Institut für Parallele und Verteilte Systeme

Abteilung Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany

Diplomarbeit Nr. 3420

Konzeption und Implementierung eines OpenFlow-basierten IP-Multicast-Dienstes für Datenzentren

Jonas Danzl

Studiengang:	Informatik
Prüfer:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer:	Dr. rer. nat. Frank Dürr
begonnen am:	10.12.2012
beendet am:	10.06.2013
CR-Klassifikation:	C.2.1, C.2.3, C.2.2, C.4, G.2.2, G.1.6

Kurzfassung

Die rasante Verbreitung von Mobile-Computing, Cloud-Services und Big-Data Anwendungen führt zu immer größeren Datenmengen, die in Datencentern verwaltet werden müssen. Für ein effizientes Verteilen und Verarbeiten dieser Daten ist eine Multicast-Kommunikation inzwischen unabdingbar geworden. Dennoch führt der rapide Anstieg von Speicher und Rechenkapazitäten sowie ein hoher Grad an Virtualisierung zu immer höheren Anforderungen an die Netzinfrastruktur. Die heutigen statischen Netzwerke benötigen eine aufwändige, manuelle Administration und passen nicht mehr ins Systemumfeld moderner Datencenter. Es sind Lösungen gefordert, die dynamisch auf Ausfälle, Datenverkehrsänderungen oder auf die individuellen Bedürfnisse einzelner Anwendungen reagieren können. Mit *Software defined Networking* (SDN), in Zusammenhang mit dem *OpenFlow-Protokoll*, existiert eine flexible Netzwerkarchitektur, die diesen Anforderungen begegnen kann. Während bisheriges Multicast-Routing durch verteilte, nicht-optimale und komplexe Algorithmen eingeschränkt ist, bietet ein OpenFlow-basierter IP-Multicast-Dienst die Möglichkeit, einfachere und effizientere Lösungen zu finden. Austauschbare Softwaremodule ermöglichen es, mehrere Routingalgorithmen zu implementieren und diese zur Laufzeit beliebig zu wechseln. Gleichzeitig können die speziellen Anforderungen in Datenzentren einbezogen werden. Im Zuge dieser Diplomarbeit wird ein solcher Dienst konzipiert, implementiert und ausgewertet. Hierfür werden auf Grundlage des Steinerbaumproblems optimierte Routing-Lösungen betrachtet, die in bisherigen, verteilt verwalteten Netzen, als nicht praktikabel galten. Durch proaktive Vorberechnung und Einrichtung sämtlicher Multicast-Routen, wird dabei eine Paket-Weiterleitung in *Line Rate* ermöglicht. Um vorhandene Netz-Ressourcen innerhalb eines Datencenters möglichst optimal auszunutzen, werden Methoden aus dem Bereich des *Traffic Engineerings* untersucht. Die Evaluation zeigt, dass der Dienst effizient arbeitet und skalierbar ist. Gleichzeitig wird eine eine Optimierung der Multicastbäume und eine Verbesserung der Lastverteilung erreicht.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In diesem Kapitel wird in Abschnitt 1.1 in den Themenbereich dieser Diplomarbeit eingeführt und die Relevanz für Forschung und Praxis motiviert. Anschließend wird in Abschnitt 1.2 ein Überblick über den Aufbau und die nachfolgenden Kapitel gegeben.

1.1 Motivation

Die Verbreitung von mobilen Anwendungen, Cloud-Services und sozialen Netzwerken haben in den letzten Jahren zu stark wachsenden Anforderungen an Datencenter geführt. Ein Grund dafür ist der stetige Anstieg des weltweiten Datenaufkommens. Das betrifft vor allem unstrukturierte Daten, die nicht nur gespeichert, sondern auch verteilt und verarbeitet werden müssen. Wie die IDC Studie „Digital Universe“ [GC08] aus dem Jahr 2011 zeigt, verdoppelt sich das Datenvolumen alle zwei Jahre. Die Gesamtmenge an Daten, die 2011 erstellt oder repliziert worden ist, entsprach 1,8 Zettabyte. 2012 waren es bereits 2,7 Zettabyte. Umgerechnet sind das 2,7 Billionen Gigabyte. Dies würde einem Äquivalent von 70 Millionen Jahren an HD-Filmmaterial entsprechen. Die IDC prognostiziert, dass sich die Datenmengen in heutigen Datencentern bis zum Jahr 2020 noch verfünfzigfachen werden. Eine solche Datenexplosion stellt nicht zuletzt enorme Anforderungen an die Netzinfrastruktur. Optimale Ausnutzung vorhandener Bandbreiten, effiziente Kommunikationsdienste und intelligente Lastverteilungen sind unabdingbar, um einem drohenden Kollaps der Netze vorzubeugen.

Da sich die Systemlandschaft durch Server- und Speichervirtualisierung, Cloud-Services und veränderter Kommunikationspatterns in Datencentern gewandelt hat, ist die Konfiguration der Netzwerke heute komplexer denn je. Oftmals müssen selbst bei kleinen Änderungen eine Reihe von zeitaufwändigen, manuellen Aufgaben durchgeführt werden, die herstellerabhängig über eigene Konsolen und Befehle konfiguriert werden müssen. Dynamische und flexiblere Ansätze sind gefordert, damit sich die Netzwerke automatisch an die jeweilige Anwendung anpassen können. Ein Lösungsansatz, der das Potential hat diesen Problemen begegnen zu können, ist das *Software Defined Networking* (SDN).

SDN ist eine aufkommende Netzwerkarchitektur, die direkt programmierbar ist [Fou12]. Die Kontrolle und Verwaltung des Netzwerkes ist von der eigentlichen Weiterleitung der Datenpakete entkoppelt. Ein Kontrollprogramm, der *SDN-Controller*, besitzt eine zentrale Sicht auf die unterliegende Infrastruktur und abstrahiert diese für diverse SDN-Anwendungen. Solche Anwendungen kennen nicht nur die unterliegende Netzstruktur, sondern haben auch die Möglichkeit, Gerätedaten, wie Weiterleitungstabellen, direkt zu manipulieren. Für diese Abstrahierung besteht mit OpenFlow [MAB+08] eine API sowie ein zugehöriges Protokoll,

die zusammen einen einheitlichen Befehlssatz auf den Netzgeräten definieren. In traditionellen Systemen ist die Verwaltungslogik auf die einzelnen Netzgeräte, wie den Routern oder den Switches, verteilt. Die durch OpenFlow hinzugewonnene Zentralität eröffnet völlig neue Möglichkeiten, einheitliche Netzanwendungen, wie neue Routingalgorithmen, Firewalls, Netzwerkmonitore, oder sogar *Traffic Engineering*, in Form von erweiterbaren Softwaremodulen zu implementieren. Außerdem ist die Netzwerkvirtualisierung, mit dem Ziel global kontrollierbare, logisch isolierte Netzwerke in die virtuelle Systemlandschaft heutiger Datencenter zu integrieren, ein oft diskutierter Anwendungfall. Die vormals komplexen Aufgaben der Systemadministration können durch herstellerunabhängige Software, die selbst oder durch Drittanbieter entwickelt wird, übernommen werden. Erste Anwendungen von SDN werden schon von Google und Facebook eingesetzt, die bereits OpenFlow in ihren Datencentern unterstützen [goo]. Zusätzlich bieten zahlreiche IT-Dienstleister, wie Big Switch Networks [Net], HP [Pac] oder IBM [IBM] ihren Kunden inzwischen OpenFlow-basierte Produktlösungen an.

Eine sinnvolle Nutzung von SDN in Datencentern setzt allerdings voraus, dass effiziente Kommunikationsdienste für darauf aufbauende SDN-Anwendungen zur Verfügung stehen. Die meisten Controller bringen nativ ein Standard-Routingverfahren auf Basis einer einfachen Entfernungsmetrik mit. Für die anspruchsvollen Anforderungen an Datencenternetzwerke ist dies jedoch unzureichend. Der Datenverkehr muss dort effizient über die verfügbaren Ressourcen aufgeteilt und die verfügbare Bandbreite möglichst optimal ausgenutzt werden. So fehlt in vielen SDN-Plattformen die Umsetzung eines Multicastdienstes. Dabei ist diese besonders effiziente Art der Gruppenkommunikation für viele verteilte Aufgaben unabdingbar. Multicast wird für das Versenden von einer Datenquelle an mehrere spezifische Empfänger verwendet und bietet die Grundlage für zahlreiche Anwendungen wie Video-Streaming, Datenreplikation oder die Datenverteilung in Hochleistungsrechenzentren. War das Multicast-Routing jedoch seither durch verteilte, nicht-optimale und komplexe Algorithmen eingeschränkt, bietet die zentrale Sichtweise des SDN-Controllers die Möglichkeit, einfachere und effizientere Multicastlösungen zu finden.

