Hybrider zustandsbehafteter Paketfilter mit P4-Hardwarebeschleunigung

Bachelorarbeit

von

Felix Maurer

Hochschule für angewandte Wissenschaften München
Fakultät für Informatik und Mathematik
Studiengang Informatik (Bachelor of Science)

Prüfer:	Prof. Dr. Alf Zugenmaier
Betreuer:	Claas Lorenz, genua GmbH
Einreichung:	7. März 2019

Erklärung

Felix Maurer	15.05.1997	IF7, WS 2018
Name	*Geburtsdatum*	*Studiengruppe*

Hiermit erkläre ich, dass ich die Bachelorarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

München, den 7. März 2019	
Ort, Datum	*Unterschrift*

Zusammenfassung

Durch den immer höheren Grad der Vernetzung von immer mehr Geräten, wachsen die verfügbaren und genutzten Bandbreiten innerhalb von und zwischen Netzwerken ständig an. In Firewalls, die zur Absicherung dieser Netze verwendeten werden, entwickelt sich die Verarbeitung der Pakete im Paketfilter jedoch zu einem Flaschenhals, der zu einer Begrenzung der Geschwindigkeit führt. Damit Pakete weiterhin ausreichend schnell verarbeitet werden können, stellt eine zumindest teilweise Auslagerung dieser Funktionalität in Hardware einen vielversprechenden Ansatz dar. Die Implementierung in einer programmierbaren Hardware scheint dabei einen Geschwindigkeitsvorteil gegenüber einer reinen Softwarelösung zu bieten, ohne die Flexibilität aufzugeben, die nötig ist, um mit den sich schnell ändernden Anforderungen und Protokollen Schritt halten zu können.

Ziel dieser Arbeit ist es, anhand der Implementierung eines hybriden Paketfilters mit Hardwarebeschleunigung durch eine programmierbare Netzwerkkarte, die Erhöhung der Verarbeitungsgeschwindigkeit eines Paketfilters beispielhaft zu zeigen. Zuerst wird ein Einblick in den Stand von Software Defined Networking gegeben und mit der Programmiersprache P4 eine aktuelle Entwicklung im Bereich der programmierbaren Data Planes beleuchtet. Im Folgenden wird ein einfacher zustandsbehafteter Paketfilter entwickelt. Dabei wird das State-Tracking auf dem Host durchgeführt, während die in der Sprache P4 entwickelte Firmware auf der Netzwerkkarte das Weiterleiten der Pakete übernimmt. Durch eine Reduktion der Menge an Paketen, die den Software-Paketfilter auf dem Host erreichen, wird dabei die Last des Host-Systems reduziert. Mit dem implementierten Paketfilter wurden in einer Testumgebung mit zustandslosen und zustandsbehafteten Regeln jeweils Messungen zur Ermittlung des Geschwindigkeitsvorteiles durchgeführt. So konnte gezeigt werden, dass sich bei zustandslosen Regeln die Datenrate durch den hybriden Paketfilter gegenüber iptables steigern lässt. Mit zustandsbehafteten Regeln konnte jedoch aufgrund von Problemen mit der verwendeten Netzwerkkarte keine Verbesserung erzielt werden.

Inhaltsverzeichnis

1. Einleitung und Motivation

Durch die voranschreitende Digitalisierung und aktuelle Trends, wie das Internet of Things, sind immer mehr Geräte mit Netzwerken verbunden und verlassen sich immer mehr Anwendungen auf eine Internetverbindung, über die steigende Datenmengen übertragen werden. Während die Datenraten ständig anwachsen, muss gleichzeitig die Sicherheit der vernetzen Systeme verstärkt werden. Sicherheitssysteme wie Firewalls müssen mit den steigenden Datenraten mithalten können, ohne den Schutz zu vernachlässigen. Trotz neuer Softwarearchitekturen und dem Einsatz von mehr Ressourcen, fällt es Softwaresystemen jedoch in steigendem Maße schwer, mit den geforderten Datenraten mitzuhalten, weshalb besonders im Hochgeschwindigkeitsumfeld spezialisierte Hardwarefirewalls zum Einsatz kommen.

Während der Einsatz spezialisierter Hardware vor allem durch hohe Parallelität der Verarbeitung von Paketen einen großen Geschwindigkeitsvorteil bietet, unterliegen die Analysemöglichkeiten engeren Beschränkungen. So können meist nur einfache Regeln angewendet und nur wenig Zustand verarbeitet werden. Des Weiteren ist die Entwicklung solcher Systeme auf Basis von Field Programmable Gate Arrays (FPGAs) oder Application-specific Integrated Circuits (ASICs) aufwändig und teuer. Eine spätere Erweiterung der Funktionalität ist oft schwierig oder unmöglich. Dagegen bietet Software breite Möglichkeiten komplexer Verarbeitungen, wie beispielsweise Deep Packet Inspection oder Intrusion Prevention Systeme. Zudem ist für den Einsatz solcher Systeme weniger Spezialwissen erforderlich und sie können zu einem späteren Zeitpunk einfach um Funktionalitäten erweitert werden. [vgl. 2]

Einen Versuch, die Vorteile bezüglich der Geschwindigkeit und der möglichen Komplexität der Verarbeitung zu vereinen, stellen hybride Architekturen dar, die Software- und Hardwarekomponenten kombinieren [8, 10]. In dieser Arbeit wird eine solche Architektur für einen zustandsbehafteten Paketfilter umgesetzt.

Die zur Unterstützung verwendete Hardware ist eine Netzwerkkarte mit einem speziel-

len Prozessor zur Verarbeitung von Netzwerkpaketen. Um den Paketfilter auch in Zukunft einfach erweitern zu können und keine Flexibilität zu verlieren, wird zur Programmierung der Netzwerkkarte die Sprache P4 verwendet. Dabei handelt es sich um eine Programmiersprache, die konzipiert wurde, um die Data Plane von Netzwerkgeräten programmieren zu können. Da sie eine starke Abstraktion darstellt, verspricht sie eine deutlich geringere Komplexität der Entwicklung als bei FPGAs, während trotzdem hohe Verarbeitungsgeschwindigkeiten möglich sind. Die Sprache ist dabei eng mit den Konzepten des Software Defined Networking verbunden, wodurch sich bei der Umsetzung von Anwendungen eine Architektur ergibt, die auf die zukünftigen Entwicklungen im Bereich der Computernetzwerke gut vorbereitet ist. [vgl. 2, 11]

Im Rahmen der Arbeit wurde eine Firmware für die Netzwerkkarte entwickelt, die eine Hardwareunterstützung für einen zustandsbehafteten Paketfilter bieten soll. Dabei bildet die Netzwerkkarte einfache Regeln ab und kann erkannte sichere Verbindungen beschleunigen, indem sie die Pakete selbstständig verarbeitet, ohne sie an den Host zu übermitteln. Auf dem Host können auf den verbleibenden Datenverkehr komplexe Regeln eines Paketfilters angewendet werden. Für das Erkennen des Zustandes von Verbindungen wurde eine Software entwickelt, die auf dem Host ausgeführt wird und erkannte Verbindungen an die Netzwerkkarte übermittelt. Die Aufteilung der Komponenten und deren Zusammenarbeit orientiert sich am Software Defined Networking.

Anschließend wurde eine Evaluation der Implementierung durchgeführt, bei der die möglichen Datenraten, die der hybride Paketfilter verarbeiten kann, ermittelt wurden. Der Durchsatz wurde sowohl mit zustandslosen als auch mit zustandsbehafteten Regeln mittels unterschiedlicher Verfahren bestimmt.

In dieser Arbeit wird zunächst eine Einführung in Software Defined Networking und die Sprache P4 gegeben und der Stand der Wissenschaft beschrieben. Darauf folgt die Darstellung des Konzeptes für den hybriden Paketfilter und dessen Umsetzung in einem Gesamtsystem. Anschließend werden die zur Evaluation verwendeten Methoden beschrieben und die Ergebnisse daraus diskutiert. Die Arbeit schließt mit einem Fazit und einem kurzen Ausblick.

2. Grundlagen

Dieses Kapitel gibt zu Beginn eine Einführung in die Entwicklungen im Bereich des Software Defined Networking gefolgt von der Beschreibung der in der Arbeit verwendeten Programmiersprache P4. Es schließt mit der Darstellung des Standes der Wissenschaft im Arbeitsgebiet.

2.1. Software Defined Networking

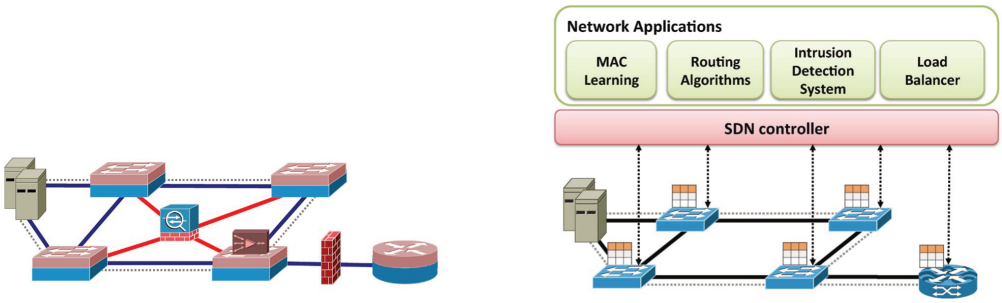
Die Informationen für die folgende Einführung in den Bereich des Software Defined Networking (SDN) wurden den umfangreichen Studien von Kreutz et al. [11] und Scott-Hayward, Natarajan und Sezer [15] entnommen.

Die Funktionalität von Computernetzwerken lässt sich in die drei Ebenen *Data Plane*, *Control Plane* und *Management Plane* einteilen. Die Data Plane ist dabei dafür verantwortlich, Datenpakete schnell und effizient weiterzuleiten. Die Control Plane wird durch Protokolle wie beispielsweise ARP oder OSPF definiert, die die Forwarding-Tabellen der Data Plane pflegen. Die Management Plane beinhaltet Software zur Konfiguration und der Überwachung des Netzwerkes und seiner Komponenten. Andersherum betrachtet gibt die Management Plane eine Richtlinie zur Verarbeitung von Netzwerkverkehr vor, die Control Plane setzt diese durch und die Data Plane führt sie nur aus, indem sie Pakete entsprechend weiterleitet. [vgl. 11]

In klassischen Netzwerken sind insbesondere Control und Data Plane eng miteinander verbunden und oft in einem Gerät untergebracht. Das führt zu einer stark dezentralisierten Struktur, die das Management komplex macht und schnelle Innovation erschwert. Außerdem werden häufig eine Vielzahl spezialisierter Geräte und sogenannter Middleboxes wie Firewalls oder Intrusion Detection Systeme eingesetzt. Die große Zahl von Geräten sorgt dafür, dass der Einsatz neuer Technologien nur mit großem Aufwand möglich ist, da diese von allen Geräten unterstützt werden müssen, was ein Softwareupdate oder sogar einen

Hardwaretausch nötig machen kann. [vgl. 11]

Im Gegensatz dazu hat SDN den Kerngedanken, Control Plane und Data Plane voneinander zu trennen. Die Netzwerkgeräte sollen nur noch dem Weiterleiten von Paketen dienen, während die Kontrolllogik in den so genannten Controller ausgelagert wird. Ein solcher Controller wird auch Network Operating System (NOS) genannt, da er weitere Anwendungen ausführen kann, die mit den Geräten der Data Plane interagieren können. Dadurch wird das gesamte Netzwerk programmierbar. Die weiterleitenden Geräte werden meist Switches genannt, obwohl sie abhängig vom Controller deutlich weitreichendere Aufgaben wie Routing oder Firewalling übernehmen können. In Abbildung 2.1 ist der Vergleich zwischen klassischen Netzwerken und SDN dargestellt. Auffällig ist dabei auch der Wegfall der Middleboxes, die nun durch Anwendungen auf dem Controller ersetzt wurden. Durch die Konzentration der Logik im Controller wird der Einsatz neuer Technologien einfacher möglich, da hierfür nur die dort ausgeführten Anwendungen erweitert werden müssen. [vgl. 11]



(a) Klassisches Netzwerk [11] (b) Software Defined Networking [11]

Abbildung 2.1.: Gegenüberstellung von klassischem Netzwerk und SDN [11]

Dieser Aufbau hat mehrere Vorteile bezüglich des Managements eines Netzwerkes und der Möglichkeit der Weiterentwicklung. Die Entwicklung der Anwendungen wird erleichtert, da ein Controller gemeinsame Abstraktionen bereitstellt. Außerdem können die Applikationen von den gemeinsamen Informationen profitieren, die im Controller vorliegen und eine globale Sicht auf das gesamte Netzwerk ermöglichen. Da die Anwendungen nur auf dem Controller ausgeführt werden, wird es einfacher, neue Funktionalitäten in das Netzwerk zu integrieren, die untereinander und mit den bestehenden zusammenarbeiten müssen. [vgl. 11]

Der Einsatz von SDN-Technologien hat jedoch auch Nachteile, die von Kreutz et al. [11] ausführlich beschrieben werden. Die erstmalige Verwendung von SDN in einem beste-

henden Netzwerk ist mit einem großen Migrationsaufwand verbunden. Das Netzwerk muss anders aufgebaut werden, es ist anderes Wissen bei den Betreibenden nötig und es kann sogar ein Austausch von vielen Geräten nötig sein. Es gibt auch Ansätze, SDN-Komponenten und klassische Netzwerkgeräte gemischt einzusetzen, was jedoch eine zusätzliche Komplexität mit sich bringt. Eine weitere Schwierigkeit stellt die Skalierbarkeit dar. Durch die Zentralisierung des Controllers bildet sich hier ein Single Point of Failure, der besonders vor Überlastung oder Ausfall geschützt werden muss. Die ständige Kommunikation der Data Planes mit dem Controller sorgt außerdem für eine höhere Last im Netzwerk. Sollen Netzwerkkomponenten in Echtzeit auf eintreffende Pakete reagieren, ist zudem die Latenz bei der Kommunikation mit dem Controller von entscheidender Bedeutung. Nicht zu unterschätzen sind auch die Herausforderungen, die sich durch den Einsatz von SDN für die Informationssicherheit stellen. Die Einführung eines zentralen Controllers öffnet Wege für neue Arten von Angriffen gegen Netzwerkkomponenten. Da dort viele Informationen zusammenlaufen, stellt er außerdem ein sehr lohnenswertes Ziel für Angreifer dar. Die beschriebenen und weitere Schwierigkeiten sind jedoch Gegenstand aktueller Forschung und industrieller Entwicklungen. [vgl. 11, 15]

Ein wichtiger Teil des SDN ist das Protokoll, welches zwischen dem Controller und der Data Plane verwendet wird. Der bekannteste Vertreter dieser Kategorie ist OpenFlow [13], das als akademisches Experiment vorgeschlagen wurde, aber mittlerweile auch in der Industrie an Bedeutung gewonnen hat. Ein OpenFlow-Switch besitzt eine oder mehrere Tabellen, die Regeln für die Paketverarbeitung beinhalten. Jede Regel trifft auf einen Teil des gesamten Datenverkehrs zu und führt damit bestimmte Aktionen durch. Je nachdem, welche Regeln durch den Controller vorgegeben werden, kann der Switch unterschiedliche Funktionen übernehmen. [vgl. 11]

2.2. Programming Protocol-Independent Packet Processors: P4

Die Beschreibungen in diesem Abschnitt basieren auf dem Paper von Bosshart et al. [2], der Spezifikation von $P4_{16}$ [40] und weiteren Quellen [5, 16, 18].

