Institut für Kommunikationsnetze und Rechnersysteme (IKR)
Universität Stuttgart
Pfaffenwaldring 47
D–70569 Stuttgart

Diplomarbeit Nr. 3157

# Implementierung von OpenFlow für Juniper Router

Filip Kostadinow

| | |
|---|---|
| **Studiengang:** | Softwaretechnik |
| **Prüfer:** | Prof. Dr.-Ing Andreas Kirstädter |
| **Betreuer:** | Dipl.-Ing. Marc Barisch<br>Dipl.-Ing. Sebastian Meier<br>Dipl.-Ing. Joachim Scharf<br>Dipl.-Inf. David Wagner |
| **begonnen am:** | 10. Januar 2011 |
| **beendet am:** | 12. Juli 2011 |
| **CR-Klassifikation:** | C.2.3, C.5.m, C.2.6 |

# Inhaltsverzeichnis

# Abbildungsverzeichnis



# Tabellenverzeichnis

# Verzeichnis der Listings

# 1. Einleitung

Für die Forschung im Bereich der Netzwerktechnologie ist die Durchführung von Experimenten in realistischen Testumgebungen mit großen Hindernissen behaftet. An erster Stelle ist die große Bandbreite verwendeter Protokolle und Geräte zu nennen [MAB+08]. Bei den meisten Netzwerkgeräten handelt es sich um geschlossene Plattformen, so dass es kaum möglich ist, diese zu erweitern oder zu modifizieren [NEC+08]. Deren Funktionsweise ist herstellerspezifisch und kann von Hersteller zu Hersteller variieren. Die Hersteller haben dabei aus Wettbewerbsgründen keine Interesse daran, die Funktionsweise und Implementierung ihrer Produkte offen zu legen.
Um den genannten Herausforderungen zu begegnen, ist ein Standard notwendig, der es erlaubt, Geräte unterschiedlicher Hersteller anzusteuern, ohne dass die interne Implementierung offengelegt wird. Das OpenFlow Consortium [opea] hat zu diesem Zweck *OpenFlow* entwickelt, mit dem Ziel eine standardisierte Schnittstelle zur Ansteuerung von Switchen und Routern unterschiedlicher Hersteller zu schaffen. Dadurch soll Forschern die Kontrolle über den Paketfluss von Netzwerkgeräten ermöglicht werden. Die Voraussetzung hierfür ist, dass Hersteller OpenFlow als Feature in ihre Produkte integrieren. OpenFlow bietet somit einen Mittelweg, so dass einerseits Forscher Experimente auf leistungsfähigen Geräten ausführen können, und Hersteller andererseits ihre interne Implementierung nicht preisgeben müssen.
Einen ersten Schritt in diese Richtung haben einige namenhafte Hersteller bereits unternommen, indem sie OpenFlow in einige ihrer Produkte integriert haben. Beispielsweise gibt es für die Cisco Catalyst 6500 Serie [Cisb] oder für die MX–Serie [JNe] von Juniper Networks versuchsweise Implementierungen von OpenFlow. Diese sind jedoch nicht offen zugänglich.
Das Ziel der vorliegenden Diplomarbeit in diesem Zusammenhang ist die Erarbeitung einer modularen, erweiterbaren Architektur, die es erlaubt, OpenFlow für einen Juniper Router zu implementieren. Anhand der Architektur soll eine prototypische Implementierung von OpenFlow erfolgen. Als Grundlage für die Implementierung von OpenFlow wird die OpenFlow–Spezifikation 1.0 [opeb] verwendet.

Die Arbeit ist folgendermaßen aufgebaut:

**Kapitel 1 – Einleitung**

**Kapitel 2 – Grundlagen:** Hier werden die Grundlagen beschrieben, die für die Implementierung von OpenFlow notwendig sind. Es wird auf OpenFlow, den Aufbau des Routers sowie auf das Junos SDK näher eingegangen.

**Kapitel 3 – Architektur:** In diesem Kapitel wird die zur Umsetzung von OpenFlow entwickelte Architektur vorgestellt, anhand derer die Implementierung erfolgt.

**Kapitel 4 – Implementierung:** Hier wird auf die Implementierung des Prototyps eingegangen, der anhand der in Kapitel 3 vorgestellten Architektur erfolgt. Es werden wichtige Abläufe sowie die Klassendiagramme der hierfür benötigten Komponenten beschrieben.

**Kapitel 5 – Test:** In diesem Kapitel wird erläutert, wie der Prototyp getestet wurde. Es wird die Testumgebung beschrieben sowie die Testfälle, die durchgeführt wurden.

**Kapitel 6 – Zusammenfassung und Ausblick** gibt eine Zusammenfassung über die Arbeit. Weiterhin wird ein Ausblick gegeben, welche Erweiterungen sich für diese Arbeit vornehmen lassen und wie auf diese Arbeit aufgebaut werden kann.

**Anhang**

# 2. Grundlagen

Das folgende Kapitel gibt zunächst eine Einführung zu *OpenFlow*, im Anschluss daran werden die Architektur des *Juniper Router M7i* sowie das *Junos SDK* näher erläutert.

## 2.1. OpenFlow

Im folgenden Abschnitt soll das Konzept von *OpenFlow* näher beschrieben werden. Nach einem kurzen Überblick der Ziele und Einsatzgebiete von OpenFlow werden die Komponenten eines OpenFlow–Switches vorgestellt. Im Anschluss daran wird die Rolle des Controllers aufgezeigt.

### 2.1.1. Überblick

OpenFlow wurde an der Universität Stanford entwickelt. Ziel von OpenFlow ist es, eine standardisierte Schnittstelle zur Ansteuerung von Netzwerkgeräten (Switchen, Router, Basisstationen) zu schaffen. Dadurch soll Forschern ermöglicht werden, Experimente in realen Testumgebungen durchzuführen, idealerweise in deren Hochschulnetzwerken, die sie jeden Tag nutzen [MAB+08].

Bei klassischen Switchen werden Routing– und Forwardingentscheidungen auf demselben Gerät getroffen. Bei OpenFlow–Switchen hingegen werden die Forwardingentscheidungen weiterhin auf dem Switch getroffen, während die Routingentscheidungen auf einen externen Controller (2.1.3) verlagert werden. Der Switch und der Controller kommunizieren über das OpenFlow–Protokoll (2.1.2.3), das u.a. dem Controller erlaubt die Forwarding–Tabelle eines Switches zu manipulieren. Netzwerkadministratoren können so den Datenverkehr in Produktions– und Forschungsströme aufteilen, indem sie die Routen angeben, auf denen die Pakete weitergeleitet werden sollen. Somit können Forscher neue Protokolle, Sicherheitsmodelle oder ein neues Adressierungsschema testen, ohne den Produktions–Datenverkehr zu beeinträchtigen. Des Weiteren können Geräte unterschiedlicher Hersteller angesteuert werden, ohne dass die Hersteller den internen Aufbau ihrer Geräte aufzeigen müssen.

### 2.1.2. Komponenten eines OpenFlow–Switch

Ein OpenFlow–Switch besteht im Wesentlichen aus einer Flow–Tabelle (2.1.2.1), einem Secure Channel (2.1.2.2), der eine Verbindung zu einem externen Controller (2.1.3) herstellt und dem OpenFlow-Protokoll (2.1.2.3), über das der Switch und der Controller kommunizieren [opeb].
Abbildung 2.1 zeigt einen OpenFlow–Switch, der über den Secure Channel mit einem externen Controller (2.1.3) kommuniziert.

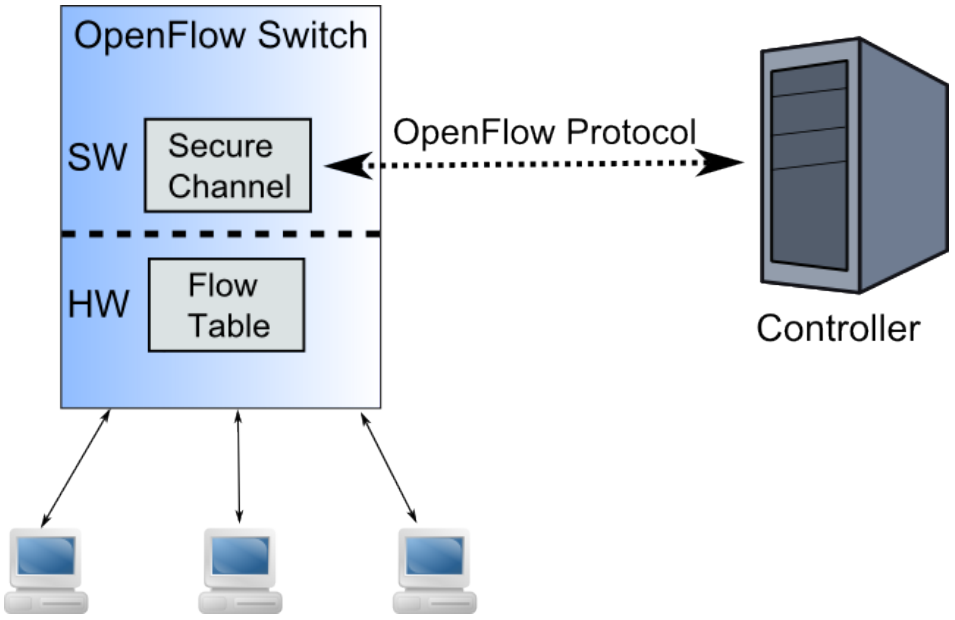


**Abbildung 2.1.:** OpenFlow Komponenten nach [opeb]

#### 2.1.2.1. Flow–Tabelle

Im folgenden Abschnitt soll auf die *Flow–Tabelle* näher eingegangen werden. Es werden hierbei die Bestandteile der Flow–Tabelle sowie der Ablauf der Verarbeitung eingehender Pakete erläutert.

Nachfolgend wird der Begriff *Flow* wie folgt verwendet: Zu einem Flow gehören alle Pakete, welche die selben Eigenschaften aufweisen. Hierzu werden die Header–Felder (Abbildung 2.3) betrachtet. Pakete mit gleichen Header–Feldern gehören zum selben Flow. Beispielsweise kann definiert werden, dass alle Pakete zum selben Flow gehören, die eine bestimmte Quell–IP–Adresse haben.
Die Flow–Tabelle besteht aus einer Menge von Flow–Einträgen. Abbildung 2.2 stellt den Aufbau eines Flow–Eintrags dar. Dieser besteht aus dem Dreiertupel Header–Felder, Aktion(en) und Zähler.

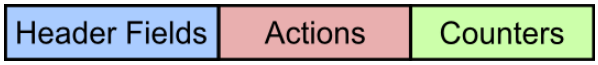


**Abbildung 2.2.:** Flow–Eintrag in der Flow–Tabelle

Die Header–Felder setzen sich aus dem in Abbildung 2.3 dargestellten Zwölftupel zusammen, das aus Paket–Headern der Schichten zwei bis vier des OSI–Schichtenmodells [Sta] sowie dem Eingangsport besteht.

| Ingress Port | Ether scr | Ether dest | Ether type | VLAN ID | VLAN priority | IP scr | IP dest | IP proto | IP ToS bits | TCP/ UDP scr port | TCP/ UDP dest port |
|---|---|---|---|---|---|---|---|---|---|---|---|

**Abbildung 2.3.:** Header–Felder nach [opeb]

Anhand der Header–Felder wird in der Flow–Tabelle nach einem passenden Eintrag gesucht. Für die jeweiligen Tupel der Header–Felder können entweder bestimmte Werte eingetragen werden oder sogenannte *wildcards*. Sind wildcards für bestimmte Felder angegeben, trifft jeder Wert zu. Einträge, bei denen wildcards angegeben sind, haben eine geringere Priorität als Einträge ohne wildcards.
Wird für ein eingehendes Paket in der Flow–Tabelle ein passender Eintrag gefunden, wird die dazugehörige Aktion ausgeführt. Es können auch mehrere Aktionen pro Eintrag definiert werden. Sind mehrere Aktionen für einen Eintrag definiert, muss die Reihenfolge beachtet werden, in der die Aktionen ausgeführt werden. Aktionen müssen in der Reihenfolge ausgeführt werden, in der sie definiert wurden. Sind für einen Eintrag keine Aktionen festgelegt, impliziert das, dass das Paket verworfen werden soll. Wird für ein Paket kein Eintrag in der Tabelle gefunden, wird der Controller kontaktiert, um festzustellen, was mit dem Paket geschehen soll.
Die Zähler dienen zur Erhebung von Statistiken. Beispielsweise wird festgehalten, wie viele Einträge in der Flow–Tabelle vorhanden sind oder wie viele Pakete empfangen und gesendet wurden. Eine genauere Auflistung der Zähler kann der OpenFlow–Spezifikation [opeb] entnommen werden.

Ein Switch muss nicht alle in der OpenFlow–Spezifikation aufgeführten Aktionstypen unterstützen. Bei der Herstellung der Verbindung mit dem Controller gibt der Switch an, welche Aktionen er unterstützt. In der OpenFlow–Spezifikation wird vorgeschrieben, welche Aktionen erforderlich sind und welche optional sind.

- Erforderliche Aktionen
  - Forward: Die Forward–Aktion gibt an, über welchen Port Pakete weitergeleitet werden. Ein Port kann ein physikalischer oder ein virtueller Port sein. Virtuelle Ports können beispielsweise „ALL“ oder „CONTROLLER“ sein. „ALL“ gibt an, dass das Paket an alle Ports weitergeleitet werden soll, mit Ausnahme des

Eingangsports. Falls „CONTROLLER“ als Port angegeben ist, soll das Paket an den Controller gesendet werden.

- Drop: Sind für einen Eintrag keine Aktionen definiert, bedeutet es, dass alle dazugehörigen Pakete verworfen werden sollen.

- Optionale Aktionen
  - Enqueue: Die Enqueue–Aktion leitet Pakete an eine Queue weiter, die einem Port vorangestellt werden kann.
  - Modify–Field: Die Aktion Modify–Field dient dazu, bestimmte Header–Felder eines Pakets zu verändern. Beispielsweise lässt sich mit dieser Aktion die VLAN–ID oder die IP–Quell– und Ziel–Adresse umschreiben.

#### 2.1.2.2. Secure Channel

Der *Secure Channel* ist die Schnittstelle, über die eine Verbindung vom OpenFlow–Switch zu einem Controller (2.1.3) hergestellt werden kann. Der Controller konfiguriert und verwaltet den Switch über diese Schnittstelle und empfängt Nachrichten vom Switch oder sendet welche an ihn. Die Kommunikation findet über das OpenFlow–Protokoll (2.1.2.3) statt.

Um eine Verbindung zum Controller aufzubauen, muss dem Switch die IP–Adresse des Controllers sowie der Port bekannt sein, an welchem der Controller auf OpenFlow–Nachrichten hört. Sobald eine Verbindung zum Controller hergestellt ist, wird der *Handshake* [opeb] durchgeführt. Damit die Kommunikation mit dem Controller stattfinden kann, muss sichergestellt werden, dass beide die selbe OpenFlow–Version unterstützen. Dabei werden Nachrichten ausgetauscht und die darin enthaltenen OpenFlow–Versionen verglichen. Stimmen die Versionen nicht überein wird der Verbindung abgebrochen.
Bricht eine zuvor hergestellte Verbindung ab und der Switch ist nicht in der Lage die Verbindung wieder herzustellen, wechselt der Switch in den sogenannten *emergency mode*. Im emergency mode werden alle Einträge aus der Flow–Tabelle entfernt, die beim Hinzufügen nicht als *emergency–Eintrag* markiert worden sind. Die Weiterleitung der Pakete geschieht nur anhand der übrig gebliebenen *emergency–Einträgen*. Sobald die Verbindung wieder hergestellt ist, arbeitet der Switch wie gewohnt weiter. Die *emergency*–Einträge bleiben in der Tabelle bestehen. Einträge, die vorher aus der Tabelle entfernt wurden, können vom Controller wieder eingetragen werden.

#### 2.1.2.3. OpenFlow Protokoll

Der Switch und der Controller kommunizieren über das OpenFlow–Protokoll (2.1.2.3), welches dem Controller unter anderem ermöglicht die Flow–Tabelle des Switches zu manipulieren. Das OpenFlow–Protokoll unterstützt drei Nachrichtentypen: *Controller–to–Switch*, *Asynchron* und *Symmetrisch*. Diese werden nachfolgend erläutert.

**Controller-to-Switch Nachrichten** Controller-to-Switch Nachrichten werden vom Controller initialisiert. Sie dienen dazu, den Switch zu konfigurieren, dessen Fähigkeiten abzufragen, Statistiken anzufordern und die Flow–Tabelle zu verwalten.

**Asynchrone Nachrichten** Asynchrone Nachrichten werden von dem Switch initialisiert und dienen dazu, den Controller über Netzwerkevents, beispielsweise Fehlermeldungen, oder Statusänderungen des Switches zu informieren. Weiterhin sendet der Switch Nachrichten an den Controller, die ein Paket beinhalten, für das keinen Eintrag in der Flow–Tabelle gefunden wurde. Eine solche Nachricht wird als *Packet-In*–Nachricht bezeichnet. Der Controller entscheidet, wie das Paket verarbeitet werden soll. Er sendet als Antwort eine *Packet–Out*–Nachricht, die angibt, dass das Paket über einen bestimmten Port versendet werden soll, oder verworfen werden soll.

**Symmetrische Nachrichten** Symmetrische Nachrichten können von beiden Seiten initialisiert werden. Sie sind in folgende Subtypen gegliedert:

- **Hello:** Beim Aufbau einer Verbindung zwischen dem Controller und dem Switch werden Hello–Nachrichten ausgetauscht. Diese enthalten die jeweils unterstützte Version des Open–Flow Protokolls.
- **Echo:** Echo–Nachrichten müssen mit einer Echo–Reply–Nachricht beantwortet werden. Sie werden genutzt, um die Latenz, Bandbreite oder die Lebendigkeit der Verbindung anzuzeigen.
- **Vendor:** Dieser Nachrichtentyp wird für zukünftige OpenFlow–Versionen bereit gestellt. OpenFlow–Switchen soll es ermöglicht werden, zusätzliche Funktionalität innerhalb der OpenFlow–Nachrichtentypen anzubieten.

### 2.1.3. Controller

Ein OpenFlow–Switch wird über einen externen Controller gesteuert. Pakete, für die der Switch keinen Eintrag in der Flow–Tabelle findet, werden an den Controller gesendet. Dieser entscheidet, wie die Pakete verarbeitet werden sollen und sendet seine Entscheidung an den Switch. Des weiteren kann der Controller Einträge in die Flow–Tabelle hinzufügen oder entfernen.

Es gibt eine Reihe frei–verfügbarer OpenFlow–Controller. Am meisten verbreitet sind Nox [GKP+08], Maestro [mae] und Beacon [ofC]. Zum Testen der prototypischen Implementierung wurde der Beacon–Controller verwendet. Dieser soll im Folgenden kurz vorgestellt werden.

**Beacon** *Beacon* ist ein OpenFlow–Controller, der an der Universität Stanford entwickelt wird und steht als Open Source zur Verfügung. Die Implementierung erfolgt in der Programmiersprache Java.

Um Beacon auszuführen ist es möglich, sowohl ein Image herunterzuladen und auf einer virtuellen Maschine abzuspielen, als auch den Quell–Code herunterzuladen, zu kompilieren und manuell zu starten. Die Kommunikation zwischen einem OpenFlow–Switch und Beacon erfolgt gemäß der OpenFlow–Spezifikation 1.0 (Stand: 01.07.2011).
Zusätzlich bietet Beacon ein Webinterface an, über das der Anwender statische Flow–Einträge erstellen kann. Der Controller sendet dann eine Nachricht an den Switch, mit dem Befehl die erstellten Einträge in die Flow–Tabelle einzutragen. Weiterhin kann sich der Benutzer einen Überblick über den aktuellen Stand des Netzwerks anzeigen lassen. Darin werden alle Hostrechner und Switchen angezeigt, die sich im Netzwerk befinden.

### 2.1.4. Manipulation der Flow–Tabelle

#### Einfügen eines Eintrags

Um einen neuen Eintrag in die Flow–Tabelle einzufügen, sendet der Controller eine *OFPT_FLOW_MOD*–Nachricht, bei der angegeben ist, dass es sich um eine Einfüge–Operation handelt. Sie enthält eine Liste von Aktionen, die angeben, wie passende Pakete verarbeitet werden sollen. Zunächst muss die Validität des einzufügenden Eintrags sichergestellt werden. Dabei wird überprüft, ob eine der definierten Aktionen einen Port referenziert, der nicht existiert oder inaktiv ist. In so einem Fall wird der Eintrag nicht in die Tabelle eingefügt und es wird eine Fehlernachricht an den Controller gesendet. Ergibt die Validitäts–Prüfung, dass der Eintrag gültig ist, so wird als nächstes untersucht, ob das *OFPFF_CHECK_OVERLAP*–Flag der Nachricht gesetzt ist. Falls es gesetzt ist, wird in der Tabelle nach einem Eintrag gesucht, der sich mit dem einzufügenden Eintrag überschneidet. Zwei Einträge überschneiden sich, wenn ein Paket auf beide Einträge passen kann und beide Einträge die selbe Priorität haben. Ist ein solcher Eintrag vorhanden, wird der Eintrag nicht eingefügt und es wird eine Fehlernachricht an den Controller gesendet. Ist das Flag nicht gesetzt oder es existiert kein überschneidender Eintrag, wird nachgeschaut, ob bereits ein Eintrag mit den selben Header–Feldern und der selben Priorität in der Tabelle existiert. Existiert kein solcher Eintrag, wird der neue Eintrag hinzugefügt. Anderenfalls wird der bereits existierende Eintrag aus der Tabelle entfernt und der neue Eintrag wird eingefügt.

#### Entfernen eines Eintrags

Einen Eintrag aus der Flow–Tabelle zu entfernen, kann aus folgenden Anlässen geschehen:

- Auf Anfrage des Controllers: Um einen Eintrag aus der Flow–Tabelle zu entfernen, sendet der Controller eine *OFPT_FLOW_MOD*–Nachricht, bei der angegeben ist, dass

es sich um eine Lösch–Operation handelt. In der Nachricht wird anhand der Header–Felder angegeben, welcher Eintrag aus der Tabelle entfernt werden soll. Wird kein passender Eintrag in der Tabelle gefunden, wird *keine* Fehlernachricht an den Controller gesendet. Ansonsten wird der Eintrag aus der Tabelle entfernt.

- Es tritt ein *Idle–Timeout* ein: Für jeden Eintrag ist ein *Idle–Timeout* angegeben. Dieser besagt, dass ein Eintrag aus der Tabelle entfernt werden soll, wenn innerhalb einer gewissen Zeitspanne keine passenden Pakete eintreffen. Ein Eintrag wird entfernt, sobald ein *Idle–Timeout* eintritt.
- Es tritt ein *Hard–Timeout* ein: Für jeden Eintrag ist ein *Hard–Timeout* angegeben. Dieser besagt, dass ein Eintrag aus der Tabelle entfernt werden soll, sobald eine bestimmte Zeit abgelaufen ist, unabhänging davon, ob passende Pakete für den Eintrag eintreffen. Ein Eintrag wird entfernt, sobald ein *Hard–Timeout* eintritt.

Jeder Eintrag entält ein *OFPFF_SEND_FLOW_REM*–Flag. Dieses gibt an, ob beim Entfernen des Eintrags aus der Flow–Tabelle, der Controller benachrichtigt werden soll. Ist das Flag gesetzt, wird eine *OFPT_FLOW_REMOVED*–Nachricht an den Controller gesendet, die Informationen über den entfernten Eintrag beinhaltet.

**Modifikation eines Eintrags**

Um einen Eintrag in der Flow–Tabelle zu modifizieren, sendet der Controller eine *OFPT_FLOW_MOD*–Nachricht, bei der angegeben ist, dass es sich um eine Modifikations–Operation handelt. In der Nachricht sind die Header–Felder angegeben, anhand derer der zu modifizierende Eintrag in der Tabelle aufgesucht wird. Wird kein passender Eintrag gefunden, wird ein neuer Eintag in die Tabelle eingefügt, der die Angaben der Modifikations–Nachricht übernimmt. Anderenfalls werden die Aktionen des bestehenden Eintrags mit den in der Modifikations–Nachricht angegebenen Aktionen ersetzt.

## 2.2. Juniper Router M7i

### 2.2.1. Überblick

Für die Implementierung von OpenFlow steht in dieser Arbeit ein *Juniper M7i Multiservice Edge Router* zur Verfügung. Nachfolgend wird die Architektur des Routers beschrieben. Die Komponenten, die für die Implementierung von OpenFlow eine wichtige Rolle spielen, werden genauer erläutert. Zum Schluss wird der Paketfluss im Router dargestellt, der für die Implementierung ebenfalls von großer Bedeutung ist.

### 2.2.2. Architektur des Juniper Routers

Der M7i Multiservice Edge Router ist ein komplettes Routingsystem, das u.a. ATM (Asynchronous Transfer Mode), kanalisiertes Ethernet, IP Services und SONET/SDH–Schnittstellen bereitstellt [JNc]. Die Haupt–Komponenten, die nachfolgend genauer beschrieben werden, sind der *Routing Engine (RE)* und *Packet Forwarding Engine (PFE)*. Im Hinblick auf die Implementierung von OpenFlow kommt der *MultiServices PIC* ebenfalls eine zentrale Bedeutung zu. Auf diese soll in diesem Abschnitt ebenfalls eingegangen werden. Abbildung 2.4 stellt die Architektur des Routers dar.

