.1. Entscheidend für diesen Test ist die Spezifikation des HTTP Befehls GET in Zeile 20 mit der Pfadangabe auf die Datei Route66.mp4 und der TCP Port 8080 in Zeile 18.

Listing 6.1: Spezifikation des HTTP Download Tests

```
1  agent {
2      name = Agent1
3      address = "88.128.137.212"
4  }
5
6  agent {
7      name = Agent2
8      address = "91.67.131.173"
9  }
10
11 probe {
12     name = HTTP_GET
13     description = "HTTP download von Route66.mp4"
14     repeat = 1
```

```
15      generated {
16          IPv4 {
17              TCP {
18                  port = 8080
19                  HTTP {
20                      GET "/data/Route66.mp4" "-v"
21                  }
22              }
23          }
24      }
25  }
26
27  test {
28      name = Route66_HTTPdownload
29      description = "HTTP download Route66.mp4 via UMTS"
30      agents = Agent1 -> Agent2
31      probes = HTTP_GET
32  }
```

Für zusätzliche Messungen wird bei diesem Test nebenbei das Werkzeug dstat ausgeführt. Protokolliert wird bei diesem Test ausschließlich auf Agent1, da Agent2 nur einen Webserver und kein Monitoring Werkzeug ausführt. Dem Test wird beim Start die eindeutige TestID '1539541592' zugewiesen. Als Testergebnisse werden zwei Dateien von Agent1 an den Server übertragen: '1539541592_HTTP-GET_Agent1' und '1539541592_HTTP-GET_dstat_Agent1'. Erste enthält die Ausgabe des intern für diesen Test genutzten Werkzeuges wget, die zweite die Ausgabe des Monitoring Werkzeuges dstat, das seine Ausgabe als Komma separierte Liste erstellt. Die wget Ausgabe ist für Untersuchungen wenig geeignet und wird hauptsächlich zur Fehlererkennung erzeugt und gespeichert. Mit den Daten von dstat zeigt Abbildung 6.1 den Verlauf der Übertragungsgeschwindigkeit während dieses Tests. Insgesamt dauerte der Test 974 Sekunden. Ein Ausschnitt des tatsächlichen dstat Protokolls mit weiteren Daten zur Systemzeit und zu Sockets findet sich im Anhang unter Abschnitt 5.

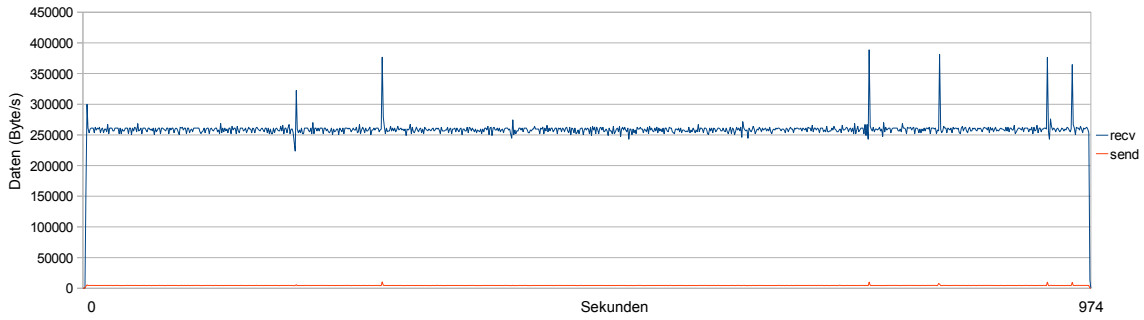


Abbildung 6.1: Übertragungsgeschwindigkeiten während des HTTP Downloads von 'Route66.mp4'

Bis auf ein paar Ausschläge zeigt die blaue Linie für die empfangenen Daten pro Sekunde einen recht geraden Verlauf bei etwa 260 kBit/s, während kaum Daten gesendet werden (rote untere Linie).

6.2 Datentransfer über BitTorrent

Bei diesem Test wird dieselbe Datei 'Route66.mp4' ebenfalls von Agent2 nach Agent1 über das Peer-to-peer Protokoll BitTorrent übertragen. Das Erzeugen der Torrent Datei und dessen Verteilung auf den Agenten muss derzeit noch manuell ausgeführt werden. Der Test wird einmal identisch zum HTTP Test aus Abschnitt 6.1 im Internet durchgeführt und einmal zur Interpretationshilfe in einem LAN.

6.2.1 BitTorrent Verkehr im Internet

Da in der Torrent Datei die Adresse des Trackers gespeichert wird und der Tracker auf dem ersten Quellagenten ausgeführt wird, muss die Torrent Datei exakt zu diesem Test erstellt werden. Dazu genügt ein Blick in die Testspezifikation aus Listing 6.2.

Listing 6.2: Spezifikation des BitTorrent Tests

```
1  agent {
2      name = Agent1
3      address = "88.128.137.212"
4  }
5
6  agent {
7      name = Agent2
8      address = "91.67.131.173"
9  }
10
11 probe {
12     name = BitTorrent
13     description = "BitTorrent Route66 transfer"
14     repeat = 1
15     generated {
16         IPv4 {
17             TCP {
18                 port = 51413
19                 TORRENT {
20                     "Route66.mp4"
21                 }
22             }
23         }
24     }
25 }
26
27 test {
28     name = Route66_torrent
29     description = "BitTorrent download"
30     agents = Agent1 <- Agent2
31     probes = BitTorrent
32 }
```

Die Agenten bleiben die gleichen, aber die Probe wird entsprechend angepasst. Nun spezifiziert Zeile 18 den Port 51413 und Zeile 20 zeigt nun innerhalb des 'TORRENT' Elements nur noch auf den Dateinamen. Wichtig ist auch die veränderte Testrichtung in Zeile 30 um das Herunterladen in die gleiche Richtung wie beim 'HTTPdownload' Test aus Abschnitt 6.1 sicherzustellen. Dadurch wird Agent2 im BitTorrent Jargon zum Seeder (Quelle) und Agent1 zum Leecher (Ziel).

Agent2 als erste und einzige Quelle startet auch den Tracker für diesen Test. Also wird eine Torrent Datei 'Route66.mp4.torrent' mit der Tracker Adresse 'http://91.67.131.173:6969/-announce' erzeugt und auf beiden Agenten abgelegt. Der Tracker Port 6969 ist der Standardport von opentracker und kann nicht über die Testspezifikation geändert werden. Der spezifizierte Port 51413 legt nur den Port des BitTorrent Clients fest (im Prototyp ist dies transmission). Die Torrent Datei muss nach der spezifizierten Datei und dem Zusatz '.torrent' benannt werden.

Für die Aufzeichnung der Übertragungsgeschwindigkeiten wird ebenfalls wieder dstat verwendet. Abbildung 6.2 zeigt den Verlauf. Anzumerken ist, dass die Torrent Downstream Geschwindigkeit über transmission auf 100 KiB/s begrenzt wurde, da eine Vollauslastung negative Auswirkungen auf den BitTorrent Verkehr haben könnte.

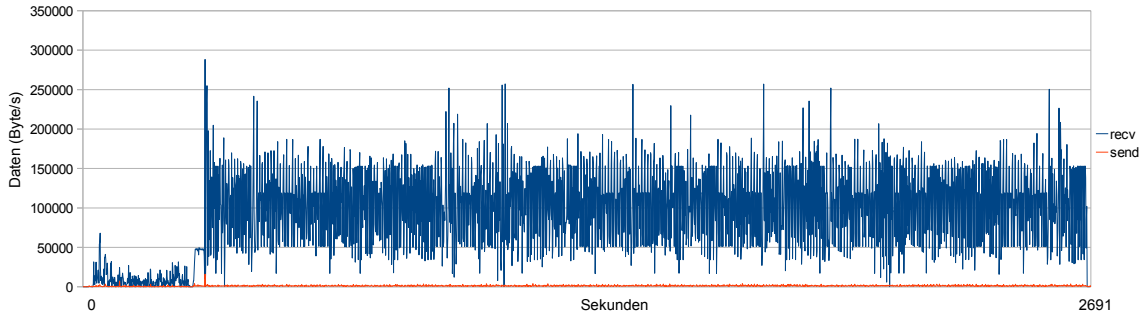


Abbildung 6.2: Übertragungsgeschwindigkeiten während des Torrent Downloads von 'Route66.mp4'

Diese erklärt die deutlich gesteigerte Testdauer auf 2691 Sekunden. Die enorme Fluktuation der Empfangsgeschwindigkeit (blaue obere Linie) und der langsame Beginn ist allerdings damit nicht zu erklären. Eine genauere Interpretation folgt in Abschnitt 6.4.

6.2.2 BitTorrent Verkehr im LAN

Als nächstes wird dieser Test in einem Kabel gebundenen lokalen Netz, getrennt vom Internet, wiederholt, um den BitTorrent Verkehr zu Vergleichszwecken ohne äussere Einflüsse messen zu können. Die Testspezifikation ändert sich hierbei nur um die nun lokalen IP Adressen der Agenten und wird daher nicht erneut präsentiert. Eine Torrent Datei mit aktualisierter Tracker Adresse muss allerdings erneut erzeugt und verteilt werden.

Ansonsten verläuft der Test mit denselben Einstellungen mit ebenfalls begrenzter Downstream Geschwindigkeit. Abbildung 6.3 zeigt den Verlauf im LAN bei einer Testdauer von 2426 Sekunden.

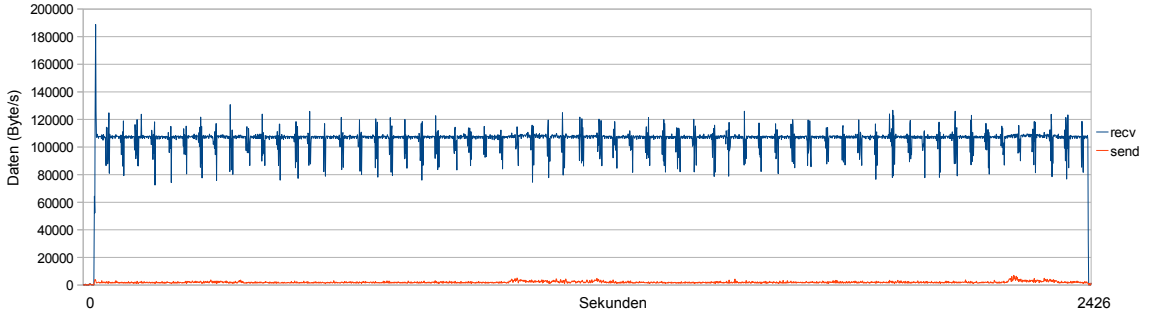


Abbildung 6.3: Übertragungsgeschwindigkeiten während des Torrent Downloads von 'Route66.mp4' im LAN

6.3 Routenermittlung nach Anstieg der RTT

Dieser Test erfolgt wieder über das Internet und führt zu Beginn einen Traceroute zur initialen Routenermittlung durch, prüft anschließend über Ping die Round Trip Time und löst ab einem bestimmten RTT Wert weitere Traceroutes aus. Der genaue Ablauf dieses Tests wurde bereits in Abschnitt 4.4 ausführlich beschrieben. Die Spezifikation des Tests ist daher auch der aus Listing 4.17 ähnlich, weist aber ein paar Unterschiede auf und wird zum besseren Verständnis in Listing 6.3 noch einmal aufgeführt.