P4 ist eine neue domänenspezifische Programmiersprache, die eine abstrakte Programmierung von Netzwerkkomponenten erlaubt und im Jahr 2014 vorgestellt wurde. Schon auf der SIGCOMM 2016 wurden fünf Paper mit Bezug zu P4 veröffentlicht [5]. Die aktuelle Spezifikation wurde 2018 veröffentlicht und dem Konsortium hinter der Sprache gehören

mittlerweile 23 Universitäten und 89 Unternehmen an [37]. Dies zeigt das große Interesse an der Sprache sowohl im akademischen, als auch im industriellen Bereich.

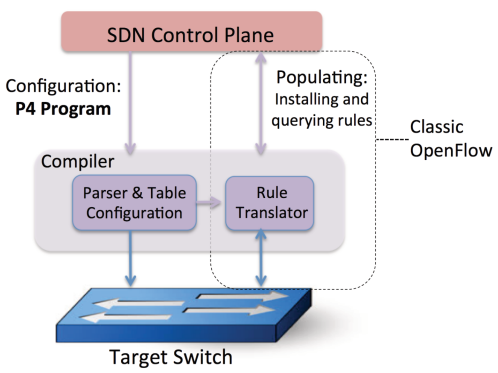
Die Grundidee hinter der Entwicklung von P4 war es, eine Programmiersprache zur Verarbeitung von Netzwerkpaketen zu entwerfen, die in Verbindung mit SDN-Kontrollprotokollen wie OpenFlow genutzt werden kann. Zusätzlich ist P4 ein Vorschlag, wie sich OpenFlow in der Zukunft entwickeln könnte, um mehr Flexibilität zu erreichen. Dass dies notwendig ist, zeigt sich daran, dass sich die Zahl der spezifizierten Header-Felder in nur vier Jahren von 12 (OpenFlow 1.0, 2009) auf 41 (OpenFlow 1.4, 2013) erhöht hat. Eine Option, eigene Header-Felder zu definieren, existiert jedoch, trotz der gestiegenen Komplexität der Spezifikationen, bis heute nicht [34]. Um nicht weiterhin immer neue Spezifikationen mit neuen Header-Feldern schreiben zu müssen, soll P4 mehr Flexibilität bieten, indem es erlaubt, auch den Parser, die Tabellen und die Verarbeitungsschritte eines Switches zu konfigurieren. Das ist durch neue Entwicklungen im Chipdesign auch bei Geschwindigkeiten im Terabit/s-Bereich möglich. Um dieses Ziel zu erreichen, wurde bei der Entwicklung der Sprache besonders auf drei Eigenschaften Wert gelegt. [vgl. 2, 18]

Rekonfigurierbarkeit Es soll möglich sein, die Verarbeitung der Pakete auch nach dem Ausrollen der Switches zu verändern. [vgl. 2, 18]

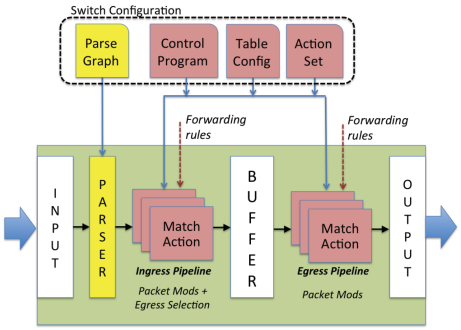
Protokollunabhängigkeit Ein Switch sollte nicht auf bestimme Protokolle festgelegt sein. Stattdessen sollte der Controller in der Lage sein, den Parser und die Verarbeitung der Pakete vorzugeben. [vgl. 2, 18]

Zielunabhängigkeit Ein P4-Programm soll eine hardwareunabhängige Beschreibung sein. Erst durch den Compiler wird daraus ein zielabhängiges Programm, dass die spezifischen Eigenschaften der Hardware berücksichtigt. [vgl. 2, 18]

In Abbildung 2.2a ist dargestellt, wie P4 als Ergänzung zum jetzigen OpenFlow positioniert ist. Die Kontrolle über einen Switch erfolgt durch zwei unterschiedliche Arten von Operationen: Konfiguration (*Configuration*) und Einpflegen (*Populating*). Bei der Konfiguration werden der Parser und die Match-Action-Tables, die die Verarbeitung der Pakete beeinflussen, programmiert. Dadurch wird festgelegt, welche Protokolle ein Switch unterstützt und wie Pakete verarbeitet werden können. Beim Einpflegen werden Einträge in die Match-Action-Tables eingefügt oder daraus entfernt. Damit wird das Regelwerk für die Verarbeitung einzelner Pakete definiert. Diesen Zweck erfüllen auch die aktuellen OpenFlow-Standards, während die Konfiguration des Switches eine neue Operation ist und in das Aufgabengebiet von P4 fallen würde. [vgl. 2, 16]



(a) Einsatz von P4 neben OpenFlow [2]



(b) Aufbau des abstrakten Forwarding-Modells, auf dem P4 basiert [2]

Abbildung 2.2.: Grafische Darstellungen zum Design der Sprache P4

Das Design der Sprache P4 basiert auf einem abstrakten Forwarding-Modell, das in Abbildung 2.2b dargestellt ist. Durch den generalisierten Aufbau, der unabhängig vom Zielgerät und dessen Architektur ist, konnte eine gemeinsame Sprache entwickelt werden, die aber für verschiedene Geräte übersetzt werden kann. Dargestellt ist auch die Trennung in Konfiguration durch P4 (blaue Pfeile) und Einpflegen von Regeln (rote Pfeile). Eingehende Pakete durchlaufen erst den konfigurierbaren Parser. Dieser extrahiert Felder aus den Headern und bestimmt somit, welche Protokolle durch den Switch unterstützt werden. Die extrahierten Felder werden an die Match-Action-Tables übergeben, die parallel oder sequentiell ausgeführt werden können. Zusätzlich können den Paketen weitere Informationen, Metadaten genannt, zugeordnet werden. In den Match-Action-Tables findet der wichtigste Teil der Verarbeitung statt, da hier abhängig von den Werten der Felder sogenannte Actions ausgeführt werden, welche die Pakete oder Metadaten verändern können. [vgl. 2, 40]

Während P4 ursprünglich, wie auch das Paper von Bosshart et al. [2] zeigt, hauptsächlich auf SDN-Switches ausgerichtet war, erreichen mittlerweile auch andere Geräte den Markt, die mit P4 programmiert werden können. In dieser Arbeit wird auf eine Netzwerkkarte mit dieser Fähigkeit zurückgegriffen.

Im Folgenden werden die einzelnen Konzepte der Sprache kurz anhand von Beispielen aus der entstandenen Implementierung beleuchtet.

2.2.1. Header

In einem Header wird die Reihenfolge und die Struktur der Felder eines Paketes angegeben. Generell wird ein Header durch eine Liste von Namen und Breiten der Felder deklariert. Es können außerdem noch weitere Einschränkungen für die Werte der einzelnen Felder angegeben werden. [vgl. 2, 16, 40]

```
typedef bit<16> port_t;
header UDP_h {
    port_t srcPort;
    port_t dstPort;
    bit<16> len;
    bit<16> checksum;
}
```

Listing 2.1.: Definition des UDP-Headers in $P4_{16}$

Ein Beispiel ist in Listing 2.1 gezeigt. Zu sehen ist, dass nur die Felder des Paketheaders angegeben sind, nicht jedoch die im Paket enthaltenen Nutzdaten.

2.2.2. Parser

```
state start {
    meta.skip_rule_table = false;
    transition parse_ethernet;
}

state parse_ethernet {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType) {
        TYPE_IPV4: parse_ipv4;
        TYPE_IPV6: parse_ipv6;
        default: accept;
    }
}
```

Listing 2.2.: Beispiel der ersten zwei Zustände eines Parsers

Der Parser ist dafür zuständig, dass die einzelnen definierten Header mit den richtigen Informationen aus einem Paket gefüllt werden. Er wird als Zustandsautomat angegeben. Die einzelnen Werte werden während des Durchlaufens des Automaten mit der Funktion

`extract()` aus den Paketen extrahiert. Die Übergänge zwischen den Zuständen werden direkt angegeben und können auch von den extrahierten Werten abhängen. [vgl. 2, 16, 40]

Ein Beispiel dafür ist in Listing 2.2 gezeigt. Das Parsen wird immer im Zustand `start` begonnen und durchgeführt, bis ein Endzustand wie beispielsweise `accept` erreicht wird [2]. Gut zu erkennen ist, wie der nächste Zustand nach dem Parsen des Ethernet-Headers durch den Ethertype im Header bestimmt wird.

2.2.3. Match-Action-Tables

Eine Kernkomponente der Sprache stellen die Match-Action-Tables dar. Sie definieren wie Felder und Metadaten eines Paketes überprüft werden sollen und welche Actions ausgeführt werden können. Mögliche Arten der Überprüfung sind zum Beispiel exakte Übereinstimmung, Bereiche oder Longest-Prefix-Matching. Im Gegensatz zu OpenFlow ist P4 durch die freie Definierbarkeit der zu überprüfenden Felder nicht auf existierende Protokolle beschränkt, sondern kann durch den Entwickler erweitert werden. [vgl. 2, 16, 40]

```
table flowsv4 {
    key = {
        hdr.ipv4.srcAddr: exact;
        hdr.tcp.srcPort: exact;
        hdr.ipv4.dstAddr: exact;
        hdr.tcp.dstPort: exact;
    }
    actions = {
        forwardv4;
        NoAction;
    }
    default_action = NoAction;
}
```

Listing 2.3.: Beispiel für eine Match-Action-Table

Ein Beispiel für die Definition einer Match-Action-Table ist in Listing 2.3 dargestellt. Die tatsächlichen Einträge, die dann den Zusammenhang zwischen den Werten und der ausgeführten Action definieren, werden erst zur Laufzeit in die Tabelle eingefügt. Das Einfügen und Entfernen von Einträgen ist jedoch nicht Teil der Spezifikation von P4, sondern muss vom Hersteller für das jeweilige Gerät entwickelt werden. In der Definition kann jedoch das Festlegen einer Standard-Action erfolgen, die ausgeführt wird, wenn kein anderer Eintrag zutrifft. [vgl. 2, 40]

2.2.4. Actions

P4 stellt nur primitive Operationen bereit, die zu komplexeren Actions kombiniert werden können. Jedes P4-Programm muss also ein Set von Actions definieren, die dann in den Match-Action-Tables verwendet werden können. Die Definition ähnelt sehr der von Funktionen in klassischen Programmiersprachen. [vgl. 2, 16, 40]

```
action forwardv4(hwport_t port) {
    meta.skip_rule_table = true;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    standard_metadata.egress_spec = port;
}
```

Listing 2.4.: Beispiel für eine Action

Ein Beispiel für eine Action ist in Listing 2.4 gezeigt. Hier ist auch zu sehen, dass Actions Parameter besitzen können. Ist dies der Fall, so müssen diese in den Einträgen der Match-Action-Tables zur Laufzeit angegeben werden. In einer Action sind jedoch keine bedingten Anweisungen erlaubt, da die Spezifikation vorsieht, dass die einzelnen Operationen parallel ausgeführt werden können [2].

2.2.5. Kontrollprogramm

Das Kontrollprogramm kombiniert die bisher geschilderten Komponenten in ein Programm, um den Kontrollfluss zu beschreiben. Sie bestehen aus den definierten Actions, weiteren Operationen, Bedingungen und Referenzen auf Tabellen. Die Beschreibung des Ablaufes erfolgt dabei für genau ein Paket. Abhängigkeiten der Pakete untereinander, wie beispielsweise die Reihenfolge des Eintreffens, können also nicht betrachtet werden. Die wichtigste Anweisung ist dabei das Anwenden von Match-Action-Tables mit `apply()`. Dadurch wird das Matching einer Tabelle mit den Headern und Metadaten des aktuellen Paketes durchgeführt und die Action des passenden Eintrages ausgeführt. [vgl. 2, 40]

Ein Ausschnitt aus einem solchen Kontrollprogramm ist in Listing 2.5 dargestellt. Auch diese Programme ähneln stark der Notation von Abläufen in klassischen Programmiersprachen. Sie sind die einzige Stelle, an denen P4 bedingte Anweisungen erlaubt [40]. Basierend auf den Headern oder Metadaten können so einzelne Operationen oder Tabellen angewendet oder ausgelassen werden.

```
if (hdr.tcp.isValid() && (hdr.tcp.flags & CONTROLLER_MASK) == 0) {
    flowsv4.apply();
}
if (!meta.skip_rule_table) {
    rulesv4.apply();
}
```

Listing 2.5.: Ausschnitt aus einem Programm

2.3. Stand der Wissenschaft

In diesem Abschnitt wird ein Überblick über bereits existierende wissenschaftliche Publikationen im Themengebiet dieser Arbeit gegeben.

Die Konzepte, die dieser Arbeit zugrunde liegen, sind sehr stark von SDN beeinflusst. Einen sehr umfassenden Einblick in diese Thema geben Kreutz et al. [11]. Es werden sowohl die Motivation hinter SDN, Grundlagen der Umsetzung und aktuelle Implementierungen als auch bestehende Herausforderungen dargestellt. Scott-Hayward, Natarajan und Sezer [15] konzentrieren sich in ihrer Studie hauptsächlich auf Aspekte der Informationssicherheit im Bereich SDN. Durch den umfassenden Ansatz der beiden Veröffentlichungen, konnten sie als Grundlage für die Ausführungen zu SDN genutzt werden.

Ein Teil der Umsetzung erfolgte in der domänenspezifischen Programmiersprache P4, die zur Programmierung der Data Plane von Netzwerkgeräten entwickelt wurde. Die erste Beschreibung der Sprache findet sich bei Bosshart et al. [2]. Auf dieser Basis und der aktuellen Spezifikation von $P4_{16}$ [40] aufbauend, wurde die Implementierung auf der Netzwerkkarte umgesetzt. Eine Darstellung der unterschiedlichen Zielplattformen, für die mit P4 entwickelt werden kann, und Möglichkeiten zu deren Vergleich findet sich in Dang et al. [5]. Liu et al. [12] zeigen die Verifizierbarkeit der Data Plane als weiteren Vorteil der Programmierung mit P4 auf.