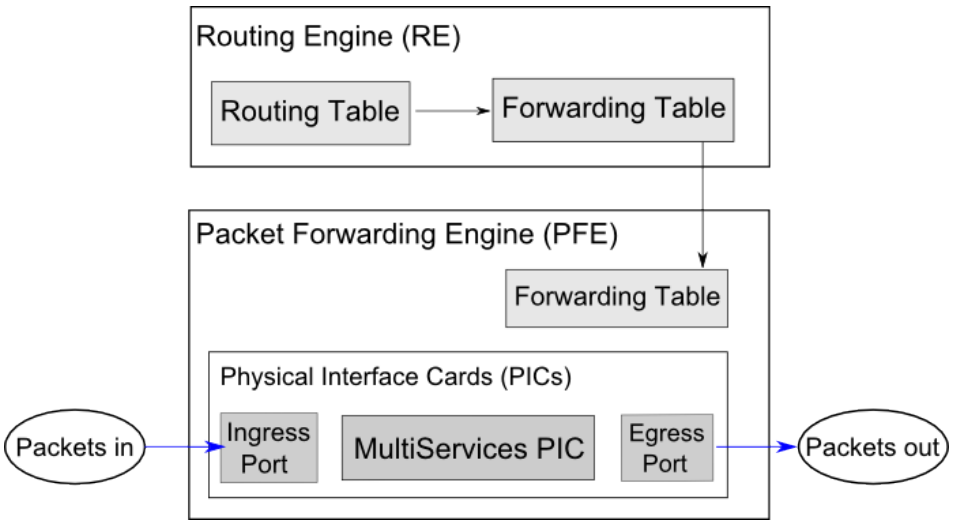


**Abbildung 2.4.:** Router–Architektur nach [JN10]

Kontrollfunktionalität und Funktionalität der Paketweiterleitung werden getrennt. Dadurch sollen Verarbeitungs– und Traffic–Engpässe vermieden werden. Kontrolloperationen werden auf der Routing–Engine (RE) durchgeführt. Die Weiterleitung der Pakete wird auf der Packet Forwarding Engine, die die Physical Interface Cards (PICs) beinhaltet, durchgeführt.
Der Router kann um bis zu vier PICs erweitert werden, über die der Router Pakete empfangen und versenden kann. Die maximale Durchsatzrate der PICs beträgt 3,2 Gbps (vollduplex) [JNd]. Die Packet Forwarding Engine kann bis zu 16 Millionen Pakete jeglicher Größe pro Sekunde weiterleiten [JNc].
Das Junos SDK, das in Abschnitt 2.3 beschrieben wird, erlaubt es Entwicklern Anwendungen zu schreiben, die auf dem Router installiert werden können. Anwendungen können sowohl auf der Routing Engine laufen, als auch auf dem MultiServices PIC.

**Routing Engine** Auf der Routing Engine läuft das Betriebssystem Junos OS [JNa]. Junos OS basiert auf dem Open–Source Betriebssystem FreeBSD [Fre] und wurde von Juniper Networks zu einem Netzwerk–Betriebssystem erweitert, so dass es in der Lage ist, eine große Anzahl von Netzwerkschnittstellen und Routen zu verwalten [Net].

Die *Routing Engine* (RE) ist der zentrale Kontrollpunkt des Systems. Authentifizierung, Systemmanagement, Systemkonfiguration und Monitoring werden normalerweise auf der RE durchgeführt. Weiterhin übernimmt die RE die Verarbeitung von Routing–Protokollen und erstellt so die Routing– und die Forwarding–Tabelle. Die Forwarding–Tabelle wird auf die Packet Forwarding Engine kopiert, so dass die Packet Forwarding Engine ausschließlich für die Weiterleitung von Datenpaketen genutzt werden kann.
Der Benutzer hat mehrere Alternativen, den Router zu verwalten und zu konfigurieren. Eine davon ist, das Junos Command–Line Interface (CLI) zu nutzen. Das CLI kann in zwei Modi betrieben werden – im Betriebsmodus und im Konfigurationsmodus [Garo6]. Im Betriebsmodus kann die ganze Router–Hardware und –Software überwacht werden. Im Konfigurationsmodus hat der Benutzer die Möglichkeit, Routereinstellungen vorzunehmen. Er kann beispielsweise die Netzwerkadressen der Schnittstellen angeben oder festlegen, welche Protokolle verwendet werden sollen. Weiterhin können Einträge in die Konfigurationsdatei des Routers gemacht werden, welche das Verhalten bestimmter Anwendungen beeinflussen. Mehr Informationen können der Junos Produktdokumentation auf der Juniper Networks Webseite [JNh] entnommen werden.

**Packet Forwarding Engine** Die *Packet Forwarding Engine* (PFE) leitet Pakete anhand der Forwarding–Tabelle weiter. Dies ist mittels Application-specific integrated circuits (ASICs) realisiert, die von Juniper Networks entwickelt wurden. Über die Physical Interface Cards (PICs) ist der PFE physikalisch mit verschiedenen Netzwerkgeräten verbunden. PICs empfangen eingehende Pakete vom Netzwerk und übertragen ausgehende Pakete ins Netzwerk. Je nach Medientyp führen diese auch Framing oder Signaling durch. Der Router unterstützt verschiedene PICs, u.a. ATM [1] und kanalisiertes Ethernet. Ein besonderer Typ der PICs sind die *MultiServices PICs*, die nachfolgend beschrieben werden.

**MultiService PIC** Die MultiServices PICs sind ein spezieller Typ der Physical Interface Cards, die zusätzlich auf dem Packet Forwarding Engine installiert werden können. Das MultiServices PIC ist mit dem PFE über eine 10 Gbps schnelle Verbindung verbunden.
Es ist möglich Datenpakete zur Verarbeitung von der PFE an die MulitServices PIC zu leiten. Anwendungen, die auf dem MultiServices PIC laufen, werden meistens für aufwendigere, zustandsorientierte Paketverarbeitungen eingesetzt, wie beispielsweise eine Stateful Firewall Anwendung, die Pakete anhand bestimmter Regeln filtert.

Die MultiServices PIC verfügt über eine 64–Bit Architektur mit 1 GB Arbeitsspeicher. Weiterhin besitzt diese acht Prozessorkerne mit jeweils vier Hardware–Threads, wodurch eine parallele Paketverarbeitung ermöglicht wird. Es ist weiterhin möglich die Anzahl der Prozessorkerne, die für eine Anwendung verwendet werden sollen, beliebig zu konfigurieren. Es wird zwischen zwei Typen von Prozessorkernen unterschieden. Zum einen gibt es die *Control Cores*, zum anderen die *Data Cores*. Die *Control Cores* dienen zur Kommunikation mit der Routing Engine, die *Data Cores* werden für die Paketverarbeitung verwendet.

[1] Asynchronous Transfer Mode

Um Pakete, die zur Verarbeitung an die MultiServices PIC–Anwendung geleitet werden, an die Threads der Anwendung zu verteilen, werden FIFO[2]–Queues verwendet. Jedem Prozessorkern (vom Typ *Data Core*) wird eine Queue zugeteilt, über die Datenpakete empfangen und nach der Verarbeitung wieder ausgesendet werden können. Die Pakete werden standardmäßig nach dem *Round Robin*[3]–Verfahren verteilt. Eine andere Möglichkeit, Pakete an die Threads zu verteilen ist, die sogenannte *Flow Affinity* zu aktivieren. Ist diese aktiviert, werden Pakete, die zum selben Flow gehören an den selben Thread geleitet. Dies geschieht durch die Bildung eines Hashwertes anhand der Ziel–, Quelladresse und des Protokolls.

### 2.2.3. Paketfluss im Router

Pakete gelangen über die Packet Forwarding Engine in den Router. Es können optional verschiedene Paketverarbeitungen vorgenommen werden, bevor die Pakete ebenfalls über die Packet Forwarding Engine weitergeleitet werden. Abbildung 2.5 stellt den Paketfluss im Router vereinfacht dar.

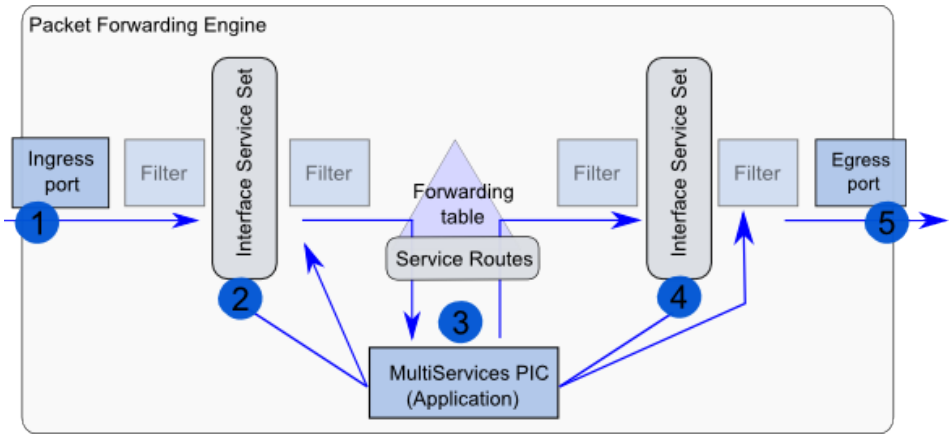


**Abbildung 2.5.:** Paketfluss im Router nach [JNg]

Pakete kommen im Schritt eins an einem Eingangsport an, durchlaufen optional definierte Filter und werden über die Forwarding–Tabelle an einen Ausgangsport geleitet. Auf dem MultiServices PIC kann eine Anwendung laufen, die bestimmte Pakete verarbeitet. Um Pakete an das MultiServices PIC, beziehungsweise an die Anwendung, zu leiten, gibt es mehrere Möglichkeiten [KAB10], von denen im Folgenden drei vorgestellt werden.
Die erste Möglichkeit ist sogenannte *Service–Routen* zu definieren (Schritt drei). Dabei werden Einträge in die Forwarding–Tabelle hinzugefügt, die besagen, dass Pakete nicht an einen Port, sondern an das MultiServices PIC geleitet werden sollen. Eine weitere Möglichkeit ist *Interface Service Sets* zu erstellen (Schritte zwei und vier). Diese können Eingangs– oder

[2]First In – First Out
[3]Pakete werden gleichmäßig über die FIFO–Queues verteilt.

Ausgangsports zugeordnet werden. Ein Interface Service Set besteht aus Regeln, anhand derer Pakete gefiltert und an die MultiServices PIC geleitet werden. Der Unterschied zu Service Routen ist, dass nur Pakete an das MultiServices PIC geleitet werden, die über Ports mit angehefteten Interface Service Sets fließen. Eine weitere Möglichkeit Pakete an die auf dem MultiServices PIC laufende Anwendung zu leiten ist, das *JUNOS Sampling Framework* zu benutzen. Dabei werden Pakete dupliziert und die Duplikate werden an das MultiServices PIC geleitet. Es kann angegeben werden, wie oft und für wie lange eingehende Pakete dupliziert werden sollen. Die Original–Pakete werden normal weitergeleitet. Das JUNOS Sampling Framework wird vorwiegend für Monitoring–Anwendungen eingesetzt.
Falls Pakete in den Schritten zwei, drei und vier zum MultiServices PIC geleitet wurden, werden sie nach der Verarbeitung über die Forwarding–Tabelle an einen Ausgangsport geleitet. Dabei durchlaufen sie wieder optional definierte Filter, bevor sie im Schritt fünf über den Ausgangsport an ihr Ziel gesendet werden.

## 2.3. Junos SDK

### 2.3.1. Überblick

Das *Junos SDK* (Software Development Kit) ermöglicht es, Entwicklern eigene Erweiterungen an Junos und Juniper Networks Plattformen vorzunehmen [JNb]. Es können neue Netzwerkanwendungen erstellt werden, die auf die eigenen Bedürfnissen zugeschneidert sind. Beispiele für solche Anwendungen sind Monitoring von Netzwerkverkehr und Routing–Performance, Verschlüsselungs– und Tunneling–Anwendungen sowie Durchführung von Load–Balancing, um bestimmte Quality of Services (QoS) zu garantieren.
Auf dem Router können Anwendungen unterschiedlicher Provider installiert werden. Aus Sicherheitsgründen wird jede Anwendung mit einem *providerspezifischen Zertifikat* versehen, das bei der Installation verifiziert wird. Jeder Provider erhält zusätzlich eine eindeutige *Provider–ID*, die dem Zertifikat entnommen werden kann. Anhand dieser ID werden Provider–Anwendungen von Juniper–Kernanwendungen unterschieden. Provider–Anwendungen werden gesondert ausgeführt und überwacht, so dass es zu keinen Kollisionen mit andern Prozessen kommen kann.
Wie in Abschnitt 2.2 dargestellt, können Anwendungen auf der Routing Engine oder auf der MultiServices PIC installiert werden. Das Junos SDK stellt für jeden Anwendungstyp jeweils eine Version bereit. Es wird zwar konzeptionell zwischen den beiden Versionen unterschieden, aber bestimmte Funktionen oder sogar ganze Programmbibliotheken werden sowohl in der Version des Routing Engine–SDK als auch in der des MultiServices PIC–SDK eingesetzt. Anwendungen, die mit dem RE–SDK entwickelt werden, sind am besten geeignet für Routing, Sampling oder Administration. Anwendungen, die mit dem MultiServices PIC–SDK erstellt werden, dienen im Allgemeinen dazu, spezifische Paketverarbeitungen vorzunehmen. Typischerweise sind dies Paketmanipulationen wie Kompression, Sampling, Prüfung oder Verschlüsselung von Paketen.
Das Junos SDK besteht aus zwei Teilen. Das sind zum einen die APIs, welche den Entwicklern zur Verfügung gestellt werden, und zum andern die Virtual Build Environment

2.3.3 . Die Virtual Build Environment wird benötigt, um Anwendungen zu entwickeln und Pakete zu erstellen, die auf dem Router installiert werden können. Im Abschnitt 2.3.2 wird ein Überblick über die APIs gegeben. Zunächst wird eine Übersicht über die APIs der RE–Version, der MultiServices PIC–Version und über die APIs, die in beiden Versionen vorhanden sind, gegeben. Anschließend werden die APIs, die bei der Umsetzung von OpenFlow eine wichtige Rolle spielen, genauer vorgestellt. Eine detailiertere Beschreibung der APIs kann der Junos SDK Dokumentation [JN10] von Juniper entnommen werden. Auf die VBE wird im Abschnitt 2.3.3 näher eingegangen.

### 2.3.2. Junos SDK APIs

Das Junos SDK baut auf einigen Standard–UNIX Programmbibliotheken auf. Die Standard C–Bibliothek ist in das SDK eingebunden. Unter anderem werden die UNIX APIs für Input/Output, Interprozesskommunikation (Sockets, Pipes), Mathematik, String–Manipulation und Signale genutzt. Zusätzlich werden APIs bereit gestellt, die auf die Junos Software zugeschnitten sind. Die APIs sind in der Programmiersprache C geschrieben, sind aber auch zu C++ kompatibel.

**APIs des Routing Engine–SDK** APIs aus der Routing Engine–Version bieten folgende Funktionalitäten:

- Auslesen der Konfigurationsdatei des Routers (2.3.2.4 *libddl-access*)
- Benachrichtigung registrierter Anwendungen beim Eintreten von bestimmten Ereignissen an Komponenten und Schnittstellen (*libisc2*)
- Hinzufügen und Entfernen von Schnittstellen
- Überwachung des Routers über SNMP (Simple Network Management Protocol) (*libjnet-snmp*, *libnet-snmp*)
- Benutzerauthentifizierung (*libjunos-aaa*)
- Sammlung von Statistiken über physikalische und logische Schnittstellen (2.3.2.5 *libjunos-stat*)
- Kontrolle des Ressourcen–Verbrauchs
- Informationsaustausch mit aktiven Knotenrechnern
- Ein– und Ausschalten von Knotenrechnern
- Erstellen einer Benutzerschnittstelle (2.3.2.3 *DDL und ODL*)

**APIs des MultiServices PIC–SDK** APIs aus der MultiServices PIC–Version bieten folgende Funktionalitäten:

- Paketverarbeitung (*libmp-sdk*)
- Locking (*libmp-sdk*)
- Speicherverwaltung (*libmp-sdk*)
- Flow Affinity (Verteilung der Pakete an die Prozessorkerne) und Processor Affinity (Verteilung der Pakete an die Hardware–Threads) (*libmp-sdk*)
- PIC to PIC Failover (Fällt eine PIC aus, kann der Verarbeitungsprozess auf eine andere PIC verlagert werden)

#### Funktionalitäten, die APIs beider SDKs bieten

- Initialisierung der Anwendungen (*libjunos–sdk*)
- Interprozesskommunikation (2.3.2.2 *libconn* )
- Unterstützung von System–Protokollierung (*libjuniper*)
- Monitoring von Prozessen (*libpmon*, *libmsvcs-pmon*)
- Hinzufügen und Löschen von Routen (*libssd*)
- Erstellung von dynamischen Firewall–Filtern (*libdfwd*)

Programmbibliotheken, die für die Implementierung von OpenFlow besonders nützlich sind, werden im Folgenden etwas genauer dargestellt.

#### 2.3.2.1. libmp–sdk

Die Bibliothek *libmp–sdk* beinhaltet fast alle wichtigen Funktionalitäten, um Anwendungen zu schreiben, die auf der MultiServices PIC laufen. Pakete, die zur Verarbeitung an die MultiServices PIC geleitet werden, werden über FIFO–Queues an die Hardware–Threads verteilt. Es werden Funktionen bereitgestellt, um in der Anwendung Schleifen zu erstellen, die Pakete aus den FIFO–Queues entnehmen und nach der Verarbeitung wieder in Ausgabe–Queues legen, um sie weiterzuleiten.
FreeBSD hat zur Speicherverwaltung von Paketen sogenannte *mbufs* [Tik] eingeführt. Im Junos SDK wird eine ähnliche Funktionalität angeboten – die jbuf Bibliothek. Die jbuf Bibliothek ist in die libmp–sdk Bibliothek integriert. Ein Paket kann durch mehrere verkettete jbufs erfasst werden. Es stehen Funktionen zum Auslesen von Informationen, wie Quell–, Ziel–IPAdresse, Eingangsport oder TCP/UDP–Port, zur Verfügung.
Da mehrere Threads auf der MultiServices PIC arbeiten, kann es notwendig sein, Zugriffe auf bestimmte Daten zu synchronisieren. Dazu können sogenannte Spinlocks verwendet werden. Eine Besonderheit der Spinlocks ist, dass Threads, die versuchen auf Daten zuzugreifen, die

gesperrt sind, nicht zum Schlafen geschickt werden, sondern sie warten in einer Schleife, bis sie Zugriff auf die Daten bekommen.

#### 2.3.2.2. libconn

Die Programmbibliothek libconn beinhaltet TCP Client und Server Bibliotheken für die Interprozesskommunikation (IPC). Kommunikation kann stattfinden zwischen Anwendungen auf der Routing Engine, zwischen Anwendungen auf MultiSerivces PICs oder zwischen Anwendungen verteilt auf der Routing Engine und der MultiServices PIC.
Um eine Verbindung zwischen zwei Anwendungen herzustellen, wird auf der einen Anwendung ein Server erstellt, mit dem sich die andere Anwendung als Client verbinden kann. Beide Kommunikationspartner sind dann in der Lage, Nachrichten zu senden und zu empfangen. Bei der Erstellung des Servers müssen der Server–Port und die maximal mögliche Anzahl an Clients, die sich mit dem Server verbinden können, angegeben werden. Außerdem werden Event–Handler für die Verbindung, die den internen Zustand der Verbindung verarbeiten, und für den Nachrichtenaustausch definiert. So können die Anwendungen auf Ereignisse wie *Verbindungsaufbau* oder *Verbindungsabbruch* entsprechend reagieren und eingehende Nachrichten verarbeiten.

#### 2.3.2.3. DDL und ODL

Wie im Abschnitt 2.2 beschrieben, ist der Benutzer in der Lage, über das Command–Line Interface die Konfigurationsdatei des Routers zu bearbeiten. Das Junos SDK bietet Entwicklern die Möglichkeit, mittels der *Data Definition Language* (DDL) und der *Output Description Language* (ODL) das Command–Line Interface zu erweitern, um anwendungsspezifische Einstellungen vornehmen zu können. Die DDL dient dazu, Befehlserweiterungen für das CLI zu definieren. Über die DDL werden Syntax, Optionen und Ausführungsdetails aller Anweisungen festgelegt. Die ODL wird verwendet, um Ausgaben zu definieren, die dem Benutzer über das CLI angezeigt werden.
Eine Funktionalität, welche die Benutzerschnittstelle betrifft, ist nur auf dem Routing Engine vorhanden. Um eine existierende Systemkonfiguration auszulesen, wird die Programmbibliothek *libddl–access* (2.3.2.4) verwendet.

#### 2.3.2.4. libddl–access

Mit Hilfe der Bibliothek *libddl–access* lassen sich Konfigurationen auslesen, welche der Benutzer über das Command–Line Interface eingegeben hat. Es werden Funktionen bereit gestellt, um die Konfigurationsdatei zu durchlaufen und die für die Anwendung relevanten Informationen auszulesen. Die ausgelesenen Informationen können lokal gespeichert werden, um nicht ständig auf die Konfigurationsdatei zugreifen zu müssen. Sobald der Benutzer Änderungen an der Konfigurationsdatei vornimmt, wird eine zuvor angegebene

Callback–Methode aufgerufen. Die Anwendung kann so jede Änderung an der Konfiguration wahrnehmen und die lokal gespeicherten Informationen aktualisieren.

#### 2.3.2.5. libjunos-stat

Mit Hilfe der Programmbibliothek *libjunos-stat* lassen sich Statistiken über den Datenverkehr an physikalischen und logischen Schnittstellen erfassen. Außerdem können Fehlerstatistiken für physikalische Schnittstellen abgefragt werden. Statistikanfragen können anhand des Namens, des Indexes oder der SNMP [4] ID gemacht werden. Folgende Statistiken können abgefragt werden:

- Anzahl empfangener Pakete
- Anzahl gesendeter Pakete
- Anzahl empfangener Bytes
- Anzahl gesendeter Bytes
- Anzahl verworfener Pakete
- Anzahl aufgetretener Fehler beim Paketempfang
- Anzahl aufgetretener Fehler beim Paketaussenden
- Anzahl aufgetretener CRC [5] Fehler
- Anzahl aufgetretener Kollisionen

Es werden zwei Typen von Statistiken angeboten.

- *Last Cleared Statistics*: Dem Benutzer wird die Möglickeit gegeben, Statistiken zurückzusetzen. Statistiken, die seit dem letzten Rücksetzten abgefragt werden, werden „Last Cleared Statistics“ genannt.
- *Absolute Statistics*: Absolute Statistics beinhalten Statistiken, die seit dem Hochfahren der Schnittstelle erfasst worden sind.

[4] Simple Network Management Protocol
[5] Cyclic Redundancy Check

### 2.3.3. Virtual Build Environment

Die Virtual Build Environment (VBE) wird benötigt, um Anwendungen zu erstellen, die auf dem Router installiert werden können. Die VBE besteht aus einem System–Image, auf dem das Betriebssystem FreeBSD [Fre] läuft. Auf diesem System–Image werden alle benötigten SDK–Pakete installiert, die zur Entwicklung gebraucht werden. Abbildung 2.6 stellt die Bestandteile der Virtual Build Environment dar. Diese beinhalten unter anderem die Programmbibliotheken, Compiler, Linker, Beispielanwendungen und Dokumentationen.

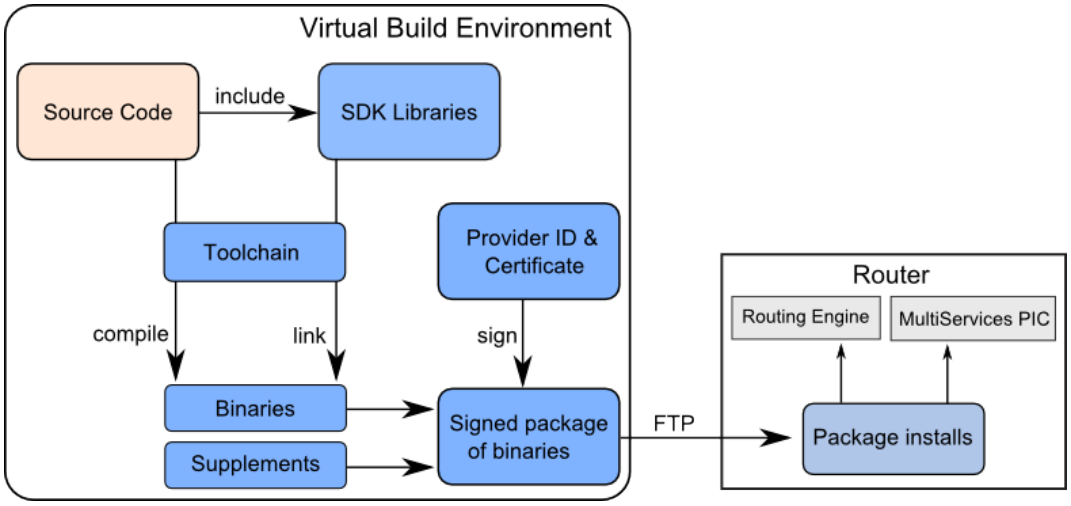


**Abbildung 2.6.:** Virtual Build Environment

Von Entwicklern geschriebener Quell–Code, einschließlich der verwendeten Junos SDK–Programmbibliotheken, wird über die Toolchain kompiliert und gelinkt. Es entstehen *Binaries* und *Supplements*. Die Supplements enthalten zusätzliche Dateien, die beispielsweise für die Erweiterung des Command–Line Interfaces gebraucht werden. Die Binaries und Supplements werden mit einem Zertifikat verknüpft und zu einem Paket gebündelt, das auf dem Router installierbar ist. Anwendungen können auch zu mehreren Paketen gebündelt werden, falls sie aus mehreren Komponenten zusammengesetzt sind. Die signierten Pakete können über FTP (File Transfer Protocol) auf den Router übertragen werden. Dort können sie bei Bedarf installiert werden.

# 3. Architektur

## 3.1. Überblick

Das vorliegende Kapitel stellt die Architektur für die Implementierung von *OpenFlow* auf einem Juniper Router vor. Der folgende Abschnitt gibt zunächst einen Überblick über die Anforderungen an die Architektur. Anschließend werden die erforderlichen Benutzereingaben beschrieben. Darauf folgend werden die einzelnen Komponenten vorgestellt. Es werden deren Funktionsweise sowie die Kommunikation zwischen den Modulen dargestellt. In den letzten Abschnitten wird auf die Erfassung von Statistiken, auf die Fehlerbehandlung und auf die Interprozesskommunikation eingegangen.
In manchen Bereichen der Architektur gibt es mehrere Alternativen für die Umsetzung. Diese werden zunächst mit ihren Vor– und Nachteilen diskutiert, anschließend wird die Auswahl der für die aktuelle Arbeit gewählten Alternativen begründet. Einige Architekturentscheidungen ergeben sich hierbei aus Gegebenheiten des Junos SDK, für andere Entscheidungen werden Performanz und Aufwand der Implementierung in Betracht gezogen.

## 3.2. Anforderungen

Das vorrangige Ziel dieser Arbeit ist die Entwicklung einer Architektur, die es ermöglicht *OpenFlow* auf Juniper Router zu implementieren. Der Schwerpunkt liegt dabei in der Umsetzung der Hauptfunktionalität von OpenFlow. Diese beinhaltet die Verwaltung der Flow–Tabelle, die Paketverarbeitung sowie die Kommunikation mit einem externen Controller.
Eine wesentliche Anforderung ist, dass die Architektur für verschiedene Juniper Hardware–Plattformen anwendbar ist. Des Weiteren muss die Wartbarkeit der Architektur gewährleistet sein. Diesbezüglich sollen sowohl Erweiterungen im Hinblick auf die Funktionalität, als auch hinsichtlich der Performanz vorgenommen werden können.
Die Performanz steht für die aktuelle Arbeit nicht im Vordergrund, vielmehr soll eine Architektur entwickelt werden, die verständlich ist und mit geringem Aufwand umgesetzt werden kann. Stehen mehrere Möglichkeiten für die Umsetzung zur Verfügung, soll diejenige gewählt werden, die leichter umsetzbar und leichter testbar ist.