Listing 6.3: Spezifikation des pathChange Tests zur Routenermittlung nach Anstieg der RTT

```
1  agent {
2      name = Agent1
3      address = "80.187.88.218"
4  }
5
6  agent {
7      name = Agent2
8      address = "91.67.131.173"
9  }
10
11 trigger {
12     name = oneTrace
13     description = "do one traceroute"
14     probes = traceroute
15 }
16
17 trigger {
18     name = max_RTT
19     description = "Triggers when RTT gets to high"
20     probes = ping
21     metric = >
22     type = RTT
23     behavior = 10
24 }
25
26 probe {
27     name = traceroute
28     description = "Traceroute"
```

```
29      repeat = 1
30      generated {
31          ICMPv4 {
32              TRACEROUTE
33          }
34      }
35  }
36
37  probe {
38      name = ping
39      description = "Round Trip Time"
40      repeat = 1000
41      generated {
42          ICMPv4 {
43              PING
44          }
45      }
46  }
47
48
49  test {
50      name = pathChange
51      description = "traceroute after ping TTL change"
52      agents = Agent1 -> Agent2
53      trigger = oneTrace {
54          value = true
55          action {
56              trigger = max_RTT {
57                  value = "170"
58                  action {
59                      probes = traceroute
60                  }
61              }
62          }
63      }
64  }
```

Wichtig für diesen Feldversuch sind die spezifizierten Werte für das Triggerkriterium, das einen Traceroute nur bei einer gemessenen RTT von über 170 ms auslöst (Zeile 57), die Anzahl an zu sendenden Pings (Zeile 40) und die maximale Anzahl an getriggerten Traceroutes (Zeile 23), die in diesem Fall mit 1000 Ping Wiederholungen und maximal 10 auslösenden Triggern maßgeblich über die Länge des Tests entscheiden.

Die gemessenen RTTs zeigt Abbildung 6.4. Die rote horizontale Linie repräsentiert den Schwellenwert von 170 ms für das Auslösen eines Traceroutes. Insgesamt acht mal wird dieser überschritten, was zusammen mit dem initialen Traceroute zu neun Routenmessungen im Ergebnis führt.

Die während dieses Tests protokollierten ersten beiden Traceroute Ausgaben zeigt Listing 6.4. Die Messung über das UMTS Netz weist leider eine Menge Routenpunkte auf, die ICMP Anfragen nicht beantworten und daher nicht ermittelt werden können.

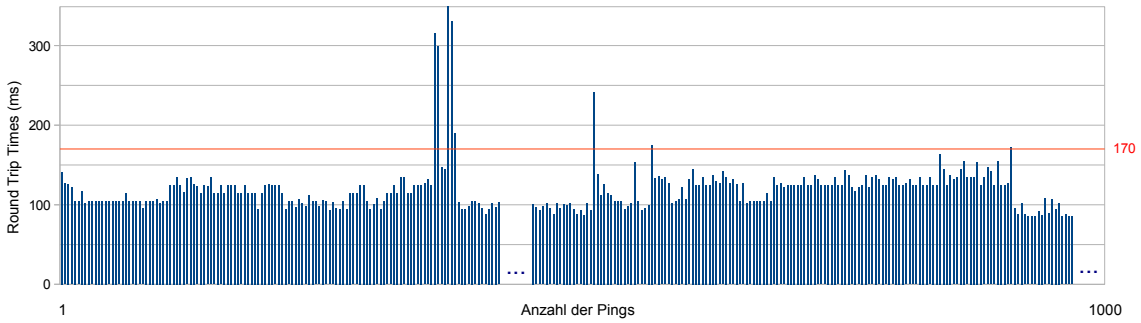


Abbildung 6.4: Round Trip Times als Kriterium für weitere Traceroutes

Listing 6.4: Protokollierte Traceroute Ausgabe des pathChange Tests

```
1  traceroute to 91.67.131.173 (91.67.131.173), 30 hops max, 60 byte packets
2   1  * * *
3   2  * * *
4   3  * * *
5   4  * * *
6   5  * * *
7   6  * * *
8   7  * * *
9   8  * * *
10  9  194.25.209.113 (194.25.209.113)  201.612 ms  269.064 ms  259.024 ms
11  10  b-ea6-i.B.DE.NET.DTAG.DE (62.154.47.69)  268.995 ms b-ea6-i.B.DE.NET.DTAG
        .DE (62.154.47.66)  268.966 ms b-ea6-i.B.DE.NET.DTAG.DE (62.154.47.69)
        291.290 ms
12  11  194.25.210.46 (194.25.210.46)  109.115 ms  109.043 ms  119.650 ms
13  12  ae-34-52.ebr2.Berlin1.Level3.net (4.69.146.153)  112.399 ms  149.864 ms
        149.816 ms
14  13  ae-4-4.car2.Munich1.Level3.net (4.69.134.5)  162.297 ms  140.050 ms
        140.514 ms
15  14  ae-11-11.car1.Munich1.Level3.net (4.69.133.253)  190.448 ms  179.930 ms
        159.918 ms
16  15  62.140.24.58 (62.140.24.58)  131.420 ms  131.362 ms  139.062 ms
17  16  83-169-128-210.static.superkabel.de (83.169.128.210)  139.020 ms *
        109.940 ms
18  17  88-134-196-210-dynip.superkabel.de (88.134.196.210)  109.895 ms  119.080
        ms  100.038 ms
19  18  83-169-128-201.static.superkabel.de (83.169.128.201)  120.846 ms  120.767
         ms  120.724 ms
20  19  83-169-135-195.static.superkabel.de (83.169.135.195)  119.066 ms  119.024
         ms  120.715 ms
21  20  * * *
22  21  * * *
23  22  * * *
24  23  * * *
25  24  * * *
26  25  * * *
27  26  * * *
28  27  * * *
29  28  * * *
30  29  * * *
31  30  * * *
```

```
32 traceroute to 91.67.131.173 (91.67.131.173), 30 hops max, 60 byte packets
33  1  * * *
34  2  * * *
35  3  * * *
36  4  * * *
37  5  * * *
38  6  * * *
39  7  * * *
40  8  * * *
41  9  194.25.209.113 (194.25.209.113)  186.572 ms  206.511 ms  216.483 ms
42 10  b-ea6-i.B.DE.NET.DTAG.DE (62.154.47.66)  216.447 ms  228.887 ms  248.843
       ms
43 11  194.25.210.46 (194.25.210.46)  248.815 ms  82.282 ms *
44 12  ae-34-52.ebr2.Berlin1.Level3.net (4.69.146.153)  92.391 ms  82.369 ms
       92.238 ms
45 13  ae-4-4.car2.Munich1.Level3.net (4.69.134.5)  109.904 ms  99.903 ms
       99.856 ms
46 14  ae-11-11.car1.Munich1.Level3.net (4.69.133.253)  89.914 ms  99.877 ms
       99.834 ms
47 15  62.140.24.58 (62.140.24.58)  98.984 ms  107.469 ms  97.472 ms
48 16  83-169-128-210.static.superkabel.de (83.169.128.210)  287.409 ms  289.927
        ms  259.906 ms
49 17  88-134-196-210-dynip.superkabel.de (88.134.196.210)  259.863 ms  250.945
       ms  222.363 ms
50 18  83-169-128-201.static.superkabel.de (83.169.128.201)  222.318 ms  109.910
        ms  97.466 ms
51 19  83-169-135-195.static.superkabel.de (83.169.135.195)  99.906 ms  99.863
       ms  99.825 ms
52 20  * * *
53 21  * * *
54 22  * * *
55 23  * * *
56 24  * * *
57 25  * * *
58 26  * * *
59 27  * * *
60 28  * * *
61 29  * * *
62 30  * * *
```

Nachfolgend werden die Ergebnisse der Feldversuche interpretiert.

6.4 Interpretation

Die Feldversuche aus Abschnitt 6.1 und 6.2 führen beide einen Datentransfer durch und werden zur Interpretation gemeinsam betrachtet. Der Traceroute Test aus Abschnitt 6.3 wird dagegen isoliert diskutiert.

6.4.1 Versuche zum Datentransfer

Bei den insgesamt drei Versuchen zum Datentransfer sind insbesondere die starken Schwankungen und der langsame Beginn im BitTorrent Test über das Internet (Abbildung 6.2) zu interpretieren.

Es liegt nahe, die Fluktuation durch die drahtlose Anbindung durch UMTS zu erklären.

Allerdings zeigte der HTTP Test in Abschnitt 6.1 trotz gleicher Ausgangslage (Agenten, Internetverbindung, Transferrichtung) keineswegs solche Schwankungen, im Gegenteil, er weist über die gesamte Testdauer von immerhin über 16 Minuten eine bemerkenswert konstante Übertragungsgeschwindigkeit auf.

Um BitTorrent Verkehr isoliert zu betrachen, wurde der BitTorrent Test noch einmal in einem LAN wiederholt. Dabei fällt sofort auf, dass weder ein langsamer Beginn noch eine starke Fluktuation auftrat (Abbildung 6.3). Etwas über dem gesetzten Limit verläuft die Übertragungsrate, abgesehen von regelmäßigen Ausschlägen sehr konstant bei etwa 106 KiB/s. Die Intervall artigen Ausschläge erklären sich durch das für BitTorrent Übertragungen übliche Senden in kleinen Teilen, in die die Datei 'Route66.mp4' automatisch zerlegt wurde.

Als Folge der Messungen wurde gezeigt, dass weder BitTorrent noch UMTS alleine für die gemessenen Schwankungen verantwortlich sind. Woher sie genau resultieren ist damit allerdings noch nicht geklärt. Wesentlich verlangsamt wurde der Verkehr jedenfalls nicht, da er über UMTS bei einer Gesamtdauer von fast 45 Minuten nur etwas über 4 Minuten länger benötigte als der Test im LAN und sich dies vor allem durch den langsamen Beginn erklären lässt.

6.4.2 Versuch zur Routenermittlung

Listing 6.4 zeigt nur die ersten beiden Traceroute Ausgaben, da sich leider keine Änderung messen lies. Bis auf Knoten Nummer 10, der manchmal noch innerhalb eines Traceroutes mehrere IP Adressen anzeigt (wie beim initialen Traceroute der Fall in Zeile 11) und damit vermutlich Lastenausgleich betreibt, sind keine Änderungen zu bemerken.