In dieser Diplomarbeit wird ein OpenFlow-basierter IP-Multicast-Dienst konzipiert, der den speziellen Anforderungen in Datenzentren gerecht werden soll. Dabei wird ein verbessertes Routing zur effizienten Nutzung der Bandbreite in der Netz-Infrastruktur angestrebt. Im Zuge dessen wird das aus der Graphentheorie bekannte Steinerbaumproblem [Pro02] mit dem Ziel untersucht, kostenoptimierte Multicastbäume in polynomieller Zeit zu approximieren. Als Metrik kommt eine Kombination aus Entfernung und verfügbarer Bandbreite zum Einsatz, was ein *Load-Balancing* über die Netzredundanzen im Datencenter ermöglichen soll. Außerdem ist die Integration in vorhandene OpenFlow-fähige Netze von Relevanz.

Sämtliche im Zuge eines Multicast durchgeführten Aufgaben, die seither verteilt durch Multicast-Router durchgeführt wurden, werden bei SDN durch entsprechende Softwaremodule im Controller implementiert. Neben der Ermittlung von Verteilbäumen und der Speicherung von Gruppenzusammensetzungen ist das vor allem die Implementierung und die Verwaltung von Multicast-Routen auf den Netzgeräten. Dafür werden die Multicastbäume proaktiv vorberechnet und anschließend auf den Netzgeräten deployed. Im Gegensatz zu einem reaktiven Einrichten der Routen, sind bei diesem Ansatz alle Hostgeräte im Netzwerk jederzeit und ohne weiteren Konfigurationsaufwand bereit, eine Gruppenkommunikation zu initiieren. Besonders in einem beschränkten Umfeld, wie in einem Datencenternetzwerk, bietet diese Lösung er-

hebliche Vorteile. Die Weiterleitung kann ab dem ersten Paket in *Line-Rate* erfolgen, so dass Verzögerungen vermieden werden. Gleichzeitig wird der Überlastung des Controllers durch einen plötzlich auftretenden *Paket-Burst*, was z. B. bei der Verwendung eines verbindungslosen Protokolls wie UDP [Pos80] geschehen kann, vorgebeugt. Zusätzlich bietet eine Software größere Flexibilität und einfachere Änderungsmöglichkeiten. Sollte beispielsweise die Menge der Routeneinträge in den OpenFlow Netzgeräten eine kritische Anzahl erreichen, bietet der Dienst die Möglichkeit den Routingalgorithmus zur Laufzeit zu wechseln. Weiterhin bezieht sich diese Flexibilität auch auf die Robustheit eines OpenFlow-Netzes. So können Ausfall bedingte Topologieänderungen durch den Controller erkannt und die betroffenen Routen neu berechnet werden.

Der Multicastdienst wird im Rahmen dieser Diplomarbeit als Erweiterung für den SDN-Controller Floodlight [flo] implementiert und die Lösungskonzeption anschließend auf einer Datencenter-Topologie ausgewertet. Die Evaluation zeigt, dass der Dienst effizient arbeitet und skalierbar ist. Gleichzeitig wird eine eine Optimierung der Multicastbäume und eine Verbesserung der Lastverteilung erreicht.

1.2 Aufbau dieser Arbeit

In Kapitel 2 werden Grundlagen bezogen auf Multicasting, SDN und OpenFlow vorgestellt sowie auf verwandte Arbeiten eingegangen. Kapitel 3 beschäftigt sich mit der Vorstellung des Systemmodells, insbesondere bezogen auf Datencenter-Topologien. Außerdem werden die Anforderungen an den Multicastdienst formuliert. In Kapitel 4 wird auf Grundlage dieser Anforderungen eine Konzeption erstellt. Dafür wird eine Architektur zur Umsetzung des Multicastdienstes herausgearbeitet und vorgestellt. Kapitel 5 beschäftigt sich mit der Implementierung des in Kapitel 4 erarbeiteten Konzeptes und geht auf implementierungsspezfische Details des Open-Source Controllers Floodlight ein. Anschließend wird die implementierte Lösung in Kapitel 6 getestet und evaluiert. Dafür werden relevante Messergebnisse herausgearbeitet und interpretiert. Zum Schluss wird eine Zusammenfassung und ein Ausblick in Kapitel 7 gegeben.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Begriffe zum Thema Multicast und *Software Defined Networking* (SDN) erläutert. Dazu gehören das *Internet Group Management Protocol* (IGMP) (Abschnitt 2.1) sowie die Untersuchung von optimierten Wegfindungsmethoden in Abschnitt 2.3. Ein besonderer Schwerpunkt soll dabei auf die Formalisierung zentralisierter Multicast-Routing-Probleme gelegt werden. Der SDN-Teil beschreibt in Abschnitt 2.4 die allgemeine Architektur und geht in Abschnitt 2.5 auf den OpenFlow-Standard als Schlüsseltechnologie ein. Zum Schluss folgt ein Überblick über verwandte Arbeiten in Abschnitt 2.6.

2.1 Multicast-Adressierung und Gruppenverwaltung

Multicast ist eine Form der Gruppenkommunikation und bezeichnet eine Nachrichtenübertragung von einer Datenquelle an mehrere Empfänger. Die Menge der Empfänger bildet eine *Multicastgruppe*, wobei ein Sender nicht Mitglied dieser Gruppe sein muss.

Die Kommunikationsformen *Broadcast* und *Unicast* können als Spezialfälle einer Multicast-Kommunikation angesehen werden. Während bei einem Broadcast alle Knoten im Netz Empfänger sind, beschreibt ein Unicast eine Kommunikation zwischen genau einem Sender und einem Empfänger. Zwar könnte man eine Gruppenverteilung auch mit mehreren Unicast-Übertragungen realisieren, das führt jedoch bei steigender Gruppengröße zu einer starken Verschwendung der Netzressourcen [Wit99]. Anstatt dieselbe Nachricht mehrfach über die gleiche Leitung zu schicken, wird sie bei einem Multicast nur einmal gesendet. In den Zwischensystemen, z. B. Router, wird sie repliziert und auf die entsprechenden Ausgangsleitungen kopiert. Die Vorteile eines Multicasts liegen demnach in einer effizienten Bandbreitennutzung und in der Entlastung der Zwischensysteme [Wit99].

Es gibt zahlreiche Anwendungen, die von den Vorteilen eines Multicast profitieren. Dazu zählen Videokonferenzen, Video-Streaming, Groupware oder Onlinespiele. Trotzdem sind im Internet nur wenige Router Multicast-fähig, d.h. sie bieten keine globale Multicastunterstützung über lokale Netzgrenzen hinweg [RES06]. Diese Arbeit konzentriert sich auf Datencenternetze, in denen Multicastübertragungen uneingeschränkt eingesetzt werden können. Eine Anwendung im Datencenter wäre beispielsweise die Replizierung von Daten an eine Menge von Servern oder die Verteilung von Software an ausgewählte Rechnergruppen [BBB07].

Die verbreiteteste Mullticast-Implementierung ist der *IP-Multicast*. *IP-Multicast* basiert auf der Erweiterung des IP-Protokolls um *Gruppenverwaltung* und *Routingprotokolle* [AFM92]. Für ersteres kommt das *Internet Group Management Protocol* (IGMP) zum Einsatz (siehe Abschnitt 2.1.3). Die Routingprotokolle sind für die Paketverteilung an die Zielrouter verantwortlich. Sie sollen dabei Redundanzen und Schleifen bei der Wegfindung verhindern [MSG+12].

Nachfolgend wird ein Überblick über die Multicast-Kommunikation in paketvermittelten Netzen gegeben und auf die Aufgaben der Gruppenverwaltung und des Routings genauer eingegangen. Ist im nachfolgenden Teil der Arbeit von *Multicast* die Rede, ist stets *IP-Multicast* gemeint, sofern dies nicht anders vermerkt ist.

2.1.1 IP-Multicast im OSI-Schichtenmodell

Die Multicasttechnik lässt sich gemäß dem *OSI-Modell* in die Vermittlungsschicht einordnen. Analog zu einer Unicast-Übertragung, schickt ein beliebiger Sender IP-Pakete an eine spezielle IP-Empfängeradresse. Diese Adresse steht stellvertretend für eine gesamte Multicastgruppe. Die dazwischen liegenden Router benutzen Tabelleneinträge für die Weiterleitung.

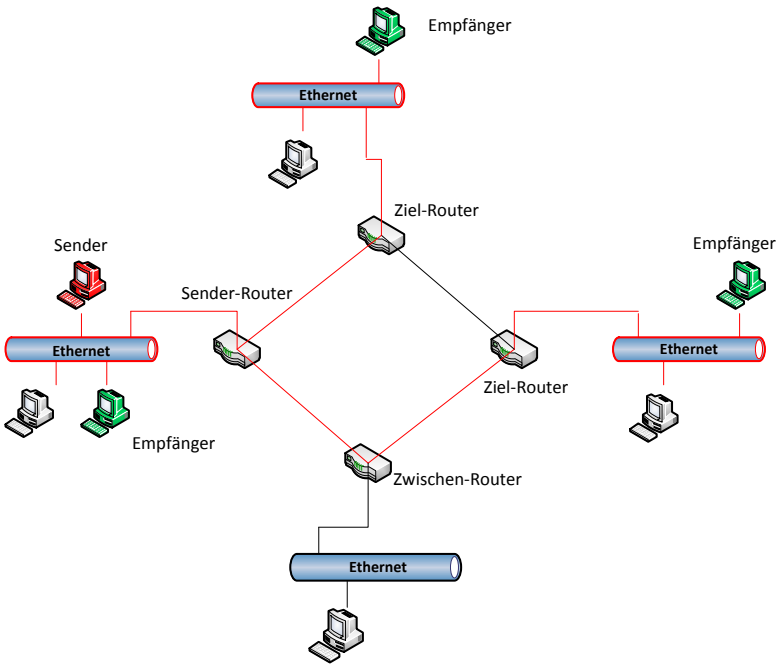

Abbildung 2.1: Ein Sender (rot) schickt ein Multicastpaket entlang eines Multicastbaumes (rote Verbindungsstücke) an eine Empfängergruppe (grün).