Die Umsetzung hybrider Netzwerkkomponenten mit Hardwareunterstützung erfolgte bisher hauptsächlich unter Verwendung von FPGAs. So beschreiben Chen et al. [4] eine solche Umsetzung, bei der die Klassifikation von Paketen in einem mehrstufigen Aufbau passiert. Durch eine FPGA-basierte Netzwerkkarte wird lediglich eine Vorfilterung durchgeführt, während eine intensive Betrachtung der Pakete mittels Deep Packet Inspection auf dem Host vorgenommen wird. Gonzalez, Paxson und Weaver [10] und Weaver, Paxson und Gonzalez [20] zeigen einen Aufbau, bei dem ein Intrusion Detection System dynamisch Regeln auf der Netzwerkkarte hinzufügt oder entfernt. Sie prägen dafür den Begriff

Shunting und zeigen auch, welcher Anteil des Datenverkehrs für eine ausschließliche Verarbeitung im FPGA geeignet ist. Bei Fiessler et al. [8] und Fiessler et al. [9] finden sich Möglichkeiten der Aufteilung von Regeln eines Paketfilters in einen einfachen Teil, der auf dem FPGA ausgeführt werden kann, und einen komplexen Teil, welcher durch den Host übernommen wird.

Die Beschreibung einer Hardwarebeschleunigung eines Paketfilters durch die Verwendung einer programmierbaren Netzwerkkarte, die eine ähnliche Architektur wie die in dieser Arbeit verwendete SmartNIC aufweist, veröffentlichten Accardi et al. [1] bereits 2005. Dabei wurde jedoch die netfilter Firewall von Linux erweitert, um den internen Zustand an die Netzwerkkarte zu übergeben. Außerdem musste hier eine komplexe, wenig abstrakte Art der Programmierung gewählt werden, da eine Technologie wie P4 noch nicht zur Verfügung stand. Eine Implementierung eines zustandsbehafteten Paketfilters unter ausschließlicher Verwendung von P4 wird von Vörös und Kiss [18] beschrieben. Dabei beschränkt sich der Zustand jedoch auf das Prüfen der Datenrate von Netzwerken oder einzelnen Adressen, um damit einen Denial of Service-Angriff erkennen und blockieren zu können. Der Zustand von TCP-Verbindungen wird nicht betrachtet. Miteff und Hazelhurst [14] zeigen eine praktische Implementierung eines hybriden Paketfilters auf Basis einer SDN-Architektur. Dabei wird das conntrack-Modul von Linux für das State Tracking verwendet und bei erkannten Verbindungen eine Regel zur schnellen Weiterleitung der Pakete dieser Verbindung angelegt. Zum Einsatz kommt hier jedoch ein SDN-Switch mit OpenFlow zur Umsetzung des beschleunigten Weiterleitens. Die Implementierung ist jedoch sehr stark auf das Einsatzgebiet, die schnelle Vernetzung bestimmter Forschungseinrichtungen, zugeschnitten.

3. Konzept

Aus dem in Kapitel 2 zusammengefassten Stand der Wissenschaft, den dort beschriebenen Hintergründen und den Einschränkungen, die sich durch die verwendeten Komponenten ergeben, wurde ein Konzept für den Aufbau eines hybriden Paketfilters entwickelt. Dessen einzelne Komponenten und deren Zusammenspiel werden in diesem Kapitel beschrieben.

Die Grundidee ist dabei, den Durchsatz durch einen Paketfilter zu erhöhen, indem dieser durch eine Hardware-Komponente unterstützt wird. Die unterstützende Komponente ist in diesem Konzept eine mit P4 programmierbare Netzwerkkarte. Die Unterstützung besteht darin, dass die Netzwerkkarte möglichst viele Pakete selber behandeln und weiterleiten kann, wodurch das Hostsystem eine geringere Anzahl an Paketen bearbeiten muss. Dabei sollen auch zustandsbehaftete Regeln unterstützt werden. Das dafür nötige Tracking der Verbindungen wird jedoch auf dem Host durchgeführt.

Die Netzwerkkarte kann hauptsächlich einfache Regeln verarbeiten, die teilweise vom Host zur Laufzeit definiert werden können. Auf dem Host hingegen kann beliebige Software ausgeführt werden, die Netzwerkverkehr verarbeitet, wodurch nahezu beliebige Verarbeitungsmöglichkeiten zur Verfügung stehen. Neben einem klassischen Paketfilter könnten hier auch ausgefeiltere Komponenten wie Intrusion Prevention Systeme oder Deep Packet Inspection zum Einsatz kommen.

3.1. Firmware auf der Netzwerkkarte

Durch den Einsatz einer programmierbaren Netzwerkkarte, einer so genannten SmartNIC, wird es möglich, selbst definierte Paketverarbeitung auf der Karte durchzuführen. Die verwendete Hardware ist dabei auf die schnelle und parallele Verarbeitung von Netzwerkpaketen ausgelegt [33]. Da diese Verarbeitung auf dem Prozessor der Netzwerkkarte passiert, bevor ein Paket an den Host übermittelt wird, wird der Host durch die Verarbeitung nicht beeinflusst. Dadurch eignet sich eine solche Netzwerkkarte sehr gut, um den

Host bei einem Paketfilter mit hybrider Architektur zu entlasten. In dieser Arbeit wird die Firmware für die Karte in der Programmiersprache P4 entwickelt, um die Vorteile einer solchen domänenspezifischen Hochsprache hinsichtlich Einfachkeit und Geschwindigkeit bei der Softwareentwicklung ausnutzen zu können.

Die im Rahmen dieser Arbeit implementierte Firmware unterstützt die Behandlung von Paketen nach einfachen, zustandslosen Paketfilterregeln und zusätzlich den Abgleich mit einer Tabelle von existierenden Verbindungen, um zustandsbehaftete Regeln abbilden zu können. Die Erkennung von Verbindungen kann jedoch nicht auf der Karte durchgeführt werden und geschieht daher auf dem Host. Es werden sowohl das IP-Protokoll in der Version 4 als auch in der Version 6 unterstützt. Trifft eine Regel auf ein Paket zu, können drei unterschiedliche Actions ausgeführt werden. Zum einen kann das Paket direkt von der Karte verworfen werden (*drop*). Zum anderen kann das Paket akzeptiert und an einen bestimmten Port weitergeleitet werden (*accept*). Sollte das Paket direkt von einem physikalischen Port kommen und an einen physikalischen Port weitergeleitet werden, so wird das Paket wie bei *accept* behandelt, es wird jedoch zusätzlich die TTL (IPv4) beziehungsweise der Hop Count (IPv6) reduziert, um den IP-Standards zu entsprechen (*forward*). Dadurch ist es möglich, gleichzeitig einen Teil des Datenverkehrs nur durch die Netzwerkkarte zu bearbeiten, während ein anderer Teil an den Host übergeben wird. Die Regeln können folgende Felder der Header der Pakete prüfen: Quell- und Zielnetz und Quell- und Zielport. Zusätzlich kann das Interface, auf dem ein Paket eingegangen ist, und das Schicht-4-Protokoll (TCP oder UDP) berücksichtigt werden.

3.2. Kommunikation zwischen Netzwerkkarte und Host

Um Daten wie Regelsätze, aber auch Netzwerkpakete, zwischen der Netzwerkkarte und dem Host auszutauschen gibt es zwei unterschiedliche Möglichkeiten. Zur Übergabe von Paketen stellt die Netzwerkkarte über Single Root Input/Output Virtualization (SR-IOV) virtuelle Netzwerkschnittstellen an das Hostsystem zur Verfügung. Für alle weitere Kommunikation ist ein Runtime Environment (RTE) des Herstellers verfügbar, das eine API bereitstellt.

3.2.1. Datenaustausch über das RTE

Da es sich bei dem RTE um proprietäre Software des Herstellers der Netzwerkkarte handelt, soll es hier nicht konkret beschrieben werden. Stattdessen werden die grundlegenden Funktionalitäten dargestellt, die für die Implementierung des Paketfilters benötigt werden. Bei einer möglichen Portierung auf die Hardware eines anderen Herstellers können diese als Anforderungen an eine vergleichbare Schnittstelle genutzt werden.

Laden eines neuen Designs/einer neuen Firmware Zu Beginn der Nutzung der Netzwerkkarte als Unterstützung für den Host muss die eigene Firmware eingespielt werden. Danach sind sowohl der Zugriff auf die Tabellen der implementierten Pipeline aus dem RTE möglich, als auch die virtuellen Netzwerkinterfaces am Host richtig initialisiert und nutzbar. Die Firmware ist im Fall der hier verwendeten Netzwerkkarte eine Binärdatei, die vom proprietären Compiler des Herstellers erzeugt wurde.

Laden eines Regelsatzes Nachdem die Firmware geladen und die Netzwerkkarte initialisiert wurde, wird vom Host ein vorher definierter Regelsatz auf die Karte geladen. Der Regelsatz besteht dabei aus den einzelnen Einträgen, die in den Match-Action-Tables der Netzwerkkarte abgelegt werden sollen.

Hinzufügen und Entfernen von einzelnen Einträgen Für das State-Tracking ist es nötig, einzelne Einträge in bestimmte Match-Action-Tables schreiben und wieder löschen zu können. Dies dient dazu, dass die Controller-Software automatisiert Regeln für die erkannten Verbindungen hinzufügen kann, welche dafür sorgen, dass die Pakete dieser Verbindung ab dann nur noch von der Karte verarbeitet werden. Wird ein Verbindungsabbau durch den Controller erkannt, wird die dazugehörige Regel wieder von der Karte gelöscht.

Abfrage von Zählern Die Abfrage der Zähler, die auf der Karte gespeichert werden, ist zwar für die Funktionalität des hybriden Paketfilters nicht kritisch, jedoch eine sehr hilfreiche Eigenschaft beim Debugging. Es erleichtert dabei sowohl die Fehlersuche in der eigenen Firmware als auch in der aktuellen Netzwerkkonfiguration.

3.2.2. Paketaustausch über SR-IOV

Bei SR-IOV handelt es sich um eine Technologie, die eigentlich dazu dient, ein über PCIe angeschlossenes Gerät performant in mehreren virtuellen Maschinen verwenden zu kön-

nen. Ein PCIe-Gerät ist im Allgemeinen ein Set von einer oder mehreren Funktionen. Diese PCIe-Funktionen sind die primären Einheiten auf dem PCIe-Bus und haben je einen eindeutigen Request Identifier (RID). Bei SR-IOV-fähigen Geräten wird eine Unterscheidung zwischen Physical Functions (PFs) und Virtual Functions (VFs) vorgenommen. Eine PF ist eine normale PCIe-Funktion, der mehrere VFs zugeordnet sind, die ebenfalls eigene eindeutige RIDs haben. Dabei teilen sich die VFs zwar die wichtigsten Ressourcen des Gerätes, stellen aber getrennte performance-kritische Ressourcen, wie beispielsweise Speicherbereiche für DMA, für die Software zur Verfügung. Dadurch ist es möglich, die unterschiedlichen VFs eines Geräts an unterschiedliche virtuelle Maschinen durchzureichen, die diese dann performant verwenden können. [vgl. 6]

Die VFs können jedoch auch vom Host direkt verwendet werden. Auf diese Art und Weise kommuniziert die hier verwendete Netzwerkkarte mit der Software auf dem Host-System. Die Karte kann mit dem verwendeten RTE bis zu 64 VFs bereitstellen [32], die vom Betriebssystem als Netzwerkinterfaces erkannt werden und verwendet werden können.

Jedem physikalischen Port der Netzwerkkarte werden zwei virtuelle Interfaces zugeordnet. Eines wird vom Host als normales Netzwerkinterface verwendet. Dadurch kann der Host sowohl notwendige Funktionalitäten, wie beispielsweise das Bearbeiten von ARP-Paketen, bereitstellen, als auch komplexe Operationen eines Routers, die auf der SmartNIC nicht abgebildet werden, durchführen. Ein Beispiel hierfür wäre das Ändern von Adressen im Rahmen der Network Address Translation. Das zweite Interface dient der Kommunikation mit dem Controller. Hierhin werden alle Pakete geleitet, die für das State-Tracking relevant sein könnten. Außerdem kann der Controller hierüber die Pakete auch wieder nach außen weiterleiten. Durch die Verwendung von zwei Interfaces wird eine saubere Trennung der Funktionalitäten erreicht.

3.3. Controller-Software auf dem Host

Auf dem Host-System, in dem die Netzwerkkarte installiert ist, wird eine Software ausgeführt, die das State-Tracking für zustandsbehaftete Verbindungen übernimmt. In dieser Arbeit wurde die Controller-Software nur für TCP-Verbindungen umgesetzt. Dazu muss der Controller möglichst alle Pakete mitlesen können, die für das State-Tracking notwendig sind. Ein möglichst großer Teil der Pakete einer Verbindung sollte jedoch nur durch die Netzwerkkarte verarbeitet werden, um einen Performance-Vorteil aus der hybriden

Architektur ziehen zu können.

Die Trennung der Funktionalität in zwei Komponenten, ähnelt stark dem Ansatz des SDN. Die Controller-Software stellt dabei die losgelöste Control Plane dar, die komplexe Operationen wie das State-Tracking durchführt. Die Netzwerkkarte entspricht der Data Plane, die anhand der vom Controller übergebenen Informationen die Pakete weiterleitet.

3.4. Zusammenspiel der Komponenten im Betrieb

In Abbildung 3.1 ist dargestellt, wie die einzelnen Komponenten beim Tracking einer TCP-Verbindung zusammenarbeiten. Der Ablauf wird in diesem Abschnitt genauer beschrieben.

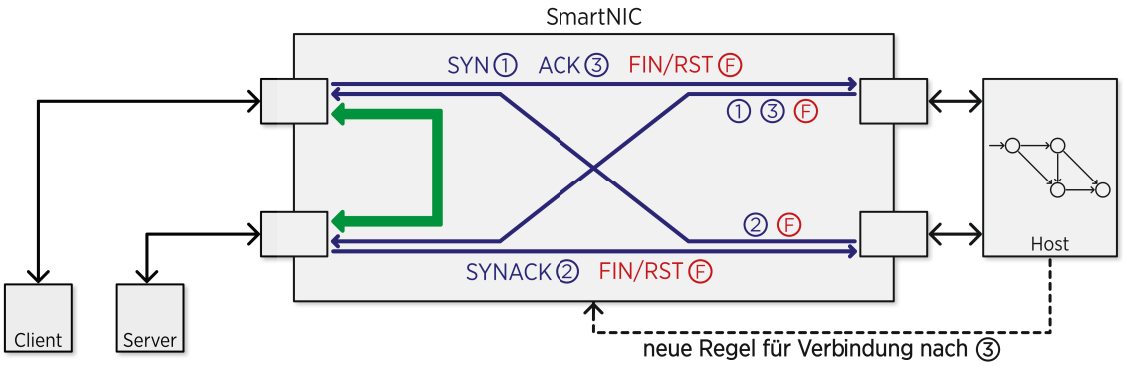


Abbildung 3.1.: Konzept des Paketfilters

Bis zum vollständigen Aufbau der Verbindung werden alle Pakete für die Regel, für die das State-Tracking erfolgen soll, über ein virtuelles Interface an den Controller geschickt. Für jeden physikalischen Port existiert ein solches virtuelles Interface, so dass eine eins-zu-eins-Zuordnung zwischen diesen möglich ist. Die Pakete werden für das State-Tracking ausgewertet und dann auf dem selben virtuellen Interface, auf dem sie empfangen wurden, wieder gesendet. Diese vom Controller erneut gesendeten Pakete werden dann von der Netzwerkkarte über den richtigen physikalischen Port nach außen geschickt. In Abbildung 3.1 ist dies durch die blauen Pfeile dargestellt.