Routenänderungen sind bei dieser Strecke innerhalb Deutschlands auch vermutlich relativ selten und würden einen sehr langen Test benötigen. Was dieser Test allerdings schön demonstriert, ist eine ereignisbasierte Reaktion nach Anforderung 6.

6.5 Evaluation des Konzepts und des Prototypen

Wie die Feldversuche zeigen, lassen sich mit dem noch recht einfachen Prototypen bereits ein paar Tests durchführen, die durchaus zu interessanten Ergebnissen führen können. Der Prototyp erfüllt bereits viele Anforderungen aus Kapitel 2, andere sind bisher nur konzeptionell ausgearbeitet und müssen noch in den Prototypen integriert werden. Die folgenden Punkte listen die neun Anforderungen noch einmal auf und bewerten deren Umsetzung:

1. **Variable Testmöglichkeiten** (Anforderung 1)

   Über die Eingabesprache spezifizieren beliebige Probes entweder Protokolle der einzelnen Schichten nach dem OSI-Referenzmodell oder aufgezeichneten Verkehr. Da dies sehr viele Varianten ermöglicht, sind bisher nur einige in der Sprachdefinition erlaubt und im Prototyp umgesetzt. Das Konzept erlaubt hier allerdings problemlos Erweiterungen. Die einzige Einschränkung in der Variabilität erfolgt aus der Entscheidung für aktive Messungen, so dass realer produktiv Verkehr zumindest nicht live getestet werden kann.

2. **Vergleichbarkeit** (Anforderung 2)

   Die zu Beginn jedes Tests zugewiesene TestID erlaubt die Zuordnung der Ergebnisse zu den jeweiligen Tests. Die Timer Komponente wurde so entworfen, dass Tests nur

sequenziell und nie gleichzeitig ausgeführt werden. Auch wenn es konzeptionell möglich ist einen Agenten für mehrere Probes gleichzeitig einzusetzen, bleibt jede Wiederholung dieses Tests vergleichbar, da der Agent immer wieder die gleichen Probes nutzt. Die einzige Weise, wie Tests verfälscht werden können, ist über einen zweiten Server, der gleichzeitig Tests mit denselben Agenten startet. Dagegen sind die Agenten im Prototyp bisher nicht abgesichert. Dieser Fall ist aber im Konzept auch nicht vorgesehen.

3. **Erweiterbarkeit** (Anforderung 3)

   Der Detektor ist so entworfen worden, dass er keine Einschränkungen bezüglich der Anzahl der Agenten oder der unterstützten Probes aufweist. Die durch die Sprachdefinition erfolgte Trennung von Spezifikation und Implementierung lässt hier prinzipiell beliebige Erweiterungen zu. Der jetzige Stand des Prototypen benötigt für fundierte Nachweise von Netzneutralitätsverstößen noch einige solche Erweiterungen. Beispielsweise sammelt das Protokoll bereits jetzt alle Daten auf dem Server und bietet daher alle Voraussetzungen für eine Erweiterung um ein Auswertungsmodul.

4. **Skalierbarkeit** (Anforderung 4)

   Agenten können einfach in der Testspezifikation hinzugefügt werden. Der Server benötigt keine weiteren Informationen und kann sofort mit einem neuen Agenten kommunizieren. Einerseits erleichtert dies den variablen Einsatz der verfügbaren Agenten, andererseits muss der Nutzer das Wissen über die vorhandenen Agenten selbst verwalten.

5. **Zeitliche Steuerung** (Anforderung 5)

   Eine zeitliche Steuerung kann in der Eingabesprache spezifiziert werden und wird durch die Timerkomponente umgesetzt. Es werden Anfangs- und Endzeitpunkte, Intervalle und die Angabe einer Laufzeit unterstützt.

6. **Ereignisbasierte Reaktion** (Anforderung 6)

   Das Triggerkonzept dient genau dieser Anforderung. Es erlaubt auf ein vorher spezifiziertes Ereignis u.a. mit weiteren Probes, Triggern oder vollständigen Tests zu reagieren. Angewendet zeigt dies der Feldversuch aus Abschnitt 6.3.

7. **Integration externer Programme** (Anforderung 7)

   Wie in Abschnitt 5.4.2 beschrieben, nutzt der Prototyp eine Reihe externer Programme zur Testdurchführung. Dabei werden sowohl externe Bibliotheken direkt eingebunden, wie im Fall des Webservers, als auch über die Shell externe Binaries ausgeführt. Generell ist das Einbinden von externen Bibliotheken vorzuziehen, da dadurch auf den Agenten keine zusätzliche Software installiert sein muss. Allerdings ist dies aufwendiger und nicht immer möglich.

8. **Einfache Bedienung** (Anforderung 8)

   Die wohldefinierte Spezifikation von Tests über die Eingabesprache und das Verzichten auf eine Interaktion mit dem Benutzer während der Tests ermöglicht eine einfache Bedienung. Dabei muss der Benutzer keine fundierten Kenntnisse über die zum Einsatz kommenden Werkzeuge besitzen. Er spezifiziert nur einen Verkehrstyp, dessen Erzeugung die Implementierung übernimmt.

9. **Sparsamer Umgang mit Ressourcen** (Anforderung 9)

   Die Konzeption unterstützt dank der Server Agenten Architektur und dem eigenen sehr schlankem Protokoll einen sparsamen Umgang mit Ressourcen. Da allerdings die Agenten trotzdem noch sehr viele Aufgaben auszuführen haben, können Agenten nicht beliebig klein und anspruchslos gehalten werden. Viel lässt sich allerdings durch eine gute Wahl der externen Werkzeuge erreichen, wie der im Prototypen verwendete minimale Webserver beweist.

Die Anforderungen werden also durch die Konzeption mit nur minimalen Einschränkungen aufgrund der Festlegung auf aktives Testen erfüllt. Der Prototyp hingegen muss noch an vielen Stellen verbessert und ausgebaut werden, um alle möglichen Testspezifikationen umsetzen und fundierte Aussagen zum Verhalten von Netzbetreibern liefern zu können.

6.6 Ansätze zur Weiterentwicklung

Wie die Evaluation in Abschnitt 6.5 andeutete, besteht noch an einigen Stellen Bedarf für Weiterentwicklungen:

- Das gesamte Detektorsystem ist nicht ausreichend gegen Fremdeinwirkung geschützt. Die für die Tests notwendige Platzierung der Agentensysteme ohne vorgeschalteter Firewall birgt in dieser Hinsicht große Gefahren und diktiert weitreichende Schutzmaßnahmen.

- Die Eingabesprache sorgt zwar bereits für eine relativ einfache Bedienung, für einen fertigen Detektor wäre allerdings eine grafische Oberfläche wünschenswert.

- Eine automatisierte Verwaltung der vorhandenen Agenten, die dem Nutzer die verfügbaren und einsatzbereiten Agenten auflistet, würde falsche Agenten Spezifikationen verhindern und die Eingabe weiter vereinfachen.

- Ein Ausbau des Protokolls um benötigte Dateien, wie z.B. Torrents, vor den Tests automatisch zu verteilen würde bei vielen Tests die Vorbereitungen vereinfachen.

- Der derzeitige Prototyp verfügt noch über keinerlei vom System unterstützte Möglichkeiten der Auswertung. Diese sind jedoch von entscheidender Bedeutung für fundierte Nachweise von Netzneutralitätsverstößen. Insbesondere die Unterscheidung von Manipulationen und Fehlern bietet hier weitere interessante Forschungsmöglichkeiten. Ein automatisiertes Erstellen von Diagrammen z.B. über Cacti [Grob] würde dem Nutzer ausserdem eine Interpretation der Ergebnisse stark erleichtern.

- Die Triggerkriterien müssen derzeit vor dem Test spezifiziert werden und erfordern daher vorab Kenntnisse über das Verhalten eines Tests. Sinnvoll wäre die Möglichkeit, Kriterien aus beobachteten Messwerten abzuleiten, um z.B. 10% über Durchschnitt spezifizieren zu können.

- Sehr wichtiger Punkt ist der Ausbau der möglichen Probes durch breitere Nutzung der Werkzeuge und dem Einsatz zusätzlicher Werkzeuge. Das bisher eingesetzte Werkzeug

traceroute z.B. ist bekannt für seine Schwächen in heutigen Netztopologien und könnte relativ einfach durch Paris Traceroute ersetzt werden [ACO+06], das verbesserte Ergebnisse bei der Ermittlung von Routen verspricht.

Ein weiteres Werkzeug, das durch einen Einsatz in dem Detektor interessante Ergebnisse erwarten lässt ist Pathchar, das Charakteristiken wie Bandbreite, Verzögerung und Verlustrate von Internet Pfaden ermittelt [Jac97].

- Allgemeine vermutlich interessante Verkehrstypen im Umfeld der Netzneutralität sind Voice over IP Verkehr (Skype im besonderen), jede Art von verschlüsseltem Verkehr, Video und Audio Streams sowie Peer to Peer Verkehr. Dazu sind Spezifikationen geeigneter Tests zu entwickeln und für Tests in der Implementierung zu realisieren.
- Im Umfeld des Fehlermanagements wurden Verfahren zur automatisierten Auswahl von Probestationen entwickelt, die auf einen möglichen Einsatz in dem Detektor für die Auswahl von Agenten überprüft werden könnten.
- Sind erst einmal einige Anomalien zu beobachten und mit hoher Wahrscheinlichkeit auf beabsichtigte Manipulationen zurückzuführen, bietet die Lokalisierung des dafür verantwortlichen Netzbetreibers ebenfalls eine interessante Herausforderung.

Im Laufe weiterer Versuche wird diese Liste vermutlich noch wachsen. Einige Punkte, wie die Implementierung eines Auswertungmoduls und die Unterstützung weiterer Probes sind essentiell für einen produktiven Einsatz des Detektors und sollten zügig umgesetzt werden. Das erarbeitete Konzept des Detektors unterstützt hier noch viele Möglichkeiten.

Das nächste Kapitel gibt eine Zusammenfassung über diese Arbeit, zieht ein Fazit und gibt einen Ausblick für die weitere Entwicklung des Detektors.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Detektor zum Erkennen von Verletzungen der Netzneutralität konzipiert, entwickelt und als Prototyp implementiert. Dazu wurden Anforderungen an den Detektor gestellt, ein geeignetes Messverfahren identifiziert und nach der Betrachtung existierender Lösungen ein eigenes, für diesen Zweck optimiertes Konzept entwickelt. Dies umfasst eine passende Server Agenten Architektur, ein eigenes Protokoll und eine eigene Sprache zur Spezifizierung von Tests. Im weiteren wurden diese Konzepte konkretisiert, in einem Prototypen umgesetzt und auch mit Hilfe von Feldversuchen gegen die Anforderungen evaluiert. Abschließend werden zahlreiche Anreize zu Weiterentwicklungen gegeben.