Abbildung 2.1 zeigt den Nachrichtenverlauf zwischen einem Sender und einer Multicastgruppe. Zuerst schickt ein Sender ein Multicastpaket über das Medium der Sicherungsschicht, z. B. Ethernet, an den Multicast-Router. Dazu bildet der Sender die IP-Multicastadresse auf

eine MAC-Multicastadresse ab. Da es sich bei Ethernet um ein Broadcast-Medium handelt, sind alle Gruppenmitglieder im selben *Lokal Area Network* (LAN) in der Lage das Paket zu empfangen. Der Multicast-Router ist nun dafür zuständig, das Paket gemäß dem verwendeten Routingprotokoll an alle Ziel-Router zu verteilen. Dies geschieht entlang eines Verteilbaumes. Ein Ziel-Router ist ein Multicast-Router, der Mitglieder der Multicastgruppe in seinen angrenzenden Netzen hat. Die nötigen Mitgliederinformationen erhält er durch das Ausführen von IGMP (siehe Abschnitt 2.1.3). Damit ein Ziel-Router ein Paket schließlich in den Zielnetzen ausliefern kann, wird die IP-Adresse erneut auf eine MAC-Multicastadresse abgebildet und über den jeweiligen Port an die entsprechenden Hosts im LAN geschickt.

Mehrere LANs können durch einen Switch verbunden werden, der das Netz in Broadcastdomänen aufteilt. Im einfachsten Fall leitet ein Switch, Rahmen mit MAC-Multicastadressen über alle Ausgangsports weiter. Bei diesem Vorgehen können diese jedoch auch in Bereiche des Netzes gelangen, in denen sich gar keine Gruppenmitglieder befinden. Eine effizientere Lösung ist das *IGMP-Snooping* [CKS06]. *IGMP-Snooping* versetzt einen Switch in die Lage, den IGMP Verkehr zwischen Host und Router mithören zu können. Dadurch kann er lernen in welchem Teil des Netzes sich Gruppenmitglieder befinden.

Für die meisten Multicastanwendungen kommt auf der Transportebene das *User Datagram Protocol* (UDP) [Pos80] zum Einsatz. UDP ist ein verbindungsloses Protokoll, d.h. es behandelt weder verloren gegangene Nachrichten noch wird die Auslieferungsreihenfolge beachtet [TW12]. Der Vorteil ist jedoch eine effiziente Übertragung an möglichst viele Empfänger bei geringem Overhead. Ähnlich dem *Transmission Control Protocol* (TCP) [Pos81] gibt es auch Protokolle, die eine zuverlässige Multicast-Kommunikation garantieren. Ein Beispiel ist der *Pragmatic General Multicast* (PGM) [SFL+98]. Weiterhin werden in RFC 3048 [WVK+01] Mechanismen zum Erkennen und Reparieren von verloren gegangenen bzw. fehlerhaften Paketen für Multicasting empfohlen.

2.1.2 Adressierung

Die IP-Multicastadressen müssen für die Übertragung auf der Sicherungsschicht in MAC-Multicastadressen abgebildet werden. Für IPv4 stehen die Adressen der Klasse D (224.0.0.0 bis 239.255.255.255) und für IPv6 jede mit FF00::/8 beginnende Adresse für die Gruppenadressierung auf der Vermittlungsschicht zur Verfügung. Die Adressvergabe ist in RFC 5771 [VC10] von der Internet Engineering Task Force (IETF) spezifiziert. Bei der 48 Bit Ethernet-Multicastadresse sind die 25 höchstwertigsten Bits fest vorgegeben, wobei das letzte Bit, des höchstwertigsten Bytes, die Unterscheidung zwischen Unicast und Multicast angibt. Somit stehen für die Gruppenadressierung mit IP-Adressen 25 Bit, für Ethernet-MAC-Adressen aber lediglich 23 Bit zur Verfügung.

Im Gegensatz zur Unicast-Kommunikation erfolgt die Abbildung Eins-zu-Eins, wobei die führenden 5 Bit einer IP-Adresse abgeschnitten werden [Wit99]. So kann auf eine aufwändige Adressauflösung wie durch das *Adress Resolution Protocol* (ARP) [Plu82] verzichtet werden. Durch die entstandene Uneindeutigkeit, müssen die Hosts beim Empfang die IP Multicastadressen vergleichen, um sicherzustellen, dass sie einen bestimmungsgemäßen Empfänger der Gruppennachricht darstellen.

2.1.3 Gruppenverwaltung mit IGMP

IP-Multicast basiert auf der Erweiterung des IP-Protokolls um Gruppenverwaltung und Routingprotokolle [AFM92]. Die Hauptaufgaben einer Gruppenverwaltung sind das Bereitstellen der Mitgliederlisten und die Handhabung von Änderungen in der Gruppenzusammensetzung. Diese Aufgaben können zentral oder verteilt durchgeführt werden. Eine zentrale Gruppenverwaltung besitzt eine globale Sicht über alle Gruppen, deren Mitglieder und auftretenden Gruppenänderungen. Verteilte Verfahren wiederum haben Vorteile bezüglich Skalierbarkeit, jedoch die Schwierigkeit eine konsistente Sicht bereitzustellen [Wit99]. Die Gruppenverwaltung beschränkt sich hier auf die Bekanntgabe der Gruppenmitgliedschaften an einen designierten Multicast-Router.

Bei IPv4 kommt das *Internet Group Protocol* (IGMP) [CDK+02] für die Gruppenverwaltung zum Einsatz. IGMP ist ein integraler Bestandteil des IP-Protokolls und muss implementiert werden, wenn Multicast unterstützt werden soll. Die entsprechende Erweiterung für IPv6 nennt sich Multicast Listener Discovery (MLD) [DFH99] und funktioniert ähnlich. Pakete zur Gruppenverwaltung sind in IP-Paketen gekapselt und tragen den vordefinierten Wert 2 im IP-Headerfeld *protocol*. Dieser gibt an, dass das Folgeprotokoll IGMP ist. Weitere mögliche Folgeprotokolle wären z. B. TCP oder UDP für ein ankommendes Unicast- oder Multicastpaket. Sämtliche Protokollnummern sind in einer Onlinedatenbank [Rey02] festgelegt. Zusätzlich wird im IP-Header der *Time to Live* (TTL) mit einem Hop-Zähler von 1 versehen, so dass IGMP Pakete das lokale Netz nicht verlassen können.

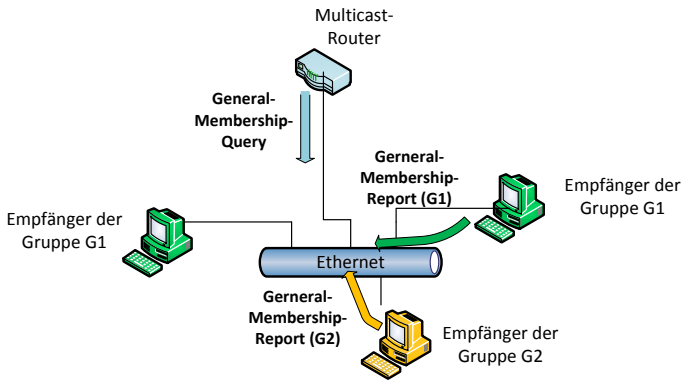

Abbildung 2.2: IGMP-Nachrichten zwischen einem Multicast-Router und Endsystemen, die jeweils zur Empfängergruppe 1 (grün) bzw der Empfängergrupe 2 (orange) angehören.

Der Ablauf ist in Bild 2.2 dargestellt. Ein Multicast-Router sendet periodisch sog. *General-Membership-Queries* an die Gruppe aller Endsysteme in seinen angeschlossenen Netzen. Die Endsysteme antworten nach einer bestimmten Wartezeit in Form eines *General-Membership-Reports*. Diese enthalten Information darüber, zu welchen Gruppen die Endsysteme angehören. Um redundante Rückmeldungen zu verhindern, wird nur geantwortet, sofern kein anderes System bereits zuvor die entsprechende Antwort gab. Der Router kennt daraufhin alle IP-Multicastadressen, dessen Pakete er in das jeweilige Netz ausliefern muss und speichert

sie in einer Tabelle. Ein Zähler zu jedem Eintrag stellt über einen *Soft-State* Mechanismus sicher, dass die Mitgliederinformationen auf dem aktuellen Stand sind. Erhält der Router für einen Eintrag über eine gewisse Zeitspanne hinweg keine *General-Membership-Reports*, läuft der Zähler ab und der Eintrag erlischt. Um die Netzlast gering zu halten, sollten die *General-Memberhship-Queries* in größeren Abständen (Standardwert 125 Sekunden) geschickt werden [Fen97]. Aufgrund dieser großen Zeitspanne ist es für Endsysteme ebenso möglich, explizit einer Multicastgruppe beizutreten, indem ohne vorherige Anfrage, ein *General-Membership-Report* an den Multicastrouter gesendet wird.