Ist ein erfolgreicher Verbindungsaufbau durch den Controller erkannt worden, fügt er über das RTE für die beiden Richtungen je einen Flow auf der Karte hinzu. Ein Flow ist dabei ein Vier-Tupel, bestehend aus Quell- und Ziel-IP-Adresse und Quell- und Zielport. Das führt dazu, dass alle Pakete, die zu dieser Verbindung gehören, von diesem Moment an nur noch durch die Karte verarbeitet werden und nicht mehr an den Controller geschickt

werden. Eine Ausnahme bilden jedoch solche Pakete, die für das State-Tracking relevant sein können. Daher werden alle Pakete, bei denen mindestens eines der Flags SYN, FIN oder RST gesetzt ist, nicht direkt weitergeleitet, sondern an den Controller übermittelt. Das ist nötig, um den Verbindungsabbau erkennen zu können, der dazu führt, dass die Flows wieder von der Karte entfernt werden.

4. Implementierung

In Kapitel 3 wurde ein Konzept für einen hybriden Paketfilter mit einer P4-programmierbaren Netzwerkkarte vorgestellt, dessen Implementierung nun erläutert wird. Die Umsetzung von Regeln wird anhand von Beispielen beschrieben und es wird auf Limitierungen der gewählten Implementierung eingegangen.

4.1. Hardware

In einem Host mit einem Intel Xeon X5680 Prozessor und 6 GB RAM wurde die programmierbare Netzwerkkarte über PCIe 2.0 angebunden. Bei der verwendeten Netzwerkkarte handelt es sich um die Agilio CX 2x40GbE des Herstellers Netronome. Sie verfügt über zwei physikalische Ports für 40 Gbit/s-Ethernet über QSFPs.

Die Netzwerkkarte basiert auf einem Prozessor mit der Bezeichnung NFP-4000, der auf die Verarbeitung von Paketen ausgelegt ist. Dieser zeichnet sich unter anderem dadurch aus, dass eine hochparallele Verarbeitung möglich ist. Er besitzt, neben einigen weiteren Komponenten, 60 sogenannte Flow Processing Cores (FPCs), die jeweils acht Threads unterstützen, wodurch bis zu 480 Pakete gleichzeitig verarbeitet werden können. Die in P4 implementierte Logik wird auf diesen FPCs abgebildet. [vgl. 31, 33]

Da die Implementierung der Komponenten des Paketfilters jedoch in hardwareunabhängigen Sprachen erfolgte, sind weitergehende Details über die verwendete Hardware nicht relevant.

4.2. Software

Die umgesetzte Software teilt sich in zwei große Bereiche auf: die Firmware für die Netzwerkkarte und die Controller-Software auf dem Host. Beide Implementierungen werden im Folgenden genauer beschrieben.

4.2.1. Firmware (P4)

Für die Implementierung der Firmware, die auf der Netzwerkkarte ausgeführt wird, kommt die Programmiersprache $P4_{16}$ zum Einsatz. Diese bietet ein hohes Abstraktionslevel gegenüber der ebenfalls unterstützten Programmierung der Netzwerkkarte in C, was die Entwicklungsgeschwindigkeit deutlich erhöht. Ein Nachteil ist jedoch, dass man an die einfachen Strukturen der domänenspezifischen Programmiersprache gebunden ist.

Header und Parser

Zu Beginn wurde eine kleine Anzahl von Headern implementiert, die nötig sind, um einen Prototypen für einen hybriden Paketfilter zu entwickeln. Umgesetzt wurden dabei die Header von Ethernet, IPv4, IPv6, UDP und TCP, wobei nur die normalen Header ohne zusätzliche Options betrachtet wurden. Der Aufbau der Header wurde jeweils aus Tanenbaum [39] übernommen.

Der implementierte Parser ist sehr unkompliziert aufgebaut. Es gibt je einen Zustand für jedes Protokoll, der den jeweiligen Header parst. Die Übergänge in den nächsten Zustand hängen dann nur von den im Header angegebenen Feldern wie Ethertype oder Next-Header ab. In den Zuständen für die Transportschicht wird jeweils noch ein Feld in den Metadaten des Pakets gesetzt, welches das Protokoll angibt.

Actions

Mit den Paketen können jeweils, wie in Abschnitt 3.1 beschrieben, die drei Actions *drop*, *accept* und *forward* ausgeführt werden. Da in einer Action keine bedingten Anweisungen enthalten sein dürfen, mussten für IPv4 und IPv6 jeweils eigenen Versionen der *forward*-Action implementiert werden, die auf die unterschiedlichen Felder der Header zugreifen. Den Actions *accept* und *forward* muss je ein Parameter übergeben werden, der angibt, auf welchem Port das Paket die Netzwerkkarte wieder verlassen soll. Welche Action mit welchem Parameter jeweils ausgeführt wird, hängt von den Einträgen in den Match-Action-Tables ab.

Match-Action-Tables

Die Match-Action-Tables stellen den wichtigsten Teil der Implementierung dar. Sie erlauben die unterschiedliche Behandlung der Pakete auf Basis unterschiedlicher Felder in ihren Headern. Außerdem können die Einträge in den Tabellen zur Laufzeit durch Software auf dem Host verändert werden, was eine Umsetzung von unterschiedlichen Regeln ohne erneutes Kompilieren der Firmware erst ermöglicht. Ein Paket durchläuft in der Implementierung bis zu zwei wichtige Tabellen: zuerst die Flow-Tabelle, die Einträge für bekannte Verbindungen enthält, und dann die Regel-Tabelle, die das durch den Paketfilter auszuführende Regelwerk enthält. Gehört ein Paket zu einer bekannten Verbindung, findet sich also ein Eintrag für das Paket in der Flow-Tabelle, so wird die Regel-Tabelle nicht mehr angewendet, sondern das Paket direkt an einen Port weitergeleitet.

Die Flow-Tabelle wurde auf der Karte in zwei Match-Action-Tables zerlegt, eine für IPv4 und eine für IPv6. Die Regel-Tabelle wurde sogar in drei Match-Action-Tables zerlegt, da hier zusätzlich zu IPv4 und IPv6 noch eine Tabelle benötigt wird, welche die Regeln enthält, die auf Nicht-IP-Pakete angewendet werden sollen.

Flow-Tabellen Üblicherweise werden Netzwerkverbindungen durch das Fünftupel aus Protokoll, Quell- und Ziel-IP-Adresse und Quell- und Ziel-Port identifiziert [39]. Auf das Protokoll wurde jedoch verzichtet, da in dieser Arbeit nur TCP-Verbindungen betrachtet werden. Die Flow-Tabellen überprüfen daher je Paket nur vier Felder in den Headern auf exakte Übereinstimmung: die Quell- und Ziel-IP-Adresse und den Quell- und Ziel-Port.

Zur Vereinfachung der Verarbeitung auf der Karte, werden pro Verbindung zwei Einträge in der Tabelle eingefügt: je einer für die Flows in Hin- und in Rückrichtung. In den beide Einträgen werden die Quelle und das Ziel vertauscht und der physikalische Ausgabeport jeweils angepasst.

Als Actions kommen die zur jeweiligen IP-Version passende *forward*-Action und die sogenannte *No-Action* in Frage, wobei diese nur der Default-Eintrag ist und ausgeführt wird, wenn kein anderer Eintrag zutrifft. Sie bewirkt keinerlei Veränderungen an dem Paket oder den weiteren Verarbeitungsschritten. In den Regeln, die zur Beschreibung eines Flows eingefügt werden, kommt daher sinnvollerweise nur die *forward*-Action vor, die das Paket direkt an einen physikalischen Port zur Ausgabe weiterleitet.

Regel-Tabellen Die Regel-Tabellen enthalten das eigentliche Regelwerk des Paketfilters. Sie bestimmen anhand verschiedener Felder, wie ein Paket verarbeitet werden soll. Angelehnt an einen einfachen Paketfilter können die Tabellen Quell- und Ziel-IP-Adresse, Quell- und Ziel-Port von UDP und TCP, das Transportprotokoll des Pakets und den Port, auf dem das Paket eingegangen ist, überprüfen. Die Werte aus den Einträgen müssen dabei exakt mit dem Paket übereinstimmen, bei den IP-Adressen kann jedoch eine Maske angegeben werden, um auch auf ganze Netzwerke prüfen zu können.

In den Einträgen können alle Actions verwendet werden. Wird jedoch von einem physikalischen Port auf einen andern physikalischen Port direkt weitergeleitet, sollte die *forward*-Action genutzt werden, um die TTL beziehungsweise das Hop Limit zu reduzieren und somit den IP-Standards zu entsprechen.

Indem bei den Regeln der Port angegeben wird, auf dem das Paket ausgegeben werden soll, werden in dieser Tabelle die eigentlichen Regeln des Paketfilters mit Routing-Informationen vermengt. Für diesen Prototypen kann eine solche Vereinfachung jedoch vorgenommen werden, da nur die grundlegende Funktionsweise demonstriert werden soll. Um die Informationen sauber zu trennen, sollte in einer Implementierung, die produktiv eingesetzt werden soll, eine weitere Tabelle eingeführt werden, die nur die Routing-Informationen enthält.

Counter

Um nachvollziehen zu können, auf wie viele Pakete eine Regel zutraf, gibt es die Möglichkeit, Zähler zu implementieren. Sogenannte *direkte Zähler* können mit einer Match-Action-Table verbunden werden und haben genauso viele einzelne Zähler, wie die Tabelle Einträge haben kann. Diese werden automatisch für jedes Paket, das durch eine Regel in der Tabelle bearbeitet wird, erhöht. In der aktuellen Version des Compilers von Netronome werden diese direkten Zähler jedoch nicht unterstützt. Es wurde im Rahmen der Arbeit auch kein Aufwand in die manuelle Implementierung einer solchen Funktionalität gesteckt.

Paketverarbeitung vor der Ausgabe

Vor dem Ausgeben der Pakete müssen für die korrekte Funktion der anderen Netzwerkgeräte die MAC-Adressen im Ethernet-Header der Pakete korrekt gesetzt werden. Da es nicht möglich ist, auf der Netzwerkkarte ARP-Pakete zu verarbeiten und automatisch eine ARP-Tabelle zu pflegen, wurde eine stark vereinfachte Lösung für den Prototyp angewendet. Es wurden zwei weitere Match-Action-Tables eingeführt. Die eine setzt die Quell-MAC-Adresse in Abhängigkeit vom Ausgabeport richtig, die andere setzt die Ziel-MAC-Adresse in Abhängigkeit von der Ziel-IP-Adresse. In einer produktiv einsetzbaren Implementierung sollte die Ziel-MAC-Adresse in Abhängigkeit von der IP-Adresse des nächsten Hops, der aus einer Tabelle mit Routing-Informationen bestimmt wird, ermittelt werden.

4.2.2. Controller

Die Implementierung einer zusätzlichen Controller-Software ist nötig, da P4 keine eigenen Mechanismen zur Verwaltung von Zustandsinformationen bereitstellt. Die Aufteilung der Funktionalitäten folgt dabei der bereits beschriebenen Idee des SDN, nach der es einen Controller, der komplexe Operationen übernimmt, und eine davon losgelöste Data Plane, welche die Weiterleitung von Paketen nach statischen Regeln durchführt, gibt. Die Umsetzung des Controllers erfolgte in Python, da hierfür eine Bibliothek zur Kommunikation mit dem RTE zur Verfügung stand.

Da für den Paketfilter wichtig ist, in welchem Zustand sich eine Verbindung befindet, um passende Regeln auf der SmartNIC einfügen zu können, ist es nötig, diesen Zustand aus den Paketen festzustellen. Dieses State-Tracking erfolgt in der Implementierung ausschließlich durch den Controller. Der verwendete Zustandsautomat für TCP-Verbindungen, der dem Verfolgen des Zustandes der Verbindung dient, unterscheidet sich von dem eines Endpunktes einer Verbindung, da der Paketfilter als Man-in-the-Middle agiert. Ein solcher Automat ist in der Literatur kaum beschrieben, weshalb er für die Arbeit aus netfilter aus dem Linuxkernel [28] übernommen wurde. Er ist in Abbildung 4.1 dargestellt. Auffällig ist besonders, dass für manche Übergänge zwischen den Zuständen die Richtung, in die das betreffende Paket geschickt wird, relevant ist. Dadurch lässt sich die korrekte Abfolge des Verbindungsaufbaus und Verbindungsabbaus abbilden. Ebenfalls erkennbar ist der Zustand *SYN_SENT2*, der eingeführt wurde, um eine Unterstützung für Simultaneous-Open, also das gleichzeitige Öffnen der Verbindung von beiden Endpunkten, zu bieten. Im Linuxkernel wird zusätzlich zu den Flags jeweils noch geprüft, ob das im Paket enthaltene

Segment zum aktuellen Übertragungsfenster passen kann. Dazu werden Grenzen definiert, innerhalb derer sich die Sequenznummern befinden müssen [17]. Diese Überprüfung findet im Controller jedoch nicht statt.

Zu Beginn erhält der Controller alle Pakete, die zu einer Verbindung gehören könnten, führt anhand dieser eventuelle Zustandsübergänge durch und leitet Pakete, welche im aktuellen Zustand der Verbindung nicht invalide sind, weiter. Erkennt der Controller eine Verbindung als aufgebaut, wechselt sie also in den Zustand *ESTABLISHED*, schreibt der Controller über das RTE je einen Flow pro Richtung in die Flow-Tabelle auf der Netzwerkkarte. Ein Flow ist dabei das in Abschnitt 4.2.1 dargestellte Viertupel, das die Verbindung identifiziert. Dadurch werden in Zukunft die meisten Pakete der Verbindung schon durch die Netzwerkkarte weitergeleitet und belasten den Host nicht. Lediglich Pakete, die für das State-Tracking relevant sein könnten, werden jetzt noch an den Controller geleitet. Dadurch kann der Controller einen Verbindungsabbau erkennen und die zu dieser Verbindung gehörigen Flows wieder von der Karte entfernen.

Die konkrete Umsetzung hat den Nachteil, dass der Controller keinerlei Informationen über die Regeln des Paketfilters hat, sondern sich darauf verlässt, dass ihm durch die Netzwerkkarte nur Pakete zu validen Verbindungen weitergeleitet werden. Für die prototypische Implementierung ist das jedoch ausreichend, zumal eine Überprüfung mit dem Regelwerk einfach implementiert werden könnte.