Die Notwendigkeit eines solchen Werkzeuges resultiert vor allem aus der Informationspolitik der Netzbetreiber, die über ihr Verkehrsmanagement keine Auskünfte geben wollen, sowie den starken Auswirkungen, die Netzneutralitätsverletzungen auf die betroffenen Verbindungen haben können.

Ausgehend von einem realen Szenario, in dem sich das Unternehmen Amadeus durch nicht neutrales Verhalten der Netzbetreiber in ihrem Geschäft bedroht sieht, wurden insgesamt neun Anforderungen an den Detektor gestellt und dessen Konzeption danach ausgelegt.

Der derzeitige Stand der Wissenschaft und Technik weist noch keine Werkzeuge auf, die diese Anforderungen erfüllen. Der hier entwickelte Detektor unterscheidet sich zu ähnlichen Werkzeugen vor allem durch seinen erweiterbaren und generischen Ansatz ohne Einschränkungen bezüglich des zu testenden Verkehrstyps und der Möglichkeit komplexe, zusammengesetzte und ereignisbasierte Messungen durchführen zu können. Diese sind insbesonders für die Differenzierung zwischen unerwünschten Fehlern und beabsichtigten Manipulationen erforderlich. Dieser entscheidende Punkt wird von anderen Arbeiten bisher gar nicht adressiert.

Ausgangspunkt der Konzeption ist die klare Spezifikation von Tests zu Netzneutralitätsmessungen, um, losgelöst von der Implementierung, Tests eindeutig beschreiben zu können. Dazu wurde eine Eingabesprache entwickelt, die in ihrer Ausdrucksmächtigkeit den Problemraum von Netzneutralitätsverstößen abdeckt. Ein Test wird dabei in einzelne Elemente mit Eigenschaften zerlegt, welche in der durch die Eingabesprache definierten Form festgelegt werden. Diese Elemente sind Agenten zur Spezifikation von Messpunkten, Probes zur Charakterisierung von Verkehrstypen, Trigger für ereignisbasierte Reaktionen und Zeit.

Um eine möglichst natürliche und übersichtliche Art der Eingabe zu erreichen, wird die Eingabesprache um Referenzen, verkürzende Schreibweisen und Mengenbildung erweitert. Referenzen ersparen unnötige Wiederholungen bereits in der Sprache formulierter Ausdrücke und helfen dadurch die Konsistenz zu bewahren. Auch die alternativen Schreibweisen und die Mengenbildung, die Gruppierung von Elementen ermöglicht, vereinfachen die Eingabe weiter. Die Eingabesprache trennt die Test Spezifikation von ihrer Umsetzung und erlaubt die Beschreibung beliebiger Tests noch vor deren Implementierung und ohne spezifischen technischen Bezug auf bestimmte Werkzeuge. Damit wird unabhängig von dem in der Implementierung eingesetzten Werkzeug ein Verkehrstyp spezifiziert und die zentrale Anforderung der Erweiterbarkeit bedient.

Auch die für den Detektor gewählte Architektur wurde mit Blick auf deren Erweiterbarkeit als Server Agenten System konzipiert und entwickelt. Die Tests verlaufen nach dem aktiv Probing Prinzip, d.h. der für die Messungen benötigte Verkehr wird künstlich generiert und basiert nicht auf produktiv Verkehr. Dies kann als Einschränkung gewertet werden, erlaubt aber im Allgemeinen flexiblere Testmöglichkeiten. Die Einschränkung auf generierten Verkehr lässt sich sogar über die in der Eingabesprache vorgesehene Möglichkeit, auch aufgezeichneten Verkehr für Messungen heranzuziehen, umgehen und erlaubt dadurch beispielsweise trotzdem Tests auf produktiv Verkehr.

Die Kommunikation zwischen Server und Agenten verläuft nach einem eigens konzipierten Protokoll, das alle Arten von Testspezifikationen und beliebig viele Ergebnisse übermitteln kann und damit unabhängig von den eigentlichen Tests ist.

Im Entwurf wird die Eingabe über zwei Parserschritte zu einer homogenen, von unterschiedlichen Eingabeschreibweisen unabhängigen, Abstract Syntax Tree Datenstruktur für die weitere Verwendung in der Implementierung konvertiert. Jeder Test wird in dem AST durch einen Teilbaum vollständig repräsentiert und als serialisierter String Tree übertragen. Die Agenten parsen diesen String Tree wieder in den AST zurück und verfügen somit stets über alle spezifizierten Informationen dieses Tests. Ausserdem werden die einzelnen Komponenten des Detektors definiert und in der Architektur dem Server oder den Agenten zugeordnet.

Zum Schluss wurde ein Prototyp implementiert, der basierend auf dem Entwurf einen funktionsfähigen Detektor mit der Fähigkeit ausgewählte Tests durchzuführen realisiert und zur Evaluation Feldversuche durchführt. Die Versuche dienten einmal dem Vergleich eines Datentransfers über die beiden unterschiedlichen Protokolle HTTP und BitTorrent und einmal der automatischen Überprüfung der Route nach der Überschreitung bestimmter Round Trip Times. Dabei wurde gezeigt, dass der Prototyp in der Lage ist, interessante Verkehrsbeobachtungen zu liefern, und geeignet ist Netzneutralitätsmessungen durchzuführen.

Die neun Anforderungen werden im Rahmen der Konzeption erfüllt, die Implementierung lässt allerdings Weiterentwicklungen und Verbesserungen zu. Besonderes Augenmerk ist dabei auf die Auswertung der Tests zu legen, um verdächtigen Verkehr fundiert von Fehlern differenzieren zu können und so eine beabsichtigte Manipulation nachweisen zu können. Dazu ist die Unterstützung weiterer Probes und der Einsatz von deutlich mehr Agenten Voraussetzung. Eventuell lassen sich zu diesem Zweck die Agenten des Detektors in Zukunft auf die in Abschnitt 3.3.2 beschriebenen Probes des RIPE ATLAS Projekts installieren.

Der in dieser Arbeit entwickelte Detektor bietet ausgewählte Tests, die sich zu Netzneutralitätsmessungen eignen, und das entwickelte Konzept birgt für die zukünftige Erkennung von Verletzungen der Netzneutralität ausserordentliches Potenzial.

Abbildungsverzeichnis

Listings

Literaturverzeichnis

[AAC+03] Akyildiz, I. F., T. Anjali, L. Chen, J. C. de Oliveira, C. Scoglio, A. Sciuto, J. A. Smith und G. Uhl: *A new traffic engineering manager for DiffServ/MPLS networks: design and implementation on an IP QoS Testbed.* Computer Communications, 26(4):388 – 403, 2003. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.112.789&rep=rep1&type=pdf [online am 29. Mai 2011].

[ACE+02] Awduche, D., A. Chiu, A. Elwalid, I. Widjaja und X. Xiao: *Overview and Principles of Internet Traffic Engineering.* RFC 3272 (Informational), Mai 2002. Aktualisiert by RFC 5462; http://www.ietf.org/rfc/rfc3272.txt [online am 28. Mai 2011].

[ACO+06] Augustin, Brice, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien und Renata Teixeira: *Avoiding traceroute anomalies with Paris traceroute.* In: *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, Seiten 153–158, New York, NY, USA, 2006. ACM. http://conferences.sigcomm.org/imc/2006/papers/p15-augustin.pdf [online am 28. Mai 2011].

[Bar09] Barczok, Achim: *T-Mobile will Skypes iPhone-App blockieren*, März 2009. http://www.heise.de/mobil/meldung/T-Mobile-will-Skypes-iPhone-App-blockieren-210347.html [online am 13. April 2011].

[BCF94] Bouloutas, A. T., S. Calo und A. Finkel: *Alarm Correlation and Fault Identification in Communication Networks.* IEEE Transactions on Communications, Vol 42, April 1994. http://www.ece.cmu.edu/~spertet/papers/finkel1994_alarms.pdf [online am 27. Mai 2011].

[BF06] Baudry, Benoit und Franck Fleurey: *Improving Test Suites for Efficient Fault Localization.* 2006. http://www.irisa.fr/triskell/perso_pro/yletraon/baudry06a.pdf [online am 28. Mai 2011].

[Bie10] Biermann, Kai: *Telekom träumt von Google-Gebühr*, März 2010. http://www.zeit.de/digital/internet/2010-03/telekom-google-netzneutralitaet [online am 14. April 2011].

[Bil10] Bilton, Nick: *One on One: Tim Wu, Author of 'The Master Switch'*, November 2010. http://bits.blogs.nytimes.com/2010/11/14/one-on-one-tim-wu-author-of-the-master-switch/ [online am 7. April 2011].

[Bil11] Bilton, Nick: *Tim Wu, Creator of the Term 'Net Neutrality', Joins the Federal Government*, Februar 2011. http://bits.blogs.nytimes.com/2011/02/09/tim-wu-creator-of-net-neutrality-term-joins-f-t-c/ [online am 7. April 2011].

[BPSM+08] Bray, Tim, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler und François Yergeau: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, November 2008. http://www.w3.org/TR/REC-xml/#idref [online am 9. Mai 2011].

[BRM01] Brodie, M., I. Rish und S. Ma: *Optimizing Probe Selection for Fault Localization*. Distributed Systems Operation and Management, Oktober 2001. http://www.simpleweb.org/ifip/Conferences/DSOM/2001/DSOM2001/proceedings/S3-2.pdf [online am 28. Mai 2011].

[BRM+02] Brodie, M., I. Rish, S. Ma, G. Grabarnik und N. Odintsova: *Active probing*. 2002. http://www.research.ibm.com/people/r/rish/papers/RC22817.pdf [online am 28. Mai 2011].

[Con09] Connors, Devin: *Canadian ISP's Intentionally Make P2P Slow*, Januar 2009. http://www.tomsguide.com/us/P2P-Internet-Torrents-Comcast,news-3340.html [online am 3. Mai 2011].

[Cot01] Cottrell, Les: *Passive vs. Active Monitoring*, März 2001. http://www.slac.stanford.edu/comp/net/wan-mon/passive-vs-active.html [online am 6. Mai 2011].

[Cro06] Crowcroft, Jon: *Net Neutrality: The Technical Side of the Debate: A White Paper*. International Journal of Communication, Juni 2006. http://ijoc.org/ojs/index.php/ijoc/article/download/159/84 [online am 27. April 2011].

[DeS11] DeSanti, Susan S.: *About the Office of Policy Planning*, April 2011. http://www.ftc.gov/opp/about.shtm [online am 7. April 2011].

[DFC05] Donnet, B., T. Friedman und M. Crovella: *Improved algorithms for network topology discovery*. Passive and Active Network Measurement, Seiten 149–162, 2005. http://www.cs.bu.edu/faculty/crovella/paper-archive/pam05-tr-at-home.pdf [online am 28. Mai 2011].