Ab IGMPv2 [Fen97] ist auch ein explizites Verlassen eines Systems mit einem *Leave-Report* möglich. Anschließend fragt der Router per *Group-Specific-Query* nach, ob sich weitere Interessenten dieser Gruppe im Subnetz befinden, bevor er den Eintrag löschen kann.

Die derzeit aktuelle Version ist IGMPv3 [CDK+02]. Ab Version 3 können Endsysteme zusätzlich feingranularere Registrierungen beim Router anfordern. So kann man in den Header-Feldern *Source Address* eine Liste von Sender IP-Adressen (Unicast) definieren, von denen man Gruppennachrichten erhalten möchte. Dazu ist eine weitere Nachricht, in Form einer *Group-and-Source-Specific-Query*, nötig, damit der Router explizit nach Mitgliedschaften zu bestimmten Sendern fragen kann.

2.2 Multicast-Routing

IP-Multicast basiert auf der Erweiterung des IP-Protokolls um Gruppenverwaltung und Routingprotokolle [AFM92]. Routingprotokolle sind für die Paketverteilung zu den Zielroutern verantwortlich. Sie haben das Ziel möglichst effiziente Pfade von dem Sender zu den Empfängern zu berechnen und dabei Redundanzen und Schleifen zu verhindern [MSG+12]. Im Fall von Multicast ist das Ergebnis ein Verteilbaum.

Ein Multicast-Routingalgorithmus muss in der Lage sein, auf Änderungen in der Gruppenzusammensetzung reagieren zu können. Nach [Wit99] kann man *inkrementelle* und *monolithische* Algorithmen unterscheiden. Im monolithischen Fall führt eine Änderung zu einer kompletten Neuberechnung des Verteilbaumes während inkrementelle Algorithmen versuchen, den Baum zu modifizieren. Eine Neuberechnung erfordert größeren Rechenaufwand resultiert aber dafür in optimaleren Pfaden. Nach der Berechnung eines neuen Verteilbaumes, wird dieser in Form von Tabelleneinträgen in den beteiligten Router gespeichert, so dass Pakete an eine Multicastgruppe dementsprechend weitergeleitet werden können.

Die Berechnung eines Baumes kann zentral oder verteilt durchgeführt werden. In heutigen Netzwerken werden verteilte Methoden, wie *Distanz-Vektor-Algorithmen* [WDP88a], *Link-State-Algorithmen* [Moy94] oder *Core-Based-Trees* [FHKH06] verwendet. Diese Dezentralität hat den Nachteil, dass die im Netz verteilten Zustände untereinander synchronisiert werden müssen und Routen nicht global-optimal berechnet werden können. Zentrale Routingalgorithmen besitzen wiederum eine komplette Sicht über das Netzwerk und können deshalb global optimierte Pfade bestimmen. Die Schwierigkeiten dabei sind eine konsistente Netzwerksicht herzustellen sowie Leistungsengpässe oder einen *Single-Point-of-Failure* zu Vermeiden [Wit99].

Durch das Aufkommen von SDN ist der Ansatz eines zentralisierten Routings erneut aufgegriffen worden (siehe Abschnitt 2.6). Im Nachfolgenden wird eine kurze Einführung zur Berrechnung von Verteilbäumen gegeben.

2.2.1 Verteilbäume

Ein Spannbaum ist ein Verteilbaum, der alle Knoten in einem Graph (Netz) verbindet. Die Berechnung erfolgt z. B. mit Dijkstras Algorithmus [Dij59]. Die Algorithmen von Prim [Pri57] und Kruskal [Kru56] können verwendet werden, um einen minimalen Spannbaum in einem ungerichteten Graphen zu finden. Im gerichteten Fall gibt es vergleichbare Algorithmen [Edm67] [Tar06]. Das Ergebnis wird als gewurzelter Baum (*Rooted Tree*) bezeichnet. Ein minimaler gewurzelten Baum hat, analog zu einem minimalen Spannbaum, minimales Kantengewicht und enthält, bei Verwendung einer Distanzmetrik, den kürzesten aufsummierten Weg zwischen einer Quelle und den restlichen Knoten im Graph. Somit stellt ein minimaler Spannbaum bzw. ein minimaler gewurzelter Baum ein optimaler Verteilbaum für eine Broadcast-Kommunikation dar. Die Berechnung geschieht dabei in Polynomialzeit. Eine Paketverteilung durch Spannbäume vermeidet unkontrolliertes Fluten und bietet so ein besseres Ausnutzen der Bandbreite [TW12].

Bei einem Multicast sind im Allgemeinen nicht alle Knoten im Netz auch Mitglied der Multicastgruppe. Ein Baum, der nur eine Teilmenge von Knoten verbindet nennt man Multicastbaum. Die Zeitkomplexität um einen optimalen (minimalen) Multicastbaum zu berechnen ist exponentiell [Pro02] und deshalb nicht praktikabel [1].

Ein effizienterer Mechanismus für die Konstruktion eines Multicastbaumes ist das Stutzen (*Pruning*). Ausgehend von einem Broadcastbaum werden Zweige, in denen sich keine Gruppenmitglieder befinden, abgeschnitten. Das Ergebnis ist ein Multicastbaum, der eine Quelle mit den Multicastmitgliedern verbindet. Das heißt, der Baum bietet eine Lösung zur Paketverteilung für einen bestimmten Sender an eine Multicastgruppe und wird als *quell-basierter Verteilbaum* bezeichnet. Diese Methode findet u. a. bei MOSPF [Moy94] und im *Dense-Mode* des *Protocol Independent Multicast* (PIM-DM) [ANS+05] Verwendung. Eine andere Möglichkeit sind Core-Based-Trees [BFC93]. Während bei einem quell-basiertem Verteilbaum ein Baum für jeden Sender pro Multicastgruppe berechnet werden muss, teilen sich hier alle Sender einen zentralen Baum. Dieser kann mehrere Wurzelknoten besitzen, ausgehend von denen der Baum aufgebaut wird. Existiert nur eine Wurzel, handelt es sich um einen *Shared-Tree*. Die Nutzung eines *Shared-Tree* spart, gegenüber quellbasierten-Bäumen, Rechenzeit und reduziert die Anzahl der benötigten Routingeinträge. Diese Technik kommt z. B. im *Sparse-Mode* des *Protocol Independent Multicast* (PIM-SM) [FHKH06] zum Einsatz.

2.2.2 Metriken

Für die Berechnung des Multicastbaumes können beliebige Routing-Metriken herangezogen werden. Eine einfache Metrik ist die Anzahl der Übertragungsschritte, die als Distanz dient.

[1] Dies gillt unter der Annahme $P \neq NP$ [Coo06]

Weitere mögliche Metriken können die Verzögerungszeit, Auslastung, Bandbreite oder Verlässlichkeit des Pfades sein.

Normalerweise werden den Kanten im Netzgraph Gewichtswerte zugeordnet. Diese können statisch oder veränderlich sein. Abhängig von der Metrik werden die Werte im dynamischen Fall an die aktuelle Netzsituation angepasst, während im statischen Fall konstante Gewichte vergeben werden. Metriken können dabei additiv, multiplikativ oder min/max sein [Kui02]. Diese Gliederung legt fest, wie der Gewichtswert eines Pfades entlang der Kantenstücke berechnet wird. Ein Beispiel für eine multiplikative Metrik ist der Paketverlust, die Pfadverzögerung hingegen würde sich additiv akkumulieren. Die verfügbare Bandbreite wäre ein Beispiel für eine einfache min/max Metrik. Ist eine einzige Metrik für die betrachtete Anwendung nicht ausreichend, können auch mehrere Faktoren gleichzeitig betrachtet betrachtet werden. Dies wird als Multimetrik bezeichnet und wird in Abschnitt 2.3.2 beschrieben.

2.3 Global optimierte Wegfindung

SDN stellt Informationen zum Netzzustand an zentraler Stelle bereit. Das ermöglicht die Verwendung von zentralisierten Routingalgorithmen wie Steinerbäume. Vor allem in leistungsfähigen Rechenzentren ist es sinnvoll, weitere Routing-Optimierungen wie Latenz, Verlässlichkeit, Durchsatz oder Energieverbrauch zu betrachten [CHZ+11]. Zusätzlich können Multicastanwendungen Anforderungen bezüglich der Dienstgüte, bzw. *Quality of Service* (QoS) stellen. Ein Ziel kann hierbei sein, die Gesamtverzögerung der Pfade zu minimieren. Ein realistisches Beispiel wäre eine Videokonferenz, bei der alle Gruppenmitglieder das Bild möglichst verzögerungsfrei bekommen sollen. Andere QoS-Anforderungen betreffen maximale Verzögerungsschwankungen oder die minimale Bandbreite [BO10b]. Eine Optimierung des Routings bezüglich solcher Faktoren nennt man *QoS-Routing*.