4.3. Umsetzung von Regeln

Das Verhalten des Paketfilters wird durch das Regelwerk bestimmt, das ihm übergeben wird. Die einzelnen Regeln eines Regelwerkes werden durch ein oder mehrere Einträge in der Regel-Tabelle abgebildet. Die Reihenfolge der Einträge entspricht auch der Reihenfolge, in der sie von der Netzwerkkarte betrachtet werden [32]. Dabei wird die First-Match-Mechanik angewendet, das heißt ein Paket wird nur vom ersten Eintrag bearbeitet, der darauf zutrifft, auch wenn weiter unten in der Tabelle noch Einträge stehen, die ebenfalls auf das Paket zutreffen würden. Im Folgenden wird zwischen „Regeln“, die die Verarbeitung bestimmter Pakete definieren und zusammen das Regelwerk bilden, und „Einträgen“ in den Match-Action-Tables auf der Netzwerkkarte unterschieden. Wird von Quell- und Ziel-Ports gesprochen, sind Ports der Transportschicht gemeint. Mit Eingangs- und Ausgabeports werden die physikalischen oder virtuellen Ports der SmartNIC bezeichnet, über

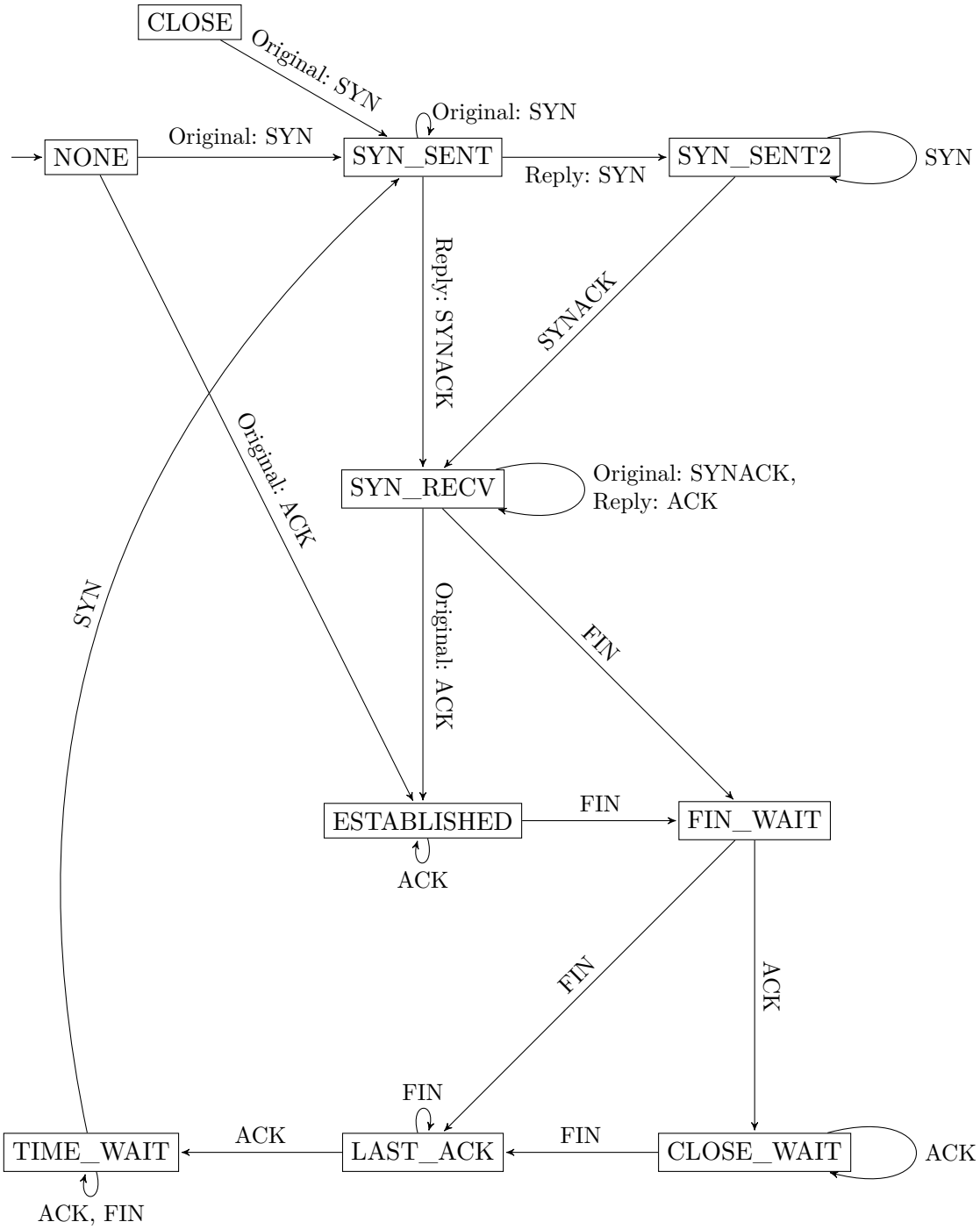


Abbildung 4.1.: Der Zustandsautomat, der für des State Tracking im Controller verwendet wird [28]. *Original* oder *Reply* geben die Richtung des Pakets an.
Aus allen Zuständen außer NONE gibt es einen zusätzlichen Übergang zu CLOSE bei einem Paket mit gesetztem RST-Flag, der hier zur Vereinfachung ausgelassen wurde.

die der Netzwerkverkehr fließt.

4.3.1. Regelgenerator

Das RTE erfordert, dass die Einträge in den Tabellen auf der Karte in einer sehr ausführlichen JSON-Notation übergeben werden müssen. Um diese nicht für jedes neue Regelwerk von Hand schreiben zu müssen, wurde ein Hilfsprogramm entwickelt, welches die Einträge automatisch erzeugen kann. Dazu wird dem Programm eine Textdatei übergeben, in der die Regeln in einer an die Konfiguration des Paketfilters pf aus OpenBSD angelehnten Notation [35] beschrieben sind. Diese Datei wird dann eingelesen und in die JSON-Darstellung überführt. Die Reihenfolge der Regeln wird dabei übernommen. Es existiert eine spezielle Notation, um auch die Match-Action-Tables zu konfigurieren, die die MAC-Adressen der ausgehenden Pakete richtig setzen. Die Regeln müssen die auszuführende Aktion, die IP-Version zur Zuordnung in die richtige Match-Action-Table und den Ausgabeport enthalten. Außerdem werden die zu überprüfenden Felder angegeben.

4.3.2. Zustandslose Regeln

Eine zustandslose Regel wird in genau einen Eintrag in einer Regel-Tabelle übersetzt. Die Regel kann dabei die Felder Quell- und Ziel-IP-Adresse, Quell- und Ziel-Port der Header, das Transportprotokoll des Pakets und den Port, auf dem das Paket eingegangen ist, überprüfen. Dabei müssen nicht alle Felder immer überprüft werden. Während die Ports und das Protokoll immer exakt mit den Werten in der Regel übereinstimmen müssen, kann bei den IP-Adressen auch geprüft werden, ob sie in einem bestimmten Netzwerk liegen. Die Netze werden in den Regeln immer in CIDR-Schreibweise angegeben und für die Netzwerkkarte in Masken übersetzt. Außerdem enthält die Regel die Action, welche ausgeführt werden soll, wenn die Regel auf ein Paket zutrifft. Zwingend erforderlich in einer Regel sind die Angaben, für welches Protokoll der Netzwerkschicht (IPv4, IPv6 oder keines) sie gelten soll und auf welchem Port das Paket wieder ausgegeben werden soll. Ein Beispiel für eine solche Regel ist in Listing 4.1 dargestellt.

```
forward inet in-on p0 proto udp to 172.16.222.0/24 port 100 out-on p4
```

Listing 4.1.: Zustandslose Regel

4.3.3. Zustandsbehaftete Regeln

Eine zustandsbehaftete Regel wird im Regelwerk mit dem Schlüsselwort `keep-state` gekennzeichnet. Sie ist in der sonstigen Verwendung den zustandslosen Regeln sehr ähnlich, jedoch muss der Eingangsport gesetzt sein und als Transportprotokoll kommt nur TCP in Frage. Eine beispielhafte Regel ist in Listing 4.2 wiedergegeben.

```
pass inet in-on p0 proto tcp to 172.16.222.0/24 port 80 out-on p4
↪ keep-state
```

Listing 4.2.: Zustandsbehaftete Regel mit dem Schlüsselwort `keep-state`

Die Pakete, auf die eine zustandsbehaftete Regel zutrifft, werden nicht direkt durch die Karte an ihren Ausgabeport geleitet, sondern müssen zuerst den Controller auf dem Host passieren wie in Abbildung 3.1 dargestellt ist. Dort ist außerdem zu sehen, dass dies auch für die Rückrichtung der Pakete beachtet werden muss, wodurch eine zustandsbehaftete Regel in insgesamt vier Einträge übersetzt wird.

Im ersten Eintrag werden die zu überprüfenden Felder aus der gegebenen Regel übernommen und lediglich der Ausgabeport durch den zum Eingangsport gehörenden Controller-Port ersetzt. Für den zweiten Eintrag wird der Eingangsport auf den Controller-Port aus dem ersten Eintrag und der Ausgabeport auf den in der Regel spezifizierten Ausgabeport gesetzt. Der dritte und der vierte Eintrag werden analog gebildet, jedoch werden dafür der Ein- mit dem Ausgabeport sowie Quell-IP-Adresse und -Port mit Ziel-IP-Adresse und -Port getauscht. Wie die Regel aus Listing 4.2 umgewandelt würde, ist beispielhaft in Listing 4.3 dargestellt.

```
pass inet in-on p0   proto tcp to 172.16.222.0/24   port 80 out-on v0.2
pass inet in-on v0.2 proto tcp to 172.16.222.0/24   port 80 out-on p4
pass inet in-on p4   proto tcp from 172.16.222.0/24 port 80 out-on v0.3
pass inet in-on v0.3 proto tcp from 172.16.222.0/24 port 80 out-on p0
```

Listing 4.3.: Zustandsbehaftete Regel aus Listing 4.2 umgewandelt in zustandslose Regeln

Durch diese Umsetzung der Regeln ergibt sich der Nachteil, dass zustandsbehaftete Regeln in beide Richtungen gelten, das heißt einen Verbindungsaufbau in beide Richtungen erlauben. Im Controller könnte jedoch eine weitere Regelprüfung implementiert werden, die den Verbindungsaufbau nur in eine Richtung erlaubt. Für diesen Prototypen wurde

darauf jedoch verzichtet.

5. Evaluation

Diese Kapitel stellt die Evaluation der Implementierung aus Kapitel 4 dar. Es werden zunächst der zum Messen verwendete Aufbau beschrieben und danach die erhaltenen Ergebnisse diskutiert. Zur Evaluation von zustandslosen sowie zustandsbehafteten Regeln wurden zwei Messreihen aufgestellt. Die Erzeugung zustandsbehafteten Datenverkehrs stellte eine besondere Herausforderung dar, sodass eine Anpassung des zustandslosen Ansatzes nicht zielführend erschien und ein anderes Verfahren gewählt wurde.

Bei der Auswertung wurde hauptsächlich die Datenrate betrachtet, die bei einer Verbindung durch den implementierten Paketfilter erzielt werden kann. Da die Pakete durch die Implementierung nur in sehr wenigen Fällen wirklich verändert werden, sondern meistens nur geparst, mit den Regeln verglichen und dann durchgeleitet werden, wurde auf eine genaue Verifikation der korrekten Funktionsweise der Paketverarbeitung verzichtet. Die korrekte Funktionalität der Bitoperationen auf den Paketen wird vorausgesetzt, da diese einen Kernbestandteil der Sprache P4 darstellen. Das erfolgreiche Aufbauen von TCP-Verbindungen durch den Paketfilter, die zur Messung nötig waren, zeigt jedoch, dass keine grundlegenden Fehler in den verarbeiteten Paketen vorhanden sind.

5.1. Aufbau

Um die erzielbare Geschwindigkeit messen zu können, werden zwei unterschiedliche Systeme verwendet, die beide in Hardware ausgeführt sind. Der genaue Aufbau des ersten Systems, das die Netzwerkkarte enthält und damit das Device under Test (DuT) darstellt, wurde bereits in Abschnitt 4.1 beschrieben. Der zweite Host, der als Lastgenerator verwendet wird, muss leistungsstark sein, um in ausreichender Geschwindigkeit Pakete generieren zu können. Daher kommt hier ein System mit zwei Intel Xeon E5-2620 v4 Prozessoren und 64 GB RAM zum Einsatz. Außerdem wurde es mit zwei AOC-S40G-i1Q Netzwerkkarten von Supermicro ausgestattet, die auf dem Intel XL710-Chipsatz basieren

und daher für hochperformante Netzwerkkommunikation mit 40 Gbit/s verwendet werden können. Die beiden Netzwerkkarten wurden über PCIe 3.0 an den selben Prozessor angebunden. Dies ist nötig, da es zur stabileren Funktion des verwendeten Paketgenerators MoonGen beiträgt. Beim Zusammenbau und der Konfiguration der performancerelevanten Einstellungen wurden die Hinweise aus [25] beachtet, um mit MoonGen, das auf DPDK basiert, möglichst gute Ergebnisse erzielen zu können. Die beiden Hosts wurden direkt über zwei Glasfasern miteinander verbunden, um eine Beeinflussung der Messung durch weitere Geräte auszuschließen. In Abbildung 5.1 ist der Aufbau noch einmal dargestellt.

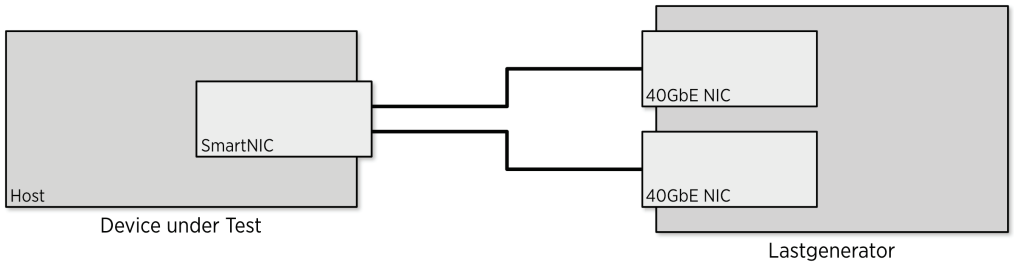


Abbildung 5.1.: Aufbau des Evaluationssetups

Die folgenden Abschnitte beschreiben die unterschiedliche zur Messung verwendete Software und die jeweiligen Gründe für ihren Einsatz.

5.1.1. MoonGen

MoonGen ist ein performanter Paketgenerator, dessen Logik durch ein vom Benutzer in Lua geschriebenes Programm definiert werden kann [7]. Dabei kann trotz der hohen Geschwindigkeiten die Erzeugung der Pakete genau durch das Programm beschrieben werden. Bei der Entwicklung der Just-In-Time-kompilierten Lua-Programme ist recht viel Freiheit möglich, wodurch sich eine Vielzahl von möglichen Messszenarien abbilden lässt. Daher bietet sich MoonGen an, um die Funktionsweise und den Durchsatz eines hochperformanten Paketfilters zu prüfen. Weitere Funktionen wie Latenzmessungen erlauben eine noch weitergehende Analyse von Netzwerkkomponenten unter hoher Last, wurden aber für diese Arbeit nicht genutzt.