## 3.3. Benutzereingaben

Zum Ausführen der OpenFlow–Anwendung werden einige Angaben von Seiten des Benutzers benötigt. Diese werden über das Command–Line Interface (CLI) in die Konfiguration des Routers eingegeben. Der Benutzer muss hierbei folgende Daten eingeben:

- *Informationen für die Kommunikation mit dem Controller:* Um eine Verbindung zu einem externen Controller herzustellen, müssen die IP–Adresse und der Port, auf welchen der Controller hört, angegeben werden.
- *Informationen über Router–Ports:* Der Benutzer muss angeben, welche Ports des Routers für OpenFlow zur Verfügung stehen. Nicht alle Ports des Routers werden notwendigerweise für den OpenFlow–Verkehr genutzt. Dazu müssen Name, Nummer und MAC–Adresse der Ports angegeben werden.
- *Definition der Interface Service Sets:* Damit Pakete zur Verarbeitung an die MS PIC–Komponente gelangen, muss der Benutzer Interface Service Sets definieren und sie den Ports zuweisen, die für den OpenFlow–Verkehr genutzt werden sollen.

Ein Beispiel, wie eine solche Konfiguration aussehen kann, ist in Listing A.24 im Anhang aufgeführt. Im nächsten Abschnitt werden die Komponenten beschrieben, die für die Implementierung von OpenFlow benötig werden.

## 3.4. Beschreibung der Komponenten

Die an der Implementierung von OpenFlow beteiligten Komponenten sind in Abbildung 3.1 dargestellt. Wie der Abbildung zu entnehmen ist, wird die Hauptfunktionalität auf zwei Komponenten verteilt, zum einen auf die *RE–Komponente*, zum anderen auf die *MS PIC–Komponente*. Die RE–Komponente läuft hierbei auf der Routing Engine (RE), während die MS PIC–Komponente auf der MultiServices PIC läuft, welche Teil der Packet Forwarding Engine (PFE) ist.

### 3.4.1. Aufteilung

Die Aufteilung der Implementierung von OpenFlow in zwei Komponenten erfolgt aus einer Reihe von Gründen. Folgende Gründe sprechen dafür, eine Komponente auf der MultiServices PIC laufen zu lassen:

- *Zugriff auf die Pakete*: Wie in Abschnitt 2.2.3 beschrieben, kommen Pakete über die Packet Forwarding Engine an. Um die Pakete verarbeiten zu können, müssen diese an eine auf der MultiServices PIC laufenden Anwendung geleitet werden. Daher muss ein Teil der Implementierung auf der MultiServices PIC laufen, da es sonst keine andere Möglichkeit gibt, auf die Pakete zuzugreifen.

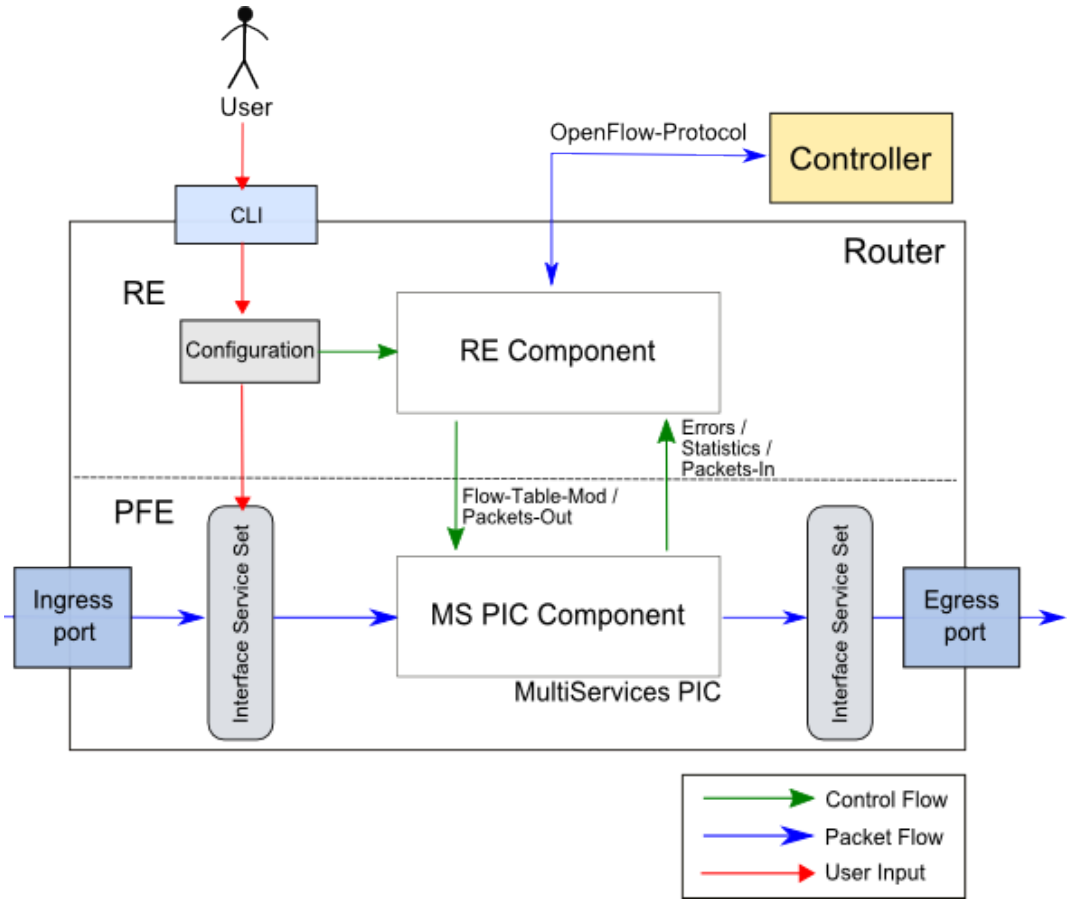


**Abbildung 3.1.:** Architektur–Übersicht

- *Verarbeitungsgeschwindigkeit*: Die MultiServices PIC verfügt über acht Prozessorkerne, somit kann die Verarbeitung der Pakete parallel erfolgen, wodurch die Verarbeitungsgeschwindigkeit der Anwendung gesteigert wird.
- *Junos Bibliothek*: Das Junos SDK stellt nützliche Bibliotheken für die Verarbeitung von Paketen zur Verfügung, welche nur von Anwendungen genutzt werden können, die auf dem MultiServices PIC installiert sind.

Die Notwendigkeit, die andere Komponente auf der Routing Engine laufen zu lassen, ergibt sich aus folgenden Gründen:

- *Erweiterung des CLI*: Damit der Benutzer die für die Ausführung der Anwendung benötigten Daten eingeben kann, sind Erweiterungen des Command–Line Interface notwendig. Dies kann nur auf der Routing Engine erfolgen.
- *Auslesen der Benutzereingaben*: Zum Auslesen der vom Benutzer getätigten Eingaben aus der Konfiguration, kann eine Bibliothek des Junos SDK genutzt werden. Da diese

nur auf der Routing Engine anwendbar ist, muss dieser Teil der Anwendung somit auf der Routing Engine laufen.

### 3.4.2. Weiterleitung der Pakete

Von den in Abschnitt 2.2.3 vorgestellten Möglichkeiten, Pakete an die MultiServices PIC zu leiten, wurde die Verwendung von *Interface Service Sets* gewählt. Dies hat den Grund, dass Interface Service Sets bestimmten Ports zugeordnet werden müssen und nur Pakete, die über diese Ports eintreffen, an die MultiServices PIC–Komponente geleitet werden. So kann der Benutzer bestimmen, welche Ports für den OpenFlow–Verkehr genutzt werden sollen. Die restlichen Ports leiten den Datenverkehr anhand der Forwarding–Tabelle weiter.

### 3.4.3. Kommunikation

Beide Komponenten müssen in der Lage sein, miteinander zu kommunizieren. Es werden hierbei Nachrichten beim Eintreten von Fehlern ausgetauscht, sowie zum Aktualisieren der Statistiken und der Flow–Tabelle. Hierfür stellt Junos einen Mechanismus zur Kommunikation zwischen verschiedenen Komponenten bereit (Interprozesskommunikation). Dieser wird im Abschnitt 3.9 beschrieben.

### 3.4.4. Synchronisation

Anwendungen, die auf der Routing Engine oder auf der MultiServices PIC laufen, brauchen unterschiedlich lange Zeiten bis sie gestartet und initialisiert sind. Bevor mit der Paketverarbeitung begonnen und bevor eine Verbindung zu dem Controller aufgebaut werden kann, müssen beide Komponenten der OpenFlow–Anwendung laufen. Um dies sicherzustellen, wird eine Synchronisierung zwischen den beiden Komponenten benötigt. Hierfür kann der Mechanismus für die Interprozesskommunikation genutzt werden. Sobald eine Verbindung zwischen den Komponenten aufgebaut ist, sendet die eine Komponente eine Nachricht an die andere Komponente. Diese antwortet mit einer Empfangsbestätigung. Sobald der Nachrichtenaustausch erfolgt ist, beginnen beide Komponenten mit der Ausführung ihrer jeweiligen Aufgaben.
Die nächsten beiden Abschnitte beschreiben die Aufgaben der jeweiligen Komponenten. Es wird zunächst auf die RE–Komponente eingegangen.

## 3.5. Routing Engine Komponente

Die Komponente, die auf der Routing Engine läuft, übernimmt folgende Aufgaben:

- Auslesen der Konfiguration (3.5.1)
- Kommunikation mit einem externen Controller (3.5.2)
- Erfassung von Statistiken (3.7)

Die einzelnen Module der Routing Engine Komponente sind in Abbildung 3.2 dargestellt. Im Folgenden wird erläutert, welche Aufgaben von welchen Modulen übernommen werden.

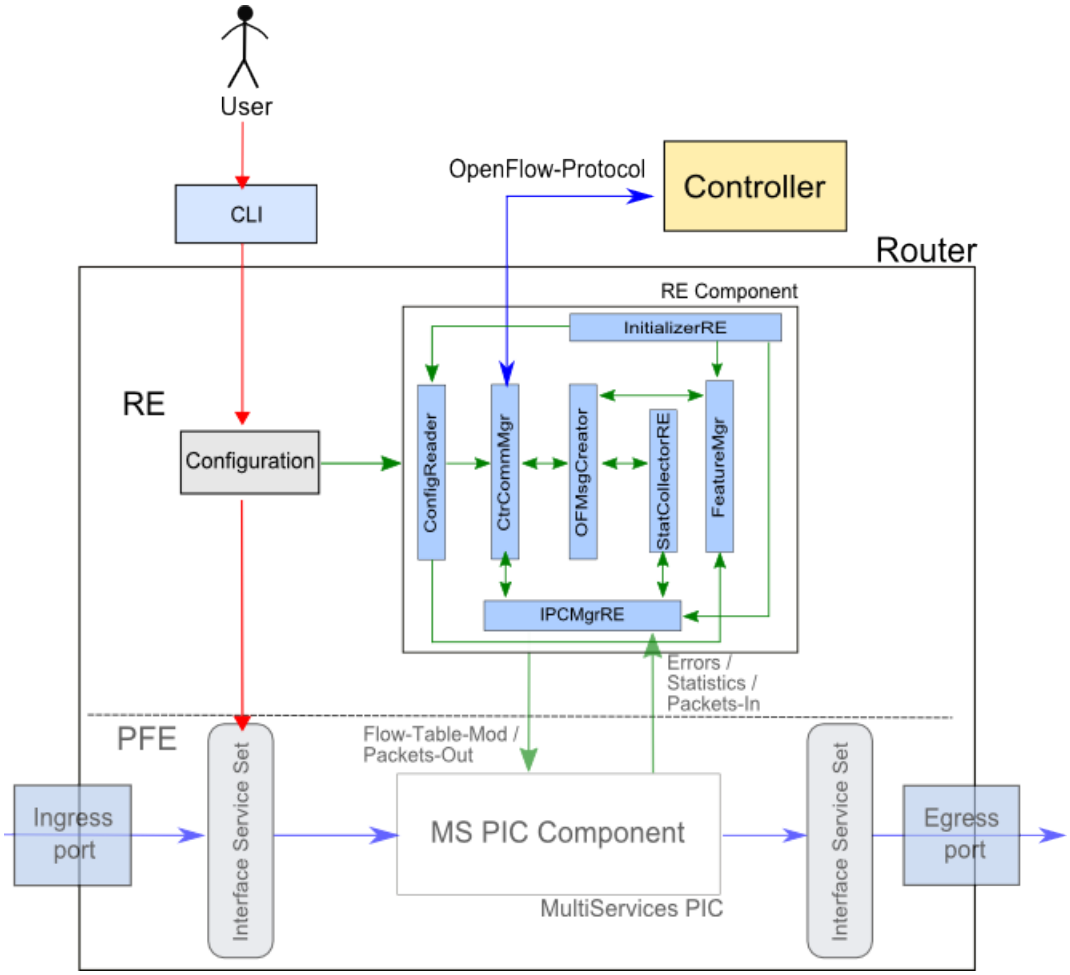


**Abbildung 3.2.:** Architektur der RE Komponente

### 3.5.1. Auslesen der Konfiguration

Das Auslesen der Konfiguration wird durch den *ConfigReader* vorgenommen. Dabei werden folgende Schritte durchlaufen:

1. Der Benutzer bestätigt die Eingaben, die er über das Command–Line Interface getätigt hat, mit einem *Commit*–Befehl.
2. Der *ConfigReader* beginnt mit dem Auslesen der Konfiguration. Hierfür stellt Junos ebenfalls einen Mechanismus mittels entsprechender Bibliothek zur Verfügung.
3. Der *ConfigReader* leitet die ausgelesenen Dateien an die dafür vorgesehenen Module weiter. Informationen zur Herstellung einer Verbindung zu einem externen Controller werden an den *CtrCommMgr* übergeben. Informationen über die Router–Ports, die für OpenFlow zur Verfügung stehen, werden an den *FeatureMgr* geleitet.

Beim Auslesen wird der gesamte Teil der Konfiguration durchlaufen, der für die OpenFlow–Anwendung relevant ist. Um keine aufwendigen Lese–Operationen ausführen zu müssen, wird zunächst geprüft, ob Änderungen an den Daten vorgenommen wurden. Es werden nur diejenigen Daten ausgelesen, die sich geändert haben.

### 3.5.2. Kommunikation mit einem externen Controller

Der *CtrCommMgr* ist zuständig für die Kommunikation mit dem externen Controller. Mit Hilfe der vom *ConfigReader* erhaltenen Informationen, kann er eine Verbindung zum Controller aufbauen. Die Kommunikation findet über das OpenFlow–Protokoll statt. Der *CtrCommMgr* empfängt Nachrichten, leitet diese an die entsprechenden Module weiter und beantwortet Anfragen des Controllers.

#### 3.5.2.1. Weiterleitung von Nachrichten

Nachrichten vom Controller, die sich auf das Weiterleiten von Paketen oder die Verwaltung der Flow–Tabelle beziehen, werden an die MS PIC–Komponente geleitet. Hierfür ist der *IPCMgrRE* zuständig. Er übernimmt die Kommunikation mit der MS PIC–Komponente.

#### 3.5.2.2. Nachrichten an den Controller

Der *OFMsgCreator* wird für das Senden von Nachrichten an den Controller benötigt, er erstellt OpenFlow konforme Nachrichten.

**Antworten auf Anfragen**
Bei Anfragen des Controllers bezüglich der Fähigkeiten des Routers holt der *OFMsgCreator* Informationen vom *FeatureMgr* ein, der die Fähigkeiten des Routers verwaltet. Im Falle von

Statistikanfragen holt sich der *OFMsgCreator* die benötigten Informationen vom *StatCollector-PFE*.

**Packet–In Nachrichten**
Für den Fall, dass der Router während der Paketverarbeitung nicht weiß, wie er ein Paket verarbeiten soll, sendet er eine Anfrage an den Controller, der darüber entscheidet, wie das Paket zu verarbeiten ist. Hierbei gibt es zwei Möglichkeiten:

- *Senden des gesamten Pakets*: Das gesamte Paket wird an den Controller gesendet. Der Controller schickt es zusammen mit seiner Entscheidung wieder zurück und das Paket wird entsprechend weiterverarbeitet.
- *Zwischenspeichern des Pakets*: Das Paket wird auf dem Router zwischengespeichert, nur ein Teil des Paket–Headers wird an den Controller gesendet. Zusätzlich wird eine *Buffer–ID* angehängt, um die Antwort des Controllers dem dazugehörigen Paket zuordnen zu können.

Die erste Möglichkeit, das gesamte Paket an den Controller zu senden, hat zwei Nachteile. Zum einen entsteht ein Overhead bei der Datenübertragung durch das Hin– und Her–Senden des Pakets. Das kann sich negativ auf die Performanz der Anwendung auswirken. Zum anderen kann es passieren, dass ein Teil des Pakets verloren geht, wenn die Größe eines Pakets die vorher vom Controller festgelegte maximale Anzahl an Bytes überschreitet, die an ihn gesendet werden können. Die zweite Möglichlichkeit, die Pakte auf dem Router zwischen zu speichern, weist die genannten Nachteile nicht auf, allerdings ist deren Umsetzung um ein Vielfaches aufwendiger. Den zwischengespeicherten Paketen muss hierzu eine eindeutige Buffer–ID zugewiesen werden, zudem entsteht durch die Verwaltung der zwischengespeicherten Pakete ein deutlicher Mehraufwand.
In Folge der genannten Überlegungen wurde für die Architektur des Prototyps die erste Möglichkeit gewählt. Diese erzeugt zwar einen Overhead, ausschlaggebend ist jedoch, dass dadurch zusätzlicher Verwaltungsaufwand erspart sowie eine weitere Fehlerquelle vermieden werden. Letzteres ist daher wichtig, da sich die Fehlersuche auf dem Router als schwer erweist, der Aufbau der einzelnen Module soll deswegen einfach, und somit leichter testbar, gehalten werden.

#### 3.5.2.3. Verbindungsabbruch

Für den Fall, dass eine zuvor hergestellte Verbindung abbricht und es nicht möglich ist, diese wiederherzustellen, wechselt der Router in den sogenannten *emergency mode* (2.1.2.2). In einem solchem Fall muss die MS PIC–Komponente davon in Kenntnis gesetzt werden. Hierfür sendet die RE–Komponente über den *IPCMgrRE* eine entsprechende Nachricht an die MS PIC–Komponente. Der *CtrCommMgr* versucht in periodischen Zeitabständen die Verbindung wiederherzustellen. Sobald die Verbindung wiederhergestellt ist, arbeiten die Komponenten wie gewohnt weiter. Die MS PIC–Komponente wird dazu über den *IPCMgrRE* informiert, in den normalen Modus zurückzuwechseln.

## 3.6. MultiServices PIC Komponente

Die Komponente, die auf der MultiServices PIC läuft, übernimmt folgende Aufgaben:

- Verwaltung der Flow–Tabelle (3.6.1)
- Paketverarbeitung (3.6.2)
- Erfassung von Statistiken (3.7)

Die einzelnen Module der MultiServices PIC Komponente sind in Abbildung 3.3 dargestellt. Im Folgenden wird erläutert, welche Aufgaben von welchen Modulen übernommen werden.

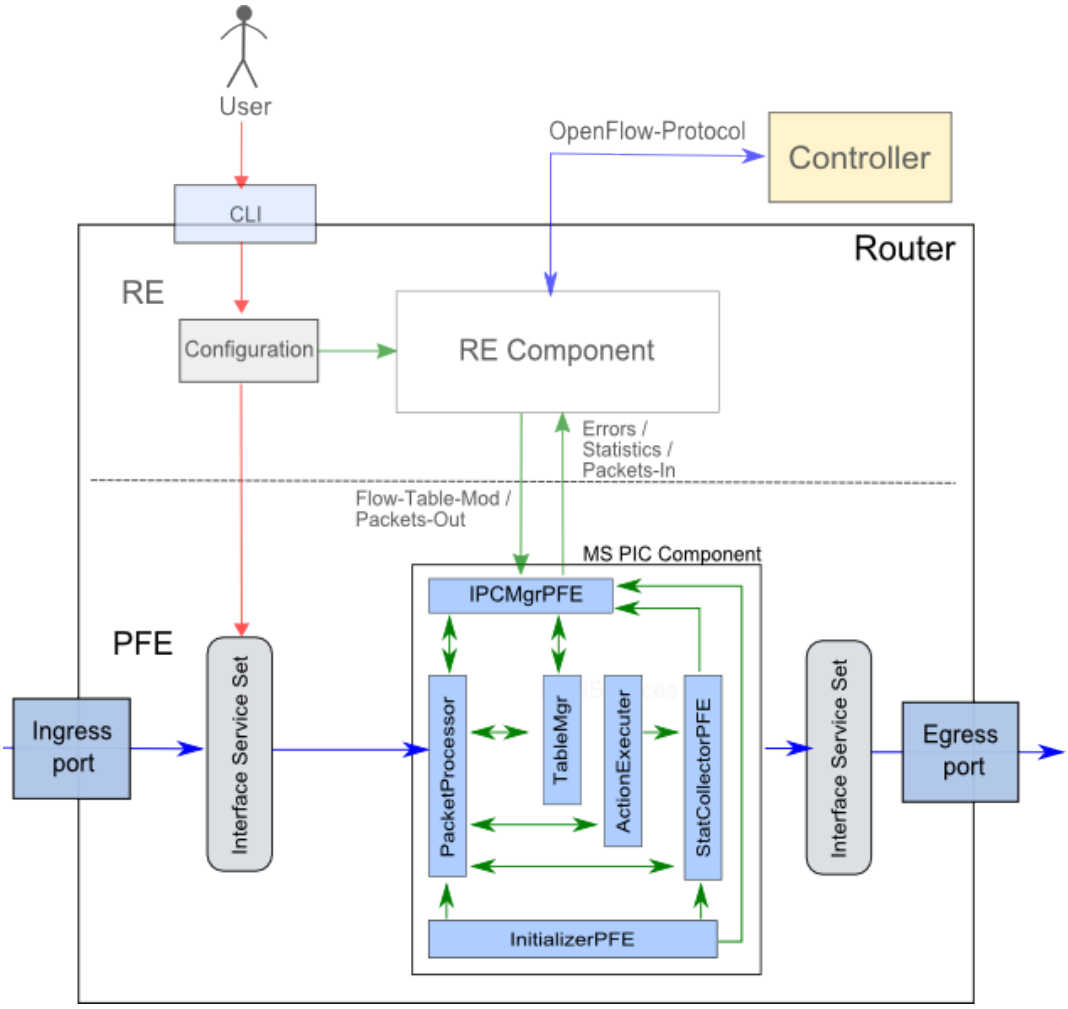


**Abbildung 3.3.:** Architektur der MultiServices PIC Komponente

### 3.6.1. Verwaltung der Flow–Tabelle

Die Verwaltung der Flow–Tabelle erfolgt durch den *TableMgr*. Über ihn kann zum einen die Flow–Tabelle manipuliert werden, zum anderen kann während der Paketverarbeitung nach passenden Einträgen gesucht werden. Da bei der Paketverarbeitung sehr häufig auf die Flow–Tabelle zugegriffen werden muss, wird diese auf der MultiServices PIC–Komponente realisiert. Dadurch entsteht weniger Kommunikationsaufwand mit der Routing Engine–Komponente, wodurch die Verarbeitungsgeschwindigkeit der Anwendung erhöht wird.

#### 3.6.1.1. Anfragen

Anfragen des Controllers, Einträge in die Tabelle einzufügen, aus der Tabelle zu entfernen oder zu modifizieren, werden von der RE–Komponente empfangen und an die MS PIC–Komponente weitergeleitet. Der *IPCMgrPFE* ist auf Seiten der MS PIC–Komponente für die Interprozesskommunikation zuständig und leitet empfangene Nachrichten bezüglich der Manipulation der Flow–Tabelle an den *TableMgr* weiter. Anders als bei der Verarbeitung von Paketen, ist die dadurch entstehende Verzögerung bei der Ausführung der Anfrage des Controllers zu vernachlässigen, da bei weitem nicht so viele Anfragen zur Modifikation der Flow–Tabelle gestellt werden, wie Pakete verarbeitet werden müssen.

#### 3.6.1.2. Operationen zur Manipulation der Flow–Tabelle

**Hinzufügen eines Eintrags**
Die notwendigen Schritte, um einen neuen Flow–Eintrag zur Flow–Tabelle hinzuzufügen, sind in Abbildung 3.4 dargestellt. Im ersten Schritt wird die Validität des einzufügenden Eintrags geprüft. Wenn der Eintrag nicht valide ist, wird eine Fehlermeldung an den Controller gesendet und der Eintrag wird nicht in die Tabelle eingetragen. Anderenfalls wird, falls gefordert, in der Tabelle nach einem Eintrag gesucht, der sich mit dem einzufügenden Eintrag überschneidet. Für denn Fall, dass ein solcher Eintrag vorhanden ist, wird der Eintrag nicht eingefügt und es wird eine Fehlermeldung an den Controller gesendet. Ansonsten wird nachgeschaut, ob bereits ein Eintrag mit den selben Header–Feldern und der selben Priorität in der Tabelle existiert. Existiert kein solcher Eintrag, wird der neue Eintrag hinzugefügt. Ist ein solcher Eintrag vorhanden, wird dieser aus der Tabelle entfernt bevor der neue Eintrag eingefügt wird.

**Entfernen eines Eintrags**
Beim Entfernen eines Eintrags wird in der Tabelle nach dem zu entfernenden Eintrag gesucht. Wird kein passender Eintrag gefunden, ist die Operation beendet. Anderenfalls wird der Eintrag aus der Tabelle entfernt. Wenn für diesen Eintrag spezifiziert ist, dass der Controller beim Entfernen benachrichtigt werden soll, so wird über den *IPCMgrPFE* die RE–Komponente aufgefordert, eine entsprechende Nachricht zu generieren und an den Controller zu senden.