[Dow99] Downey, A.B.: *Using pathchar to estimate Internet link characteristics*. ACM SIGCOMM Computer Communication Review, 29(4):250, 1999. http://nslab.ee.ntu.edu.tw/courses/summer00/topology/path-char.ps [online am 28. Mai 2011].

[DSK+07] Dorgan, Byron L., Olympia J. Snowe, John F. Kerry, Barabara Boxer, Tom Harkin, Patrick J. Leahy, Hillary Rodham Clinton, Barack Obama, Ron Wyden, Christopher J. Dodd, Richard Durbin und Bernard Sanders: *Internet Freedom Preservation Act (Introduced in Senate - IS)*, September 2007. http://thomas.loc.gov/cgi-bin/query/z?c110:S.215: [online am 29. Mai 2011].

[Dud06] Dudman, Jane: *Trouble on the line.* The Guardian, April 2006. http://www.guardian.co.uk/technology/2006/apr/06/voip.telephony [online am 3. Mai 2011].

[End] Endling, Dirk: *opentracker - An open and free bittorrent tracker.* http://erdgeist.org/arts/software/opentracker/ [online am 10. Juni 2011].

[Eul05] Euler, Stephan: *Netzwerk-Basiswissen, Teil 1*, Januar 2005. http://www.tecchannel.de/netzwerk/lan/402420/netzwerk_basiswissen_teil_1/index9.html [online am 24. Mai 2011].

[FTD10] *Schnelligkeit im Netz kostet.* Financial Times Deutschland, Dezember 2010. http://www.ftd.de/it-medien/medien-internet/:netzneutralitaet-schnelligkeit-im-netz-kostet/50208074.html [online am 24. Mai 2011].

[GDH+08] Gummadi, Krishna P., Marcel Dischinger, Andreas Haeberlen, Alan Mislove und Stefan Saroiu: *Glasnost: Bringing Transparency to the Internet*, April 2008. http://broadband.mpi-sws.org/transparency/ [online am 29. Mai 2011].

[Gmb10] GmbH, Akamai Technologies: *Akamai unterstützte 24 Fernsehsender in 65 Ländern weltweit erfolgreich bei der Online-Übertragung der Fußball-WM 2010*, Juli 2010. http://www.akamai.de/dl/Akamai_WorldCup_FINAL200710.pdf [online am 2. Mai 2011].

[Gop00] Gopal, R.: *Layered model for supporting fault isolation and recovery.* In: *NOMS 2000: IEEE/IFIP Network Operations and Management Symposium 'The Networked Planet: Management Beyond 2000'*, Seiten 729–742, 2000.

[Groa] Grothoff, Christian: *GNU libmicrohttpd.* http://www.gnu.org/software/libmicrohttpd/ [online am 10. Juni 2011].

[Grob] Group, The Cacti: *About Cacti.* http://www.cacti.net/index.php [online am 6. Juni 2011].

[GT00] Govindan, R. und H. Tangmunarunkit: *Heuristics for Internet map discovery.* 3:1371–1380, 2000. http://isi.edu/div7/publication_files/heuristics.pdf [online am 28. Mai 2011].

[Han] Hanson, Troy D.: *easily store and retrieve binary data in C.* http://tpl.sourceforge.net/ [online am 10. Juni 2011].

[Hon08] Honan, Mathew: *Inside Net Neutrality: Is your ISP filtering content?*, Februar 2008. http://www.macworld.com/article/132075/2008/02/netneutrality1.html [online am 8. April 2011].

[HPMc02] Huffaker, Bradley, Daniel Plummer, David Moore und k claffy: *Topology discovery by active probing.* 2002. http://www.caida.org/publications/papers/2002/SkitterOverview/skitter_overview.pdf [online am 28. Mai 2011].

[HZSL+08] He, Jiayue, Rui Zhang-Shen, Ying Li, Cheng-Yen Lee, Jennifer Rexford und Mung Chiang: *DaVinci: dynamically adaptive virtual networks for a customized internet*. In: *CoNEXT '08: 2008 ACM CoNEXT Conference*, Seiten 1–12, New York, NY, USA, 2008. ACM. `http://www2.ece.ohio-state.edu/~ji/1569139969.pdf` [online am 29. Mai 2011].

[IN11] Idle, Jim und Benjamin Niemann: *ANTLR3 Code Generation Targets*, März 2011. `http://www.antlr.org/wiki/display/ANTLR3/Code+Generation+Targets` [online am 24. Mai 2011].

[Jac97] Jacobson, Van: *pathchar - a tool to infer characteristics of internet paths*, 1997. `ftp://ftp.ee.lbl.gov/pathchar/msri-talk.pdf` [online am 6. Juni 2011].

[JM07] Johnson, Abigail und Paul Michelson: *Beyond Deep Packet Inspection: New chip first to perform completepacket inspection, at wire speeds to 20 GBPS and beyond*, Mai 2007. `http://www.cpacket.com/CPNewChip11May07FINAL.pdf` [online am 20. April 2011].

[Kan11] Kang, Cecilia: *Verizon challenges FCC rules on net neutrality*, Januar 2011. `http://voices.washingtonpost.com/posttech/2011/01/verizon_challenges_fcc_rules_o.html` [online am 14. April 2011].

[Kar10] Karrenberg, Daniel: *Active Measurements - A Small Probe*, August 2010. `http://labs.ripe.net/Members/dfk/a-small-probe-for-active-measurements` [online am 29. Mai 2011].

[Kes11] Kessler, Marc: *EU-Kommission stellt zwei Verfahren gegen Deutschland ein*, Mai 2011. `http://www.teltarif.de/eu-komission-verfahren-gegen-deutschland-eingestellt/news/42723.html` [online am 27. Mai 2011].

[Kre10a] Krebs, George: *Liveblogging the Vote on Rules Protecting the Open Internet*, Dezember 2010. `http://blog.openinternet.gov/?p=438` [online am 14. April 2011].

[Kre10b] Krempl, Stefan: *Geteiltes Echo auf US-Richtlinien zur Netzneutralität*, Dezember 2010. `http://www.heise.de/newsticker/meldung/Geteiltes-Echo-auf-US-Richtlinien-zur-Netzneutralitaet-1158334.html` [online am 14. April 2011].

[KWNP10] Kreibich, Christian, Nicholas Weaver, Boris Nechaev und Vern Paxson: *Netalyzr: Illuminating The Edge Network*. November 2010. `http://www.icir.org/christian/publications/2010-imc-netalyzr.pdf` [online am 29. Mai 2011].

[Lee06a] Lee, Tim Berners: *Net Neutrality: This is serious*, Juni 2006. `http://dig.csail.mit.edu/breadcrumbs/node/144` [online am 8. April 2011].

[Lee06b] Lee, Tim Berners: *Neutrality of the Net*, Mai 2006. `http://dig.csail.mit.edu/breadcrumbs/node/132` [online am 8. April 2011].

[Lin01] Linz, P.: *An introduction to formal languages and automata.* Jones and Bartlett Publishers, Inc, 4th Auflage, 2001. http://books.google.com/books?id=KOo4NpfTEAIC [online am 6. Mai 2011].

[LL00] Lemley, Mark A. und Lawrence Lessig: *The End of End-to-End: Preserving the Architecture of the Internet in the Broadband Era.* UC Berkeley Law & Econ Research Paper No. 2000-19, 2000. http://ssrn.com/paper=247737 [online am 7. April 2011].

[LM04] Li, Zhi und Prasant Mohapatra: *QRON: QoS-Aware Routing in Overlay Networks.* IEEE Journal on Selected Areas in Communications, 22(1):29–40, Januar 2004. http://140.115.52.150/98html/paper/pdf/QRON--QoS-awareroutinginoverlaynetworks.pdf [online am 29. Mai 2011].

[Mon08] Monitor.com, XiTi: *Internet Primetime in Europe*, März 2008. http://en.atinternet.com/pdf/surveys/en-US/InternetPrimetimeinEurope-March2008.pdf [online am 2. Mai 2011].

[NBBB98] Nichols, K., S. Blake, F. Baker und D. Black: *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers.* RFC 2474 (Standards Track), Dezember 1998. Aktualisiert durch RFC 3168 und 3260; http://tools.ietf.org/html/rfc2474 [online am 29. Mai 2011].

[NCC] NCC, RIPE: *Welcome to RIPE Atlas!* http://atlas.ripe.net/ [online am 11. Juni 2011].

[NS02] Natu, Maitreya und Adarshpal S. Sethi: *Probe Station Placement for Robust Monitoring of Networks.* 2002. http://www.research.ibm.com/people/r/rish/papers/RC22817.pdf [online am 28. Mai 2011].

[NS06] Natu, Maitreya und Adarshpal S. Sethi: *Active Probing Approach for Fault Localization in Computer Networks*, April 2006. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.93.8983&rep=rep1&type=pdf [online am 6. Mai 2011].

[NS07] Natu, Maitreya und Adarshpal S. Sethi: *Efficient probing techniques for fault diagnosis.* In: *International Conference on Internet Monitoring and Protection (ICIMP'07), Silicon.* Citeseer, 2007. http://www.cis.udel.edu/~natu/papers/icimp07.pdf [online am 28. Mai 2011].

[Par07] Parr, Terence: *The Definitive ANTLR Reference.* Mai 2007. http://uploadprj.googlecode.com/files/The.Definitive.ANTLR.Reference.pdf [online am 11. Mai 2011].

[Pas10] Passmore, David: *Net Neutrality in 2010: More Important than Ever*, April 2010. http://www.burtongroup.com/Research/PublicDocument.aspx?cid=1901.

[Pro] Project, Transmission: *Transmission.* http://www.transmissionbt.com/ [online am 10. Juni 2011].

[PV02] Pásztor, A. und D. Veitch: *Active probing using packet quartets*. In: *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, Seite 305. ACM, 2002. http://www.cubinlab.ee.unimelb.edu.au/~darryl/Publications/quartets_camera.pdf [online am 28. Mai 2011].

[SA] SA, Amadeus IT Group: *About Amadeus*. http://www.amadeus.com/amadeus/aboutamadeus?src=corporatehomepage [online am 20. April 2011].

[Sie11] Sietmann, Richard: *Schmalspur*. c't Magazin, März 2011. http://www.heise.de/ct/artikel/Schmalspur-1216729.html [online am 8. April 2011].

[Sin10] Singel, Ryan: *Mobile Carriers Dream of Charging per Page*. Mobile Carriers Dream of Charging per Page, Dezember 2010. http://www.wired.com/epicenter/2010/12/carriers-net-neutrality-tiers/all/1 [online am 24. Mai 2011].

[SMWA04] Spring, Neil, Ratul Mahajan, David Wetherall und Thomas Anderson: *Measuring ISP topologies with rocketfuel*. IEEE/ACM Trans. Netw., 12(1):2–16, 2004. http://pages.cs.wisc.edu/~akella/CS740/S08/740-Papers/SMW02.pdf [online am 28. Mai 2011].