Um die Messungen durchzuführen wurde ein einfaches Skript für MoonGen entwickelt, dass stets die gleichen UDP-Pakete in angegebener Größe mit einer gegebenen Anzahl von Threads generiert. Diese werden auf einem Interface gesendet und sollen auf einem weiteren Interface wieder empfangen werden. Es werden sowohl die Daten- als auch die

Paketrate der Sende- und Empfangsseite ermittelt, die spätere Auswertung berücksichtigt jedoch nur die Empfangsraten. Für die Messreihen wurde das Skript mit unterschiedlichen Paketgrößen jeweils für 60 Sekunden ausgeführt.

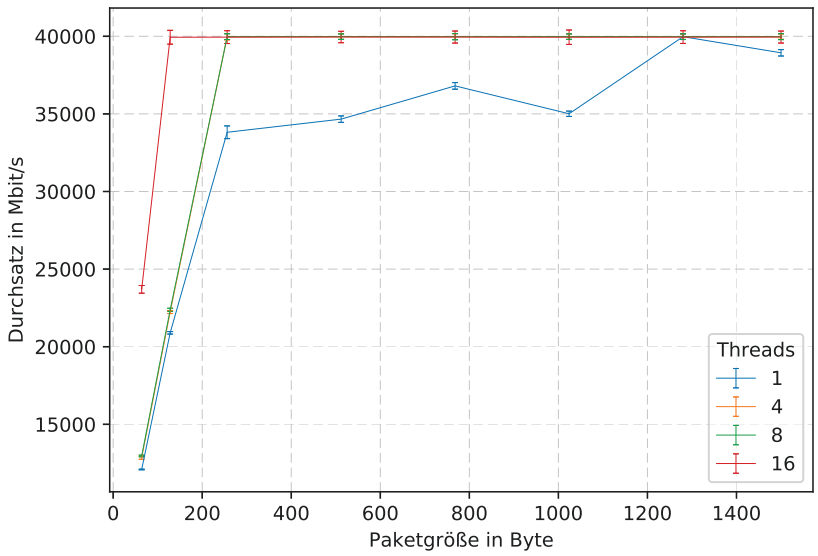
Da jedoch auch mit einer optimierten Software wie MoonGen das Erzeugen von Paketen mit einer Datenrate von 40 Gbit/s nicht trivial möglich ist, wurden zuerst die Fähigkeiten des Lastgenerators ermittelt. Dazu wurden die beiden Netzwerkinterfaces mit einer Glasfaser direkt verbunden, sodass sich kein weiteres Gerät dazwischen befand.

Die Ergebnisse der Evaluierung des Lastgenerators sind in Abbildung 5.2 dargestellt. Gezeigt sind die erzielbaren Daten- und Paketraten bei unterschiedlichen Paketgrößen mit der Standardabweichung bei der jeweiligen Messung. Um eine Aussage über die Performance des Lastgenerators treffen zu können wurden die Pakete mit unterschiedlich vielen Threads generiert. Es lässt sich erkennen, dass ein einzelner Thread zum Erreichen von 40 Gbit/s nicht ausreichend ist. Mit einer größeren Anzahl von Threads kann ab einer Paketgröße von 256 Byte die maximale Datenrate erreicht werden. Bei einer Paketgröße von 128 Byte werden 16 Threads benötigt, um die Verbindung voll auszulasten. Bei einer Paketgröße von 64 Byte ist es auch mit 16 Threads nicht möglich, 40 Gbit/s zu erreichen. Dies kann darauf zurückgeführt werden, dass die verwendeten Netzwerkkarten von Intel aufgrund von Hardwarelimitationen nicht in der Lage sind, die dafür notwendige hohe Paketrate zu verarbeiten [7, 27]. Da diese Einschränkung nicht zu umgehen ist, wird sie für den Lastgenerator akzeptiert.

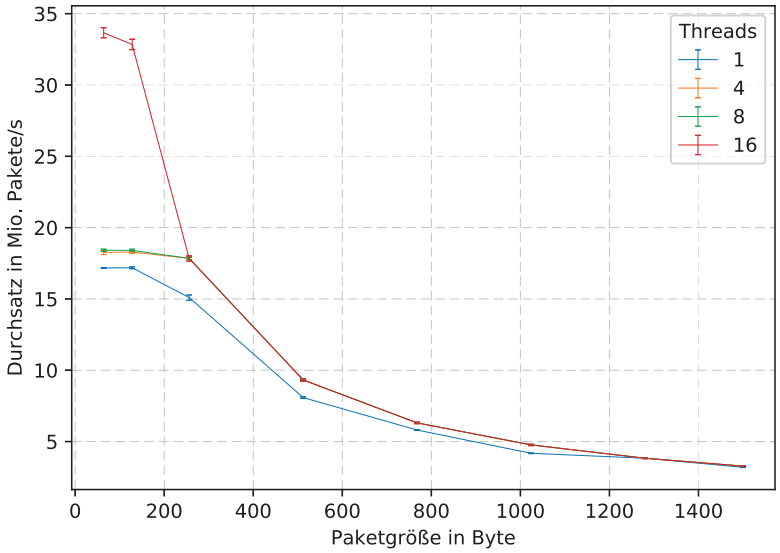
Alle Messungen mit zustandslosem Traffic von MoonGen werden aufgrund dieser Ergebnisse mit 16 Threads durchgeführt, da hier mit möglichst kleinen Paketen schon 40 Gbit/s erreicht werden können und keine zusätzlichen Einschränkungen auftreten.

5.1.2. ApacheBench

Mit MoonGen ist es zwar möglich, TCP-Pakete in hoher Geschwindigkeit zu erzeugen, jedoch gestaltet sich das Aufbauen einer vollständigen TCP-Verbindung eher schwierig, da dazu der gesamte Zustandsautomat in Lua implementiert werden müsste. Daher wurde für die Evaluation der zustandsbehafteten Regeln das Programm ApacheBench [21] verwendet. Dabei handelt es sich eigentlich um ein Programm, welches für Lasttests von Webservern entwickelt wurde [29]. Da dazu aber viele TCP-Verbindungen zu dem Server geöffnet werden müssen, eignet es sich auch als Lasttest zum Test eines zustandsbehafteten Paketfilters.



(a) Erzielte Datenrate



(b) Erzielte Paketrate

Abbildung 5.2.: Evaluation des zustandslosen Lastgenerators

Zur Durchführung aller Tests mit ApacheBench waren Lastgenerator und DuT, wie in Abbildung 5.1 dargestellt, miteinander verbunden. Auf dem Lastgenerator wurde der Webserver nginx ausgeführt. Dieser konnte unterschiedliche Dateien in Größen von 1 KB bis 1 GB ausliefern. Da ApacheBench nur einen Thread startet, was zur Erzeugung von 40 Gbit/s Datenverkehr nicht ausreicht, wurden vier Instanzen parallel gestartet. Diese

haben über eine Zeit von 60 Sekunden jeweils eine bestimmte Zahl von Anfragen gleichzeitig an den Server geschickt. Nach Ablauf der Zeit gibt ApacheBench die erzielte Datenrate und weitere Informationen aus. In die Datenrate werden jedoch nur die übertragenen Nutzdaten mit einbezogen. Auf dem Lastgenerator mussten einige Routen manuell angelegt, iptables-Regeln erzeugt und weitere Einstellungen getroffen werden, damit ApacheBench und der Webserver auf dem gleichen Host ausgeführt werden können, aber der Datenverkehr trotzdem über das DuT fließt [23].

Auch dieser Testaufbau wurde zu Beginn überprüft, um die maximale Datenrate zu bestimmen, die der Lastgenerator erreichen kann. Dazu wurde das DuT in der Verbindung belassen, aber ein Regelwerk auf die SmartNIC geschrieben, das den gesamten TCP-Datenverkehr an den Webserver weiterleitet. Wichtige Ausschnitte des verwendeten Regelwerks sind in Listing 5.1 dargestellt. Dabei sind die Einträge, die Nicht-IP-Verkehr an den Host weiterleiten, wichtig, da dadurch ARP-Requests richtig beantwortet werden, was für die korrekte Funktionalität des Netzwerkstacks des Lastgenerators nötig ist.

```
forward inet in-on p0 proto tcp to 172.16.222.0/24 port 80 out-on p4
forward inet in-on p4 proto tcp from 172.16.222.0/24 port 80 out-on p0

pass noip in-on p0 out-on v0.0
pass noip in-on v0.0 out-on p0

pass noip in-on p4 out-on v0.1
pass noip in-on v0.1 out-on p4
```

Listing 5.1.: Regelwerk für die SmartNIC bei der Evaluation von ApacheBench

In Abbildung 5.3 sind die Ergebnisse der Messungen zur Evaluation des zustandsbehafteten Lastgenerators dargestellt. Es wird deutlich, dass es auch mit großen übertragenen Dateien nicht möglich ist, 40 Gbit/s zu erreichen. Erst ab einer Größe der übertragenen Datei von mindestens 10 MB konnten in einigen Konfigurationen 35 Gbit/s erreicht werden. Insbesondere bei kleinen Dateien mit 1 KB oder 10 KB konnten mit keiner Messmethode mehr als 5 Gbit/s erzielt werden. Dennoch wurde der Lastgenerator in dieser nicht optimalen Form zur Evaluation der Implementierung verwendet. Dabei wurden bei den Messungen ebenfalls vier Instanzen von ApacheBench gestartet, die einmal je vier und einmal je 64 gleichzeitige Anfragen gestellt haben. Die beiden Anzahlen gleichzeitiger Verbindungen wurden gewählt, da mit einer großen und einer kleinen Zahl gleichzeitiger Verbindungen gemessen werden sollte und mit diesen beiden vergleichsweise hohe Daten-

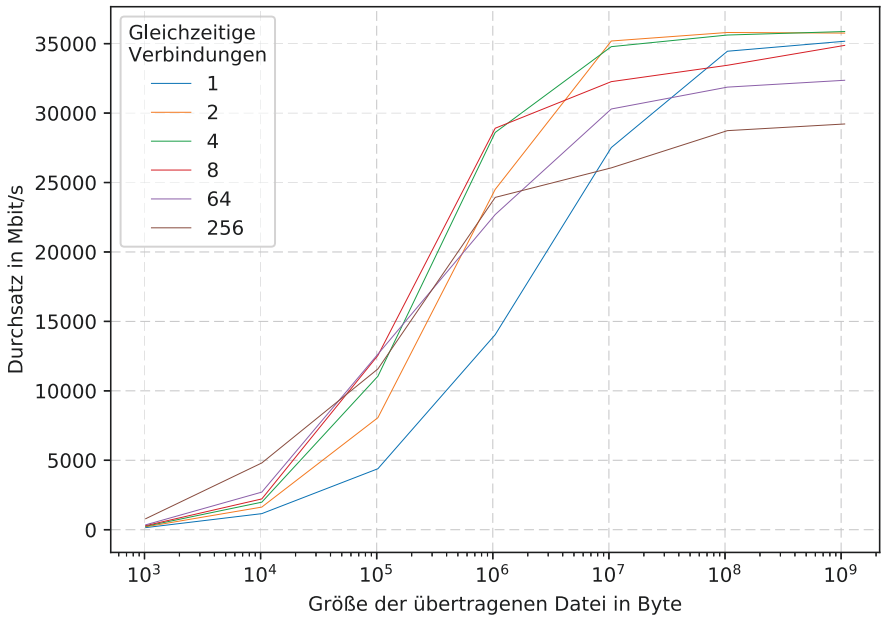


Abbildung 5.3.: Evaluation des Lastgenerators für TCP-Verbindungen

raten erzielt werden konnten, wie im Diagramm zu sehen ist.

5.2. Ergebnisse und Diskussion

Dieser Abschnitt stellt die unterschiedlichen Messungen und die erzielten Ergebnisse dar und diskutiert deren Einzelheiten.

5.2.1. Zustandslose Regeln

Für die Evaluation der Performance des Paketfilters wurde je Messreihe mit der implementierten Firmware auf der SmartNIC und mit dem Softwarepaketfilter iptables auf Linux zum Vergleich durchgeführt. Die Ergebnisse der Messungen sind in Abbildung 5.4 dargestellt.

Für die Messungen des Paketfilters auf der SmartNIC wurde der in Abschnitt 5.1 beschriebene Aufbau mit Lastgenerator und DuT verwendet. Auf der Netzwerkkarte wurde die entwickelte Firmware ausgeführt. In Listing 5.2 ist das Regelwerk dargestellt, das für die Messreihe zum Einsatz kam. Die Einträge der ARP-Tabellen wurden jedoch ausgelassen. Es enthält nur eine Regel, welche die von MoonGen generierten Pakete direkt auf den anderen physikalischen Port weitergeleitet, und eine weitere, die alle anderen Pakete

verwirft. Es werden keine Pakete zur Verarbeitung an den Host weitergeleitet.

```
forward inet in-on p0 proto udp to 172.16.222.0/24 port 100 out-on p4
drop inet in-on p0
```

Listing 5.2.: Regelwerk für die SmartNIC beim Test mit MoonGen

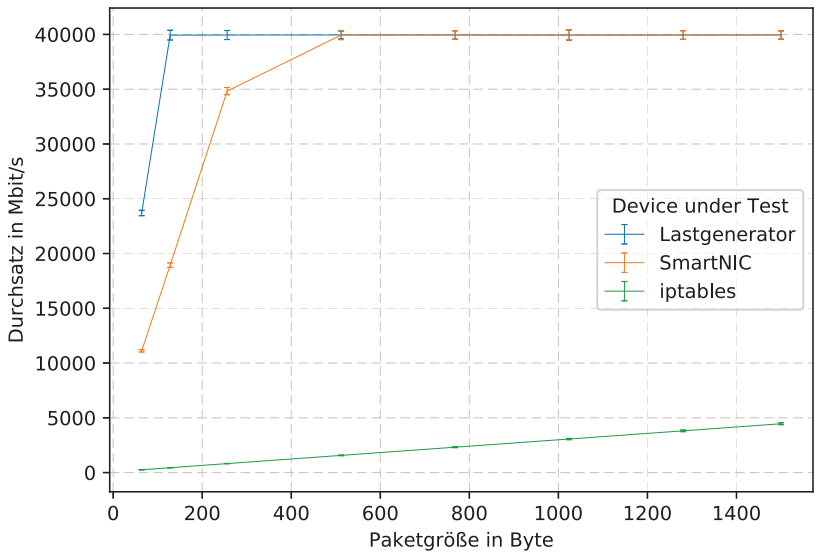
Um die Vergleichsmessung mit dem Softwarepaketfilter iptables durchzuführen, wurde ebenfalls der Aufbau aus Abschnitt 5.1 verwendet. Auf der Netzwerkkarte wurde die so genannte Basic-Firmware von Netronome ausgeführt. Mit dieser registriert der Host je physikalischem Port ein Interface, das sich wie gewohnt verwenden lässt. Außerdem bietet die Karte damit viele der üblichen Features performanter Netzwerkkarten wie beispielsweise Receive Side Scaling, Segment- oder Checksum-Offloading. [vgl. 31] Damit die ankommenden Pakete richtig weitergeleitet werden, wurden auf den Interfaces Adressen konfiguriert und manuelle Einträge in der ARP-Tabelle vorgenommen, da das verwendete Messskript selber keine ARP-Requests beantworten würde. Das verwendete Regelwerk ist in Listing 5.3 dargestellt und entspricht dem Regelwerk, welches zuvor auf der Netzwerkkarte verwendet wurde.

```
iptables -A FORWARD -i enp132s0np0 -p udp -d 172.16.222.0/24 --dport
↪  100 -j ACCEPT
iptables -A FORWARD -i enp132s0np0 -j DROP
```