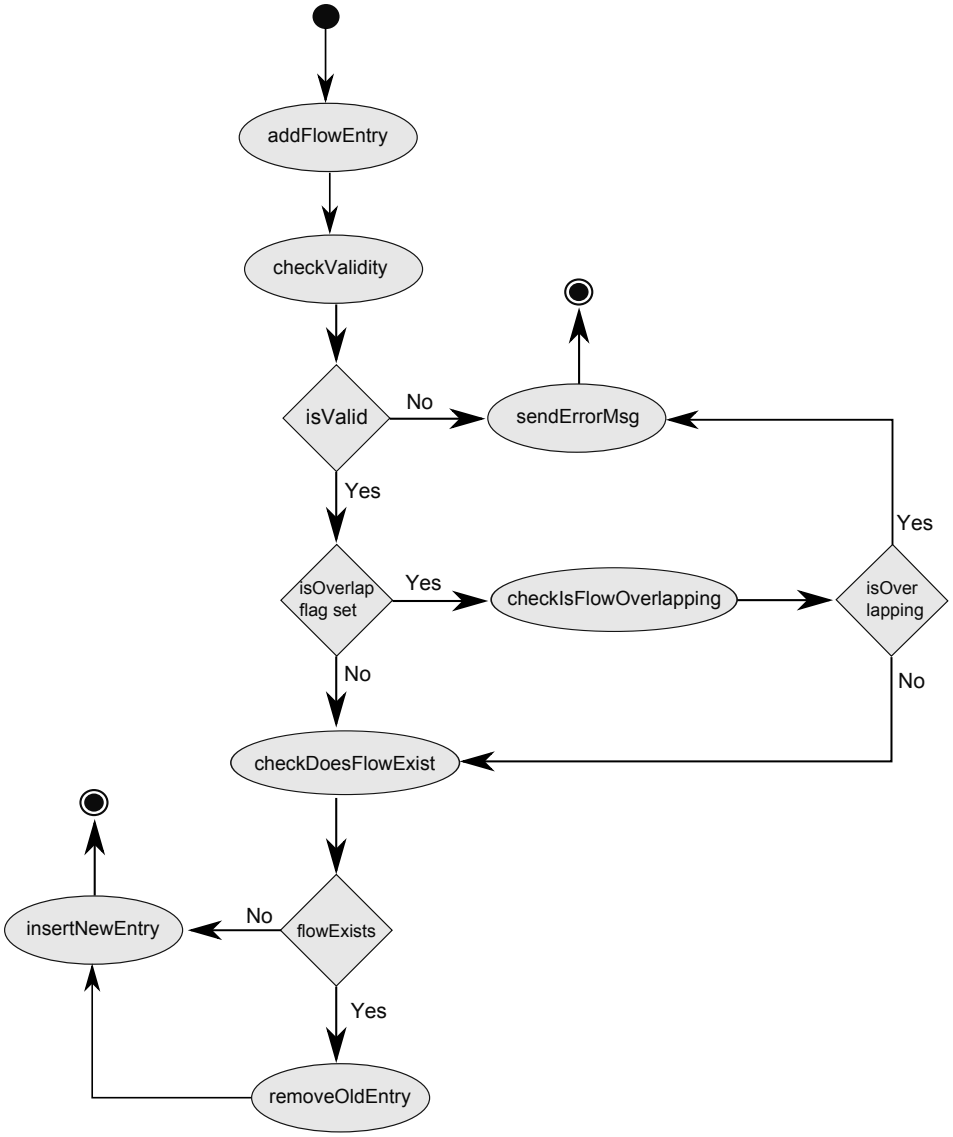


**Abbildung 3.4.:** Einfügen eines Eintrags in die Flow–Tabelle

**Modifizieren eines Eintrags**
Ein neuer Eintrag wird in die Flow–Tabelle eingetragen, falls bei der Modifikations–Operation kein passender Eintrag gefunden wird. Die Informationen der Modifikations–Anfrage werden hierbei für den neuen Eintrag übernommen. Für den Fall, dass ein passender Eintrag gefunden wird, werden die definierten Modifikationen vorgenommen.

#### 3.6.1.3. Locking

Die Verarbeitung der Pakete wird durch mehrere parallel laufende Threads verrichtet, die gleichzeitig auf die Flow–Tabelle zugreifen. Damit es während einer Änderung an der Flow–Tabelle nicht zu Inkonsistenzen kommt, wird *Locking* verwendet. Der Zugriff auf die Tabelle wird hierbei solange gesperrt, bis die Änderung an der Tabelle abgeschlossen ist. In dieser Zeit ist die Verarbeitung der Pakete stillgelegt. Dies wirkt sich bei Szenarien mit wenigen Manipulationen der Flow–Tabelle nur geringfügig auf die Verarbeitungsgeschwindigkeit aus. Bei Szenarien mit vielen Tabellenänderungen und hohem Datenverkehr könnte dies jedoch unter Umständen zu Problemen führen. So kann es neben einer verringerten Verarbeitungsgeschwindigkeit auch zu Paketverlusten kommen. Hintergrund hierfür ist, dass Pakete über gemeinsame FIFO[1]–Queues an die paketverarbeitenden Threads verteilt werden. Die Größe der Queues ist hierbei begrenzt, so dass bei längeren Verzögerungen die Queues überlaufen und Pakete verloren gehen können.

#### 3.6.1.4. Aufspalten der Tabelle

Das Aufspalten der Tabelle stellt einen Ansatz zur Beschleunigung der Nachschlageoperation dar. Die Tabelle wird dabei in zwei Tabellen aufgeteilt, Flow–Einträge mit wildcards und Flow–Einträge ohne wildcards werden in jeweils unterschiedliche Tabellen eingetragen. Da Flow–Einträge ohne wildcards eine höhere Priorität haben, wird zuerst die Tabelle durchsucht, in der diese eingetragen sind. Wird ein passender Eintrag gefunden, ist es nicht mehr nötig, in der anderen Tabelle nachzuschlagen. Auf diese Weise kann das Nachschlagen eines Eintrags beschleunigt werden.
Eine weitere Beschleunigung kann durch eine zusätzliche Aufspaltung der beiden Flow–Tabellen erzielt werden. Die Aufteilung der Flow–Einträge erfolgt nach deren Priorität, so dass jede Tabelle eine Priorität zugewiesen bekommt. Das Nachschlagen der Einträge erfolgt in absteigender Priorität der Flow–Tabellen. Sobald in einer Tabelle ein passender Eintrag gefunden wird, ist die Suche beendet.
Die Verwendung mehrerer Flow–Tabellen verbessert zwar die Performanz der Anwendung, jedoch entsteht dadurch ein Mehraufwand durch die Verwaltung mehrerer Tabellen. Daher wurde für die prototypische Umsetzung auch hier der Ansatz mit dem geringsten Aufwand gewählt, indem die Flow–Einträge in einer Tabelle gespeichert werden.

### 3.6.2. Paketverarbeitung

#### 3.6.2.1. Beteiligte Module

Die Module *PacketProcessor*, *TableMgr* und *ActionExecutor* übernehmen die Verarbeitung der Pakete. Der *PacketProcessor* empfängt die Pakete und extrahiert die benötigten Informationen,

[1] First In – First Out

um in der Flow–Tabelle nach einem passenden Eintrag zu suchen. Für den Fall, dass die Anfrage beim *TableMgr* einen Eintrag zurück liefert, wird der *ActionExecutor* beauftragt, die dazugehörigen Aktionen auszuführen. Falls kein Eintrag in der Tabelle gefunden wird, wird über den *IPCMgr* eine Nachricht an die RE–Komponente gesendet, um den Controller diesbezüglich zu kontaktieren. Um die Performanz dieses Vorgangs zu beschleunigen, wird die Gegebenheit genutzt, dass die MultiServices PIC über acht Prozessorkerne verfügt, von denen eine bestimmte Anzahl der OpenFlow–Anwendung zugewiesen werden können. Dies ermöglicht eine parallele Verarbeitung der Pakete.

#### 3.6.2.2. Gemeinsam genutzte Ressourcen

Da die Verarbeitung der Pakete durch mehrere, parallel laufende Threads erfolgt, muss je nachdem, ob es sich um lesende oder schreibende Zugriffe handelt, der Zugriff auf gemeinsam genutzte Ressourcen synchronisiert werden.

**Lesende Zugriffe**
Mehrere Threads können versuchen gleichzeitig auf die Flow–Tabelle zuzugreifen. Es handelt sich hierbei um lesende Zugriffe, so dass keine Synchronisierung notwendig ist.

**Schreibende Zugriffe**
Falls mehrere Threads gleichzeitig versuchen, eine Nachricht über den *IPCMgrPFE* an die RE–Komponenten zu senden, ist eine Synchronisierung notwendig, da nur eine Verbindung zur RE–Komponente existiert. Die Synchronisierung erfolgt über den *Locking*–Mechanismus, der durch das Junos SDK bereitgestellt wird.
Bei der Synchronisierung kann es zu Verzögerungen kommen, die im Regelfall zu vernachlässigen sind. Eine Ausnahme stellen Nachrichten an die RE–Komponente dar, die besagen, dass eine *Packet–In* Nachricht an den Controller gesendet werden muss. In so einem Fall kann es zu einer erheblichen Behinderung der Paketverarbeitung kommen. Durch Aktivierung der in Abschnitt 2.2.2 beschriebenen *Flow Affinity* können die Auswirkungen der Behinderungen reduziert werden, indem Pakete eines Flows immer vom selben Prozessorkern verarbeitet werden. Dadurch sind nur Prozessorkerne betroffen, welche Flows bearbeiten, für die es keine Einträge in der Tabelle gibt und somit der Controller über den *IPCMgrPFE* kontaktiert werden muss. Die Paketverarbeitung von Flows, bei denen nicht der Controller kontaktiert werden muss, läuft dann ungehindert weiter.

#### 3.6.2.3. Parsen eines Pakets

In Abbildung 3.5 ist der Parsvorgang eines Pakets dargestellt, um die benötigten Informationen für das Nachschlagen in der Flow–Tabelle zu extrahieren. Wie der Abbildung zu entnehmen ist, wird zunächst der Port bestimmt, an dem das Paket eingetroffen ist. Anschließend werden die Ethernet Quell– und Zieladresse sowie der Ethernettyp ausgelesen. Das weitere Vorgehen hängt vom Ethernettyp des Pakets ab. Folgende Typen werden hierbei betrachtet:

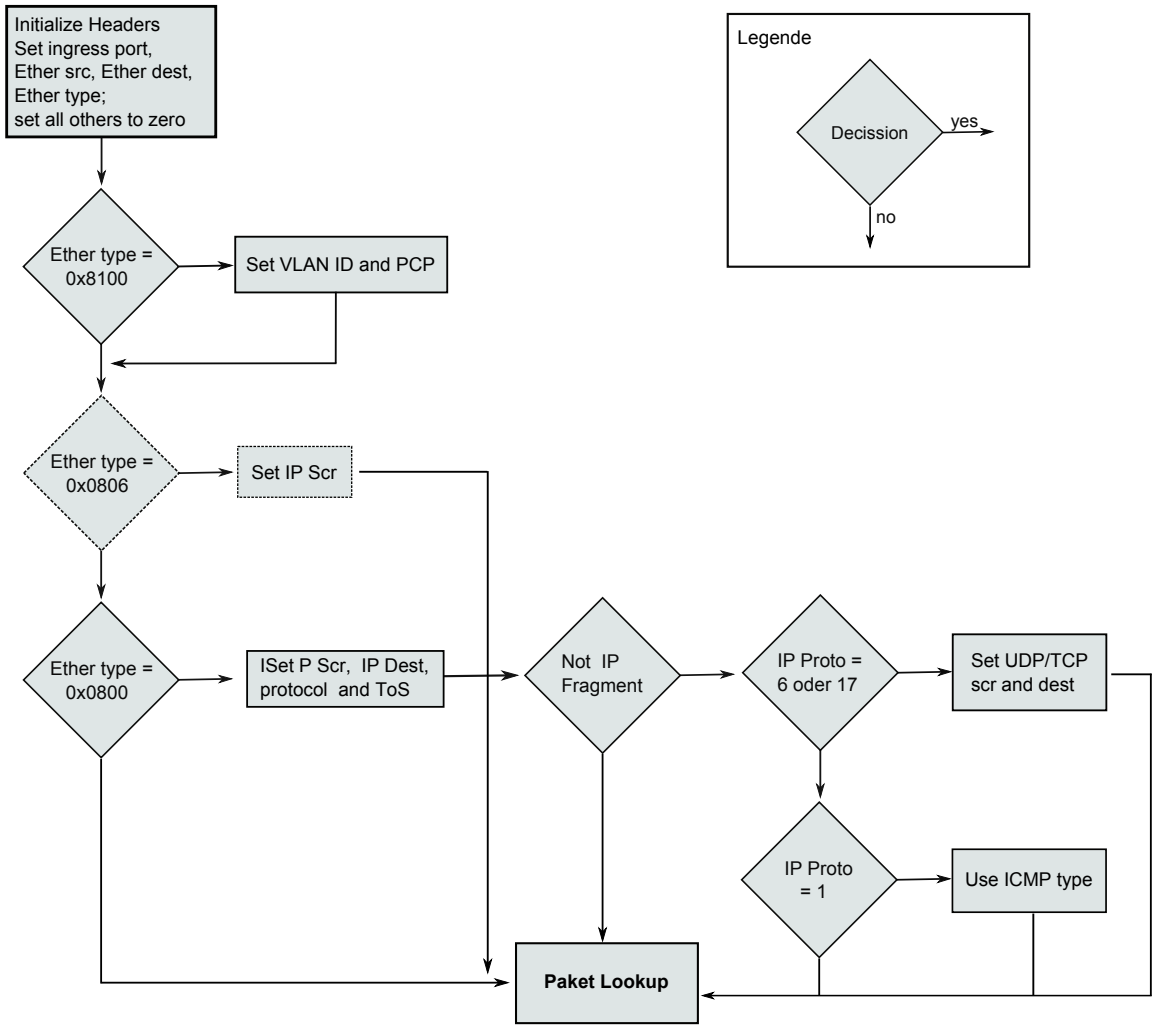


**Abbildung 3.5.:** Parsen der Header–Felder [opeb]

- Ethernettyp 0x8100 – VLAN : Handelt es sich um ein VLAN[2]–Paket, wird die VLAN ID und die VLAN Priorität (PCP) für das Nachschlagen in der Tabelle verwendet.
- Ethernettyp 0x0806 – ARP : Bei ARP[3]–Paketen wird die Quell–IP Adresse extrahiert.
- Ethernettyp 0x0800 – IP : Für IP[4]–Pakete werden die Quell– und Ziel–IP Adresse, das Protokoll sowie das ToS[5]–Feld extrahiert. Weiterhin wird, falls es sich nicht um ein

[2]Virtual Local Area Network
[3]Address Resolution Protocol
[4]Internet Protocol
[5]Type of Service

IP–Fragment handelt, das IP–Protokoll untersucht. Dabei werden folgende Protokolle unterschieden:

- Bei TCP[6]– oder UDP[7]–Paketen werden die Quell– und Zielports ausgelesen.
- Bei ICMP[8]–Paketen werden das Typ–Feld sowie das Code–Feld ausgelesen.

Ist das Paket vollständig geparst, wird anhand der extrahierten Header–Felder in der Tabelle nach einem passenden Eintrag gesucht.

## 3.7. Erfassung von Statistiken

Sowohl die Routing Engine–Komponente als auch die MultiServices PIC–Komponente erfassen Statistiken. Hierfür ist auf Seiten der RE–Komponente der *StatCollectorRE* und auf Seiten der MS PIC–Komponente der *StatCollectorPFE* verantwortlich.

### 3.7.1. Statistiken der RE–Komponente

Die RE–Komponente sammelt Statistiken über den Datenverkehr an den physikalischen Schnittstellen. Das Junos SDK bietet hierfür eine Bibliothek an, die *libjunos–stat* (2.3.2.5), die es ermöglicht, alle bis auf eine in der OpenFlow–Spezifikation definierten Statistiken über den Datenverkehr an physikalischen Schnittstellen zu erfassen. Die durch die Bibliothek ermöglicht e Erfassung von Statistiken ist für die prototypische Realisierung von OpenFlow ausreichend, somit wird diese Bibliothek für die Erfassung der Statistiken auf Seiten der RE–Komponente verwendet.

### 3.7.2. Statistiken der MS PIC–Komponente

Die MS PIC–Komponente erfasst Statistiken über die Flow–Tabelle und über einzelne Flows. Diese werden vom *TableMgr* während der Paketverarbeitung erfasst und an den *StatCollector-PFE* übergeben.

[6]Transmission Control Protocol
[7]User Datagram Protocol
[8]Internet Control Message Protocol

### 3.7.3. Verwaltung der Statistiken

Die Verwaltung der Statistiken kann entweder über eine verteilte oder über eine zentrale Verwaltung erfolgen.

**Verteilte Verwaltung**
Bei der verteilten Verwaltung werden die Statistiken auf der jeweiligen Komponente verwaltet. Die RE–Komponente antwortet umgehend bei Statistikanfragen seitens des Controllers, die sie selbst beantworten kann, anderenfalls muss sie die benötigten Daten von der MS PIC–Komponente abfragen. Dadurch entsteht eine Verzögerung bei der Beantwortung der Anfrage. Damit die RE–Komponente nicht blockiert wird, muss weiterhin die Anfrage auf der RE–Komponente zwischengespeichert werden, bis die MS PIC–Komponente die Daten gesendet hat. Außerdem müssen für denn Fall, dass der Controller mehrere Statistikanfragen sendet, diese synchronisiert werden. Dies ist mit zusätzlichem Mehraufwand verbunden.

**Zentrale Verwaltung**
Die Einführung einer zentralen Sammelstelle der Statistiken stellt einen alternativen Ansatz dar, der mit weniger Mehraufwand einhergeht. Der *StatCollectorRE* der RE–Komponente übernimmt hierbei die Funktion der zentralen Sammelstelle.
Da die RE–Komponente im Gegensatz zur MultiServices PIC–Komponetne über einen persistenten Speicher verfügt, wird die zentrale Sammelstelle auf der RE–Komponente realisiert. Auf dies Weise gehen Statistiken auch bei einem Absturz oder bei Neustart der MultiServices PIC nicht verloren und können über einen längeren Zeitraum hinweg gespeichert werden.
Ein weiterer Vorteil der Verwaltung der Statistiken auf der RE–Komponente ergibt sich daraus, dass die Kommunikation mit dem Controller ebenfalls auf dieser Komponente realisiert ist. Statistikanfragen seitens des Controllers können dadurch direkt beantwortet werden, ohne erst die Daten von der MS PIC–Komponente anzufordern.
Um die zentrale Sammelstelle auf einem aktuellen Stand zu halten, sendet die MS PIC–Komponente in periodischen Zeitabständen die erfassten Statistiken an die RE–Komponente. Dies erfolgt durch einen separaten Prozess, so dass die Paketverarbeitung und die Verwaltung der Flow–Tabelle nicht beeinträchtigt werden. Hierzu wird der Mechanismus zur Interprozesskommunikation genutzt. Der *IPCMgrRE* empfängt die von der MS PIC–Komponente gesendeten Statistiken und leitet sie an den *StatCollectorRE* weiter.

Zu beachten ist, dass bei beiden Ansätzen die Schwierigkeit besteht, die angeforderten Daten zeitlich abzugleichen, wenn Statistikanfragen Statistiken beider Komponenten beinhalten.
Für die prototypische Umsetzung von OpenFlow wird der Ansatz der zentralen Verwaltung gewählt. Ausschlag gebend hierbei ist der geringere Aufwand bei der Umsetzung sowie die Möglichkeit der persistenten Speicherung der Statistiken.

## 3.8. Fehlerbehandlung

Dieser Abschnitt befasst sich mit der Behandlung von Fehlern, die bei der Kommunikation mit einem externen Controller sowie während der Paketverarbeitung auftreten können.
Beim Auftreten eines Fehlers wird eine entsprechende Fehlernachricht an den Controller gesendet. Die Nachricht enthält einen *Fehler–Typ* und einen *Fehler–Code*. Der *Fehler–Typ* gibt an, welche Art von Fehler aufgetreten ist, der *Fehler–Code* gibt eine Beschreibung des Fehlers an.
Auf Seiten der RE–Komponente können zwei Fehlertypen auftreten. Zum einen kann ein Fehler beim *Handshake* eintreten, falls die unterstützten OpenFlow–Versionen nicht übereinstimmen. Zum anderen tritt ein Fehler ein, falls der Controller eine Anfrage sendet, die vom Router nicht unterstützt wird. Beispielsweise können dies Anfragen bestimmter Statistiken sein, die von der Anwendung nicht erfasst werden.
Weitere Fehler können auf Seiten der MS PIC–Komponente eintreten. Diese werden über den *IPCMgrPFE* an die RE–Komponente gesendet, die sie an den Controller weiterleitet. Solche Fehler können beim Einfügen neuer Einträge in die Flow–Tabelle auftreten, falls versucht wird, einen Eintrag einzufügen, der bereits in der Tabelle existiert.

## 3.9. Interprozesskommunikation

In diesem Abschnitt wird die Interprozesskommunikation (IPC) zwischen der Routing Engine–Komponente und der MultiServices PIC–Komponente vorgestellt. Hierfür wird die Bibliothek *libconn* des Junos SDK genutzt. Wie in 2.3.2.2 beschrieben, wird dazu das Client–Server–Modell verwendet. Die Rolle des Servers übernimmt die RE–Komponente, da diese im Vergleich zur MS PIC–Komponente in der Regel weniger Zeit braucht, bis sie gestartet ist. Sobald die MS PIC–Komponente gestartet ist, erstellt sie eine Clientverbindung zur RE–Komponente. Die Verbindung kann synchron oder asynchron erfolgen. Für die Realisierung von OpenFlow wird eine asynchrone Verbindung gewählt, da die Komponenten für die jeweils gesendete Nachricht keine Antwort erwarten.
Hierbei ist zu beachten, dass die MultiServices PIC eine 64–Bit Architektur hat. Das bedeutet, dass sich die Größe der Strukturen, die von der RE–Komponente gesendet werden, von denen auf der MS PIC–Komponente unterscheiden können.

Nachfolgend wird aufgeführt, in welchen Situationen Nachrichten zwischen beiden Komponenten ausgetauscht werden. Die entsprechende Umsetzung der Nachrichtentypen ist in 4.6 beschrieben.

Die RE–Komponente sendet folgende Nachrichten an die MS PIC–Komponente:

- Anfragen des Controllers, Flow–Einträge in die Tabelle einzufügen, zu löschen oder zu modifizieren werden an die MS PIC–Komponente weitergeleitet

- Beim Abbruch der Verbindung und erfolglosen Versuchen, diese wiederherzustellen, wird eine Nachricht gesendet, in der gefordert ist in den *emergency mode* zu wechseln
- Anfragen des Controllers, bestimmte Pakete über einen angegebenen Port auszusenden, werden an die MS PIC–Komponente weitergeleitet

Die MS PIC–Komponente sendet folgende Nachrichten an die RE–Komponente:

- Nach erfolgreichem Start der Anwendung
- Beim Auftreten von Fehlern während der Manipulation der Flow–Tabelle
- Beim Entfernen eines Flow–Eintags aus der Flow–Tabelle
- Während der Paketverarbeitung wird für ein Paket kein passender Eintrag in der Flow–Tabelle gefunden
- Statistiken werden in periodischen Zeitabständen gesendet

# 4. Implementierung

## 4.1. Überblick

Nachdem im vorherigen Kapitel die Architektur für die Umsetzung von OpenFlow beschrieben wurde, soll in diesem Kapitel der anhand dieser Architektur entwickelte Prototyp vorgestellt werden. Es werden zunächst die Rahmenbedingungen definiert, anschließend wird auf die Implementierung der einzelnen Module eingegangen.

## 4.2. Rahmenbedingungen

**Programmiersprache**
Die von Junos bereitgestellten APIs sind, wie in Kapitel 2 beschrieben, in der Programmiersprache C geschrieben. Der Prototyp wird daher ebenfalls in der Programmiersprache C implementiert.

**Zielsetzung**
Der Prototyp soll als Machbarkeitsnachweis der entwickelten Architektur aus Kapitel 3 dienen. Hierfür werden Ansprüche an Leistungsmerkmalen, wie die Verarbeitungsgeschwindigkeit von Paketen oder Nachrichten vom Controller, zunächst in den Hintergrund gestellt. Als vorrangiges Ziel wird in erster Linie die Implementierung der Basisfunktionalität verfolgt. Im Vordergrund stehen hierbei die Kommunikation mit einem externen Controller, die Verwaltung der Flow–Tabelle sowie die Verarbeitung von Paketen.

**Einschränkungen**
Der Juniper M7i Multiservice Edge Router, der für diese Arbeit zur Verfügung steht (siehe 2.2), unterstützt kein *Level 2* Routing. Das heißt, es ist nicht möglich auf *Schicht 2–Daten* [1] der Pakete zuzugreifen, so wie es in der OpenFlow–Spezifikation [opeb] vorgesehen ist. Für die Implementierung hat dies folgende Auswirkungen:

- *Nachschlagen von Einträgen in der Flow–Tabelle:* Das Nachschlagen von passenden Einträgen für Pakete in der Flow–Tabelle erfolgt anhand der Header–Felder der jeweiligen Pakete. Das beinhaltet unter anderem die Ziel– sowie Quell– MAC Adresse eines Pakets. Aufgrund der genannten Einschränkung werden Schicht 2 – Daten beim Nachschlagen in der Tabelle nicht berücksichtigt.

[1] Als Schicht 2 wird die Sicherungsschicht im OSI—Schichtenmodel [Sta] bezeichnet. Schicht 2–Daten, die für OpenFlow benötigt werden sind insbesondere die Ziel– und Quell– MAC Adresse der Pakete.

- *Senden von OFPT_PACKET_IN –Nachrichten an den Controller:* Wird ein Paket an den Controller gesendet, so fehlen die Informationen über Schicht 2–Daten. Für die Ziel– und Quell– MAC Adresse werden daher Dummy–Daten angegeben.