[Spe02] Speta, James B.: *A Common Carrier Approach to Internet Interconnection*, 2002. http://www.law.indiana.edu/fclj/pubs/v54/no2/Speta.pdf [online am 7. April 2011].

[SRL98] Schulzrinne, H., A. Rao und R. Lanphier: *Real Time Streaming Protocol (RTSP) - RFC 2326*, April 1998. http://tools.ietf.org/html/rfc2326 [online am 26. Mai 2011].

[SS02] Steinder, M. und AS Sethi: *End-to-end Service Failure Diagnosis Using Belief Networks*. In: *Proc. Network Operation and Management Symposium*, Seiten 375–390. Citeseer, 2002. http://www.cis.udel.edu/~sethi/papers/02/noms02.pdf [online am 28. Mai 2011].

[SS04] Steinder, Malgorzata und Adarshpal S. Sethi: *A survey of fault localization techniques in computer networks*. Science of Computer Programming, Juli 2004. http://www.sciencedirect.com/science/article/pii/S0167642304000772 [online am 27. Mai 2011].

[SS10] Shaqvitt, Yuval und Yaron Singer: *Limitations and Possibilities of Path Trading between Autonomous Systems*, 2010. http://www.cs.berkeley.edu/~yaron/papers/PathTrading.pdf [online am 29. April 2011].

[Tan02] Tanenbaum, Andrew: *Computer Networks*. Prentice Hall Professional Technical Reference, 4th Auflage, 2002. http://www.de9.ime.eb.br/~mpribeiro/redes/PrenticeHall-ComputerNetworksTanenbaum4ed.pdf [online am 4. Mai 2011].

[Tau10] Tauber, Peter: *Netzneutralität: Voraussetzung für Innovationen*, März 2010. http://blogfraktion.de/2010/03/19/netzneutralitat-voraussetzung-fur-innovationen/ [online am 14. April 2011].

[Tel09] Telekom, Deutsche: *T-Mobile bietet Option für Internet-Telefonie an*, Juni 2009. http://www.telekom.com/dtag/cms/content/dt/de/595698?archivArticleID=673552 [online am 13. April 2011].

[TMFA09] Tariq, Mukarram Bin, Murtaza Motiwala, Nick Feamster und Mostafa Ammar: *Detecting network neutrality violations with causal inference.* In: *CoNEXT '09: Proceedings of the 5th international conference on Emerging networking experiments and technologies*, Seiten 289–300, New York, NY, USA, 2009. ACM. http://conferences.sigcomm.org/co-next/2009/papers/Tariq.pdf [online am 29. Mai 2011].

[udG10] Gesellschaft, Enquete-Kommission Internet und digitale: *Projektgruppe Netzneutralität*, August 2010. http://www.bundestag.de/internetenquete/dokumentation/2010/Netzneutralitaet/NEU_Mitglieder_Netzneutralit__t_05_08_2010.pdf [online am 14. April 2011].

[Uhl07] Uhls, Anna: *Digital Divide: The Issue of Net Neutrality*, April 2007. http://www.imprintmagazine.org/life_and_style/digital_divide_issue_net_neutrality [online am 12. April 2011].

[Wie] Wieërs, Dag: *Dstat: Versatile resource statistics tool.* http://dag.wieers.com/home-made/dstat/ [online am 10. Juni 2011].

[Wik] Wikipedia, the free encyclopedia: *Network neutrality.* http://en.wikipedia.org/wiki/Network_neutrality [online am 12. April 2011].

[Wro97] Wroclawski, J.: *The Use of RSVP with IETF Integrated Services.* RFC 2210 (Standards Track), September 1997. http://www.ietf.org/rfc/rfc2210 [online am 29. Mai 2011].

[Wu] Wu, Tim: *Network Neutrality FAQ.* http://timwu.org/network_neutrality.html [online am 12. April 2011].

[Wu11] Wu, Tim: *Biography*, April 2011. http://timwu.org/bio.html [online am 7. April 2011].

[WW97] Wolf, Misha und Charles Wicksteed: *Date and Time Formats*, September 1997. http://www.w3.org/TR/NOTE-datetime [online am 9. Mai 2011].

[YRK+08] Yuksel, Murat, Kadangode K. Ramakrishnan, Shivkumar Kalyanaraman, Joseph D. Houle und Rita Sadhvani: *Class-of-service in ip backbones: informing the network neutrality debate.* SIGMETRICS Perform. Eval. Rev., June 2008. http://portal.acm.org/citation.cfm?id=1375508 [online am 27. Mai 2011].

[ZMZ09] Zhang, Ying, Zhuoqing Morley Mao und Ming Zhang: *Detecting traffic differentiation in backbone ISPs with NetPolice.* In: *IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, Seiten 103–115, New York, NY, USA, 2009. ACM. http://www-personal.umich.edu/~wingying/Papers/imc09.pdf [online am 29. Mai 2011].