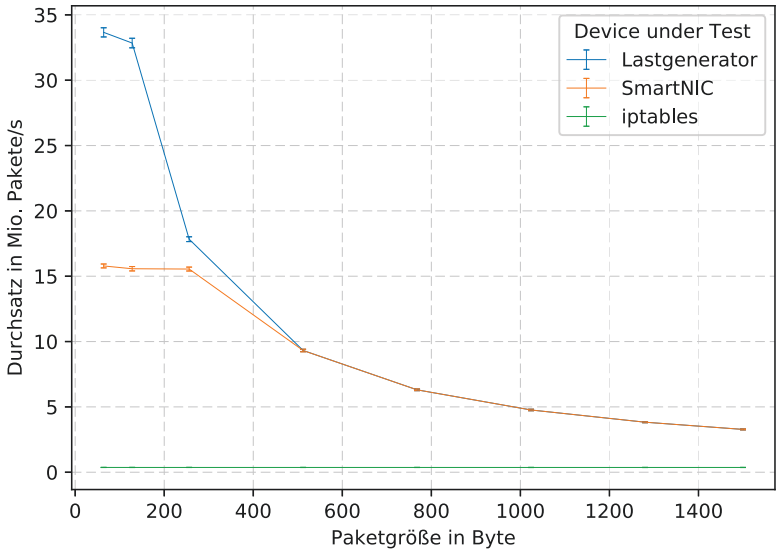
Listing 5.3.: Regelwerk für iptables beim Test mit MoonGen

In Abbildung 5.4 sind die erzielten Ergebnisse der beiden Messungen und die maximale Performance des Lastgenerators zum Vergleich abgebildet. In Tabelle A.1 sind die Mittelwerte der einzelnen Messungen aufgeführt. Die geringere Performance, die der Lastgenerator bei kleinen Paketen erreicht, fällt nicht ins Gewicht, da auch keiner der Paketfilter an die Raten herankommt.

An der Datenrate ist zu erkennen, dass der Paketfilter auf der SmartNIC die volle Performance erst ab einer Paketgröße von 512 Byte erreicht hat. Von 64 bis 256 Byte großen Paketen lässt sich ein linearer Anstieg der Datenrate erkennen. Das Abflachen des Graphen zwischen 256 und 512 Byte lässt sich mit der geringen Anzahl der Datenpunkte begründen. Der lineare Anstieg der Datenrate und die konstante Paketrate bei kleinen Paketen erlauben die Vermutung, dass die Netzwerkkarte unabhängig von der Paketgröße einen maximalen Durchsatz von knapp über 15 Millionen Paketen pro Sekunde erreichen



(a) Erzielte Datenrate



(b) Erzielte Paketrate

Abbildung 5.4.: Vergleich zwischen iptables und SmartNIC mit zustandslosen Regeln

kann. Da bei größeren Paketen weniger Pakete pro Sekunde verarbeitet werden müssen, um 40 Gbit/s zu erreichen, fällt diese Beschränkung dann nicht mehr ins Gewicht.

Die erreichte Datenrate von iptables ist durchgängig sehr gering und erreicht auch mit der maximalen Paketgröße von 1500 Byte nicht ganz einen Durchsatz von 5 Gbit/s. Es ist jedoch deutlich ein linearer Anstieg der Datenrate mit steigender Paketgröße zu erkennen.

Die Paketrate bleibt für alle Paketgrößen konstant. Während des Testes konnte beobachtet werden, dass ein Prozessorkern des DuT zu 100% ausgelastet war, die anderen jedoch überhaupt nicht. Dadurch erklärt sich die konstante Paketrate, welche das Maximum der auf einem Kern durch Linux verarbeitbaren Pakete darstellt. Die schlechte Verteilung der Last kann darauf zurückgeführt werden, dass von MoonGen immer die gleichen Pakete generiert wurden. Da der Kern, auf dem die Verarbeitung eines Paketes stattfindet, aus den Feldern der Header bestimmt wird, wird bei identischen Paketen stets der selbe Kern ausgewählt und dadurch überlastet [3, 26].

5.2.2. Zustandsbehaftete Regeln

Auch bei der Evaluation mit zustandsbehafteten Regeln wurden sowohl Messungen mit dem implementierten hybriden Paketfilter als auch Vergleichsmessungen mit iptables durchgeführt. Die Ergebnisse sind in Abbildung 5.5 dargestellt.

Für die Messungen des hybriden Paketfilters wurde der in Abschnitt 5.1 beschriebene Aufbau mit Lastgenerator und DuT verwendet. Auf der Netzwerkkarte wurde die entwickelte Firmware ausgeführt. In Listing 5.4 ist die zustandsbehaftete Regel dargestellt, die für die Messreihe verwendet wurde. Auf dem Host wurde die entwickelte Controller-Software ausgeführt, die das State-Tracking durchführt und über das RTE mit der Netzwerkkarte kommuniziert.

```
pass inet in-on p0 proto tcp to 172.16.222.0/24 port 80 out-on p4
↪  keep-state
```

Listing 5.4.: Im Test verwendete zustandsbehaftete Regel

Um die Vergleichsmessung mit iptables durchzuführen, wurde ebenfalls der Aufbau aus Abschnitt 5.1 verwendet. Auf der Netzwerkkarte wurde wie schon im Test der zustandslosen Regeln die Basic-Firmware von Netronome ausgeführt. Die verwendeten Regeln sind in Listing 5.5 dargestellt.

In Abbildung 5.5 findet sich eine grafische Darstellung der Ergebnisse der Messungen mit zustandsbehafteten Regeln. In Tabelle A.2 sind die Messwerte aufgeführt. Dabei sind, insbesondere auch im Vergleich zu den Messungen mit zustandslosen Regeln, einige Auffälligkeiten zu erkennen. So erzielt iptables jetzt im Vergleich deutlich höhere Datenraten. Während bei kleinen Dateien etwa die Performance des Lastgenerators erreicht wird, kann

```
iptables -A FORWARD -i enp132s0np0 -p tcp -d 172.16.222.0/24 --dport 80
↪  -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
iptbles -A FORWARD -i enp132s0np1 -p tcp -s 172.16.222.0/24 --sport 80
↪  -m conntrack --ctstate ESTABLISHED -j ACCEPT
iptables -A FORWARD -i enp132s0np0 -j DROP
iptables -A FORWARD -i enp132s0np1 -j DROP
```

Listing 5.5.: Regelwerk für iptables beim Test mit zustandsbehafteten Regeln

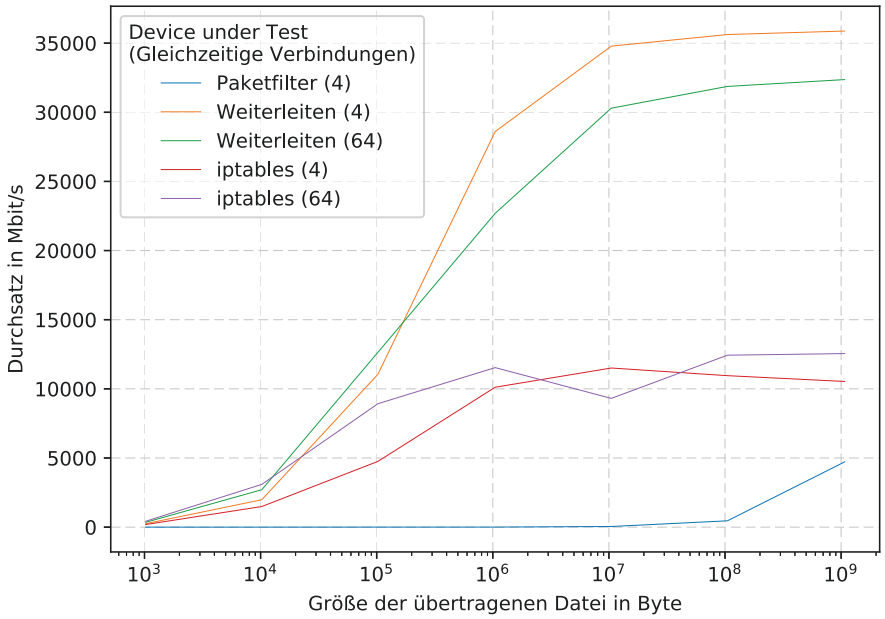


Abbildung 5.5.: Evaluation des hybriden Paketfilters mit zustandsbehafteten Regeln

iptables ab einer Dateigröße von 100 KB nicht mehr mithalten und erreicht im Maximum je nach Anzahl unterschiedlicher Verbindungen zwischen 10 und 15 Gbit/s. Die Last auf dem DuT war außerdem deutlich besser über alle Kerne der CPU verteilt. Das kann darauf zurückgeführt werden, dass durch die unterschiedlichen Verbindungen die Verteilungsmechanismen des Linux-Kernels besser ausgenutzt werden können und die Verarbeitung der Pakete nun auf unterschiedlichen Kernen stattfindet [26]. Außerdem können weitere Features der Basic Firmware wie Segmentation Offloading oder Checksum Offloading bei TCP sehr gut genutzt werden, was die Verarbeitung zusätzlich beschleunigt.

Bei den Messungen des hybriden Paketfilters zeigt sich ein ganz anderes Bild. Bis zu einer Dateigröße von 100 MB bewegt sich die Übertragungsrate auf einem sehr geringen Niveau. Erst mit großen Dateien ist eine höhere Übertragungsrate möglich. Nur mit Dateien der Größe 1 GB lies sich ein Durchsatz von knapp 5 Gbit/s erzielen. Mit 64 parallelen Verbindungen pro Instanz konnte kein vollständiger Testlauf durchgeführt werden. Im

Folgenden werden die mögliche Gründe für die schlechte Performance bei der Messreihe beleuchtet.

5.2.3. Probleme des Paketfilters im Test

Bei Tests während der Entwicklung wurde festgestellt, dass die Übertragung einzelner Dateien durch den Paketfilter meist ohne Probleme möglich ist. Dabei ist zwar eine recht hohe Latenz aufgrund des Umweges über den Controller zu bemerken, die Datenrate lag jedoch bei großen Dateien im Bereich von mehreren Gbit/s. Werden jedoch viele Verbindungen in schneller Abfolge im Rahmen der Messungen aufgebaut, ist der Paketfilter auf die Dauer nicht wirklich zuverlässig. Das äußert sich in abbrechenden, hängenden oder sehr langsamen Verbindungen. Dadurch werden die Messungen erheblich beeinflusst. Insbesondere bei 64 gleichzeitigen Verbindungen je ApacheBench-Instanz war aufgrund der großen Zahl abbrechender Verbindungen keine sinnvolle Messreihe durchzuführen.

Beobachtet werden konnte, dass manchmal Datenverkehr, für den ein Eintrag in einer Flow-Tabelle existiert, trotzdem an den Controller geleitet wurde. Das Einfügen der neuen Einträge in die Flow-Tabelle hat also die Behandlung der Pakete nicht verändert, stattdessen wurden sie weiterhin nach den Filterregeln bearbeitet. Zusammen mit dem unregelmäßigen Auftreten dieses Verhaltens wird vermutet, dass ein Caching-Problem der verwendeten Hardware für die falsche Verarbeitung verantwortlich ist. Die Netzwerkkarte stellt einen sogenannten Flowcache bereit. Dieser dient dazu, die Performance der Verarbeitung zu erhöhen, indem er bestimmte Informationen zu dem Fünftupel eines Flows vorhält [30, 38]. Leider war ein Abschalten des Caches beim Kompilieren der Firmware nicht möglich, wodurch es nicht möglich war, diesen als Ursache zu bestätigen oder auszuschließen.

Es wurde jedoch bereits von Shukla et al. [38] und in *Compromising P4-Enabled SmartNICs in Less than a Minute!* [24] gezeigt, dass dieser Flowcache Probleme aufweist. Dabei wurde der Cache durch das Erzeugen einer großen Zahl unterschiedlicher Pakete, für die jeweils ein neuer Eintrag angelegt werden muss, zum Überlaufen gebracht. Danach lässt sich ein erheblicher Paketverlust feststellen, sodass kaum noch Daten übertragen werden können.

Obwohl während der Messungen deutlich weniger unterschiedliche Pakete auf der SmartNIC eintreffen, liegt die Vermutung nahe, dass das fehlerhafte Caching für die schlechte Performance zumindest mitverantwortlich ist. Daher wird vermutet, dass das Konzept des

hybriden Paketfilters theoretisch einen Geschwindigkeitsvorteil bringen kann, jedoch mit dieser Netzwerkkarte nicht umsetzbar ist.

6. Fazit und Ausblick

Die domänenspezifische Programmiersprache P4 gewinnt im Bereich des SDN in letzter Zeit viel an Bedeutung. Sie ist dabei für die Programmierung von performanten Data Planes gut geeignet, da ihr Aufbau der Architektur von Switches nachempfunden ist. Außerdem ist die Komplexität der Entwicklung im Vergleich zu FPGAs oder ASICs deutlich reduziert. Durch die Orientierung des Designs an den Konzepten des SDN wird beim Umsetzen von Anwendungen eine moderne Softwarearchitektur gefördert, die Funktionalitäten sauber in einzelne Komponenten aufteilt. Dadurch wird auch die Erweiterbarkeit in der Zukunft unterstützt.

Der Einsatz einer Programmiersprache wie P4 zur Implementierung der Data Plane bietet auch noch weitere Vorteile, die in Zukunft genutzt werden könnten. So ist es mittlerweile bereits möglich, ein solches Programm mittels Software zu verifizieren, anstatt wie bisher auf ausgiebige Tests mit echtem Datenverkehr zurückgreifen zu müssen [12]. Die Unabhängigkeit der Sprache von der Zielplattform erlaubt auch das Übersetzen eines P4-Programmes in eine Konfiguration für ein FPGA, um sie direkt zu verwenden oder als Basis für weitere Entwicklungen zu nutzen [19]. Während sich die Werkzeuge zur Entwicklung von Software mit P4 weiterentwickeln, gibt es auch Wünsche nach einer Weiterentwicklung der Sprache. So zeigen Sivaraman et al. [16] einige mögliche Veränderungen auf, welche die Einsatzmöglichkeiten in der Zukunft fördern sollen.