## 4.3. Erweiterung des Command–Line Interface

Das Command–Line Interface (CLI) muss erweitert werden, damit der Benutzer die benötigten Daten für die Ausführung der Anwendung eingeben kann. Hierfür muss die DDL–Konfigurationsdatei angepasst werden. Die DDL–Konfigurationsdatei besteht aus hierarchisch angeordneten *Konfigurations–Objekten*. Diese legen fest, welche Einstellungen der Benutzer vornehmen kann. Für alle einzugebenden Daten, die in Abschnitt 3.4 näher beschrieben sind, werden in der DDL–Konfigurationsdatei Objekte mit den dazugehörigen Attributen definiert, die im Folgenden beschrieben werden:

- *controller-object*: Das Objekt dient zur Eingabe von Informationen, die für die Verbindung mit einem externen Controller benötigt werden. Als Attribute hat es zum einen die *controller-ip-adr*, zum anderen den *controller-port*.
- *port-object*: Das Objekt dient zur Eingabe der Ports, die für den OpenFlow–Verkehr verwendet werden sollen. Als Attribute werden *port–name*, *port–number* und *port–hw–address* definiert.
- *rule-object*: Das *rule-object* wird für die Erstellung der Interface Service Sets (2.2.3) benötigt. Es legt die Regeln fest, welche Pakete über die Interface Service Sets an die MultiServices PIC–Komponente geleitet werden sollen. Folgende Attribute müssen angegeben werden:
  - *rule–name*: Legt den Namen der Regel fest. Regeln werden anhand des Namens eindeutig identifiziert.
  - *match–direction*: Gibt an, für welche Art von Paketen die Regel angewandt werden soll. Es wird zwischen eingehenden und ausgehenden Paketen unterschieden.
  - *from*: Legt die Vergleichskriterien fest, anhand derer Pakete gefiltert werden. Das Attribut *from* ist vom Typ *match-object*, das im nächsten Unterpunkt beschrieben wird.
  - *then*: Legt fest, was mit einem Paket geschehen soll, das die Vergleichskriterien erfüllt. Als Aktion kann *allow* oder *deny* angegeben werden. Das erstere gibt an, dass das Paket an die MS PIC–Komponente geleitet werden soll. Das letztere gibt an, dass das Paket normal über die Forwarding–Tabelle weitergeleitet werden soll.
- *match-object*: Beinhaltet Vergleichskriterien, die im Attribut *from* angegeben werden können. Es müssen nicht alle vorhandenen Attribute angeben werden. Ein Paket kann anhand folgender Kriterien gefiltert werden:
  - *source–prefix*: Gibt den Netzpräfix der Quell–IP Adresse an.

- *destination–prefix*: Gibt den Netzpräfix der Ziel–IP Adresse an.
- *protocol*: Gibt das Kommunikationsprotokoll an, welches als Vergleichskriterium angegeben werden kann.
- *source–port*: Gibt den Quell–Port eines Pakets an.
- *destination–port*: Gibt den Ziel–Port eines Pakets an.

Damit die beschriebenen Objekte im Command–Line Interface genutzt werden können, müssen sie in die Juniper Konfigurations-Hierarchie integriert werden. Dies wird ebenfalls über die DDL–Konfigurationsdatei gemacht. Hierbei werden die neu definierten Objekte unterhalb des höchsten Hierarchieelementes *juniper–config* eingefügt. Da Erweiterungen von verschiedenen Providern und für verschiedene Anwendungen vorgenommen werden können, werden zwei zusätzliche Hierarchieelemente erstellt. Zum einen wird ein weiteres Element *ikr* hinzugefügt, um kenntlich zu machen, dass die Erweiterung zum Provider *IKR (Institut für Kommunikationsnetze und Rechnersysteme)* gehört. Zum anderen wird unterhalb des *ikr*–Elementes das Objekt *openflow* eingefügt, um zu kennzeichnen, dass die Erweiterungen von der OpenFlow–Anwendung benötigt werden. Alle weiteren Elemente werden unterhalb des *openflow*–Elementes eingefügt.
Um Interface Service Sets für bestimmte Ports des Routers zu definieren, muss das von Juniper bereitgestellte Hierarchieelement *services* ergänzt werden. Unterhalb davon befindet sich das Element *service–set*. Das Element *service–set* wird um ein Element *extension–service* ergänzt, das die oben beschriebenen *rule-object*–Elemente beinhaltet. Ein solcher *Service* kann dann einem Interface des Routers zugeordnet werden. Listing A.25 im Anhang zeigt einen Ausschnitt der DDL–Konfigurationsdatei, in dem das *Service*–Element dargestellt wird.

## 4.4. Routing Engine Komponente

### 4.4.1. Auslesen der Konfiguration

Das Auslesen der Konfiguration übernimmt der *ConfigReader*. Der Ablauf zum Auslesen der Konfiguration wird in Abbildung 4.1 dargestellt. Der Benutzer muss die Eingaben, die er über das Command–Line Interface getätigt hat, mit einem *Commit*–Befehl bestätigen. Sobald der Benutzer den *Commit*–Befehl ausgeführt hat, wird die zuvor definierte Callback–Methode *readConfigCB()* des *ConfigReaders* aufgerufen. Diese durchläuft die Konfiguration des Routers und liest die für die Anwendung relevanten Informationen aus. Die Informationen werden an die dafür vorgesehenen Module weitergeleitet. Die IP–Adresse und der Port des Controllers werden über die Methode *setCtrInfo()* an den *CtrCommMgr* übergeben. Angaben über die Ports des Routers, die für OpenFlow genutzt werden können, werden über die Methode *setPortDefinitions()* an den *FeatureMgr* übergeben.

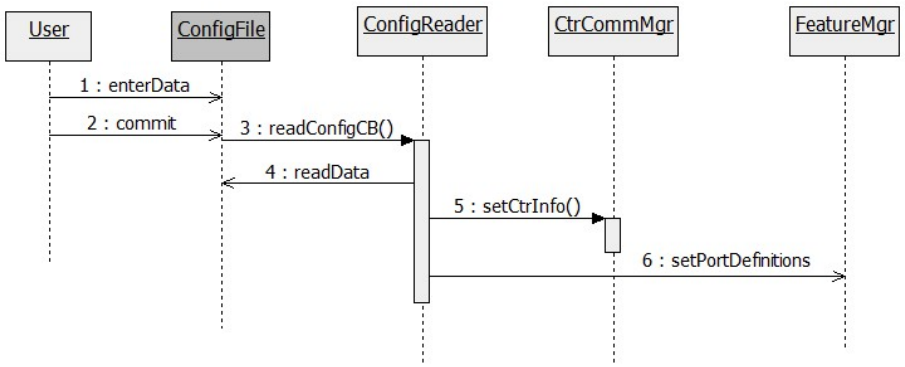


**Abbildung 4.1.:** Ablauf–Auslesen der Konfigurationsdatei

### 4.4.2. Kommunikation mit dem Controller

Die Kommunikation zwischen dem Router und einem externen Controller findet über eine TCP–Socketverbindung statt. Als Kommunikationsprotokoll wird TCP gewählt, da es einen zuverlässigen Datenaustausch ermöglicht. Dies ist für die Funktionalität der Anwendung von hoher Bedeutung. Hierbei übernimmt der Router die Rolle des Clients, der sich zum Controller verbindet. Für den Aufbau der TCP–Socketverbindung werden Funktionen aus der Standard C–Bibliothek verwendet.
Sobald die Verbindung zum Controller hergestellt ist, ist der Router bereit, Nachrichten zu senden und zu empfangen. Der Nachrichtenempfang wird durch eine Ereignissteuerung realisiert. Hierbei können Funktionen aus der Standard C–Bibliothek genutzt werden, um Events und entsprechende Event–Handler anzugeben. Der Vorteil dieses Vorgehens gegenüber einem *Polling* am Socket, bei dem kontinuierlich überprüft wird, ob neue Nachrichten eingetroffen sind, liegt in einer geringeren Beanspruchung des Prozessors.

#### a) Kommunikationsaufbau zum Controller

Der Ablauf des Kommunikationsaufbaus mit einem externen Controller ist in Abbildung 4.2 dargestellt. Sobald die RE–Komponente sowie die MS PIC–Komponente gestartet sind, wird eine Verbindung zum Controller aufgebaut. Die MS PIC–Komponente sendet hierbei eine Nachricht an die RE–Komponente, um dies zu signalisieren. Empfängt der *IPCMgrRE* eine solche Nachricht, ruft er die Methode *connectToCtr()* des *CtrCommMgrs* auf, die eine TCP–Verbindung zum Controller aufbaut. Wird die Verbindung erfolgreich hergestellt, beginnt der *Handshake* [opeb] mit dem Controller. Hiefür werden *OFPT_HELLO*–Nachrichten ausgetauscht. Ist die Prüfung der jeweils unterstützten OpenFlow–Version erfolgreich, sendet der Controller eine *OFPT_FEATURE_REQUEST*–Nachricht. Als Antwort schickt der *CtrCommMgr* eine *OFPT_FEATURE_REPLY*–Nachricht. Diese wird vom *OFMsgCreator* über die Methode *createFeatureReply()* erstellt. Die Informationen, die er hierfür benötigt, holt er sich über die Methode *getFeatures()* vom *FeatureMgr*. Nach dem Austausch dieser Nachrichten ist der *Handshake* beendet und der *CtrCommMgr* wartet auf weitere Nachrichten vom Controller.

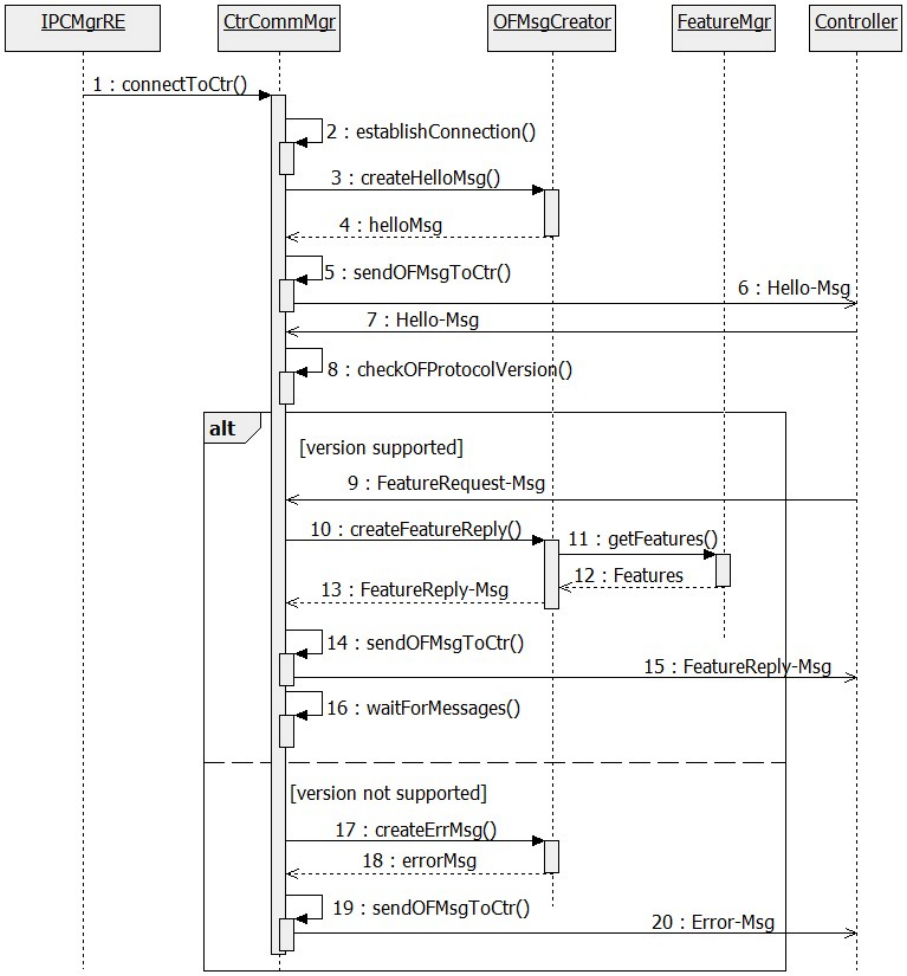


**Abbildung 4.2.:** Ablauf des Kommunikationsaufbaus zum Controller

**b) Nachrichtenaustausch mit dem Controller**
Der Ablauf zur Verarbeitung von Nachrichten vom Controller ist in Abbildung 4.3 dargestellt. Nachfolgend werden die Verarbeitungsschritte für die jeweiligen Nachrichtentypen im Einzelnen erläutert.

- OFPT_STATS_REQUEST: Sobald der Controller eine Anfrage sendet, um bestimmte Statistiken abzufragen, fordert der *CtrCommMgr* den *OFMsgCreator* auf eine entsprechende Antwortnachricht zu generieren. Die Informationen, die der *OFMsgCreator* dafür benötigt, fragt er über die Methode *getStats()* beim *StatCollectorRE* ab. Die generierte Nachricht wird vom *CtrCommMgr* an den Controller gesendet.

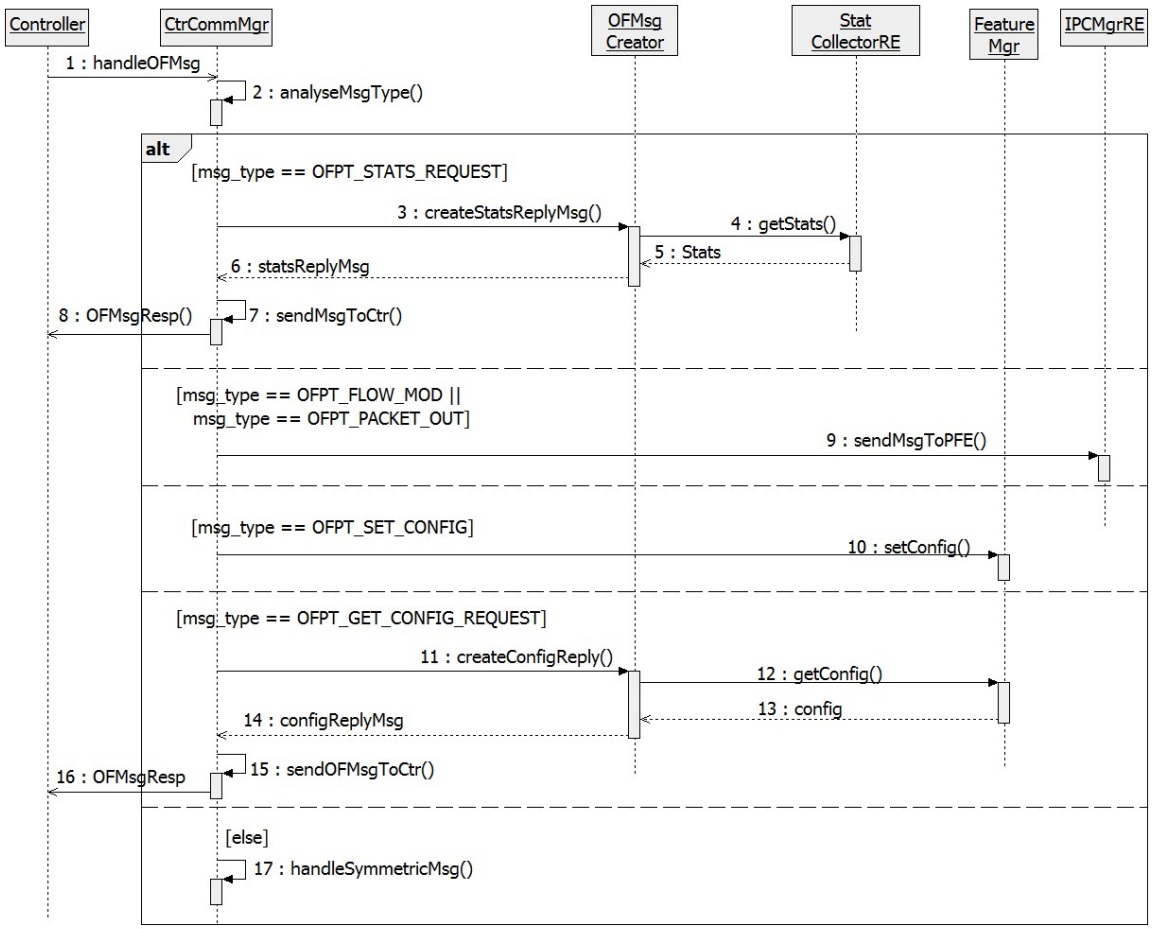


**Abbildung 4.3.:** Verarbeitung von Controller–Nachrichten

- OFPT_FLOW_MOD: Anfragen zur Manipulation der Flow–Tabelle werden über den *IPCMgrRE* an die MS PIC–Komponente weitergeleitet, da diese für die Verwaltung der Flow–Tabelle zuständig ist.
- OFPT_PACKET_OUT: Der *CtrCommMgr* leitet die *OFPT_PACKET_OUT*–Nachricht über den *IPCMgrRE* an die MS PIC–Komponente zur Verarbeitung weiter.
- OFPT_SET_CONFIG: Beim Empfang einer *OFPT_SET_CONFIG*–Nachricht werden über die Methode *setConfig()* die Konfigurationen an den *FeatureMgr* übergeben.
- OFPT_GET_CONFIG_REQUEST: Als Antwort auf diese Nachricht lässt sich der *CtrCommMgr* vom *OFMsgCreator* über die Methode *createConfigReply()* eine Nachricht generieren, die er an den Controller sendet. Der *OFMsgCreator* holt sich die benötigte Information vom *FeatureMgr* über die Methode *getConfig()* .
- Symmetric Messages: Symmetrische Nachrichten, wie *Echo–* und *Echo-Reply–* Nachrichten werden von der Methode *handleSymmetricMsg()* verarbeitet. *Echo–*

Nachrichten werden gesendet, um sicherzustellen, dass die Verbindung zum Kommunikationspartner noch steht.

### 4.4.3. Klassendiagramm

Abbildung 4.4 zeigt das Klassendiagramm der Routing Engine–Komponente, die einzelnen Module werden nachfolgend näher beschrieben.

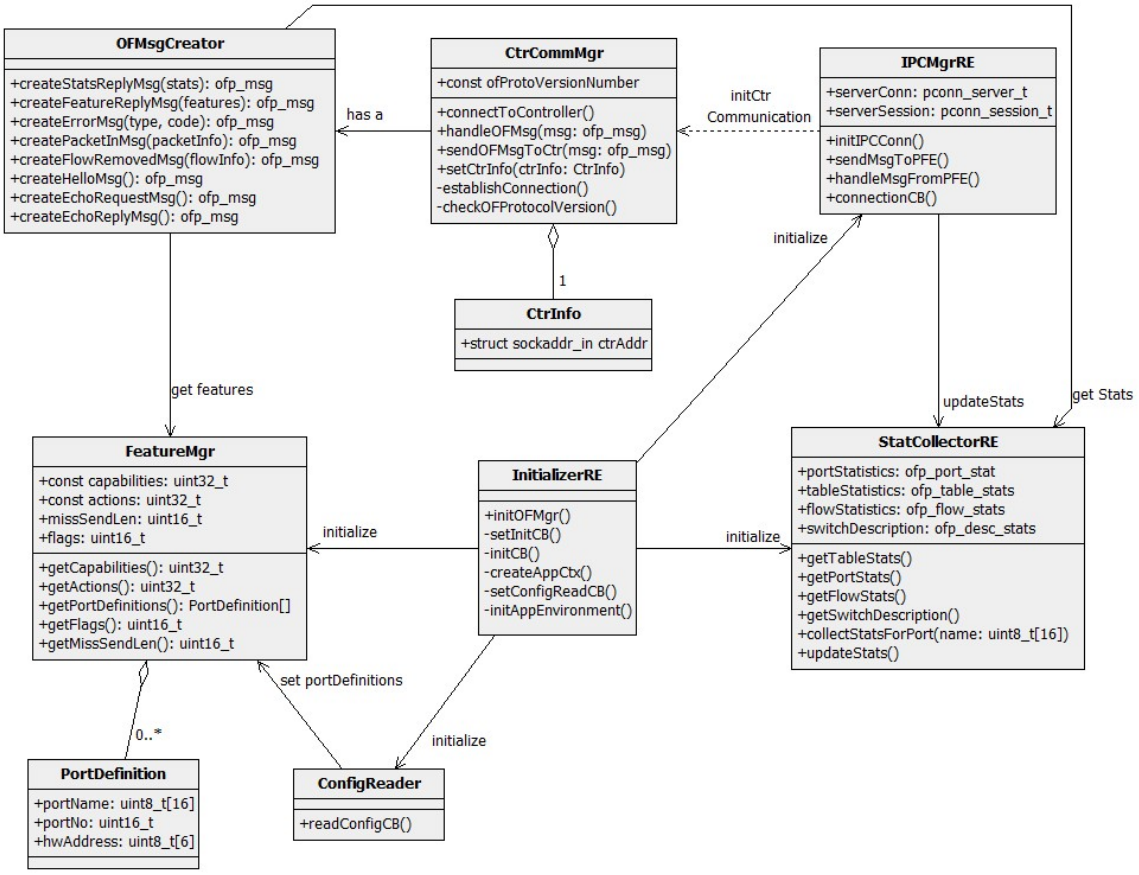


**Abbildung 4.4.:** Klassendiagramm–RE Komponente

- InitializerRE: Der *InitializerRE* initialisiert die RE-Komponente. Dazu wird die Methode *initRE()* aufgerufen. Diese wiederum ruft folgende Methoden in der angegebenen Reihenfolge auf:

  - *createAppCtx()*: Die Methode erstellt einen *Application–Context*[2] für die Komponente. Dazu nutzt sie die Methode *junos_createapp_ctx()*, die das Junos SDK zur Verfügung stellt.
  - *setConfigReadCB()*: Die Methode registriert die Callback–Methode, die das Auslesen der Konfigurationsdatei übernimmt. Bei Änderungen der Konfigurationsdatei wird ebenfalls die angegebene Methode aufgerufen. Hierzu wird die Methode *junos_set_app_cb_config_read()* des Junos SDKs genutzt. Als Parameter wird die Methode *readConfigCB()* vom *ConfigReader()* angegeben.
  - *setInitCB()*: Diese Methode registriert, welche Methode als Callback aufgerufen wird, um applikationsspezifische Initialisierungen durchzuführen. Dazu wird die Methode *junos_set_app_cb_init()* aus dem Junos SDK verwendet. Als Parameter wird die Methode *initCB()* mitgegeben, welche während dem Startup der Komponente als erstes aufgerufen wird. Sie dient dazu, die einzelnen Module zu initialisieren. Es werden die Attribute des *FeatureMgr* und des *StatCollectorRE* sowie die Interprozesskommunikation der RE–Komponente initialisiert.
  - *initAppEnvironment()*: Die Methode dient dazu, die RE–Komponente auf dem Router zu initialisieren und zu starten. Sie nutzt die vom Junos SDK bereitgestellte Methode *junos_app_init()*, die nacheinander die registrierten Callback–Methoden aufruft. Zunächst wird die Initialisierungs-Methode *initCB()* aufgerufen, dann die *readConfigCB()*–Methode.

- ConfigReader: Der *ConfigReader* stellt die Callback–Methode *readConfigCB()* bereit, welche die Konfigurationsdatei ausliest. Diese wird beim Initialisieren der Komponente durch den *InitializerRE* als Callback-Methode registriert.
- CtrCommMgr: Der *CtrCommMgr* ist zuständig für die Kommunikation mit dem externen Controller. Die Informationen, die er zur Herstellung der Verbindung zum Controller benötigt, werden über die Methode *setCtrInfo()* vom *ConfigReader* gesetzt. Diese werden in einem Attribut vom Typ *CtrInfo* gespeichert. Über die Methode *connectToCtr* wird der Kommunikationsaufbau angestoßen. Hierfür wird zunächst in der Methode *establishConnection()* eine TCP–Verbindung zu dem Controller aufgebaut. Der *CtrCommMgr* speichert weiterhin die OpenFlow–Version, die vom Router unterstützt wird. Diese wird in der Methode *checkOFProtocolVersion()* mit der Version verglichen, die der Controller unterstützt. Sobald eine Nachricht vom Controller eintrifft, wird die Methode *handleOFMsg()* aufgerufen. Diese parst den Typ der Nachricht und beantwortet die Anfrage oder leitet gegebenenfalls die Nachricht über den *IPCMgrRE* an die MS PIC–Komponente weiter.
- StatCollectorRE: Der *StatCollectorRE* speichert die erfassten Statistiken als Attribute, die über entsprechende *get()*–Methoden abgefragt werden können. Über die Methode *collectStatsForPort()* erfasst er Statistiken über den Datenverkehr an einem bestimmten

[2]Für jede Komponente wird ein Application Context erstellt. Dieser leitet auftretende Ereignisse an die dafür vorgesehenen Module.

Port. Alle weiteren Statistiken werden ihm von dem *StatCollectorPFE* der MS PIC–Komponente über die Methode *updateStats()* übermittelt.

- OFMsgCreator: Der *OFMsgCreator* wird vom *CtrCommMgr* genutzt, um OpenFlow–Nachrichten zu erstellen, die er an den Controller sendet. Der *OFMsgCreator* bietet folgende Methoden zur Erstellung von Nachrichten:
  - **createHelloMsg():** Erstellt eine *OFPT_HELLO*–Nachricht, die zum Kommunikationsaufbau mit dem Controller gebraucht wird.
  - **createFeatureReplyMsg():** Erstellt eine *OFPT_FEATURE_REPLY*–Nachricht, die an den Controller als Antwort auf eine *OFPT_FEATURE_REQUEST*–Nachricht gesendet wird.
  - **createConfigReply():** Erstellt eine *OFPT_CONFIG_REPLY*–Nachricht, die an den Controller als Antwort auf eine *OFPT_GET_CONFIG_REQUEST*–Nachricht gesendet wird.
  - **createEchoReplyMsg():** Erstellt eine *OFPT_ECHO_REPLY*–Nachricht, die an den Controller als Antwort auf eine *OFPT_ECHO_REQUEST*–Nachricht gesendet wird.
  - **createEchoRequest():** Erstellt eine *OFPT_ECHO_REQUEST*–Nachricht, die an den Controller gesendet wird, um sicherzustellen, dass die Verbindung zum Controller nicht abgebrochen ist.
  - **createErrorMsg():** Erstellt eine *OFPT_ERROR*–Nachricht, die an den Controller gesendet wird, sobald ein Fehler auftritt. Die Nachricht enthält einen *Fehler–Typ* und einen *Fehler–Code*. Der *Fehler–Typ* gibt an, was für eine Art von Fehler aufgetreten ist und der *Fehler–Code* gibt eine Beschreibung des Fehlers an.
  - **createFlowRemMsg():** Erstellt eine *OFPT_FLOW_REMOVED*–Nachricht, die an den Controller gesendet wird, um ihm mitzuteilen, dass ein Eintrag aus der Flow–Tabelle entfernt wurde. Sie enthält Informationen über den entfernten Eintrag und den Grund, warum der Eintrag aus der Tabelle entfernt worden ist.
  - **createPacketInMsg():** Erstellt eine *OFPT_PACKET_IN*–Nachricht, die an den Controller gesendet wird. Sie enthält ein Paket und den Grund, warum das Paket an den Controller gesendet wird.
  - **createStatsReplyMsg():** Erstellt eine *OFPT_STATS_REPLY*–Nachricht, die an den Controller als Antwort auf eine *OFPT_STATS_REQUEST*–Nachricht gesendet wird. Sie entält die angeforderten Statistiken.
- FeatureMgr: Der *FeatureMgr* speichert die Informationen über die Fähigkeiten, über die Ports und über die Konfiguration des Routers. Die Fähigkeiten geben die „Capabilities“ an und welche Aktionen vom Router unterstützt werden. „Capabilities“ beinhalten unter anderem Informationen, welche Statistiken erhoben werden und von Controller abgefragt werden können.
  Der Controller kann zwei Arten von Konfigurationen vornehmen. Zum einen kann

er Angaben darüber machen, wie IP–Fragmente zu verarbeiten sind. Dies wird im Attribut *flags* gespeichert. Zum anderen kann er die maximale Anzahl an Bytes eines Pakets angeben, das an ihn geschickt werden sollen. Dies wird im Attribut *missSendLen* gespeichert. Die Namensgebung der Attribute ist an die OpenFlow–Spezifikation angelehnt.
Die Informationen über die Ports werden in einer Liste gespeichert, die aus Elementen vom Typ *PortDefinition* besteht. Diese speichert den *Name*, die *Portnummer* und *Hardware–Adresse* (MAC–Adresse) eines Ports. Einträge in die Liste werden vom *ConfigReader* während dem Auslesen der Konfigurationsdatei hinzugefügt.