Anhang

1 ANTLR NNTester Lexer und Parser Grammatik

```
grammar NNTester;


options {
    language = C;
    output=AST;
}

tokens {
    DEF = '=' ;
    RIGHT   = '->';
    LEFT    = '<-';
    // Trees for collecting inputs together
    TestTree;
    AgentTree;
    AgentSetTree;
    ProbeTree;
    ProbeSetTree;
    PeriodTree;
    TriggerTree;

}

@includes{
    #include "NNTesterParserHelper.h"
}
/*------------------------------------------------------------------
 * PARSER RULES
 *------------------------------------------------------------------*/
start
    :
     (COMMENT | test | agent | agentset | probe | probeset | period | trigger
         ) *
    ->
    ^(AgentTree agent*)
    ^(AgentSetTree agentset*)
    ^(ProbeTree probe*)
    ^(ProbeSetTree probeset*)
    ^(PeriodTree period*)
    ^(TriggerTree trigger*)
    ^(TestTree test*)  // Test needs to be at the end!
    COMMENT*
     ;

test    : TEST! '{'! name^ description agent_refs (probe_refs | trigger_refs)
     time* '}'!
```

```
45      ;
46
47  test_refs   : TESTS^ DEF! idrefs ;
48              // default time = test starts instantly and is done once !  ;
49
50  agent
51      : AGENT! '{'! name^ address location? '}'! ;
52      name    : NAME! DEF! id ;
53      address : ADDRESS! DEF! string ;
54      location    : LOCATION! '{'! company street zip city country as cc grid
            '}'! ;
55          company : COMPANY^ DEF! string ;
56          street  : STREET^ DEF! string ;
57          zip : ZIP^ DEF! INT ;
58          city    : CITY^ DEF! string ;
59          country : COUNTRY^ DEF! string ;
60          as  : AS^ DEF! INT ;
61          cc  : CC^ DEF! INT ;
62          grid    : GRID^ DEF! string ;
63
64  agent_refs  : AGENTS DEF ((s_ids+=id(',' s_ids+=id)*) | (s_all=ALL (s_ex=
        EXCEPT (s_exids+=id(',' s_exids+=id)*) )?))
65           ((RIGHT | LEFT )  ((d_ids+=id(',' d_ids+=id)*) | (d_all=ALL (d_ex=
                EXCEPT (d_exids+=id(',' d_exids+=id)*) )?)))?
66      -> {$RIGHT != NULL}? ^(AGENTS ^(NET ^(SOURCE $s_ids* $s_all? ^($s_ex
            $s_exids*)?) ^(DESTINATION $d_ids* $d_all? ^($d_ex  $d_exids*)?)))
67      -> {$LEFT != NULL}? ^(AGENTS ^(NET ^(SOURCE $d_ids* $d_all? ^($d_ex
            $d_exids*)?) ^(DESTINATION $s_ids* $s_all? ^($s_ex  $s_exids*)?)))
68      ->^(AGENTS ^(NET ^(SOURCE $s_ids* $s_all? ^($s_ex $s_exids*)?) ^(
            DESTINATION $s_ids* $s_all? ^($s_ex $s_exids*)?)))
69       ;
70
71  agentset    : AGENTSET '{' name description net+ '}'
72      -> ^(name description ^(AGENTS net+))
73       ;
74      description : DESCRIPTION! DEF! string ;
75      net : NET^ '{'! source destination '}'! ;
76          source  : SOURCE DEF ( ir=idrefs | (ALL (EXCEPT ex=idrefs )?))
77              ->^(SOURCE $ir? ALL? ^(EXCEPT $ex )?)
78              ;
79          destination : DESTINATION DEF ( ir=idrefs | (ALL (EXCEPT ex=idrefs)?)
                )
80              ->^(DESTINATION $ir? ALL? ^(EXCEPT $ex )?)
81              ;
82
83
84  probe   : PROBE '{' name description repeat? (recorded | generated) '}'
85      -> ^(name description recorded generated repeat) ;
86      repeat  : REPEAT^ DEF! INT ;
87      recorded    : RECORDED^ '{'! 'file' DEF! string '}'! ;
88      generated   : GENERATED^ '{'! l3 '}'! ;
89      // LAYER 3
90      l3 : (ipv4 | ipv6 | icmpv4 | icmpv6) ;
91      ipv4 : IPv4^ '{'! l4 '}'! ;
92      ipv6 : IPv6^ '{'! l4 '}'! ;
93      icmpv4 : ICMPv4^ '{'! (PING | TRACEROUTE) '}'! ;
94      icmpv6 : ICMPv6^ '{'! (PING | TRACEROUTE) '}'! ;
```

```
95          // LAYER 4
96          l4 : (tcp | udp ) ;
97          tcp : TCP^ '{'! port l7tcp '}'! ;
98          udp : UDP^ '{'! port l7udp '}'! ;
99          port : PORT^ DEF! INT {checkPort($INT.text);} ;
100             // LAYER 7
101             l7tcp : ( http | ftp | torrent) ;
102             http : HTTP '{' cmd+ '}'
103             -> ^(HTTP cmd+ ) ;
104                 cmd : (GET string param?)
105                  -> ^(GET string param?)
106                 | (POST string param?)
107                 -> ^(POST string param?) ;
108             ftp : FTP ;
109             torrent : TORRENT '{' string+ '}'
110             -> ^(TORRENT string+) ;
111             param : string ;
112
113             l7udp : rtp ;
114             rtp : RTP ;
115
116 probe_refs  : PROBES^ DEF! idrefs ; // references probesets as well!!
117
118 probeset    : PROBESET! '{'! name^ description probe_refs '}'! ;
119
120
121 // TIME
122 // http://www.w3.org/TR/NOTE-datetime
123 time        : begin ((end interval) | period_ref)
124          ->^(TIME begin end? interval? period_ref?) ;
125
126     begin   : START^ DEF! ( DATETIME | NOW) ;
127     end : END^ DEF! DATETIME ;
128     interval    : INTERVAL^ DEF! DURATION_TYPE ;
129         // nach W3C
130
131 // PERIOD
132 period  : PERIOD! '{'! name^ description duration interval '}'! ;
133     duration    : DURATION^ DEF! DURATION_TYPE ;
134 period_ref  : PERIOD^ DEF! id ;
135
136
137 // TRIGGER
138
139
140 trigger : TRIGGER! '{'! (name^ description probe_refs (metric ttype behavior)
        ? ) '}'! ;
141     metric  : METRIC^ DEF! METRIC_TYPE ; // left means measured result, right
            means input value
142     behavior    : BEHAVIOR^ DEF! ('once' | 'always' | INT) ;
143     ttype   : TYPE^ DEF! TRIGGER_VALUES ;
144
145
146 action  : ACTION^ (('{'! (trigger_refs | test_refs | probe_refs)+ '}'!) | (
        DEF! ('delete' | 'restart' | 'stop'))) ;
147
148
```

```
149 trigger_refs:   TRIGGER DEF  id '{' value action '}' (',' id '{' value
        action '}')* -> ^(TRIGGER ^(id value action)+) ;
150     value : VALUE^ DEF! (string | TRUE) ;
151
152 string  : STRING ;
153         // remove " from Strings!
154
155 path    : address (','! address)+ ;
156
157 idrefs  : id (',' id)* -> id+ ;
158 id  : ID ;
159 /*------------------------------------------------------------------
160  * LEXER RULES
161  *------------------------------------------------------------------*/
162
163 // Tokennames
164
165 TEST    : 'test' ;
166 TESTS   : 'tests' ;
167 AGENT   : 'agent' ;
168 NAME    : 'name' ;
169 ADDRESS : 'address' ;
170 PORT    : 'port' ;
171 TYPE    : 'type' ;
172 LOCATION    : 'location' ;
173 COMPANY : 'company' ;
174 STREET  : 'street' ;
175 ZIP : 'zip' ;
176 CITY    : 'city' ;
177 COUNTRY : 'country' ;
178 AS  : 'AS' ;
179 CC  : 'CC' ;
180 GRID    : 'grid' ;
181
182 AGENTS  : 'agents' ;
183 AGENTSET    : 'agentset' ;
184 AGENTSETS   : 'agentsets' ;
185 NET : 'net' ;
186 SOURCE  : 'source' ;
187 DESTINATION : 'destination' ;
188 ALL : 'all' ;
189 EXCEPT  : 'except' ;
190
191 PROBE   : 'probe' ;
192 PROBES  : 'probes' ;
193 PROBESET    : 'probeset' ;
194 DESCRIPTION : 'description' ;
195 REPEAT  : 'repeat' ;
196 GENERATED   : 'generated' ;
197 RECORDED    : 'recorded' ;
198
199 IPv4    : 'IPv4' ;
200 IPv6    : 'IPv6' ;
201 ICMPv4  : 'ICMPv4' ;
202 ICMPv6  : 'ICMPv6' ;
203 PING    : 'PING' ;
204 TRACEROUTE  : 'TRACEROUTE' ;
```

```
205
206 TCP : 'TCP' ;
207 UDP : 'UDP' ;
208
209 HTTP    : 'HTTP' ;
210 GET : 'GET' ;
211 POST    : 'POST' ;
212 FTP : 'FTP' ;
213 TORRENT : 'TORRENT' ;
214
215 RTP : 'RTP' ;
216
217 START   : 'start' ;
218 END : 'end' ;
219 DURATION    : 'duration' ;
220 INTERVAL    : 'interval' ;
221 PERIOD  : 'period' ;
222 NOW : 'now' ;
223
224 TRIGGER : 'trigger' ;
225 METRIC  : 'metric' ;
226 BEHAVIOR    : 'behavior' ;
227 VALUE   : 'value' ;
228 TRIGGER_VALUES
229     : 'RTT' | 'TROUGHPUT' | 'TIME' ;
230 ACTION  : 'action' ;
231 TRUE    : 'true' ;
232
233 // Data Types
234
235
236 METRIC_TYPE
237     :   '<' | '>' | '==' | '<=' | '>='
238     ;
239
240 INT :   (NUMBER)+
241     ;
242 ID      :   (CHAR | NUMBER)+
243         ;
244 STRING  :   '"' (CHAR | NUMBER | ' ' | SPECIALS)* '"'
245         ;
246 DATETIME
247     :   DATE 'T' TIME TZD
248     ;
249 DURATION_TYPE
250     :    (DAYS ':')? TIME?
251     ;
252 fragment
253 DATE    :   (NUMBER NUMBER NUMBER NUMBER) '-' (('0' NUMBER) | ('1' '0'..'2'))
        '-' (('0'..'2' NUMBER) | ('3' '0'..'1'))
254     ;
255 fragment
256 TIME    :   (('0'..'1' NUMBER)|('2' '0'..'3'))':'('0'..'5' NUMBER)
        ':'('0'..'5' NUMBER)
257     ;
258 fragment
259 TZD :   (('Z') | (      // Time Zone Designator
```

```
260                 ('-'|'+')
261                     (
262                         (
263                             (('0' NUMBER)|('1' '0'..'1'))
264                             ':'('0'..'5' NUMBER)
265 
266                         )
267                      | '12:00'
268                      )
269                 )
270             )
271         ;
272 fragment
273 DAYS    :   NUMBER+ 'd'
274         ;
275 
276 fragment
277 NUMBER  :   '0'..'9'
278             ;
279 
280 COMMENT
281             :       '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
282                 |   '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN;}
283             ;
284 
285 WS          :       ( ' '
286             |   '\t'
287             |   '\r'
288             |   '\n'
289             )   {$channel=HIDDEN;}
290             ;
291 
292 fragment
293 CHAR    :   'a'..'z' | 'A'..'Z' | '_' | '<' | '>' | '-' | '+'
294         ;
295 
296 fragment
297 SPECIALS:   '.' | '/' | ':' | ',' | '!' | '@' | '#' | '\\' | '|' | '%' | '\''
        | '{' | '}' | '$' | '=' | '&'
298         ;
```

2 ANTLR NNTester Tree Walker Grammatik

```
tree grammar NNTesterTreeWalker;

options {
    language=C;
    tokenVocab=NNTester;
    output=AST;
    ASTLabelType    = pANTLR3_BASE_TREE;
}

scope global{
    pANTLR3_BASE_TREE testTree;
    pANTLR3_BASE_TREE agentTree;
    pANTLR3_BASE_TREE agentSetTree;
    pANTLR3_BASE_TREE probeTree;
    pANTLR3_BASE_TREE probeSetTree;
    pANTLR3_BASE_TREE periodTree;
    pANTLR3_BASE_TREE triggerTree;
}
@includes{
    #include "NNTesterTreeWalkerHelper.h"
}


/*------------------------------------------------------------------
 * PARSER RULES
 *------------------------------------------------------------------*/
start
scope global;
    :
    (COMMENT
    | ^(TestTree {$global::testTree=$TestTree;} test*)
    | ^(AgentTree {$global::agentTree=$AgentTree;} agent*)
    | ^(AgentSetTree {$global::agentSetTree=$AgentSetTree;} agentset* )
    | ^(ProbeTree {$global::probeTree=$ProbeTree;} probe*)
    | ^(ProbeSetTree {$global::probeSetTree=$ProbeSetTree;} probeset*)
    | ^(PeriodTree {$global::periodTree=$PeriodTree;} period*)
    | ^(TriggerTree {$global::triggerTree=$TriggerTree;} trigger*)
    )*
    //{printf("probeTree: \%s",$ProbeTree->toStringTree($ProbeTree)->chars);}
    ;

test    :
    ^(name description agent_refs (probe_refs time*| trigger_refs time*))
    ;

test_refs   : ^(TESTS idrefs) ->
    {
        refs2id($TESTS,  $global::testTree, NULL)
    }
    ;

            // default time = test starts instantly and is done once !  ;

agent   : ^(name address location?)
    ;
```

```
56      name    : id ;
57      address : STRING ;
58      //typ   : (FULL | REQUEST) ;
59      location    :  company street zip city country as cc grid  ;
60          company     : ^(COMPANY STRING) ;
61          street      : ^(STREET STRING) ;
62          zip     : ^(ZIP INT) ;
63          city        : ^(CITY  STRING) ;
64          country     : ^(COUNTRY  STRING) ;
65          as      : ^(AS  INT) ;
66          cc      : ^(CC  INT) ;
67          grid        : ^(GRID  STRING) ;
68
69  agent_refs  :
70      ^(AGENTS ^(NET ^(SOURCE (s_ids=idrefs | ALL | (ALL ^(s_ex=EXCEPT s_exids=
            idrefs))))
71          (^(DESTINATION (d_ids=idrefs| ALL | (ALL ^(d_ex=EXCEPT d_exids=idrefs
                ))) ))? ))
72      -> {isSet($SOURCE,$global::agentSetTree)}? ^(AGENTS {agent_setrefs2id(
            $AGENTS,$global::agentSetTree)})
73      -> ^(AGENTS ^(NET  {agent_refs2id($SOURCE,$global::agentTree,$s_ex)}
74          {agent_refs2id($DESTINATION,$global::agentTree,$d_ex)}
75           ))
76      ;
77
78  agentset    : ^(name description  ^(AGENTS  net+))
79      ;
80      description : STRING ;
81      net     : ^(NET source destination) ;
82          source  : ^(SOURCE (idrefs| ALL | (ALL ^(EXCEPT idrefs))))
83              ->  {agent_refs2id($SOURCE,$global::agentTree,$EXCEPT)}
84              ;
85          destination : ^(DESTINATION  (idrefs | ALL | (ALL ^(EXCEPT idrefs))))
86               ->  {agent_refs2id($DESTINATION,$global::agentTree,$EXCEPT)}
87              ;
88
89
90  probe       : ^(name description (recorded | generated) repeat? ) ;
91      repeat  : ^(REPEAT INT) ;
92      recorded    : ^(RECORDED 'file' STRING) ;
93      generated   : ^(GENERATED l3) ;
94      // LAYER 3
95      l3 : (ipv4 | ipv6 | icmpv4 | icmpv6) ;
96      ipv4 : ^(IPv4 l4) ;
97      ipv6 : ^(IPv6 l4) ;
98      icmpv4 : ^(ICMPv4 (PING | TRACEROUTE)) ;
99      icmpv6 : ^(ICMPv6 (PING | TRACEROUTE)) ;
100         // LAYER 4
101         l4 : (tcp | udp ) ;
102         tcp : ^(TCP port l7tcp) ;
103         udp : ^(UDP  port l7udp) ;
104         port : ^(PORT INT) {checkPort($INT);} ;
105             // LAYER 7
106             l7tcp : ( http | ftp | torrent) ;
107             http : ^(HTTP (^(cmd STRING param?))+ ) ;
108             cmd : GET | POST ;
109             ftp : FTP ;
```