Bei der Implementierung des Paktfilters zeigte sich trotz der gebotenen Flexibilität eine gute erzielbare Performance der verwendeten Netzwerkkarte bei zustandslosen Regeln. Die erzielten Ergebnisse für den gesamten hybriden Paketfilter zeigen eine schlechte Performance, was jedoch auf Probleme des Caches dieser Hardware zurückgeführt wird und nicht auf die grundlegende Architektur. So ist es bei einzelnen Tests sehr wohl möglich, Daten über die Karte schnell zu weiterzuleiten. Der Zunahme der Latenz beim Verbindungsaufbau könnte durch eine performantere Implementierung des Controllers entgegengewirkt werden. Dabei sollte keine Skriptsprache mehr zum Einsatz kommen und es könnte eine

Technologie wie DPDK als Basis genutzt werden.

In der Zukunft könnte auch die Implementierung auf einer anderen, mit P4 programmierbaren Hardwareplattform getestet werden. Ein Beispiel für eine solche Plattform wäre ein Switch auf Basis des Tofino-Chips von Barefoot [22]. Eine weitere Möglichkeit wäre die Portierung auf ein FPGA. Durch die zunehmende Verbreitung von P4 wird voraussichtlich auch die Auswahl von möglichen Plattformen größer werden.

Im Jahr 2018 wurde bekanntgegeben, dass das P4-Projekt zukünftig ein Teil der Open Networking Foundation sein wird, die auch hinter den OpenFlow-Spezifikationen steht [36]. Daraus könnte sich ein weiterer Schub für die Verbreitung von P4 im Bereich des SDN ergeben. Damit scheint es auch eine Möglichkeit zu geben, dass P4 tatsächlich einen großen Einfluss auf die Entwicklung von „OpenFlow 2.0“ hat, wie Bosshart et al. [2] vorgeschlagen haben. Sollte dies der Fall sein, wäre eine Implementierung eines Paketfilters mit der hier vorgestellten Architektur auf einem zukünftigen NOS eine Option, die möglicherweise eine große Vereinfachung des Controllers und breiten Hardwaresupport bringen würde.

A. Messergebnisse

Paketgröße	Lastgenerator		SmartNIC		iptables	
Byte	$\frac{Mbit}{s}$	$\frac{Mio.Pakete}{s}$	$\frac{Mbit}{s}$	$\frac{Mio.Pakete}{s}$	$\frac{Mbit}{s}$	$\frac{Mio.Pakete}{s}$
64	23697.17	33.6608	11113.21	15.7858	258.28	0.3669
128	39937.52	32.8434	18935.87	15.5723	445.20	0.3661
256	39945.09	17.8326	34828.59	15.5485	820.99	0.3665
512	39952.80	9.3174	39944.10	9.3153	1569.86	0.3661
768	39950.39	6.3053	39948.99	6.3051	2318.55	0.3659
1024	39940.59	4.7639	39940.72	4.7639	3059.97	0.3650
1280	39947.50	3.8293	39949.20	3.8295	3808.84	0.3651
1500	39950.56	3.2768	39950.69	3.2768	4457.20	0.3656

Tabelle A.1.: Mittelwerte der Messungen zum Vergleich zwischen iptables und SmartNIC mit zustandslosen Regeln(*Lastgenerator* zur Evaluation des Lastgenerators)

	Paketfilter	Weiterleiten		iptables	
	4	4	64	4	64
1KB	0.009	247.792	350.563	187.902	425.142
10KB	0.066	1981.842	2707.233	1490.174	3092.625
100KB	0.619	11050.896	12640.642	4753.076	8925.323
1MB	4.653	28604.658	22692.745	10118.066	11535.503
10MB	46.165	34783.850	30295.824	11506.721	9309.749
100MB	457.200	35620.793	31869.385	10953.755	12434.129
1GB	4720.111	35873.908	32363.264	10534.287	12550.179

Tabelle A.2.: Durchsatz in Mbit/s bei unterschiedlichen Messungen mit zustandsbehafteten Regeln (*Weiterleiten* zur Evaluation des Lastgenerators)

B. Referenzen

B.1. Wissenschaftliche Quellen

[1] Kristen Accardi et al. „Network Processor Acceleration for a Linux* Netfilter Firewall“. In: *Proceedings of the 2005 Symposium on Architecture for Networking and Communications Systems - ANCS '05*. The 2005 Symposium. Princeton, NJ, USA: ACM Press, 2005, S. 115. DOI: 10.1145/1095890.1095906.

[2] Pat Bosshart et al. „P4: Programming Protocol-Independent Packet Processors“. In: *ACM SIGCOMM Computer Communication Review* 44.3 (28. Juli 2014), S. 87–95. DOI: 10.1145/2656877.2656890.

[3] Silas Boyd-Wickizer et al. „An Analysis of Linux Scalability to Many Cores“. In: *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*. 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI '10). Vancouver, BC, Canada, Okt. 2010, S. 16.

[4] Mou-Sen Chen et al. „Using NetFPGA to Offload Linux Netfilter Firewall“. In: 2nd North American NetFPGA Developers Workshop. Stanford, CA, 13. Aug. 2010, S. 7.

[5] Huynh Tu Dang et al. „Whippersnapper: A P4 Language Benchmark Suite“. In: *Proceedings of the Symposium on SDN Research - SOSR '17*. The Symposium. Santa Clara, CA, USA: ACM Press, 2017, S. 95–101. DOI: 10.1145/3050220.3050231.

[6] Yaozu Dong et al. „High Performance Network Virtualization with SR-IOV“. In: *Journal of Parallel and Distributed Computing* 72.11 (Nov. 2012), S. 1471–1480. DOI: 10.1016/j.jpdc.2012.01.020.

[7] Paul Emmerich et al. „MoonGen: A Scriptable High-Speed Packet Generator“. In: ACM Press, 2015, S. 275–287. DOI: 10.1145/2815675.2815692.

[8] Andreas Fiessler et al. „HyPaFilter: A Versatile Hybrid FPGA Packet Filter“. In: *Proceedings of the 2016 Symposium on Architectures for Networking and Communi-*

cations Systems - ANCS '16. The 2016 Symposium. Santa Clara, California, USA: ACM Press, 2016, S. 25–36. DOI: 10.1145/2881025.2881033.

[10] Jose M. Gonzalez, Vern Paxson und Nicholas Weaver. „Shunting: A Hardware/Software Architecture for Flexible, High-Performance Network Intrusion Prevention". In: *Proceedings of the 14th ACM Conference on Computer and Communications Security - CCS '07*. The 14th ACM Conference. Alexandria, Virginia, USA: ACM Press, 2007, S. 139. DOI: 10.1145/1315245.1315264.

[11] Diego Kreutz et al. „Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1 (Jan. 2015), S. 14–76. DOI: 10.1109/JPROC.2014.2371999.

[12] Jed Liu et al. „P4v: Practical Verification for Programmable Data Planes". In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication - SIGCOMM '18*. The 2018 Conference of the ACM Special Interest Group. Budapest, Hungary: ACM Press, 2018, S. 490–503. DOI: 10.1145/3230543.3230582.

[13] Nick McKeown et al. „OpenFlow: Enabling Innovation in Campus Networks". In: *ACM SIGCOMM Computer Communication Review* 38.2 (31. März 2008), S. 69. DOI: 10.1145/1355734.1355746.

[14] Simeon Miteff und Scott Hazelhurst. „NFShunt: A Linux Firewall with OpenFlow-Enabled Hardware Bypass". In: *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. 2015 IEEE Conference on Network Function Virtualization and Software-Defined Networks (NFV-SDN). San Francisco, CA: IEEE, Nov. 2015, S. 100–106. DOI: 10.1109/NFV-SDN.2015.7387413.

[15] Sandra Scott-Hayward, Sriram Natarajan und Sakir Sezer. „A Survey of Security in Software Defined Networks". In: *IEEE Communications Surveys & Tutorials* 18.1 (2016), S. 623–654. DOI: 10.1109/COMST.2015.2453114.

[16] Anirudh Sivaraman et al. „DC.P4: Programming the Forwarding Plane of a Data-Center Switch". In: *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research - SOSR '15*. The 1st ACM SIGCOMM Symposium. Santa Clara, California: ACM Press, 2015, S. 1–8. DOI: 10.1145/2774993.2775007.

[17] Guido van Rooij. „Real Stateful TCP Packet Filtering in IP Filter". In: 2nd International SANE Conference. Maastricht, Mai 2000.

[18] Péter Vörös und Attila Kiss. „Security Middleware Programming Using P4“. In: *Human Aspects of Information Security, Privacy, and Trust*. Hrsg. von Theo Tryfonas. Bd. 9750. Cham: Springer International Publishing, 2016, S. 277–287. DOI: 10.1007/978-3-319-39381-0_25.

[19] Han Wang et al. „P4FPGA: A Rapid Prototyping Framework for P4“. In: *Proceedings of the Symposium on SDN Research - SOSR '17*. The Symposium. Santa Clara, CA, USA: ACM Press, 2017, S. 122–135. DOI: 10.1145/3050220.3050234.

[20] Nicholas Weaver, Vern Paxson und Jose M. Gonzalez. „The Shunt: An FPGA-Based Accelerator for Network Intrusion Prevention“. In: *Proceedings of the 2007 ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays - FPGA '07*. The 2007 ACM/SIGDA 15th International Symposium. Monterey, California, USA: ACM Press, 2007, S. 199. DOI: 10.1145/1216919.1216952.

B.2. Weitere Quellen

[21] *Ab - Apache HTTP Server Benchmarking Tool - Apache HTTP Server Version 2.4*. URL: https://httpd.apache.org/docs/2.4/programs/ab.html (besucht am 22. 02. 2019).

[22] Barefoot Networks. *Tofino Product Brief*. URL: https://www.barefootnetworks.com/products/brief-tofino/ (besucht am 27. 02. 2019).

[23] cmcginty. *Linux - Force Local IP Traffic to an External Interface (Antwort von Cmcginty)*. URL: https://serverfault.com/questions/127636/force-local-ip-traffic-to-an-external-interface/128680#128680 (besucht am 08. 01. 2019).

[24] *Compromising P4-Enabled SmartNICs in Less than a Minute!* URL: https://www.youtube.com/watch?v=j544b_rl-yE (besucht am 21. 02. 2019).

[25] DPDK Project. *How to Get Best Performance with NICs on Intel Platforms — DPDK 16.04.0 Documentation*. URL: https://doc.dpdk.org/guides-16.04/linux_gsg/nic_perf_intel_platform.html (besucht am 23. 10. 2018).

[26] Tom Herbert und Willem de Bruijn. *Scaling in the Linux Networking Stack*. URL: https://www.kernel.org/doc/Documentation/networking/scaling.txt (besucht am 21. 02. 2019).

[27] Intel. *Intel Ethernet Controller XL710 10/40 GbE - Product Brief*. 2015. URL: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/xl710-10-40-gbe-controller-brief.pdf (besucht am 13. 02. 2019).

[28] *Linux Kernel*. Version v4.20. linux/net/netfilter/nf_conntrack_proto_tcp.c. URL: https://github.com/torvalds/linux/blob/v4.20/net/netfilter/nf_conntrack_proto_tcp.c (besucht am 11. 12. 2018).

[29] Colm MacCárthaigh. *Scaling Apache 2.x beyond 20,000 Concurrent Downloads*. 21. Juli 2005. URL: http://pds5.egloos.com/pds/200704/12/18/scaling-apache-handout.pdf (besucht am 22. 02. 2019).

[30] Netronome. *Agilio CX 2x40GbE SmartNIC - Product Brief*. URL: https://www.netronome.com/media/documents/PB_Agilio_CX_2x40GbE.pdf (besucht am 21. 02. 2019).

[31] Netronome. *Basic Firmware User Guide*. URL: https://help.netronome.com/support/solutions/articles/36000049975-basic-firmware-user-guide (besucht am 20. 02. 2019).

[32] Netronome. *Netronome Network Flow Processor 6xxx: Development Tools User's Guide*.

[33] Netronome. *NFP-4000 Theory of Operation*. URL: https://www.netronome.com/media/documents/WP_NFP4000_TOO.pdf (besucht am 22. 11. 2018).

[34] Open Networking Foundation. *OpenFlow Switch Specification: Version 1.5.1*. URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[35] *OpenBSD PF: Packet Filtering*. URL: https://www.openbsd.org/faq/pf/filter.html (besucht am 19. 02. 2019).

[36] *P4 Gains Broad Networking Industry Adoption, Joins Open Networking Foundation (ONF) and Linux Foundation (LF) to Accelerate Next Phase of Growth and Innovation*. 16. März 2018. URL: https://www.opennetworking.org/news-and-events/press-releases/p4-joins-onf-and-linux-foundation/ (besucht am 27. 02. 2019).

[37] *P4 Language Consortium Members*. URL: https://p4.org/members/ (besucht am 16. 02. 2019).

[38] Apoorv Shukla et al. *Compromising P4-Enabled SmartNICs in Less than a Minute!* Nicht veröffentlicht, 2018.

[39] Andrew S. Tanenbaum. *Computer Networks*. 5th ed. Pearson Education, 2011. ISBN: 978-0-13-255317-9.

[40] The P4 Language Consortium. *P4-16 Language Specification: Version 1.1.0*. 30. Nov. 2018. URL: `https://p4.org/p4-spec/docs/P4-16-v1.1.0-spec.pdf`.

C. Glossar

ASIC Application-specific Integrated Circuit: Auf eine bestimmte Anwendung spezialisierter integrierter Schaltkreis. 1, 41

DuT Device under Test: Gerät, dessen Eigenschaften bei einem Test oder einer Messung überprüft werden sollen. 29, 32–34, 37, 38

FPGA Field Programmable Gate Array: In einer Hardwarebeschreibungssprache programmierbarer integrierter Schaltkreis. 1, 2, 11, 12, 41, 42

NOS Network Operating System: Zentrale Komponente des Software Defined Networking, die eine Abstraktionsschicht für Anwendungen bereitstellt. 4, 42

PF Physical Function: Stellt eine Funktionalität eines Gerätes auf dem PCIe-Bus zur Verfügung. 16

RID Request Identifier: Nummer zur eindeutigen Identifizierung einer PCIe-Funktion auf einem PCIe-Bus. 16

RTE Runtime Environment: Software, die die Ausführung anderer Programme in einer bestimmten Umgebung ermöglicht, indem es eine Abstraktionsschicht bereitstellt (auch: Laufzeitumgebung). 14–17, 23, 24, 26, 37

SDN Software Defined Networking: Neue Technologie zum effizienteren Betrieb von Computernetzwerken. 3–7, 11, 12, 17, 23, 41, 42

SR-IOV Single Root Input/Output Virtualization: Technologie zur effizienten Verwendung eines PCIe-Gerätes in mehreren virtuellen Maschinen. 14–16

VF Virtual Function: PCIe-Funktion zur Verwendung mit SR-IOV, die einer Physical Function zugeordnet ist und sich Ressourcen mit dieser teilen kann. 16