- IPCMgrRE: Der *IPCMgrRE* kümmert sich um die Interprozesskommunikation auf Seiten der Routing Engine–Komponente. Über die Methode *initIPCConn()* wird der *IPCMgrRE* initialisiert. Dabei wird ein Server erstellt, mit dem sich die MS PIC–Komponente verbinden kann. Hierfür kann die Methode *pconn_server_create()* aus der Bibliothek *libconn* (2.3.2.2) des Junos SDKs verwendet werden. Bei der Erstellung des Servers muss zum einen angegeben werden, welcher Port für die Verbindung mit der MS PIC–Komponente verwendet werden soll, zum anderen die maximale Anzahl an Verbindungen, die mit dem *IPCMgrRE* aufgebaut werden können. Weiterhin müssen Callback–Methoden registriert werden. Eine Callback–Methode kümmert sich um Ereignisse, welche die Verbindung zur MS PIC–Komponente betreffen. Eine Andere wird benötigt, um eingehende Nachrichten zu verarbeiten. Als Callback–Methode für die Verbindung zur MS PIC–Komponente wird die Methode *connectionCB()* verwendet. Diese wird aufgerufen, sobald die Verbindung hergestellt ist oder wenn die Verbindung abbricht. Für den Empfang von Nachrichten ist die Methode *handleMsgFromPFE()* zuständig.

## 4.5. MultiServices PIC Komponente

### 4.5.1. Flow–Table Management

Die Flow–Einträge werden in einer zentrale Flow–Tabelle gespeichert, diese wird vom *TableMgr* verwaltet und ist als eine einfach verkettete Liste implementiert. Die Liste besteht aus *FlowEntry*–Elementen, die in Kapitel 2 genauer beschrieben sind. Diese Art der Implementierung wurde gewählt, da sie am einfachsten zu realisieren ist, während die Leistungsfähigkeit des Prototyps nicht im Vordergrund steht. Auf diese Weise werden unnötige Fehlerquellen vermieden, welche aufgrund der Cross–Compilierung[3] nur schwer zu entdecken sind. Neue Elemente werden immer zu Beginn der Liste eingefügt, so dass nicht die gesamte Liste durchlaufen werden muss. Letzteres ist allerdings beim Löschen oder Modifizieren von Einträgen der Fall.
Während der Verarbeitung von Paketen kann es dazu kommen, dass Änderungen an der

[3]Bei der Cross-Compilierung läuft der Compiler auf einem Sysytem (hier: auf der Virtual Build Environment), aber die ausführbaren Programme auf einem anderen System (hier: auf dem Router)

Flow–Tabelle vorgenommen werden. Es können neue Einträge eingefügt werden, bestehende Einträge gelöscht oder modifiziert werden. Damit es nicht zu Inkonsistenzen kommt, wird der Locking–Mechanismus verwendet, der vom Junos SDK bereitgestellt wird. In der Bibliothek *libmp-sdk* werden sogenannte *Spinlocks* zur Verfügung gestellt. Für die Flow–Tabelle wird ein globales Lock erstellt. Bevor auf die Flow–Tabelle zugegriffen werden kann, muss dieses Lock über die Methode *msp_spinlock_lock()* erlangt werden. Das Lock wird sowohl für Nachschlageoperationen, als auch für Manipulationsoperationen gebraucht. Nach Beendigung der Operation wird das Lock über die Methode *msp_spinlock_unlock()* freigegeben. Verwendet wird dieser Locking–Mechanismus in zwei Modulen der MS PIC–Komponente:

- *PacketProcessor*: In der Methode *processPacket()* wird in der Flow–Tabelle nach einem Eintrag gesucht, nachdem das Paket geparst wurde. Da es sich hierbei um einen lesenden Zugriff handelt, muss die Tabelle nicht gesperrt werden, sondern lediglich überprüft werden, ob die Tabelle bereits gesperrt ist. Hierfür kann die Methode *msp_spinlock_trylock()* verwendet werden. Ist dies der Fall, heißt das, dass Änderungen an der Tabelle vorgenommen werden. Der Zugriff auf die Tabelle muss dann verschoben werden, bis die Tabelle nicht mehr gesperrt ist.
- *IPCMgrPFE*: Die Methode *handleMsgFromRE()* verarbeitet Nachrichten von der RE–Komponente. Handelt es sich dabei um eine Nachricht zur Manipulation der Tabelle, so muss das Lock für die Tabelle angefordert werden, bevor Änderungen an der Tabelle vorgenommen werden können. Sobald die Änderungen durchgeführt wurden, muss das Lock wieder freigegeben werden.

### 4.5.2. Paketverarbeitung

Die Verarbeitung von Paketen übernehmen die Module *PacketProcessor*, *TableMgr* und *ActionExecutor*. Der Ablauf zur Verarbeitung eines Pakets ist in Abbildung 4.5 dargestellt.

Der *PacketProcessor* empfängt Pakete, die zur Verarbeitung an die MS PIC–Komponente geschickt werden. Hierbei wird die zuvor definierte Callback–Methode *processPacketCB()* aufgerufen. Diese Methode ruft die Methode *parsePacket()* auf, in der die Header–Felder des Pakets extrahiert werden, die benötig werden, um nach einem passenden Eintrag in der Flow–Tabelle zu suchen. Dieser Parsvorgang wird im nächsten Abschnitt genauer beschrieben. Nach erfolgreicher Durchführung des Parsvorgangs, wird mittels der Methode *getFlowEntry()* nach einem passenden Eintrag in der Flow–Tabelle gesucht. Der *TableMgr* übernimmt die Aktualisierung der Statistiken. Hierfür nutzt er die Methode *updateStats()* des *StatCollectorPFE*. Liefert die Anfrage an den *TableMgr* keinen Eintrag zurück, wird das Paket über den *IPCMgrPFE* an die RE–Komponente gesendet, die es an den Controller weiterleitet. Wird ein passender Eintrag in der Tabelle gefunden, wird das Paket an den *ActionExecutor* übergeben, der die im Eintrag definierten Aktionen ausführt. Sind keine Aktionen im Eintrag definiert, impliziert dies, dass das Paket verworfen werden soll. Ansonsten wird das Paket im Falle einer *FORWARD*–Aktion über den angegebenen Port weitergeleitet oder im Falle einer *MODIFY*–Aktion entsprechen modifiziert.
Das Parsen der Header–Felder eines Pakets wird vom *PacketProcessor* in der Methode

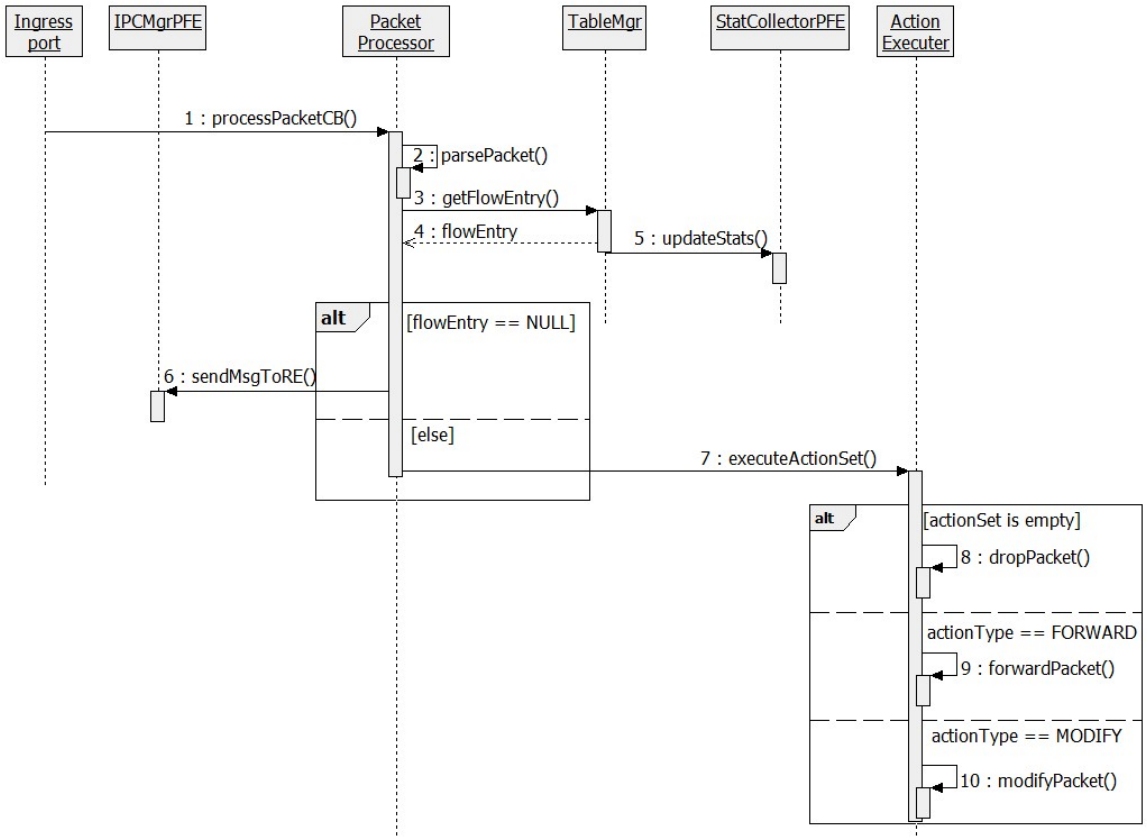


**Abbildung 4.5.:** Ablauf bei der Paketverarbeitung

*parsePacket()* durchgeführt. Pakete werden durch *jbufs* (2.3.2.1) repräsentiert. Das Junos SDK bietet folgende Methoden an, um aus den *jbufs* die gewünschten Informationen auszulesen:

- *jbuf_get_rcvidx()*: Liefert den Eingangsport des Pakets zurück.
- *jbuf_to_d()*: Liefert den IP–Header des Pakets zurück, aus dem sich die Quell– und Ziel–IP Adresse sowie das IP Protokoll auslesen lassen.
- *jbuf_getptr()*: Über diese Methode kann der TCP–Header ausgelesen werden. Dieser beinhaltet den Quell– und den Zielport.

### 4.5.3. Klassendiagramm

Abbildung 4.6 zeigt das Klassendiagramm der MultiServices PIC–Komponente. Nachfolgend werden die einzelnen Module näher beschrieben.

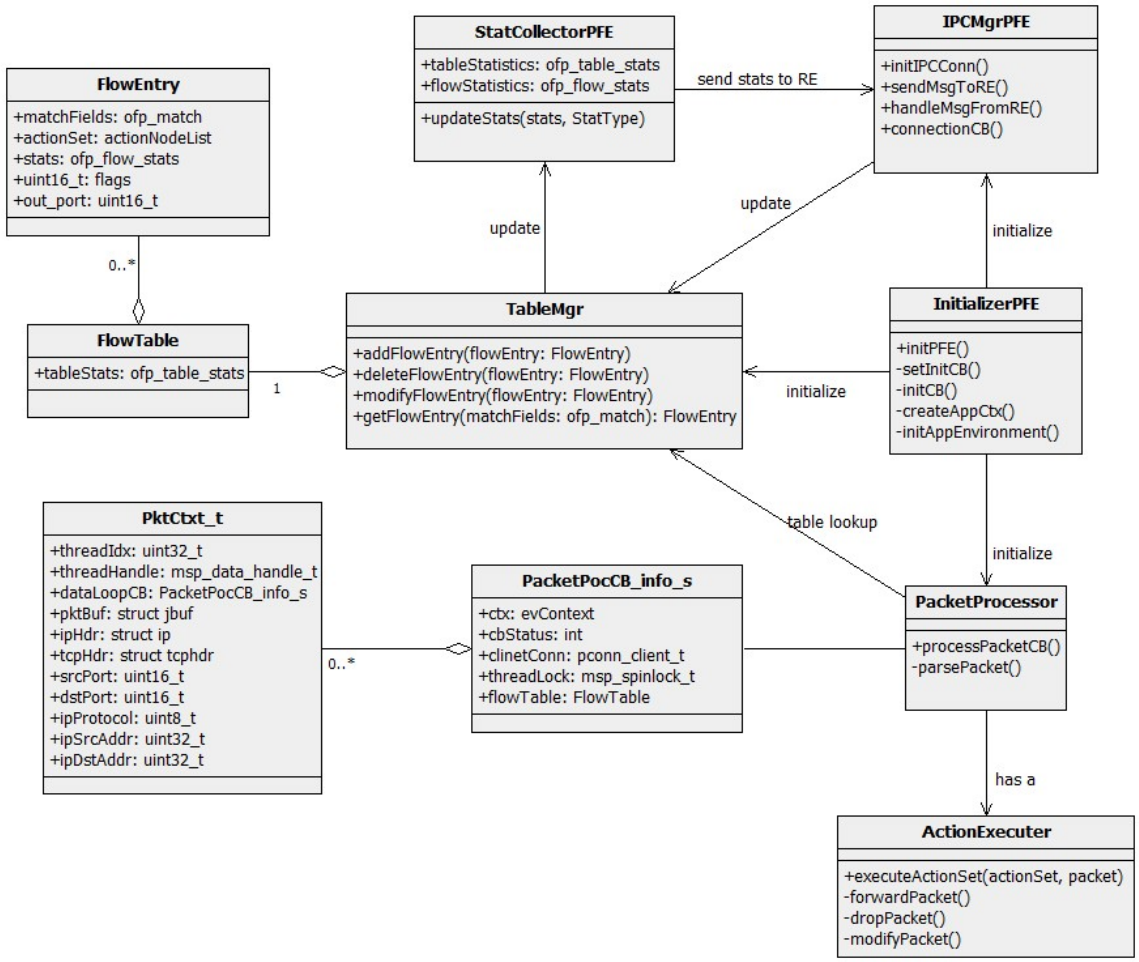


**Abbildung 4.6.:** Klassendiagramm–MS PIC Komponente

- InitializerPFE: Der *InitializerPFE* initialisiert die MS PIC–Komponente. Hierbei wird die Methode *initPFE()* aufgerufen. Diese wiederum ruft folgende Methoden in der angegebenen Reihenfolge auf:
  - *createAppCtx()*: Die Methode erstellt einen *Application–Context* für die Komponente. Dazu nutzt sie die Methode *msp_create_app_ctx()*, welche das Junos SDK zur Verfügung stellt.
  - *setInitCB()*: Die Methode registriert die Callback–Methode, welche aufgerufen wird, um applikationsspezifische Initialisierungen vorzunehmen. Dazu wird die

Methode *msp_set_app_cb_init()* aus dem Junos SDK verwendet. Als Parameter wird die Methode *initCB()* mitgegeben. *initCB()* wird während dem Startup der Komponente als erstes aufgerufen. Sie dient dazu, die einzelnen Module zu initialisieren. Es werden die Attribute des *ActionExecuter*, *TableMgr* sowie die Verbindung zur Interprozesskommunikation initialisiert. Weiterhin werden die Threads für die Paketverarbeitung erstellt. Hierzu wird die Methode *msp_data_create_loops()* aus dem Junos SDK verwendet. Als Parameter wird die Methode *processPacketCB()* des *PacketProcessor* angegeben.

  - *initAppEnvironment()*: Die Methode dient dazu, die Applikation zu initialisieren und zu starten. Sie nutzt die vom Junos SDK bereitgestellte Methode *msp_app_init()*.

- TableMgr: Der *TableMgr* ist für die Verwaltung der Flow–Tabelle zuständig. Die Flow–Tabelle (*FlowTable*) speichert Statistiken über die Tabelle und eine Liste von Flow–Einträgen (*FlowEntry*). Weiterhin bietet der *TableMgr* entsprechende Methoden zum Einfügen, Löschen, Modifizieren und Nachschlagen von Einträgen.

- PacketProcessor: *PacketProcessor* empfängt Pakete, die zur Verarbeitung an die MS PIC–Komponente geleitet werden. Hierfür wird die Methode *processPacketCB()* bereitgestellt. Weiterhin verfügt der *PacketProcessor* über die Methode *parsePacket()*. Sie dient dazu die benötigten Informationen aus einem Paket zu extrahieren.

- PacketPocCB_info_s: Die Verarbeitung von Paketen erfolgt durch mehrere parallel laufenden Threads. *PacketPocCB_info_s* ist die Datenstruktur, die alle Informationen beinhaltet, die von den Threads benötigt werden. Unter anderem wird der Status der Anwendung im Attribut *cbStatus* gespeichert. Sobald die Anwendung initialisiert ist und die IPC–Verbindung zu der RE–Komponente hergestellt ist, wird der Status auf *STATUS_UP* gesetzt. Tritt ein Fehler auf oder wird die Anwendung durch einen anderen Grund beendet, wechselt der Status auf *STATUS_SHUTDOWN*. Ein weiterer Status, in dem sich die Anwendung befinden kann, ist *STATUS_EMERGENCY_MODE*.

- ActionExecuter: Der *ActionExecutor* führt für ein Paket die im *actionSet* angegeben Aktionen durch. Hierfür wird die Methode *executeActionSet()* bereitgestellt. Als Parameter werden das *actionSet* sowie das Paket mitgegeben. Je nach Aktion werden die Methoden *forwardPacket()*, *dropPacket()* oder *modifyPacket()* aufgerufen.

- StatCollectorPFE: Der *StatCollectorPFE* verwaltet die Statistiken, die auf der MS PIC–Komponente erfasst werden. Diese werden in den Attributen *tableStatistics* und *flowStatistics* gespeichert und können über die Methode *updateStats()* aktualisiert werden.

- IPCMgrPFE: Der *IPCMgrPFE* kümmert sich um die Interprozesskommunikation auf Seiten der MultiServices PIC–Komponente. Über die Methode *initIPCConn()* wird die Verbindung zur RE–Komponente initialisiert. Dabei wird ein Client erstellt, um sich zum Server zu verbinden, der auf der RE–Komponente erstellt wurde. Hierfür wird die Methode *pconn_client_connect_async()* des Junos SDK verwendet. Weiterhin müssen Callback–Methoden registriert werden. Eine Callback–Methode kümmert sich um Ereignisse, die die Verbindung zu der RE–Komponente betreffen. Eine andere wird

zur Verarbeitung eingehender Nachrichten benötigt. Als Callback–Methode für die Verbindung zur RE–Komponente wird die Methode *connectionCB()* verwendet. Diese wird aufgerufen, sobald die Verbindung hergestellt wurde oder wenn die Verbindung abbricht. Für den Empfang von Nachrichten ist die Methode *handleMsgFromRE()* zuständig.

## 4.6. Interprozesskommunikation

Dieser Abschnitt stellt Implementierungsdetails der Interprozesskommunikation zwischen der RE– und der MS PIC–Komponente dar. Hierfür wird die Bibliothek *libconn* des Junos SDK verwendet. Die Rollenverteilung der Komponenten (Client/Server) ist in 3.9 beschrieben. Sobald die Verbindung zwischen den beiden Komponenten aufgebaut ist, können Nachrichten gesendet werden. Hierfür kann die Methode *pconn_server_send()* von Seiten der RE–Komponente und die Methode *pconn_client_send()* von Seiten der MS PIC–Komponente genutzt werden. Beide Methoden erwarten als Parameter die zu sendenden Daten sowie den Typ der Nachricht. Für die in 3.9 aufgelisteten Situationen, in denen eine Nachricht gesendet wird, werden Nachrichtentypen definiert, anhand derer die jeweilige Komponente die Nachricht entsprechend verarbeiten kann. Diese werden nachfolgend erläutert. Die Namensgebung ähnelt den Nachrichtentypen in der OpenFlow–Spezifikation, haben jedoch eine andere Bedeutung.

### 4.6.1. Nachrichten von der RE– an die MS PIC– Komponente

Im Folgenden werden die Nachrichten beschrieben, die von der Routing Engine–Komponente an die MultiServices PIC–Komponente gesendet werden.

- *MSG_FLOW_MOD*: Dieser Nachrichtentyp gibt an, dass eine Modifikation an der Flow–Tabelle vorgenommen werden soll. Die Details sind in den mitgelieferten Daten enthalten.
- *MSG_SEND_PACKET_OUT*: Dieser Nachrichtentyp macht kenntlich, dass die mitgelieferten Daten ein Paket beinhalten, dass über einen bestimmten Port versendet werden soll.
- *MSG_ENTER_EMERG_MODE*: Dieser Nachrichtentyp weist die MS PIC–Komponente an, in den *emergency mode* zu wechseln.

### 4.6.2. Nachrichten von der MS PIC– an die RE–Komponente

Im Folgenden werden die Nachrichten beschrieben, die von der MultiServices PIC–Komponente an die Routing Engine–Komponente gesendet werden.

Treten Fehler beim Hinzufügen eines neuen Eintrags in die Flow–Tabelle auf, kann die MS PIC–Komponente folgende Nachrichten an die RE–Komponente senden:

- *MSG_ERROR_FLOW_MOD_INVALID_PORT*: Dieser Nachrichtentyp gibt an, dass die *checkValidity()*–Methode fehlgeschlagen ist. Die RE–Komponente erstellt über den *OFMsgCreator* eine Fehlernachricht, bei der als Fehlertyp *OFPET_BAD_ACTION* und als Fehlercode *OFPBAC_BAD_OUT_PORT* angegeben werden und sendet sie an den Controller.
- *MSG_ERROR_FLOW_MOD_OVERLAP*: Dieser Nachrichtentyp gibt an, dass die *checkIsFlowOverlapping()*–Methode fehlgeschlagen ist. Es wird eine Fehlernachricht an die RE–Komponente gesendet. Diese erstellt über den *OFMsgCreator* eine Fehlernachricht, bei der als Fehlertyp *OFPET_FLOW_MOD_FAILED* und als Fehlercode *OFPFMFC_OVERLAP* angegeben werden und sendet sie an den Controller.

Nachfolgende Nachrichten werden an die RE–Komponente gesendet, falls ein Eintrag aus der Flow–Tabelle entfernt wurde, für den festgelegt wurde, den Controller darüber zu informieren.

- *MSG_FLOW_ENTRY_REMOVED_IDLE_TO*: Dieser Nachrichtentyp gibt an, dass der Eintrag aufgrund eines *Idle–Timeout* aus der Tabelle entfernt wurde. Die RE–Komponente erstellt über den *OFMsgCreator* eine Flow–Removed Nachricht, bei der als Grund *OFPRR_IDLE_TIMEOUT* angegeben wird und sendet sie an den Controller.
- *MSG_FLOW_ENTRY_REMOVED_HARD_TO*: Dieser Nachrichtentyp gibt an, dass der Eintrag aufgrund eines *Hard–Timeout* aus der Tabelle entfernt wurde. Die RE–Komponente erstellt über den *OFMsgCreator* eine Flow–Removed Nachricht, bei der als Grund *OFPRR_HARD_TIMEOUT* angegeben wird und sendet sie an den Controller.
- *MSG_FLOW_ENTRY_REMOVED_DELETE*: Dieser Nachrichtentyp gibt an, dass der Eintrag auf Anfrage des Controllers aus der Tabelle entfernt wurde. Die RE–Komponente erstellt über den *OFMsgCreator* eine Flow–Removed Nachricht, bei der als Grund *OFPRR_DELETE* angegeben wird und sendet sie an den Controller.

Weitere Nachrichten, die an die RE–Komponente gesendet werden, sind:

- *MSG_HELLO*: Eine *Hello*–Nachricht dient zur Synchronisation der beiden Komponenten. Sobald die IPC–Verbindung zu der RE–Komponente erfolgreich aufgebaut wurde, sendet die MS PIC–Komponente eine *MSG_HELLO*–Nachricht.
- *MSG_ACTION_SEND_PACK_TO_CTR*: Diese Art von Nachricht wird an die RE–Komponente gesendet, wenn eine im Flow–Eintrag angegebene Aktion besagt, dass ein passendes Paket an den Controller gesendet werden soll. Das entsprechende Paket wird in die Nachricht eingebunden und wird von der RE–Komponente an den Controller weitergeleitet.

- *MSG_UPDATE_STATS*: Die MS PIC–Komponente sendet in periodischen Zeitabständen ihre gesammelten Statistiken an die RE–Komponente, welche die Verwaltung aller Statistiken übernimmt. Mit dieser Art von Nachricht wird angegeben, dass ein Update von Statistiken gesendet wurde.
- *MSG_NO_ENTRY_FOR_PACKET_FOUND*: Diese Art von Nachricht gibt an, dass für ein Paket kein Eintrag in der Flow–Tabelle existiert. Das Paket ist in der Nachricht eingebunden und wird an den Controller weitergeleitet.

# 5. Test

## 5.1. Überblick

In diesem Kapitel wird der Test des Prototypen dargestellt. Es wird zunächst die für die Durchführung von Tests eingerichtet Testumgebung beschrieben. Anschließend wird auf die durchgeführten Tests eingegangen.
Aufgrund des Cross–Compiling der Anwendung gestaltet sich die Durchführung von Tests als schwierig, so dass sich nur funktionale Tests durchzuführen lassen. Die Überprüfung, ob ein Funktionstest positiv oder negativ verlaufen ist, erfolgt über eine *Log–Datei*. Für alle wichtigen Schritte wird geloggt, ob die Ausführung erfolgreich war. Weiterhin wurden spezielle Debug–Methoden geschrieben, die bestimmte Variablenwerte oder Listeninhalte in die Log–Datei schreiben. Eine solche Debug–Methode wurde beispielsweise für das Auslesen der Konfiguration geschrieben. Alle ausgelesenen Dateien werden in eine Liste gespeichert, deren Inhalt in die Log–Datei geschrieben wird.
Eine weitere Schwierigkeit ist die Abhängigkeit von dem externen Controller. Zum Testen des Prototyps wurde der in Kapitel 2, Abschnitt 2.1.3 beschriebene *Beacon*–Controller verwendet. Dieser befindet sich selbst noch in der Entwicklung, so dass Fehler sowohl von Seiten des Prototyps, als auch von Seiten des Controllers auftreten können. Eine Eigenart des *Beacon*–Controllers ist, dass er sofort nach erfolgreicher Durchführung des *Handshake* (siehe 2.1.2.2) eine *OFPT_FLOW_MOD*–Nachricht sendet, welche besagt, dass die gesamte Flow–Tabelle geleert werden soll. Bei der Durchführung der Tests gilt es, dies zu beachten.
Zur zusätzlichen Überprüfung der Testergebnisse wurde Wireshark [wirb] verwendet. Hierbei handelt es sich um ein Werkzeug, mit dem es möglich ist, den Datenverkehr einer Netzwerk–Schnittstelle aufzuzeichnen und in einer Form darzustellen, die für den Menschen lesbar ist. Für Wireshark ist ein Plugin verfügbar [wira], mit welchem OpenFlow–Nachrichten analysiert werden können, welche zwischen Controller und Router ausgetauscht werden.