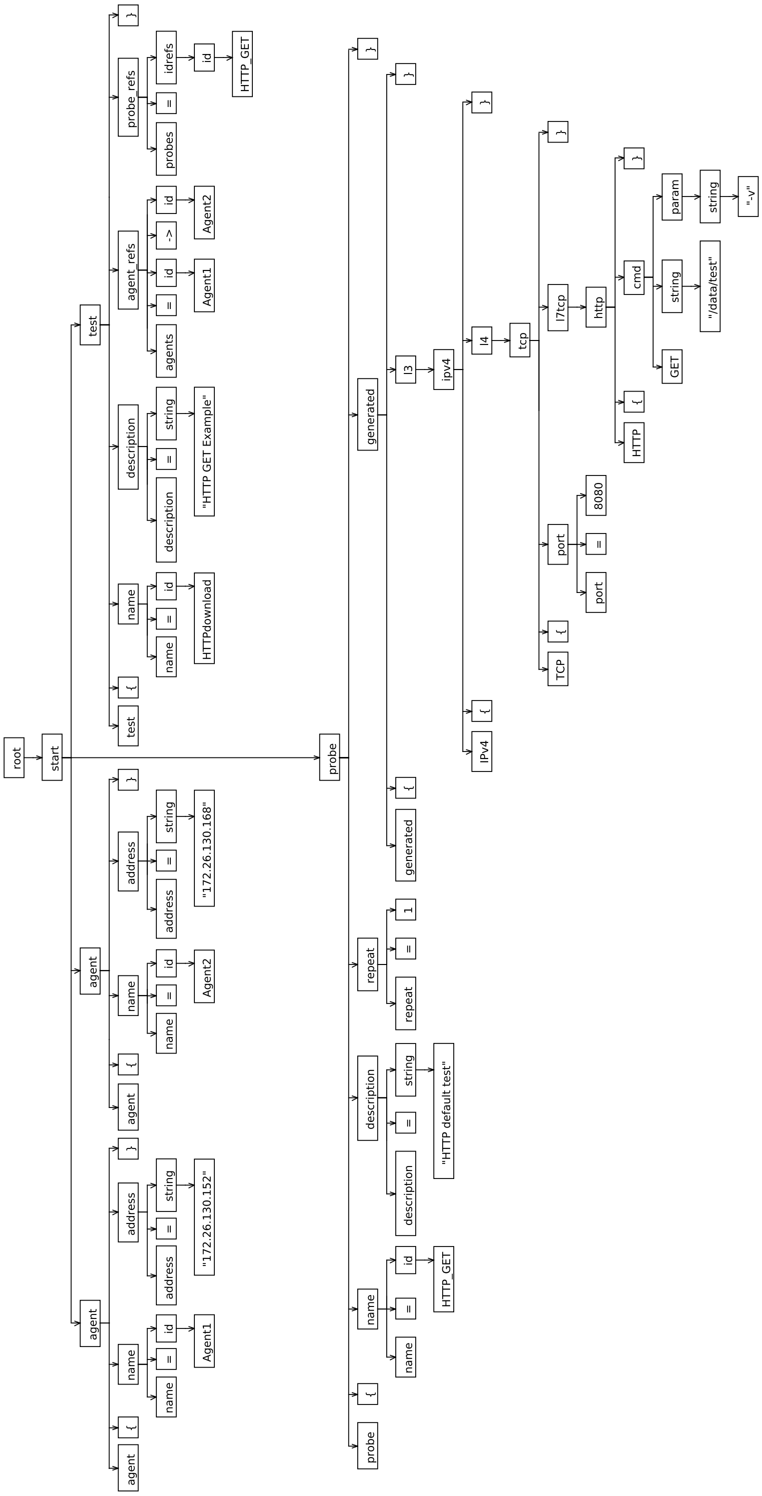
```
110             torrent : ^(TORRENT STRING+);
111             param : STRING ;
112 
113             l7udp : rtp ;
114             rtp : RTP ;
115 
116 
117 probe_refs  : ^(PROBES idrefs) ->
118     {
119         refs2id($PROBES,  $global::probeTree, $global::probeSetTree)
120     }
121     ; // references probesets as well!!
122 
123 probeset    : ^( name description ^(PROBES idrefs)) -> ^(name description
124     {
125         refs2id($PROBES,  $global::probeTree , NULL)
126     } )
127     ;
128 
129 
130 
131 // TIME
132 // http://www.w3.org/TR/NOTE-datetime
133 time        : ^(TIME begin ((end interval) | period_ref)) ;
134 
135     begin   : ^(START ( DATETIME | NOW)) ;
136     end : ^(END  DATETIME) ;
137     interval    : ^(INTERVAL DURATION_TYPE) ;
138         // nach W3C
139 
140 // PERIOD
141 period      : ^(name description duration interval) ;
142     duration    : ^(DURATION DURATION_TYPE) ;
143 period_ref  : ^(PERIOD id) ->
144     {
145         refs2id($PERIOD,$global::periodTree,NULL)
146     };
147 
148 
149 // TRIGGER
150 
151 
152 trigger : ^(name description probe_refs (metric ttype behavior)? ) ;
153     metric  : ^(METRIC METRIC_TYPE) ;  // left means measured result, right
        means value
154     behavior    : ^(BEHAVIOR ('once' | 'always' | INT)) ;
155     ttype   : ^(TYPE TRIGGER_VALUES) ;
156 
157 action  : ^(ACTION (((trigger_refs | agent_refs | probe_refs)+ ) | ('delete'
    | 'restart' | 'stop'))) ;
158 
159 
160 trigger_refs    : ^(TRIGGER (^( id value action))+)
161     ->  { trigger_refs2id($TRIGGER,$global::triggerTree)} ;
162     value   : ^(VALUE (STRING | TRUE)) ; // definition over inputtype in
        trigger definition
163 
```

```
164
165 path          :    address+ ;
166
167 id        :    ID
168           ;
169 idrefs       :    id+
170           ;
```

3 ANTLR NNTesterAgent String Tree Grammatik

```
1  grammar fromStringTree;
2
3  options {
4      language = C;
5      output=AST;
6  }
7
8
9  start    : root ;
10
11 root     : '('  node  child+ ')' -> ^(node child+) ;
12
13 child    : root |node ;
14
15 node     : NODE |STRING {$STRING->setText($STRING, $STRING.text->subString(
       $STRING.text,1,$STRING.text->len-1));} ;
16 NODE     : (CHAR | SPECIALS | NUMBER)+ ;
17 STRING   : '"' (CHAR | SPECIALS | NUMBER | ' ')* '"';
18
19 fragment
20 CHAR     : 'a'..'z' | 'A'..'Z' | '_' | '<' | '>' | '-' | '+' ;
21
22 fragment
23 SPECIALS     : '.' | '/' | ':' | ',' | '!' | '@' | '#' | '\\' | '|' | '%' |
       '\'' | '{' | '}' | '$' | '=' | '&' ;
24
25 fragment
26 NUMBER   : '0'..'9' ;
27
28 WS       :    ( ' '
29           | '\t'
30           | '\r'
31       | '\n'
32           ) {$channel=HIDDEN;}
33         ;
```

4 Vollständiger Parse Tree des HTTPdownload Tests



5 Ausschnitt aus dem dstat Messergebnis des HTTPdownloads

```
"Dstat 0.7.0 CSV output"
"Author:","Dag Wieers <dag@wieers.com>",,,,"URL:","http://dag.wieers.com/home-made/dstat/"
"Host:","ubuntu",,,,"User:","root"
"Cmdline:","dstat -t --socket --tcp -n --output ./measurements/1539541592_HTTP-GET_dstat_Agent1",,,,"Date:","08 Jun 2011 11:57:19 CET"

"system","sockets",,,,,"tcp sockets",,,,,"net/total",
"date/time","tot","tcp","udp","raw","frg","lis","act","syn","tim","clo","recv","send"
08-06 11:57:19,596.0,9.0,2.0,0.0,0.0,6.0,3.0,0.0,2.0,2.0,0.0,0.0
08-06 11:57:20,596.0,9.0,2.0,0.0,0.0,6.0,3.0,0.0,2.0,2.0,0.0,0.0
08-06 11:57:21,597.0,10.0,2.0,0.0,0.0,6.0,3.0,1.0,2.0,2.0,64.0,296.0
08-06 11:57:22,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,2.0,2.0,159052.0,3328.0
08-06 11:57:23,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,2.0,2.0,300000.0,5408.0
08-06 11:57:24,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,2.0,2.0,261000.0,4524.0
08-06 11:57:25,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,2.0,2.0,253552.0,4316.0
08-06 11:57:26,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,2.0,2.0,259500.0,4524.0
08-06 11:57:27,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,2.0,2.0,261000.0,4524.0
08-06 11:57:28,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,2.0,2.0,261052.0,4524.0
08-06 11:57:29,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,2.0,2.0,261000.0,4524.0
08-06 11:57:30,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,2.0,2.0,253604.0,4368.0
08-06 11:57:31,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,2.0,2.0,262500.0,4576.0
08-06 11:57:32,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,2.0,2.0,258000.0,4472.0
08-06 11:57:33,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,2.0,2.0,261000.0,4524.0
08-06 11:57:34,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,2.0,2.0,259500.0,4472.0
08-06 11:57:35,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,2.0,2.0,262500.0,4524.0
08-06 11:57:36,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,253500.0,4368.0
08-06 11:57:37,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,259548.0,4524.0
08-06 11:57:38,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,261000.0,4524.0
08-06 11:57:39,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,253552.0,4368.0
08-06 11:57:40,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,259548.0,4524.0
08-06 11:57:41,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,259500.0,4524.0
08-06 11:57:42,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,255000.0,4420.0
08-06 11:57:43,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,267156.0,4680.0
08-06 11:57:44,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,252000.0,4368.0
08-06 11:57:45,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,261000.0,4524.0
08-06 11:57:46,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,261000.0,4524.0
08-06 11:57:47,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,261052.0,4524.0
08-06 11:57:48,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,256552.0,4420.0
08-06 11:57:49,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,256500.0,4472.0
08-06 11:57:50,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,261000.0,4524.0
08-06 11:57:51,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,261000.0,4524.0
08-06 11:57:52,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,261000.0,4524.0
08-06 11:57:53,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,261052.0,4524.0
08-06 11:57:54,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,252052.0,4368.0
08-06 11:57:55,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,261000.0,4420.0
08-06 11:57:56,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,252000.0,4368.0
08-06 11:57:57,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,259500.0,4472.0
08-06 11:57:58,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,256500.0,4472.0
08-06 11:57:59,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,256500.0,4420.0
08-06 11:58:00,597.0,10.0,2.0,0.0,0.0,6.0,4.0,0.0,0.0,2.0,261000.0,4524.0
```