Der Prozess, mit dem eine bestimmte Dienstgüte bereitgestellt wird, ist das *Traffic Engineering* (TE). *Traffic Engineering* beschäftigt sich mit der Analyse, Gestaltung und Optimierung von Datenflüssen und dem globalen Ziel, die Netzwerknutzung zu optimieren [Wik]. Ein Beispiel dafür ist die gleichmäßige Verteilung der Leitungsauslastungen (*Load Balancing*) [BO10b]. Das *Point-to-Multipoint Multiprotocol Label Switching* (P2MP MPLS) [Yas06] ist eine Technik, um *Traffic Engineering* für Multicastanwendungen in paketvermittelten Netzwerken umzusetzen. Verwandte Arbeiten machen sich die durch SDN neugewonnene Zentralität zu Nutze, um TE-Anwendungen zu implementieren [KWE+11] [MYLSP07]. Bisher aufwändige TE-Aufgaben werden dadurch stark vereinfacht, was sich in reduzierter Komplexität und vereinfachtem Deployment auswirkt.

Die nachfolgenden Teilkapitel beleuchten einen theoretischen Ansatz für die globale Optimierung der Wegfindung. Dafür wird das Multicast-Routing verallgemeinert und formalisiert. Das Ziel ist es, die Komplexität für optimierte zentralisierte Routingalgorithmen aufzuführen, die in dieser Arbeit diskutiert und implementiert werden. Das im Anschluss definierte Steinerbaumproblem bildet hierfür die Grundlage.

2.3.1 Das Steinerbaumproblem

Ein Steinerbaum S_T ist ein Verteilbaum, der eine Teilmenge $T \subseteq V$ aller Knoten eines Graphen $G(V, E)$ umspannt. Dabei wird mit V die Menge der Knoten und mit E die Menge der Kanten bezeichnet. T ist die Menge der Terminalknoten, $V \setminus T$ definiert die Menge der möglichen Steinerknoten. Ein Steinerbaum kann beliebig viele Steinerknoten enthalten, muss aber nicht. Ein Spannbaum ist ein Spezialfall davon, der alle Knoten des Graphen umfasst. Abbildung 2.3 zeigt einen Graphen und einen zugehörigen Steiner- und Spannbaum im Vergleich.

Ein minimaler Steinerbaum ist ein Multicastbaum, der die Kosten der Kanten in S_T global minimiert. Das Finden eines minimalen Steinerbaumes für einen gegebenen Graph G wird *Steinerbaumproblem* genannt. Es existieren zahlreiche Heuristiken, mit denen man in einem zentral verwalteten Netzwerk bestimmte Lösungsapproximationen erzielen kann. Eine davon wird in Abschnitt 2.3.1 betrachtet. Das Finden einer exakten Lösung ist nicht praktikabel, da für das Problem die NP-Vollständigkeit bewiesen wurde [Pro02] [1]. Eine Instanz des Steinerbaumproblems wird mit $(G(V, E, w), K)$ bezeichnet. Im Anschluss wird wird das Problem folgendermaßen definiert:

Gegeben ist ein zusammenhängender, ungerichteter Graph $G(V, E, w)$, mit Kantengewichten $w(e) \in \mathbb{R}$, $e \in E$, ein Sendeknoten $s \in V$ und eine Menge von Empfängern $T \subseteq V$. Es ist ein Steinerbaum S_K gesucht, der $K := T \cup \{s\}$ umspannt und dabei die Kosten bezüglich w minimiert:

$$w(K) \; = \; min \, \{w(K^*) |\; K^* \; ist \; ein \; Steinerbaum \; der \; K \; umspannt\} \tag{2.1}$$

Ändert sich die Terminalmenge T, ändert sich auch der minimale Steinerbaum. Somit handelt es sich bei einem Steinerbaum um einen monolithischen Ansatz. Es sind keine lokalinkrementellen Modifikationen möglich, ohne dabei die Optimalitätseigenschaft zu verlieren [Wit99]. Das minimale Steinerbaumproblem kann direkt auf das Routing per Multicastbaum abgebildet werden. Die Terminalmenge T entspricht den Ziel-Routern einer Multicastgruppe, während der Router des Senders mit s gegeben ist. Weiterhin entsprechen die Steinerknoten den Routern, die zwischen der Quelle und den Empfängern liegen.

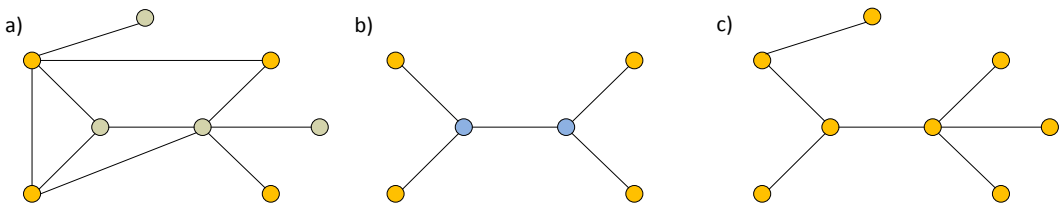

Abbildung 2.3: Teilbild a) zeigt einen Graphen G mit 4 Terminalknoten (gelb), b) ein minimaler Steinerbaum zu G mit 2 Steinerknoten (blau) und c) ein Spannbaum zu G. Die Restknoten (grau) sind in Teilbild b) aus Übersichtsgründen weggelassen worden.

[1] Dies gillt unter der Annahme P $\neq$ NP [Coo06]

Es sei angemerkt, dass in der ursprünglichen Problemformulierung die Steinerknoten nicht fest vorgegeben sind, sondern beliebig eingefügt werden können. Außerdem wird nicht zwischen Senderknoten und Terminalen unterschieden. Das stellt jedoch keine Einschränkung dar, da ein Senderknoten lediglich ein weiteres Terminal ist.

Steinerbaum-packing-Problem Eine Generalisierung des Steinerproblems ist das *Steinerbaum-packing-Problem* [Kos10]. Es beschreibt das Routing mehrerer Steinerbäume in einem Netzwerk, sodass der Netzwerkfluss nach einer bestimmten Bedingung global optimiert wird. Beispielsweise für eine maximale oder optimale Bandbreitenausnutzung [Kos10], oder eine möglichst fairen Verteilung der Datenflüsse nach dem *Min-Max Prinzip* [NNDKB08] [KRT99]. So gibt es in Datencenternetzen häufig redundante Verbindungsleitungen gleicher Länge, die durch Lastverteilung optimal ausgenutzt werden können (*Multipathing*). Eine hinreichende Lösung für solche Probleme kann aber auch durch die Wahl der Routingmetrik gefunden werden. Ein Routingalgorithmus, der Steinerbäume berechnet und als Metrik die Restbandbreite mit einbezieht, wird für neue Verbindungen automatisch eine weniger stark ausgelastete Kante bevorzugen.

Heuristiken Gegeben ist eine Instanz des Steinerbaumproblems $(G(V, E, w), K)$. Ein einfacher Algorithmus listet alle Teilmengen von E auf. Danch prüft er, ob sie K umspannen und dabei einen Steinerbaum bilden. Das Ergebnis mit dem kleinsten Kantengewicht ist eine exakte Lösung für das Steinerbaumproblem. Ein solcher Algorithmus hat jedoch keine polynomielle Laufzeit. Einen Überblick über weitere exakte Lösungsalgorithmen sowie der Berweis der NP-Vollständigkeit wird in [Kos10] gegeben. Aufgrund der Problemschwere gibt es zahlreiche Heuristiken, die eine optimale Lösung lediglich approximieren. Die durch SDN hinzugewonnene zentrale Sicht in Netzwerken ermöglicht es, einfache Heuristiken für ein Routing zu nutzen, die seither als nicht praktikabel galten.

Ein Beispiel hierfür ist der Algorithmus von Kou et al. [KMB81] (KMB), der auf minimalen Spannbäumen basiert. Die optimale Lösung wird hier 2-approximiert. Das heisst, dass die Kosten des berechneten Steinerbaumes maximal doppelt so hoch sind, wie die des minimalen Steinerbaumes. Auswertungen nach [DL93] belegen, dass in der Praxis der schlechteste Fall aber nur selten auftritt, so dass das Ergebnis im Schnitt nur um 5% vom Optimum abweicht. Die Zeitkomplexität von KMB ist mit $O(|K||V|^2)$ gegeben.