## 5.2. Testumgebung

Die Testumgebung ist in Abbildung 5.1 dargestellt. Die Testumgebung beseht aus dem Router, auf dem der OpenFlow–Prototyp läuft, und drei Rechnern, die über den Router verbunden sind. Auf einem Rechner ist der *Beacon*–Controller installiert, der mit dem OpenFlow–Prototyp kommuniziert. Die andern beiden Rechner, *PC1* und *PC2*, tauschen Nachrichten aus. Diese sollen über den OpenFlow–Prototypen geleitet werden.

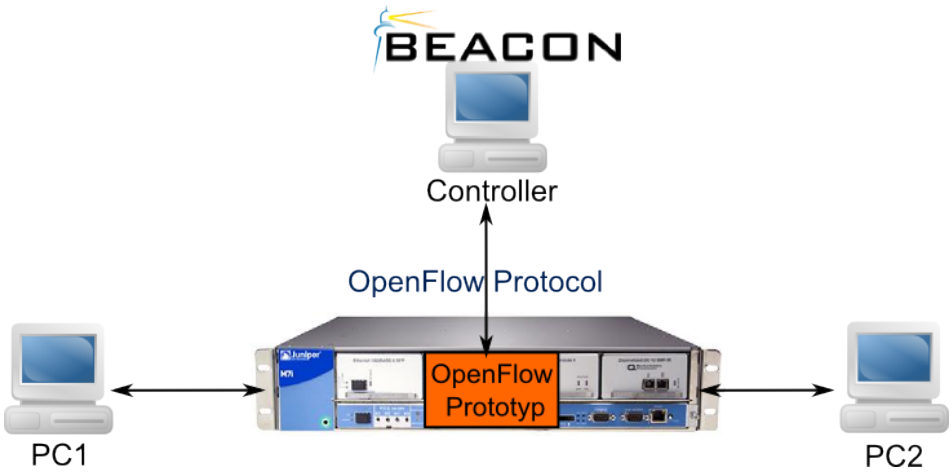


**Abbildung 5.1.:** Testumgebung

## 5.3. Testfälle

Dieser Abschnitt beschreibt die Testfälle sowie das Ergebnis der Testdurchführung. Einige der Funktionalitäten können erst getestet werden, wenn die Funktionalität anderer Module sichergestellt ist. So kann beispielsweise der *Handshake* erst getestet werden, nachdem die Verbindung zu einem externen Controller hergestellt ist. Es wird für alle Testfälle davon ausgegangen, dass eine Verbindung zu einem externen Controller hergestellt ist. Dies wird nicht explizit als Voraussetzung in der Beschreibung jedes Testfalls vermerkt.

### 5.3.1. Auslesen der Konfiguration

Das Auslesen der Konfiguration des Routers wird vom *ConfigReader* durchgeführt. Tabelle 5.1 zeigt den Testfall, um die Funktionaliät zu prüfen.

**Tabelle 5.1.:** Testfall – Auslesen der Konfiguration

| **Testfall** | Auslesen der Konfiguration |
|---|---|
| **Vorbedingung** | Die Konfiguration enthält keine Einträge, die zum OpenFlow–Prototyp gehören. |
| **Ablauf** | 1. Benutzer gibt folgende Daten ein:<br>• Controller Informationen: IP Adresse und Port<br>• Port Informationen: Name, Nummer und HW–Adresse zweier Ports<br>Beispieleingaben können dem Listing A.24 entnommen werden.<br>2. Benutzer bestätigt seine Eingaben durch *Commit*–Befehl. |
| **Soll–Resultat** | Daten sind in der Konfiguration eingetragen. Die Eingaben des Benutzers wurden in die Log–Datei geschrieben. |
| **Ist–Resultat** | Stimmt mit dem Soll–Resultat überein. |

### 5.3.2. Paketverarbeitung

Die Paketverarbeitung wird durch die Module *PacketProcessor* und *ActionExecutor* realisiert. Die Methode *parsePacket()* des *PacketProcessor* extrahiert die benötigten Informationen aus einem Paket. Diese Funktionalität soll nachfolgend mit dem in Tabelle 5.2 beschriebenen Testfall überprüft werden. Weiterhin wird in den Testfällen 5.4 und 5.5 die Weiterleitung von Paketen durch den *ActionExecutor* getestet. Das Verwerfen von Paketen wird im Testfall 5.6 geprüft.

#### 5.3.2.1. Paketverarbeitung – Parsen eines Pakets

**Tabelle 5.2.:** Testfall – Parsen eines Pakets

| **Testfall** | Parsen eines Pakets |
|---|---|
| **Vorbedingung** | Der Prototyp ist auf dem Router installiert und gestartet. Die Konfiguration des Routers enthält definierte Interface Service Sets, die den Schnittstellen zugeordnet sind, an denen *PC1* und *PC2* angeschlossen sind. *PC1* und *PC2* sind in der Lage sich gegenseitig einen Ping zu senden. |
| **Ablauf** | Es wird ein Ping von *PC1* nach *PC2* durchgeführt. |

| | |
|---|---|
| **Soll–Resultat** | Es wurden folgende Informationen in die Log–Datei geschrieben:<br>• Ziel IP Adresse: IP Adresse von *PC2*<br>• Quell IP Adresse: IP Adresse von *PC1*<br>• Eingangsport des Pakets<br>• IP Protokoll : 1 (ICMP) |
| **Ist–Resultat** | Stimmt mit dem Soll–Resultat überein. |

#### 5.3.2.2. Paketverarbeitung – Erstellen einer *Packet–In*–Nachricht

Dieser Testfall testet das Verhalten, wenn für ein Paket kein passender Eintrag in der Flow–Tabelle gefunden wird. Tabelle 5.3 gibt eine Beschreibung des Testfalls.

**Tabelle 5.3.:** Testfall – Erstellen einer Packet-In Nachricht

| | |
|---|---|
| **Testfall** | Erstellen einer Packet-In Nachricht |
| **Vorbedingung** | Der Prototyp ist auf dem Router installiert und gestartet. Die Flow–Tabelle ist leer. |
| **Ablauf** | Es wird ein Ping von *PC1* nach *PC2* durchgeführt. Es wird eine *Packet-In*–Nachricht erstellt und an den Controller gesendet. |
| **Soll–Resultat** | Die Log–Datei enthält folgende Einträge:<br>• Nachschlagen in der Flow–Tabelle: Es wurde kein passender Eintrag gefunden.<br>• *Packet-In*–Nachricht enthält folgende Daten:<br>– reason: OFPR_NO_MATCH<br>– Eingangsport<br>– Inhalt des Pakets<br>– Buffer–ID: -1 |
| **Ist–Resultat** | Stimmt mit dem Soll–Resultat überein. |
| **Wireshark** | Zeigt dem Inhalt der Packet-In Nachricht an, der mit den angegebenen Daten übereinstimmt. |

#### 5.3.2.3. Paketverarbeitung – Paketweiterleitung anhand der Flow–Tabelle

**Tabelle 5.4.:** Testfall – Weiterleitung eines Pakets anhand der Flow–Tabelle

| **Testfall** | Weiterleitung eines Pakets anhand der Flow–Tabelle |
|---|---|
| **Vorbedingung** | Der Prototyp ist auf dem Router installiert und gestartet. Die Konfiguration des Routers enthält definierte Interface Service Sets, die den Schnittstellen zugeordnet sind, an denen *PC1* und *PC2* angeschlossen sind. *PC1* und *PC2* sind in der Lage sich gegenseitig einen Ping zu senden. In der Flow–Tabelle befindet sich weiterhin ein Eintrag, der für Pakete passt, die von *PC1* gesendet werden. Als Aktion ist für den Eintrag angegeben, das Pakete an *PC2* weiter zu leiten sind. |
| **Ablauf** | Es wird ein Ping von *PC1* nach *PC2* durchgeführt. |
| **Soll–Resultat** | Das Paket wurde erfolgreich geparst und es wurde hierfür ein passender Eintrag in der Flow–Tabelle gefunden. Das Paket wurde über den *ActionExecutor* weitergeleitet. *PC2* hat das Paket empfangen. |
| **Ist–Resultat** | Stimmt mit dem Soll–Resultat überein. |
| **Wireshark** | Zeigt den Empfang des Pakets am *PC2* an. |

#### 5.3.2.4. Paketverarbeitung – Paketweiterleitung aufgrund einer *Packet–Out*–Nachricht

**Tabelle 5.5.:** Testfall – Weiterleitung eines Pakets aufgrund einer *Packet–Out*–Nachricht des Controllers

| **Testfall** | Weiterleitung eines Pakets anhand der Flow–Tabelle |
|---|---|
| **Vorbedingung** | Der Prototyp ist auf dem Router installiert und gestartet. Die Konfiguration des Routers enthält definierte Interface Service Sets, die den Schnittstellen zugeordnet sind, an denen *PC1* und *PC2* angeschlossen sind. *PC1* und *PC2* sind in der Lage sich gegenseitig einen Ping zu senden. In der Flow–Tabelle befinden sich keine. |
| **Ablauf** | Es wird ein Ping von *PC1* nach *PC2* durchgeführt. Da keine Einträge in der Tabelle vorhanden sind, wird das Paket an den Controller gesendet. Der Controller sendet daraufhin eine *Packet–Out*–Nachricht, die angibt das Paket weiterzuleiten. |
| **Soll–Resultat** | Das Paket wurde über den *ActionExecutor* weitergeleitet. |
| **Ist–Resultat** | Das Paket wurde nicht weitergeleitet. In der Methode zum Weiterleiten des Pakets ist ein Fehler aufgetreten. |

#### 5.3.2.5. Paketverarbeitung – Verwerfen eines Pakets

**Tabelle 5.6.:** Testfall – Verwerfen eines Pakets

| **Testfall** | Verwerfen eines Pakets |
|---|---|
| **Vorbedingung** | Der Prototyp ist auf dem Router installiert und gestartet. Die Konfiguration des Routers enthält definierte Interface Service Sets, die den Schnittstellen zugeordnet sind, an denen *PC1* und *PC2* angeschlossen sind. *PC1* und *PC2* sind in der Lage sich gegenseitig einen Ping zu senden. In der Flow–Tabelle befindet sich weiterhin ein Eintrag, der für Pakete passt, die von *PC1* gesendet werden. Es sind keine Aktionen für den Eintrag definiert (impliziert das Verwerfen des Pakets). |
| **Ablauf** | Es wird ein Ping von *PC1* nach *PC2* durchgeführt. |
| **Soll–Resultat** | Das Paket wurde erfolgreich geparst und es wurde hierfür ein passender Eintrag in der Flow–Tabelle gefunden. Das Paket wurde über den *ActionExecutor* verworfen und in der Log–Datei dokumentiert. *PC2* hat das Paket nicht empfangen. |
| **Ist–Resultat** | Stimmt mit dem Soll–Resultat überein. |

### 5.3.3. Handshake

Beim *Handshake* (2.1.2.2) sind die Module *CtrCommMgr*, *OFMsgCreator* und *FeatureMgr* beteiligt. Der *CtrCommMgr* empfängt Nachrichten vom Controller und sendet vom *OFMsgCreator* erstellte Nachrichten zurück an den Controller. Der *OFMsgCreator* holt sich im Falle einer *OFPT_FEATURE_REPLY*–Nachricht die benötigten Informationen vom *FeatureMgr*. In Tabelle 5.7 ist der Testfall dargestellt, welche dieses Zusammenspiel der Module überprüft.

**Tabelle 5.7.:** Testfall – Handshake

| **Testfall** | Durchführung des *Handshake* |
|---|---|
| **Vorbedingung** | Der Prototyp ist auf dem Router installiert und gestartet. Die Konfiguration des Routers enthält Informationen über die OpenFlow–Ports des Routers. Diese wurde vom *ConfigReader* ausgelesen und die Daten an den *FeatureMgr* übergeben. |
| **Ablauf** | Nach erfolgreicher Herstellung der TCP–Verbindung zum Controller wird eine *OFPT_HELLO*–Nachricht vom Router an den Controller gesendet. |

| | |
|---|---|
| **Soll–Resultat** | Es wurden folgende Nachrichten in der angegebenen Reihenfolge ausgetauscht:<br>• Controller sendet eine *OFPT_HELLO*–Nachricht.<br>• Controller sendet eine *OFPT_FEATURE_REQUEST*–Nachricht<br>• Router sendet eine *OFPT_FEATURE_REPLY*–Nachricht, die Informationen über die Ports, Capabilities, unterstützen Aktionen und Anzahl der Flow–Tabellen beinhaltet.<br>Beim Empfang der *OFPT_HELLO*–Nachricht vom Controller wird im *CtrCommMgr* überprüft, ob die OpenFlow–Versionen übereinstimmen. Diese stimmen überein.<br>Empfang und Senden von Nachrichten, samt deren Inhalt, wurde in der Log–Datei festgehalten. |
| **Ist–Resultat** | Stimmt mit dem Soll–Resultat überein. |
| **Wireshark** | Zeigt Reihenfolge und Inhalt der Nachrichten an. Diese stimmen mit den Angaben im Soll–Resultat überein. |

### 5.3.4. Verwaltung der Flow–Tabelle

Die folgenden Testfälle prüfen die Verwaltung der Flow–Tabelle. Um das Hinzufügen und Löschen eines Eintrags in die Flow–Tabelle zu testen, wurde eine Testmethode implementiert, die einen Dummy–Eintrag erstellt. Mittels diesen Dummy–Eintrags soll die Funktionalität des *TableMgr* getestet werden. Der Flow–Eintrag hat als Vergleichskriterium die IP Adresse von *PC1*, d.h. der Eintrag passt zu allen Paketen, die an *PC1* gesendet werden. Das Hinzufügen eines Eintrags ist in Tabelle 5.8 beschrieben, das Löschen in Tabelle 5.9. In beiden Testfällen wird gleichzeitig die Funktionalität zum Nachschlagen von Einträgen getestet.

#### 5.3.4.1. TableMgr – Hinzufügen eines Eintrags

**Tabelle 5.8.:** Testfall – Hinzufügen eines Eintrags in die Flow–Tabelle

| | |
|---|---|
| **Testfall** | Hinzufügen eines Eintrags in die Flow–Tabelle |
| **Vorbedingung** | Der Prototyp ist auf dem Router installiert und gestartet. Die Testmethode wurde (temporär) in den Code des Prototypen eingebaut, so dass nachdem der *Handshake* durchgeführt worden ist, die Methode zum Hinzufügen eines Eintrags (*addFlowEntry()*) des *TableMgr* mit dem Dummy–Eintrag als Parameter aufgerufen wird. |
| **Ablauf** | Es wird ein Ping von *PC1* nach *PC2* durchgeführt. |

| | |
|---|---|
| **Soll–Resultat** | Die Log–Datei enthält folgende Einträge:<br>• Den Ablauf beim Einfügen des Eintrags in die Flow–Tabelle:<br>– Tabelle war anfangs leer<br>– Eintrag wurde als neuer Kopf der Liste eingetragen (Die Flow–Tabelle ist als einfach verkettete Liste implementiert).<br>• Nachschlagen in der Flow–Tabelle: Es wurde ein Eintrag gefunden, der zum Paket mit der Ziel–IP Adresse des *PC1* passt. |
| **Ist–Resultat** | Stimmt mit dem Soll–Resultat überein. |

#### 5.3.4.2. TableMgr – Löschen eines Eintrags

**Tabelle 5.9.:** Testfall – Löschen eines Eintrags aus der Flow–Tabelle

| | |
|---|---|
| **Testfall** | Löschen eines Eintrags aus der Flow–Tabelle |
| **Vorbedingung** | Der Prototyp ist auf dem Router installiert und gestartet. Die Testmethode wurde in die Initialisierungsmethode des *InitializerPFE* eingebaut, so dass der Dummy–Eintrag in der Tabelle eingetragen ist, bevor der *Handshake* durchgeführt wird. |
| **Ablauf** | Nachdem die Anwendung gestartet ist, wird der *Handshake* mit dem Controller durchgeführt. Dieser sendet nach der Durchführung eine Nachricht, die besagt, dass alle Einträge aus der Tabelle entfernt werden sollen. Anschließend wird ein Ping von *PC1* nach *PC2* durchgeführt. |
| **Soll-Resultat** | Die Log–Datei enthält folgende Einträge:<br>• Den Ablauf beim Entfernen des Eintrags aus der Flow–Tabelle:<br>– Tabelle war anfangs nicht leer<br>– Eintrag wurde gefunden und entfernt<br>• Nachschlagen in der Flow–Tabelle, nachdem der Ping durchgeführt wurde: Es wurde kein passender Eintrag gefunden. |
| **Ist–Resultat** | Stimmt mit dem Soll–Resultat überein. |

## 5.4. Diskussion der Ergebnisse

Das wesentliche Ziel bei der Entwicklung des OpenFlow–Prototypen war, die Hauptfunktionalität von OpenFlow zu implementieren und anschließend zu testen. Es wurden hierbei die Kommunikation mit einem externen Controller, die Verwaltung der Flow–Tabelle sowie die Paketverarbeitung getestet.
Der Aufbau der Verbindung sowie die Durchführung des *Handshake* wurden anhand von Tests der Kommunikation mit einem externen Controller erfolgreich überprüft. Diese beiden Funktionalitäten sind die Voraussetzung für die weitere Kommunikation mit dem Controller, welche ihrerseits bei der Verwaltung der Flow–Tabelle eine wesentliche Rolle spielt. Das Entfernen von Einträgen wurde mittels entsprechender Anfragen von Seiten des Controllers überprüft. Nachdem das Einfügen von Einträgen in die Flow–Tabelle mittels eines Dummy–Eintrags erfolgreich getestet worden ist, wurde auf Anfrage des Controllers das Entfernen dieses Dummy–Eintrags fehlerfrei druchgeführt. Das Nachschlagen von Einträgen in der Tabelle wurde im Zusammenhang mit der Paketverarbeitung geprüft. Zunächst wurde sichergestellt, dass das Parsen der Pakete keine Fehler aufzeigt. Alle benötigten Informationen wurden erfolgreich aus dem Paket extrahiert, anhand dieser Informationen konnte in der Tabelle nach einem passenden Eintrag gesucht werden. Auch das Nachschlagen in der Tabelle erwies sich als fehlerfrei.
Der einzige aufgetretene Fehler ergab sich bei der Verarbeitung der *Packet–Out*–Nachricht des Controllers. Dabei konnte das Paket nicht weitergeleitet werden. Bei der Methode, die das Paket weiterleiten sollte, handelt es sich um eine vom Junos SDK bereitgestellte Methode. In der Dokumentation des Junos SDK ist diese nur sehr knapp beschrieben, so dass keine Aussage darüber gemacht werden kann, warum der Fehler aufgetreten ist. Eine Möglichkeit diesen Fehler zu umgehen besteht darin, für den Fall, dass für ein Paket kein Eintag in der Flow–Tabelle gefunden wird, nicht das gesamte Paket an den Controller zu senden, sondern dieses zwischenzuspeichern und anhand der Antwort des Controllers weiterzuleiten.

# 6. Zusammenfassung und Ausblick

## 6.1. Zusammenfassung

Bei *OpenFlow* handelt es sich um einen Standard, der als Feature zu einem Switch oder Router hinzugefügt werden kann. Dieser ermöglicht die Ansteuerung von Switchen oder Routern. Über das OpenFlow–Protokoll lassen sich die Forwardingtabellen über das Netzwerk bearbeiten. Somit wird es ermöglicht, experimentelle Netzwerkprotokolle für Forschungszwecke zu testen, ohne dass Hersteller die interne Funktionsweise ihrer Geräte preisgeben müssen. Eine Reihe namenhafter Hersteller, wie Juniper Networks [JNf], Hewlett–Packard [hp] oder Cisco [cisa] haben mit der Integration von OpenFlow in ihre Produkte begonnen, ohne dass bisher jedoch öffentlich zugängliche Arbeiten verfügbar sind.
Die vorliegende Arbeit beschäftigt sich mit der Frage, ob und wie sich OpenFlow auf einem Juniper Networks Router mit Hilfe des Junos SDKs realisieren lässt. Die Aufgabenstellung umfasst in einem ersten Schritt die Einarbeitung in die Architektur des Routers sowie in das Junos SDK, um mögliche Abbildungen von OpenFlow auf den Juniper Router herauszuarbeiten. Darauf aufbauend wird eine Architektur entwickelt, mit der OpenFlow auf einen Juniper Router umgesetzt werden kann. Die Architektur sieht hierbei für die Realisierung von OpenFlow zwei Komponenten vor, welche auf verschiedenen Schichten der Routerarchitektur ausgeführt werden. Die Hauptaufgabe der einen Komponente ist die Kommunikation mit einem externen Controller, während die andere Komponente die Paketverarbeitung übernimmt. Anhand dieser Architektur erfolgt eine prototypische Implementierung mit anschließend durchgeführten Tests. Im einzelnen wurden hierbei Tests zur Überprüfung der Hauptfunktionalitäten Paketverarbeitung, Verwaltung der Flow–Tabelle sowie Kommunikation mit einem externen Controller durchgeführt. Die Tests verliefen mit einer Ausnahme fehlerfrei. Ein Fehler trat bei der Weiterleitung von Paketen auf, die in *Packet–Out*–Nachrichten des Controllers eingebunden waren.

## 6.2. Ausblick

Die in dieser Diplomarbeit entwickelte Architektur sowie der implementierte Prototyp lassen sich in mehrerer Hinsicht erweitern. Zunächst wurden bei der Entwicklung der Architektur nicht alle Bestandteile der OpenFlow–Spezifikation berücksichtigt. So wurden etwa einige der als *optional* gekennzeichneten Bestandteile im Rahmen dieser Arbeit außen vorgelassen. Dazu gehört beispielsweise die Aktion *Enqueue* und die damit verbundene Verwendung von Queues an bestimmten Ports. Darüber hinaus wurde mittlerweile die OpenFlow Spezifikation 1.1 [opec] mit weiteren neu hinzugekommenen Funktionalitäten veröffentlicht. Da

die aktuelle Arbeit auf die OpenFlow Spezifikation 1.0 basiert, bleibt es künftigen Arbeiten vorbehalten, neben den optionalen Bestandteilen auch die neuen Funktionalitäten zu integrieren.
Eine dringende Erweiterung ist im Bereich der Paketverarbeitung notwendig. Der bei der Verarbeitung von *Packet–Out*–Nachrichten von Controller aufgetretene Fehler soll hierbei beseitigt werden. Dies kann durch eine alternative Möglichkeit zum Senden von *Packet–In*–Nachrichten an den Controller erreicht werden. Diese ist in 3.5.2 beschrieben und sieht vor, Pakete zwischenzuspeichern, anstatt komplett an den Controller zu senden.
Ein weiterer Ansatzpunkt für künftige Erweiterungen stellt die Performanz dar, welche für die Implementierung des Prototypen nicht im Fokus stand. Daher wurde in dieser Arbeit auf die Durchführung vom Messungen der Performanz verzichtet. Für künftige Arbeiten könnte sich dies jedoch als ein zentraler Punkt erweisen. So sind beispielsweise im Bereich des Tablemanagements Verbesserungen hinsichtlich der Performanz durchaus denkbar.
Eine weitere interessante Fragestellung ergibt sich daraus, dass die Architektur so konzipiert ist, dass sie auf verschiedenen Juniper–Plattformen anwendbar ist. Der Prototyp kann daher auch auf einem Router der Juniper MX–Serie [JNe] ausgeführt werden, welche ein Level 2 Routing ermöglicht. Dies war auf dem M7i Router, auf dem der Prototyp getestet wurde, nicht möglich, daher konnte die Level 2–Funktionalität bisher noch nicht getestet werden.
Im Hinblick auf die Interoperabilität des OpenFlow–Prototypen können weitere Tests mit unterschiedlichen OpenFlow–Controllern, wie in Kapitel 2 vorgestellt, durchgeführt werden.
Die weitere Entwicklung kann durch Erweiterungen im Bereich des Commmand–Line Interface (CLI) erleichtert werden. Beispielsweise bietet es sich für die Bereiche Tabellenmanipulation und Monitoring der Anwendung an, Befehlserweiterungen vorzunehmen. Besonders im Hinblick auf die Durchführung von Tests, verringert das Hinzufügen und Entfernen von Flow–Einträgen der Flow–Tabelle über das CLI den Konfigurationsaufwand. Die Ausgabe von Informationen, beispielsweise über den Inhalt der Flow–Tabelle oder über bestimmte Statistiken, kann zum Monitoring der Anwendung genutzt werden.

# A. Anhang

Im Folgenden werden die einzelnen Vorgänge erläutert, um den OpenFlow–Prototyp zu kompilieren und auf dem Router zu installieren. Weiterhin wird beschrieben, welche Konfigurationen für die Ausführung der Anwendung vorgenommen werden müssen.
Ausführliche Informationen zur Konfiguration des Routers können dem *JUNOS Cookbook* [Garo6] entnommen werden.

## A.1. Kompilierung und Installation der Anwendung

Um selbstgeschriebene Anwendungen auf dem Router zu installieren, müssen zunächst der Code kompiliert und installierbare Pakete erstellt werden. Hierfür muss die Virtual Build Environment eingerichtet sein. Eine ausführliche Anleitung kann der Installationsbeschreibung [Net10] entnommen werden. Die Virtual Build Environment läuft auf einem Entwicklungs–PC in einer Virtuellen Maschine. Über den Entwicklungs–PC kann über eine Telnet–Verbindung auf den Router zugegriffen werden.

Nachdem die Virtual Build Environment eingerichtet wurde, kann der Code der Anwendung in das *src*–Verzeichnis der *Sanbox*[1] [Net10] kopiert werden. Der OpenFlow–Prototyp besteht aus zwei Komponenten, für die jeweils ein Unterordner angelegt wird, in den der Code der entsprechenden Komponente gespeichert wird. Der Code der MS PIC–Komponente befindet sich im Ordner *ikr-openFlow-data*, der Code der RE–Komponente befindet sich im Unterordner *ikr-openFlow-mgmt*. Um dem Code zu kompilieren und installierbare Pakete zu erstellen, muss in das *src*–Verzeichnis der *Sandbox* gewechselt werden und folgender Befehl ausgeführt werden:

```
mk release
```

**Listing A.1:** Befehl: Kompilieren und Paketerstellung

Die installierbaren Pakete werden in das *ship*–Verzeichnis generiert. Um diese zu installieren, müssen sie zunächst auf den Router kopiert werden. Dies kann nur ein Benutzer mit *root*–Rechten durchführen. Hierfür müssen folgende Schritte ausgeführt werden.

1. Installierbare Pakete auf den Entwicklungs–PC kopieren

[1] Der Code des OpenFlow–Prototypen befindet sich in der Sandbox *OPENFLOW_newCerts*

2. SSH–Verbindung zu router@labserv1 herstellen
3. Login als *root*
4. In das Verzeichnis */var/home/netlab* wechseln
5. FTP–Verbindung zum Entwicklungs–PC herstellen und mittles *get*–Befehl die installierbaren Pakete auf den Router übertragen
6. FTP–Verbindung trennen
7. SSH–Verbindung trennen durch Tastenkombination *Steuerung* + *AltGR* + *backslash*, *q*

Für alle weiteren Schritte sind keine *root*–Rechte notwendig. Um die Anwendung zu installieren, muss eine Telnet–Verbindung zum Router hergestellt werden und folgender Befehl im Betriebsmodus ausgeführt werden:

```
> request system software add <Name der Anwendung>
```

**Listing A.2:** Befehl: Anwendung installieren

Falls eine Anwendung aus mehreren Komponenten besteht, kann entweder ein *Bundle* erstellt werden, welches alle Komponenten beinhaltet. In diesem Fall muss nur das *Bundle* installiert werden. Ansonsten muss jede Komponente einzeln über den in Listing A.2 dargestellten Befehl installiert werden.
Für den OpenFlow–Prototyp wurde ein *Bundle* namens *ikr–openFlow–bundle–10.2I20110617_1439_root.tgz* erstellt, das beide Komponenten beinhaltet. Der Name setzt sich zusammen aus dem Namen der Anwendung [2], der Version des Junos SDK (10.2), dem Datum (17.06.2011, 14:39 Uhr) und dem Benutzer (root), der die Pakete generiert hat.

## A.2. Ausführung der Anwendung

Um die installierte Anwendung auszuführen, muss der Router konfiguriert werden. In den folgenden Unterabschnitten wird die Konfiguration des Routers beschrieben.
Der Router wird über das Command–Line Interface konfiguriert. Hierfür muss in den Konfigurationsmodus gewechselt werden. Dies kann über den in Listing A.3 aufgeführten Befehl erreicht werden. Das Zeichen „>“ in der Kommandozeile bedeutet, dass man sich im Betriebsmodus befindet. Im Konfigurationsmodus steht das Zeichen „#“ am Anfang der Zeile.

```
> configure
```

**Listing A.3:** Befehl: In den Konfigurationsmodus wechseln

[2] Name der Anwendung ist *ikr–openFlow*. *Bundle* bedeutet, dass in diesem Paket sowohl die RE–Komponente, als auch die MS PIC Komponente vorhanden sind. So müssen nicht beide Komponenten einzeln installiert werden.

Damit die modifizierte Konfiguration aktiviert wird, müssen die Eingaben mit einem *Commit*–Befehl bestätigt werden. Um sicherzugehen, dass die Eingaben korrekt sind, kann der Befehl *commit check* verwendet werden. Falls dieser Befehl Fehler anzeigt, kann die Eingabe korrigiert werden. *Commit*–Befehle werden in den Listings nicht angegeben.

### A.2.1. Grundkonfigurationen

Nachdem in den Konfigurationsmodus gewechselt wurde, müssen die in Listing A.4 aufgeührten Konfigurationen angegeben werden. Das Listing beinhaltet die System–Konfigurationen sowie die Konfiguration der Interfaces.

```
system {
    host-name labrt7;
    domain-name netlab.ikr.uni-stuttgart.de;
    root-authentication {
        encrypted-password "$1$p15w0lhZ$v11JtxdQ0ElR.3QOVM7VH1"; ## SECRET-DATA
    }
    name-server {
        10.31.1.254;
    }
    login {
        user netlab {
            uid 2000;
            class super-user;
            authentication {
                encrypted-password "$1$7654Ro28$8a5HFvOQU84eFyIrvFxt80"; ## SECRET-DATA
            }
        }
    }
    services {
        telnet;
        xnm-clear-text;
        web-management {
            http;
        }
    }
    syslog {
        file test {
            any any;
        }
    }
    extensions {
        providers {
            stuttgartuniv {
                license-type evaluation deployment-scope private;
            }
        }
    }
}
interfaces {
```

```
40      ge-0/0/1 {
41          unit 0 {
42              family inet {
43                  address 192.168.1.253/24;
44              }
45          }
46      }
47      ge-0/0/2 {
48          unit 0 {
49              family inet {
50                  address 192.168.3.254/24;
51              }
52          }
53      }
54      ge-0/0/3 {
55          unit 0 {
56              family inet {
57                  address 192.168.2.254/24;
58              }
59          }
60      }
61      ms-0/1/0 {
62          unit 0 {
63              family inet;
64          }
65      }
66  }
```

**Listing A.4:** Konfiguration: System

Alle Einträge bis auf *Syslog* (Zeile 26) und *extensions* (Zeile 31) sollten vorhanden sein. Listing A.5 zeigt den Befehl, um den *extensions*–Eintrag vorzunehmen. Dieser Eintrag ist notwendig, um eigene Anwendungen installieren zu können.

```
# set system extensions providers stuttgartuniv license-type evaluation deployment-scope
    private
# commit
```

**Listing A.5:** Befehl: Extensions–Eintrag hinzufügen

Listing A.6 zeigt den Befehl, um den *Syslog*–Eintrag vorzunehmen.

```
# set system syslog file test any any
# commit
```

**Listing A.6:** Befehl: Syslog–Eintrag hinzufügen

Dieser Befehl gibt an, dass alle Logs in die Datei „test“ geschrieben werden. Der Pfad zu dieser Datei is */var/log/test*. Die Datei kann mit den nachfolgenden Befehlen betrachtet werden. Listing A.7 zeigt den Befehl, um die Datei aus dem Konfigurationsmodus zu betrachten.

```
# run file show /var/log/test
```

**Listing A.7:** Befehl: Logdatei aus Konfigurationsmodus betrachten

Listing A.8 zeigt den Befehl, um die Datei aus dem Betriebsmodus zu betrachten.

```
> show log test
```

**Listing A.8:** Befehl: Logdatei aus Betriebsmodus betrachten

Falls die Interfaces nicht in der Konfiguration vorhanden sind, lassen diese sich über den in Listing A.9 aufgeführten Befehl hinzufügen. Dieser Befehl fügt das Interface namens *ge-0/0/1* hinzu.

```
# set interfaces ge-0/0/1 unit 0 family inet address 192.168.1.253/24
```

**Listing A.9:** Befehl: Konfiguration eines Interfaces

Wichtig ist weiterhin die Konfiguration des Interfaces *ms-0/1/0*. Dieses steht für die MultiServices PIC. Listing A.10 zeigt, wie man dieses hinzufügt. Es muss keine IP–Adresse angegeben werden.

```
# set interfaces ms-0/1/0 unit 0 family inet
```

**Listing A.10:** Befehl: Konfiguration des MultiServices PIC Interface

### A.2.2. Anwendungsspezifische Konfiguration

Nachdem die Grundkonfiguration durchgeführt worden ist, erfolgt die anwendungsspezifische Konfiguration für den OpenFlow–Prototyp.

#### A.2.2.1. Konfiguration der MultiServices PIC–Komponente

Ein Teil der Anwendung läuft auf der MultiServices PIC. Daher müssen die in Listing A.11 aufgefürten Konfigurationen vorgenommen werden.

```
1  chassis {
2      fpc 0 {
3          pic 0 {
4              adaptive-services {
5                  service-package layer-3;
6              }
7          }
8          pic 1 {
9              adaptive-services {
10                 service-package {
11                     extension-provider {
```

```
12                      control-cores 2;
13                      data-cores 2;
14                      object-cache-size 512;
15                      policy-db-size 64;
16                      package ikr-openFlow-data;
17                      syslog {
18                          pfe {
19                              any;
20                              destination routing-engine;
21                          }
22                      }
23                  }
24              }
25          }
26      }
27  }
28  }
```

**Listing A.11:** Konfiguration der MultiServices PIC–Komponente

Die MultiSerives PIC, die im Router M7i des IKR verfügbar ist, befindet sich unter dem *fpc 0*, *pic 1*. Es müssen die Anzahl der *Control Cores* sowie der *Data Cores* angegeben werden (Zeile 13 und 14). Die Befehle hierführ sind in Listing A.12 angegeben.

```
# set chassis fpc 0 pic 1 adaptive-services service-package extension-provider control-cores
    <Anzahl cores>

 # set chassis fpc 0 pic 1 adaptive-services service-package extension-provider data-cores
     <Anzahl cores>
```

**Listing A.12:** Befehl: Konfiguration des MultiServices PIC Interface

Weiterhin muss angegeben werden, dass die MS PIC–Komponente des OpenFlow–Prototypen auf der MultiServices PIC ausgeführt werden soll (Zeile 16). Für die MS PIC–Komponente wurde das Paket *ikr–openFlow–data* erstellt. Listing A.13 zeigt den Befehl, der hierführ ausgeführt werden muss.

```
# set chassis fpc 0 pic 1 adaptive-services service-package extension-provider package
    ikr-openFlow-data
```

**Listing A.13:** Befehl: Konfiguration des MultiServices PIC Interface

#### A.2.2.2. Benutzereingaben

Für die Ausführung des OpenFlow–Prototyps müssen zusätzliche Informationen angegeben werden, wie in Abschnitt 3.3 beschrieben ist. Listing A.14 zeigt hierfür eine Beispielkonfiguration.

```
1  services {
2      service-set s1 {
3          interface-service {
4              service-interface ms-0/1/0;
5          }
6          extension-service openflow {
7              rules r1;
8              rules r2;
9          }
10     }
11 }
12 ikr {
13     openflow {
14         rule r1 {
15             match-direction input;
16             from {
17                 source-prefix 0.0.0.0/0;
18                 destination-prefix 0.0.0.0/0;
19             }
20             then allow;
21         }
22         rule r2 {
23             match-direction output;
24             from {
25                 source-prefix 0.0.0.0/0;
26                 destination-prefix 0.0.0.0/0;
27             }
28             then allow;
29         }
30         controller {
31             controller-ip-adr 192.168.1.4;
32             controller-port 6633;
33         }
34         port ge-0/0/2 {
35             port-number 2;
36             port-hw-address 50:c5:8d:c0:7c:02;
37         }
38         port ge-0/0/3 {
39             port-number 3;
40             port-hw-address 50:c5:8d:c0:7c:03;
41         }
42     }
43 }
```

**Listing A.14:** Beispielkonfiguration Benutzereingaben

Die einzelnen Eingaben können über entsprechende *set*–Befehle eingegeben werden.

#### A.2.2.3. Konfiguration der Interfaces

Um Pakete an die MS PIC–Komponenten zu leiten, müssen die in Listing A.14, Zeilen 1–11 definierten Services an Interfaces geheftet werden. Listing A.15 zeigt den entsprechenden

Befehl dafür. Als Beispiel wird das Service Set *s1* dem Interface *ge-0/0/2* zugeordnet.

```
# set interfaces ge-0/0/2 unit 0 family inet service input service-set s1
# set interfaces ge-0/0/2 unit 0 family inet service output service-set s1
```

**Listing A.15:** Befehl:Service Set einem Interface zuweisen

Listing A.16 zeigt einen Ausschnitt der Konfiguration. In diesem ist ein Interface mit angehängtem Service Set dargestellt.

```
1  interfaces {
2      ge-0/0/2 {
3          unit 0 {
4              family inet {
5                  service {
6                      input {
7                          service-set s1;
8                      }
9                      output {
10                         service-set s1;
11                     }
12                 }
13                 address 192.168.3.254/24;
14             }
15         }
16     }
17
18 }
```

**Listing A.16:** Ausschnitt aus Konfiguration: Interface mit Service Set

## A.3. Deinstallation der Anwendung

Um den OpenFlow–Prototypen zu deinstallieren, können die in Listing A.17 aufgeführten Befehle verwendet werden. Diese werden im Betriebsmodus des CLI ausgeführt.

```
> request system software delete ikr-openFlow-data
> request system software delete ikr-openFlow-mgmt
```

**Listing A.17:** Befehl: Deinstallation von Anwendungen

Bei *ikr–openFlow–data* handelt es sich um den Namen des Pakets der MS PIC–Komponente, *ikr–openFlow–mgmt* ist der Name des Pakets der RE–Komponente.

Zuvor sollte das Paket der MS PIC–Komponente aus der Konfiguration der MultiServices PIC entfernt werden. Listing A.18 zeit den hierfür benötigten Befehl.

```
# delete chassis fpc 0 pic 1 adaptive-services service-package extension-provider package
    ikr-openFlow-data
```

**Listing A.18:** Befehl: Paket aus MS PIC Konfiguration entfernen

## A.4. Debuggen der Anwendung

Für das Debuggen der Anwendung kann Logging eingesetzt werden. Hierfür müssen die in Listing A.11, Zeilen 17-22, aufgeführten Einträge gemacht werden. Diese geben an, dass alle Log–Daten der Routing Engine übergeben werden, welche sie in eine entsprechenden Log–Datei schreibt. Die Angabe, in welche Log–Datei geschrieben werden soll, wird in der Grundkonfiguration vorgenommen. Der Befehl hierfür ist in Listing A.6 zu finden.
Es bietet sich weiterhin an, Wireshark [wirb] einzusetzen und das OpenFlow–Plugin [wira] zu installieren.
Falls der Beacon–Controller verwendet wird, können dessen Log–Dateien betrachtet werden. Diese befinden sich im Verzeichnis */opt/beacon/logs* der VM. Es werden hierbei Informationen über die Verbindung mit dem Controller geloggt sowie Informationen über empfangene und gesendete Nachrichten vom Controller.

## A.5. Nützliche Befehle

### A.5.1. Kopieren der Konfiguration über ein Terminal

Damit nicht alle Konfigurationen einzeln eingeben werden müssen, kann eine existierende Konfiguration kopiert werden. Hierfür wird der in Listing A.19 dargestellte Befehl ausgeführt.

```
# load merge terminal
```

**Listing A.19:** Befehl: Öffnenen eines Terminals zum Kopieren der Konfiguration

Dieser Befehl öffnet ein Terminal, in welches ein bestimmter Teil der Konfiguration kopiert werden kann. Nachdem der gewünschte Text reinkopiert wurde, wird die Eingabe mit der Tastenkombination *Ctrl + d* beendet. Um die Konfiguration zu aktivieren, muss der *commit*–Befehl ausgeführt werden. Es empfiehlt sich, davor den *commit check* Befehl auszuführen, um mögliche Fehler abzufangen.

### A.5.2. Router neustarten

Für einen Neustart des Routers kann der in Listing A.20 aufgeführte Befehl verwendet werden.

```
> request system reboot
```

**Listing A.20:** Befehl: Neustart des Routers

### A.5.3. Prozess / Anwendung neustarten

Der in Listing A.21 aufgefürte Befehl kann verwendet werden, um einen Prozess oder eine Anwendung neu zu starten.

```
> restart <Name des Prozesses / der Anwendung>
```

**Listing A.21:** Befehl: Neustart eines Prozesses

### A.5.4. Prozess / Anwendung aktiveren und deaktivieren

Der in Listing A.23 aufgeführte Befehl zeigt, wie ein Prozess oder eine Anwendung aktiviert oder deaktiviert werden kann.

```
# set system processes <Name des Prozesses> [enable|disable]
```

**Listing A.22:** Befehl: De- / Aktivierung von Prozessen

### A.5.5. Aktuelle Konfiguration anzeigen

Mit dem in Listing A.23 aufgeführten Befehl kann die aktuelle Konfiguration des Routers angezeigt werden.

```
> show configuration
```

**Listing A.23:** Befehl: Konfiguration des Routers anzeigen

## A.6. Komplette Routerkonfiguration

Das folgenden Listing A.24 zeigt eine komplette Beispielkonfiguration, welche die Ausführung des OpenFlow–Prototyps ermöglicht. Darin enthalten sind alle Informationen, die für die Kommunikation mit einem externen Controller benötigt werden, Informationen für die Erstellung von Interface Service Sets, Informationen über die OpenFlow–Ports des Routers sowie auf welche Weise ein Interface Service Set einem Interface des Routers zugeordnet wird (Interface ge–0/0/2, Zeile 77).

```
## Last commit: 2011-04-26 08:50:28 UTC by netlab
version 10.2R1.8;
system {
    host-name labrt7;
    domain-name netlab.ikr.uni-stuttgart.de;
    root-authentication {
        encrypted-password "$1$p15w0lhZ$v11JtxdQ0ElR.3QOVM7VH1"; ## SECRET-DATA
    }
    name-server {
        10.31.1.254;
    }
    login {
        user netlab {
            uid 2000;
            class super-user;
            authentication {
                encrypted-password "$1$7654Ro28$8a5HFvOQU84eFyIrvFxt80"; ## SECRET-DATA
            }
        }
    }
    services {
        telnet;
        xnm-clear-text;
        web-management {
            http;
        }
    }
    syslog {
        file test {
            any any;
        }
    }
    extensions {
        providers {
            stuttgartuniv {
                license-type evaluation deployment-scope private;
            }
        }
    }
}
chassis {
    fpc 0 {
        pic 0 {
            adaptive-services {
                service-package layer-3;
            }
        }
        pic 1 {
            adaptive-services {
                service-package {
                    extension-provider {
                        control-cores 2;
                        data-cores 2;
                        object-cache-size 512;
```

```
                    policy-db-size 64;
                    package ikr-openFlow-data;
                    syslog {
                        pfe {
                            any;
                            destination routing-engine;
                        }
                    }
                }
            }
        }
    }
}
interfaces {
    ge-0/0/1 {
        unit 0 {
            family inet {
                address 192.168.1.253/24;
            }
        }
    }
    ge-0/0/2 {
        unit 0 {
            family inet {
                service {
                    input {
                        service-set s1;
                    }
                    output {
                        service-set s1;
                    }
                }
                address 192.168.3.254/24;
            }
        }
    }
    ge-0/0/3 {
        unit 0 {
            family inet {
                address 192.168.2.254/24;
            }
        }
    }
    ms-0/1/0 {
        unit 0 {
            family inet;
        }
    }
    fxp0 {
        unit 0 {
            family inet {
                address 10.31.1.217/24;
            }
        }
```

```
110     }
111 }
112 services {
113     service-set s1 {
114         interface-service {
115             service-interface ms-0/1/0;
116         }
117         extension-service openflow {
118             rules r1;
119             rules r2;
120         }
121     }
122 }
123 ikr {
124     openflow {
125         rule r1 {
126             match-direction input;
127             from {
128                 source-prefix 0.0.0.0/0;
129                 destination-prefix 0.0.0.0/0;
130             }
131             then allow;
132         }
133         rule r2 {
134             match-direction output;
135             from {
136                 source-prefix 0.0.0.0/0;
137                 destination-prefix 0.0.0.0/0;
138             }
139             then allow;
140         }
141         controller {
142             controller-ip-adr 192.168.1.4;
143             controller-port 6633;
144         }
145         port ge-0/0/2 {
146             port-number 2;
147             port-hw-address 50:c5:8d:c0:7c:02;
148         }
149         port ge-0/0/3 {
150             port-number 3;
151             port-hw-address 50:c5:8d:c0:7c:03;
152         }
153     }
154 }
```

**Listing A.24:** Komplette Router–Konfiguration

## A.7. Ausschnitt aus der DDL–Konfigurationsdatei

```
object services {

      flag no-struct;

      object service-set {

            notify DNAME_IKR_OPENFLOWD;

            flag no-struct;

            object extension-service {

                 flag no-struct;

                 object rules {
                      help "One or more ikr-openFlow rules";
                      flag setof list;
                      define DDLAID_SVCS_SVC_SET_EXT_SERVICE_RULES;

                      attribute rule-name {
                          help "Rule name";
                          flag nokeyword identifier;
                          type ranged string 1 .. 64;
                          path-reference "ikr openflow rule <*>";
                          must "ikr openflow rule $$";
                          must-message "referenced ikr-openflow rule must be defined";

                      }
                  }
            }
        }
   }
}
```

**Listing A.25:** Beschreibung Service–Element aus der DDL–Konfiguration

# Literaturverzeichnis


[cisa] Cisco Systems, Inc. (Zitiert auf Seite 73)

[Cisb] Cisco. Catalyst 6500 series overview website. URL http://www.cisco.com/en/US/products/hw/switches/ps708/. (Zitiert auf Seite 9)

[Fre] FreeBSD web site. URL http://www.freebsd.org/. (Zitiert auf den Seiten 18 und 26)

[Garo6] A. Garrett. *Junos Cookbook (Cookbooks (O'Reilly))*. O'Reilly Media, Inc., 2006. (Zitiert auf den Seiten 19 und 75)

[GKP+08] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker. NOX: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38:105–110, 2008. doi:http://doi.acm.org/10.1145/1384609.1384625. URL http://doi.acm.org/10.1145/1384609.1384625. (Zitiert auf Seite 15)

[hp] HP Deutschland Startseite. URL http://www.hp.com/. (Zitiert auf Seite 73)

[JNa] I. Juniper Networks. Junos OS web site. URL http://www.juniper.net/us/en/products-services/nos/junos/. (Zitiert auf Seite 18)

[JNb] I. Juniper Networks. Junos sdk web site. URL http://www.juniper.net/us/en/products-services/junos-developer/junos-sdk/. (Zitiert auf Seite 21)

[JNc] I. Juniper Networks. M7i Multiservice Edge Router - Hardware Guide. URL http://www.juniper.net/techpubs/en_US/release-independent/junos/information-products/topic-collections/hardware/m-series/m7i/hwguide/m7i-hwguide.pdf. (Zitiert auf Seite 18)

[JNd] I. Juniper Networks. M7i Multiservice Edge Router End-of-Life PIC Guide. URL http://www.juniper.net/techpubs/en_US/release-independent/junos/information-products/topic-collections/hardware/m-series/m7i/eol-pics/m7i-pic-eol.pdf. (Zitiert auf Seite 18)

[JNe] I. Juniper Networks. MX Series 3D Universal Edge Router - Enterprise Ethernet Service Provider Router. URL http://www.juniper.net/us/en/products-services/routing/mx-series/. (Zitiert auf den Seiten 9 und 74)

[JNf] I. Juniper Networks. Network Security Solutions-Networking Performance Optimization-Juniper Networks. URL http://www.juniper.net/us/en/. (Zitiert auf Seite 73)

[JNg] I. Juniper Networks. Packet Flow Through the Adaptive Services or MultiServices PIC – JUNOS 10.2 Services Interface Configuration Guide. URL http://www.juniper.net/techpubs/en_US/junos10.2/information-products/topic-collections/config-guide-services/topic-41108.html#id-10058055. (Zitiert auf den Seiten 6 und 20)

[JNh] I. Juniper Networks. Technical Documentation. URL http://www.juniper.net/techpubs. (Zitiert auf Seite 19)

[JN10] I. Juniper Networks. Juniper Networks Partner Solution Development Platform Documentation, 2010. (Zitiert auf den Seiten 6, 18 und 22)

[KAB10] J. Kelly, W. Araujo, K. Banerjee. Rapid service creation using the JUNOS SDK. *SIGCOMM Comput. Commun. Rev.*, 40:56–60, 2010. doi:http://doi.acm.org/10.1145/1672308.1672320. URL http://doi.acm.org/10.1145/1672308.1672320. (Zitiert auf Seite 20)

[MAB+08] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008. doi:http://doi.acm.org/10.1145/1355734.1355746. (Zitiert auf den Seiten 9 und 11)

[mae] Maestro-Platform- A scalable control platform written in Java which supports OpenFlow switches. URL http://code.google.com/p/maestro-platform/. (Zitiert auf Seite 15)

[NEC+08] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, N. McKeown. Implementing an OpenFlow switch on the NetFPGA platform. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, pp. 1–9. ACM, New York, NY, USA, 2008. doi:http://doi.acm.org/10.1145/1477942.1477944. URL http://doi.acm.org/10.1145/1477942.1477944. (Zitiert auf Seite 9)

[Net] J. Networks. Junos Software, Technical Documentation. URL http://www.juniper.net/techpubs/software/junos/index.html. (Zitiert auf Seite 18)

[Net10] J. Networks. Installing the JUNOS SDK. 2010. (Zitiert auf Seite 75)

[ofC] Beacon. OpenFlow Controller by the Standford University. URL http://www.openflowhub.org/display/Beacon/Beacon+Home. (Zitiert auf Seite 15)

[opea] OpenFlow Consortium. OpenFlow website. URL http://www.openflow.org. (Zitiert auf Seite 9)

[opeb] OpenFlow Specification. URL http://www.openflow.org/wp/documents/. (Zitiert auf den Seiten 6, 9, 12, 13, 14, 39, 45 und 48)

[opec] OpenFlow Specification 1.1. URL http://www.openflow.org/wp/documents/. (Zitiert auf Seite 73)

[Sta] I. O. for Standardization. Open systems interconnection (OSI). (Zitiert auf den Seiten 13 und 45)

[Tik] Y. Tikhiy. UNIX man pages : mbuf (9. (Zitiert auf Seite 23)

[wira] OpenFlow Wireshark Dissector - OpenFlow Wiki. (Zitiert auf den Seiten 63 und 83)

[wirb] Wireshark - Go deep. (Zitiert auf den Seiten 63 und 83)

Alle URLs wurden zuletzt am 08.07.2011 geprüft.

**Erklärung**

Mir ist bekannt und ich erkenne an, dass ich an den Ergebnissen meiner Studienarbeit/Diplomarbeit keine eigenständigen Verwertungsrechte habe. Eine Verwertung der Arbeit einschließlich der Ausarbeitung darf nur nach Zustimmung durch das Institut für Nachrichtenvermittlung und Datenverarbeitung erfolgen.
Ich erkläre weiterhin, dass ich diese Arbeit selbständig verfasst und keine anderen als die in meiner Ausarbeitung angegebenen Hilfsmittel für die Durchführung der Arbeit verwendet habe.

---

(Filip Kostadinow)