Es gibt derzeit keinen Polynomialzeit-Algorithmus, der dass Problem besser als 2-approximiert (im *worst case*) [Kos10]. Zahlreiche weitere Arbeiten beschäftigten sich lediglich mit der Verbesserung der Zeitkomplexität. Durch Adaption von [KMB81] kann eine leicht über-lineare Laufzeit von $O(|E| + |V|log|V|)$ in der Knotenmenge erreicht werden [Meh88]. Eine ausführliche Übersicht über Steiner-Approximationsalgorithmen findet sich in [Kos10]. Ein Vergleich der verschiedenen Laufzeiten ist in [OP05] gegeben.

Das dynamische Steinerbaumproblem In einem dynamischen Umfeld, in dem sich die Multicastgruppen und die Sender beliebig ändern, müsste bei jeder Änderung eine Neuberechnung des Steinerbaumes stattfinden. Trotz effizienter Heuristiken kann die ständige Neuberechnung ein Effizienzproblem darstellen. Dies führt dazu, dass viele Routingtabellen häufig geändert werden müssen. Ein inkrementeller Algorithmus nimmt lokale Änderungen am Multicastbaum vor und vermeidet so eine Neuberechnung. Im Vergleich zur optimalen Lösung verschlechtert sich das Ergebnis aber nach einer bestimmten Anzahl von Änderungen [PPX98].

Ein einfacher Greedy-Algorithmus verbindet bei einer Änderung den hinzugekommenen Knoten mit der alten Terminalmenge auf dem kürzesten Pfad. Das entspricht dem Knoten aus dem existierenden Spannbaum, der am nächsten zum neuen Knoten liegt. Beim Verlassen eines Knotens wird der Zweig einfach gestutzt (*Pruning*). Imaze und Waxmann [IW91] haben gezeigt, dass das *Competitive Ratio* eines solchen Algorithmus bei $log_2(N)$ liegt. Das *Competitive Ratio* ist das Kostenmaximum aller Änderungsanfragen eines Online-Algorithmus zum Verhältnis der Kosten für einen optimalen Offline-Baum, bei dem die Änderungen schon im Voraus bekannt waren. Online-Algorithmen zum dynamischen Steinerbaumproblem wurden in zahlreichen Ausarbeitungen diskutiert und verfeinert [BC97] [WY93] [Wax88] [PPX98].

2.3.2 QoS und Multimetriken

Ein Routingalgorithmus mit einer Metrik optimiert den Multicastbaum nach genau einem Kriterium. Das Steinerbaumproblem beschreibt in diesem Fall die optimale Lösung [Kui02]. Für das Finden einer möglichst global-optimalen Lösung, die zusätzlich Anforderungen (Zielfunktionen), wie kürzester Weg und maximale Bandbreite, oder Constraints wie eine garantierte minimale Bandbreite, mitbringt, müssen mehrere Metriken mit einbezogen werden.

Beim *Multiobjective Multicast Routing Problem* (MMRP) ist in einem zusammenhängenden, ungerichteten Graphen ein Multicastbaum gesucht, der mehrere Zielfunktionen und Constraints aus dem QoS und *Traffic Engineering* Umfeld erfüllt und optimiert [BO10b]. Dabei können verschiedene Bedingungen im Konflikt zueinander stehen, so dass eine global-optimale Lösung nicht unbedingt jedes Ziel lokal optimiert. Das MMRP leitet sich aus der *Pareto-Optimierung* [Brü07] ab, einem aus der Mathematik und Volkswirtschaftslehre bekanntem Optimierungsproblem. Das Steinerbaumproblem kann als der Spezialfall des MMRP mit nur einer Zielfunktion und keinen Constraints gesehen werden.

Ein einfacher Ansatz für mehrere Zielfunktionen ist eine gewichtete Summe [CB04]. Mehrere Routingmetriken werden normiert, mit entsprechenden Faktoren gewichtet und zu einem kombinierten Wert addiert. Der Vorteil für den Routingalgorithmus ist, dass er nur eine Metrik behandeln muss. Andere Verfahren nutzen einen evolutionären Algorithmus wie SPEA2 [ZLT01], um eine Pareto-optimale Lösung für mehrere Zielfunktionen zu finden [BO10b] [BO10a] [CB04].

Werden zusätzlich Constraints betrachtet, muss eine minimale Lösung gefunden werden und dabei bestimmte lokale Bedingungen eingehalten werden. Kuipers und Mieghem [Kui02] betrachten in ihrem Algorithmus mehrere Constraints mit dem Ziel, die maximale Summe mehrerer Metriken zu minimieren. Zusätzlich dazu müssen eine Reihe von QoS-Anforderungen erfüllt sein.

Kompella et al. [KPP93] wählen den Ansatz, eine vorhandene Steiner Heuristik um Verzögerungs-Constraints zu erweitern. Die Heuristik arbeitet sehr ähnlich wie der KMB-Algorithmus aus Abschnitt 2.3.1.

Der *Constrained Shortest Path Tree Algorithm* (SCPT) aus [CW98] berechnet zwei Bäume und kombiniert sie. Einen mit minimalen Kosten und einen mit den minimalen Pfadverzögerungen. Ein solcher Algorithmus ist für große Netzwerke besser geeignet, während [KPP93] bei Graphen mit weniger Knoten effizienter arbeitet [PZH05].

Weiterhin wird in der Fachliteratur auch das dynamische Steinerbaumproblem (vgl. Abschnitt 2.3.1) im Zusammenhang mit verzögerungskritischen Anwendungen untersucht [RMSRM99].

2.4 Software Defined Networking

Der rapide Anstieg von Speicher und Rechenkapazitäten in modernen Rechenzentren führt zu größeren Anforderungen an das Netzwerk. Die aufwändige, manuelle Netzadministration erschwert zunehmend die Anpassung vorhandener Netze an neue Anforderungen. Auch die steigende Komplexität durch Virtualisierung von Rechenressourcen und dadurch resultierende Skalierbarkeitsprobleme haben die Entwicklung von *Software Defined Networking* motiviert [Fou12].

Software Defined Networking (SDN) ist eine Netzwerkarchitektur, bei der das Netzwerk direkt programmierbar ist und die Kontrollebene von der Datenebene entkoppelt wird (Definition nach [Fou12]). Auf Datenebene geschieht die Weiterleitung des Datenverkehrs. Eine logisch zentralisierte Kontrollebene regelt, wohin dieser fließen soll. In klassischen Netzen hat jedes Gerät sich selbst verwaltet und musste individuell programmiert werden. Bei SDN übernimmt dies ein zentraler Controller, der das gesamte oder einen bestimmten Teil des Netzwerks verwaltet. Die Programmierbarkeit wird erreicht, indem die unterliegende heterogene Infrastruktur von den SDN-Anwendungen abstrahiert und über APIs eine vereinfachte und zentrale Sicht des Netzwerkes zur Verfügung gestellt wird. Mit OpenFlow, siehe Abschnitt 2.5, existiert hierzu ein etabliertes Standardprotokoll, das die Kommunikation zwischen Kontroll- und Datenebene übernimmt. Bisher komplexe Netzwerkaufgaben wie Konfiguration, Verwaltung, Sicherheit oder Optimierungen, wie *Traffic Engineering*, können bei SDN herstellerunabhängig mit einem einfachen Softwareprogramm gelöst werden. Das macht es zu einer dynamisch flexiblen Netzwerkarchitektur, die sich schnell und einfach an die eigenen Anforderungen im Netzwerk anpassen lässt.

In Abschnitt 2.4.1 wird die nachfolgend die allgemeine Architektur von SDN beschrieben. Abschnitt 2.5 stellt anschließend den OpenFlow-Standard vor.

2.4.1 SDN-Architektur

Abbildung 2.4 zeigt die SDN-Architektur nach [Fou12]. Sie besteht aus der Anwendungsschicht, Kontrollschicht und der Datenschicht sowie den zugehörigen APIs.

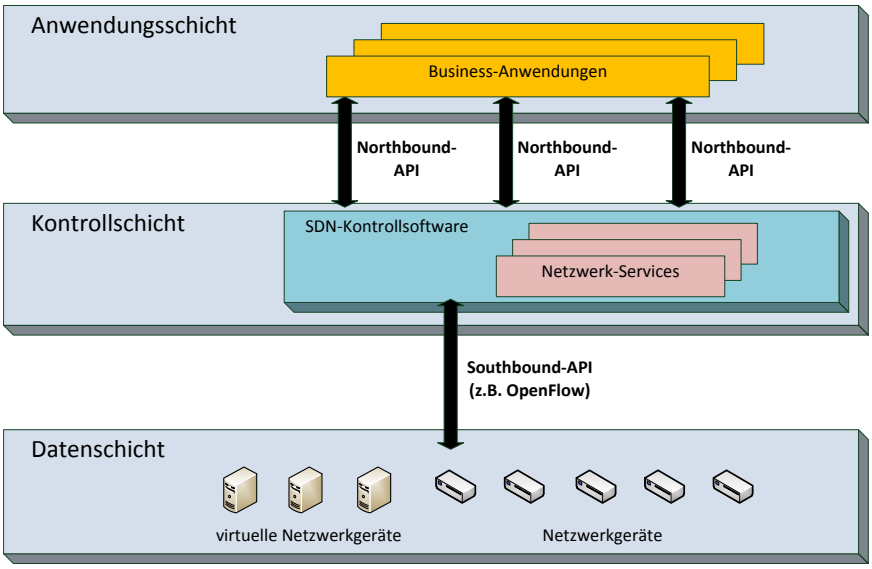

Abbildung 2.4: SDN-Architektur nach [Fou12]

Die unterste Schicht besteht aus den Netzwerkgeräten. Auf Kontrollebene wird kein Unterschied zwischen physikalischen und virtuellen Switches gemacht. Deshalb zählen hier auch virtuelle Geräte zur Datenschicht. Das sog. *Southbound-Protokoll* regelt die Kommunikation zwischen Kontroll- und Datenschicht über die *Southbound-Schnittstelle*. Um SDN zu unterstützen, müssen die teilnehmenden Switches das *Southbound-Protokoll* beherrschen. Hersteller müssen ihre Hardware nicht öffnen, sondern lediglich den Minimalbefehlssatz des *Southbound-Protokolls* unterstützen. OpenFlow (siehe Abschnitt 2.5) hat sich hierfür als Quasi-Standard etabliert [Onl]. Die Hauptaufgabe des Southbound-Protokolls ist das Umsetzen eines Grundbefehlsatzes zur Manipulation der Flow-Tables in den Switches, um damit die Komplexität und Heterogenität auf Datenebene transparent zu machen. Ähnlich der Funktionsweise eines Compilers.

Auf der Kontrollschicht findet sich die SDN-Kontrollsoftware, oder auch *SDN-Controller*. Man kann den Controller als Middleware ansehen, der die Geräte der Datenschicht für SDN-Anwendungen abstrahiert. Außerdem können in der Kontrollschicht, Netzwerkdienste über eine Menge von APIs, implementiert werden. Das können beispielsweise Dienste für *Routing*, *Multicast*, *Security* oder *Traffic Engineering* [Fou12] sein. Solche Anwendungen profitieren stark von der zentralen Netzwerksicht und ermöglichen global-optimierte Lösungen. Erste Controller sind bereits auf dem Markt. Unter anderem von Herstellern wie Big Switch Networks [Net], HP [Pac] oder IBM [IBM]. Bekannte Open-Source-Controller sind NOX [GKP+08] und Floodlight [flo].

Auf der Kontrollschicht aufbauend befinden sich, auf der obersten Schicht, die SDN -Anwendungen wie *Firewalls*, *Netzwerk-Monitore* oder *Load Balancer*. Über die *Nothbound-APIs* werden die Netzwerkdienste und eine Netzabstraktion für die Anwendungsebene zur Verfügung gestellt. Im Gegensatz zur *Southbound-API*, hat sich zum Zeitpunkt dieser Diplomarbeit noch kein klarer Standard hierfür herauskristallisiert. Mögliche Gründe dafür könnten

sein, dass verschiedene Anwendungen unterschiedlich Anforderungen an die Granularität der Netzinformationen oder Effizienzvorgaben haben, sowie die Konkurrenz unter den Herstellern, die ihre Produkte auf dem Markt etablieren wollen [Gui].

2.5 OpenFlow

OpenFlow [MAB+08] ist das erste Standard-Interface, das speziell für SDN entwickelt wurde [Fou12]. Ursprünglich wurde es in Stanford und Berkley entwickelt, wird aber mittlerweile von der Open Networking Foundation (ONF) [ONF] vertreten. OpenFlow beschreibt eine *Southbound-API*, die die Kontrollschicht von der Datenschicht trennt und einen effizienten Datenverkehr über heterogene Netzwerkgeräte hinweg unterstützt. Die Kommunikation zwischen Switch und dem OpenFlow-Controller geschieht über das OpenFlow-Protokoll.

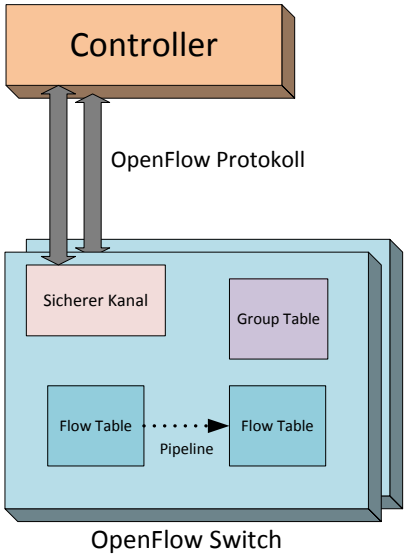

Abbildung 2.5: Spezifikation eines OpenFlow-Switches und dessen Kommunikation über das OpenFlow-Protokoll nach [P+11]

Ingress Port	Metadata	Ether src	Ether dst	Ether type	VLAN id	VLAN priority	MPLS label	MPLS traffic class	IPv4 src	IPv4 dst	IPv4 proto / ARP opcode	IPv4 ToS bits	TCP / UDP / SCTP src port / ICMP Type	TCP / UDP / SCTP dst port / ICMP Type

Abbildung 2.6: Headerfelder in OpenFlow für den Abgleich zwischen Flow-Table und Paketrahmen nach [P+12]

2.5.1 OpenFlow-Switches

In Abbildung 2.5 ist der Aufbau eines OpenFlow-Switches zu sehen. Die Kommunikation mit dem Controller erfolgt über einen sicheren Kanal. Nach der OpenFlow Spezifikation (Version 1.3.0) [P+12] muss ein OpenFlow-Switch eine Gruppentabelle (*Group-Table*) und eine bzw. mehrere Flow-Tabellen (*Flow-Tables*) enthalten.

Ein Eintrag in einer Flow-Table besteht aus 3 Komponenten: *Headerfelder*, *Zähler* und *Aktionen*. Einem ankommenden Paket wird mit Hilfe der Headerfelder eine oder mehrere Aktionen zugeordnet. Es existieren 12 Headerfelder, mit denen auf unterschiedlichen Granularitäten ein Abgleich stattfinden kann. Das kann z. B. der Eingangsport, die Ethernet-Adresse, die IP-Adresse oder der TCP/UDP-Port sein. Ein Vergleich aufgrund feingranularer Faktoren erhöht den Kontroll-Overhead, erreicht dafür aber genauere Routingergebnisse [KSIT11]. Die vollständige Liste aller 12 Headerfelder findet sich in Abbildung 2.6.

Existieren mehrere Flow-Tables, sind diese in einer Pipeline angeordnet. Alle Pakete werden immer zuerst mit Tabelle 0 abgeglichen. Wird ein passender Eintrag gefunden, werden die zugehörigen Aktionen ausgeführt oder diese modifiziert und anschließend erneut abgeglichen. Existiert kein Tabelleneintrag, wird das Paket abhängig von der Tabellenkonfiguration ignoriert, an den Controller gesendet oder entlang der Pipeline im Switch weitergeleitet. Kommt das Paket beim Controller an, muss dieser dafür sorgen, dass ein passender Flow-Eintrag im passenden Switch eingerichtet wird. Jedem dieser Einträge ist eine Menge von Aktionen zugeordnet, die vom Switch verarbeitet werden. Vom Standard vorgeschriebene Aktionen sind *Output*, *Group* und die Möglichkeit ein Paket zu ignorieren (*Drop*). *Output* lässt das Paket über einen bestimmten Port weiterleiten, *Group* lässt das Paket durch einen spezifizierten Gruppeneintrag verarbeiten. Außerdem gibt es zahlreiche optionale Aktionen, die ein Hersteller zusätzlich implementieren kann. Für eine ausführliche Liste aller Aktionen sei auf [P+12] verwiesen.

Zusätzlich sind jeder Flow-Table und jedem darin enthaltenen Eintrag gewisse Zähler zugeordnet. Sie zählen unter Anderem die Anzahl der weitergeleiteten Bytes, die Zeit die der Eintrag existiert oder die Anzahl aller Einträge in einer Flow-Table. Die Zählerinformationen können mit einer OpenFlow-Nachricht abgefragt und für Statistikanwendungen oder für eine dynamische Routingmetrik weiterverwendet werden.

Die Group-Table wurde erst ab der Version 1.1.0 in die OpenFlow-Spezifikation aufgenommen. In der im Moment gängigen Version 1.0 ist lediglich die Flow-Table definiert. Über die Group-Table werden einem Gruppeneintrag mehrere sog. *Action Buckets* zugeordnet, wobei jeder *Bucket* aus einer Menge von Aktionen besteht. Über den Gruppentyp wird die Semantik festgelegt. So bedeutet der Typ *all*, dass ein ankommendes Paket, welches zu der Gruppe passt, geklont wird. Pro Klon wird daraufhin genau ein *Bucket* ausgeführt. So ist es möglich, dass ein Paket an mehrere Ports weitergeleitet werden kann, was die Anwendung von Broad- und Multicast ermöglicht.

Ab Version 1.3.0 ist es unter anderem möglich mit einer weiteren Tabelle (*Meter-Table*) einfache QoS-Operationen zu implementieren. Diese können einem Flow zugeordnet werden. Beispiele sind Beschränkungen in Übertragungsrate oder auch komplexere Dienste wie DivServ.

2.5.2 OpenFlow-Protokoll

Die Kommunikation zwischen dem Switch und dem Controller erfolgt, wie in Abbildung 2.5 angedeutet, über einen sicheren Kanal. Er agiert als Schnittstelle, über die der Controller die Switches verwaltet, Ereignisse bekommt und Pakete zurücksendet. Meist geschieht dies über ein gesondertes Netzwerk (*Out-of-band*). In OpenFlow-Protokoll sind drei Nachrichtentypen definiert: *Controller-to-switch*, *Asynchronous*, und *Symmetric* mit jeweils mehreren Subtypen.

Controller-to-Switch-Nachrichten werden vom Controller genutzt, um Switchkonfigurationen zu ändern oder auszulesen. Das beinhaltet auch die Möglichkeit, eine Flow-Table zu modifizieren (*Flow_Modifikation*-Nachricht) oder ihren aktuellen Zustand zu erfragen. Außerdem kann der Controller ein Paket an einen Switch senden (*Packet_out-Nachricht*).

Der Typ *Asynchronous* definiert Nachrichten, die ein Switch unaufgefordert an den Controller sendet. Darunter fallen Fehler- und Statusmeldungen. Sie können auftreten, falls sich ein Portzustand ändert oder ein Flow-Eintrag durch Timeout oder Löschen-Aktion entfernt wird. Außerdem kann mit Hilfe einer *Packet_in*-Nachricht ein Paket an einen Controller weitergeleitet werden.

Eine Nachricht vom Typ *Symmetric* kann sowohl vom Switch als auch vom Controller initiiert werden. Dabei handelt es sich um *Hello*- oder *Echo*-Anfragen, die für den anfänglichen Verbindungsaufbau wichtig sind. Außerdem können sie zum Testen von Latenz und Bandbreite der Controller-Switch Verbindung genutzt werden.

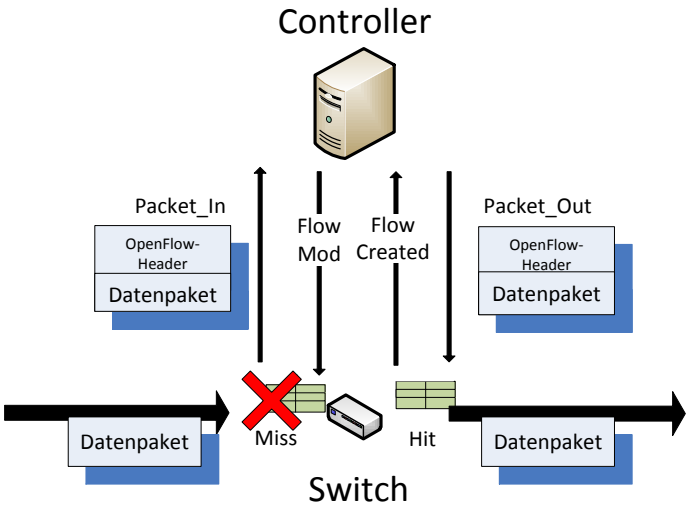

Abbildung 2.7: Beispielkommunikation zwischen Switch und Controller gemäß dem OpenFlow-Protokoll

Abbildung 2.7 zeigt eine Beispielkommunikation zwischen Switch und Controller. Ein Switch erhält ein Paket, für das kein passender Eintrag in den Flow-Tables gefunden wird. Dieser ist so konfiguriert, dass er daraufhin eine *Packet_in*-Nachricht an den Controller schickt. Das Paket wird an die *Packet_in*-Nachricht angehängt und mit einem Grund, hier *Reason = No_Match*, versehen. Außerdem fügt der Switch noch den Port hinzu, auf dem das Paket ursprünglich eingegangen ist. So ist später beim Zurücksenden eine Zuordnung möglich. Der Controller antwortet mit einer *Flow_Modifikation*-Nachricht, woraufhin entsprechende Einträge in der Flow-Table des Switches erzeugt werden. Sobald eine Statusmeldung quittiert, dass der Eintrag erfolgreich erstellt wurde, sendet der Controller das Paket mit einer *Packet_out*-Nachricht wieder zurück. Zuvor kopiert er den Port aus der *Packet_in*-Nachricht. So weiß der Switch, auf welchem seiner Ports das Paket ursprünglich gekommen ist und kann einen Tabellenabgleich durchführen. Nachkommende, gleichartige Pakete werden nun direkt weitergeleitet.

2.6 Verwandte Arbeiten

Verteilte Routingprotokolle für IP-Multicast wie DVMRP [WDP88b] oder MOSPF [Moy94] bauen auf Unicast-Verfahren auf. DVMRP nutzt die Tabelleneinträge des Unicast-Routings, um Distanzen zu anderen Routern zu erhalten. Der Multicastbaum wird durch periodisches Fluten (*Flooding*) und Stutznachrichten (*Pruning*) aufgebaut. Das führt zu einer erhöhten Netzbelastung und reduzierter Skalierbarkeit. *Protocol Independent Multicast* (PIM-DM) [ANS+05] ist nicht von einem Unicast-Protokoll abhängig, nutzt aber ebenfalls das Stutzen eines Broadcastbaumes. *Multicast Open Shortest Path First* (MOSPF) verteilt Gruppeninformationen an alle Router, um eine konsistente Sicht auf die Topologie zu geben. Dabei müssen Gruppenänderungen nicht nur im gesamten Netz geflutet, sondern auch sämtliche Multicastbäume von allen Routern redundant berechnet werden. Der *Sparse-Mode* des *Protocol Independent Multlicast* (PIM-SM) [FHKH06] nutzt wurzelbasierte Bäume (*Core-Based-Trees*) [BFC93]. Zwar wird das Fluten vermieden, dafür führt es zu längeren Wegen als in quellenbasierten Bäumen. Außerdem muss zusätzlicher Aufwand aufgebracht werden, um die Rendezvous-Knoten zu platzieren und zu verwalten.

Im Gegesatz hierzu berechnen zentralisierte Ansätze die Routen an einer oder mehreren zentralen Stellen und verhindern dadurch redundante Berechnungen. Außerdem vermeiden sie zusätzliche Netzlast, die durch den Austausch von verteilten Zustandsinformationen entstehen würde.

Keshav und Paul [KP99] konzentrieren sich auf ein Multicast-Routing im Internet. Die Autoren schlagen vor, für jede Domain eine zentrale Einheit zu haben, die das Routing durchführt. Außerdem existieren sog. *Root-Controller*, die für die Verteilung der Multicastinformationen über Domaingrenzen hinweg zuständig sind, vergleichbar mit DNS. Für die Routenberechnung kann ein quellenbasierter oder ein geteilter Baum verwendet werden.

Marcondes et al. [MSG+12] nutzen OpenFlow für einen zentralisierten Multicastdienst für IPTV-Anwendungen im Internet. Sie berechnen und speichern alle kürzeste Pfade der IPTV-Sender zu allen Endgeräten im Voraus. Dies hat den Vorteil, schnell auf Änderungen in der

Gruppenzusammensetzung reagieren zu können. Kommt ein neuer Empfänger hinzu, wird der Pfad von der Quelle zum neuem Knoten nachgeschlagen, mit dem aktuellen Baum verglichen und an diesen angebunden. Beim Löschen wird der Baum gestutzt. Die Autoren ersetzen außerdem IGMP als Gruppenverwaltungsprotokoll. Dies hat eine Änderung in den Endgeräten zur Folge, was bedeutet, dass der Ansatz nicht ohne weiteres in bestehende IP-Netze integriert werden kann. In dieser Diplomarbeit hingegen wird IGMP beibehalten, um eine möglichst einfache Integration in vorhandene Datencenternetze zu ermöglichen. Dabei wird von einer eher konstanten Anzahl von Sendern und Emfpängern in einem beschränkten Umfeld ausgegangen, die eine im Vergleich geringerer Gruppendynamik ausweisen. Der Fokus liegt hier auf der optimalen Ausnutzung der Netzressourcen und effizienten Berechnung von optimierten Multicastbäumen, während für eine IPTV-Anwendung im Internet das häufige Umschalten der Kanäle (*Zapping*) und die damit verbundene Gruppendynamik im Vordergrund steht.

In [KSS12] präsentieren Kotani et al. ebenfalls ein Design für einen OpenFlow Controller, der IP Multicast implementiert. Im Vordergrund steht das Reduzieren von Paketverlusten durch Switch-Ausfälle, hervorgerufen durch Fehler oder Wartungsarbeiten. Dazu werden für jede Multicastgruppe zwei Verteilbäume vorberechnet, die man bei Bedarf umschalten kann. Jeder Redundanzbaum hat eine ID, die vom Sender-Switch in den Paket Header geschrieben wird. Soll auf einen anderen Baum umgeschaltet werden, muss lediglich der Flow Eintrag im Sender-Switch ersetzt werden. Die anderen Switches haben beide Bäume bereits vorinstalliert und machen nur einen Abgleich auf die ID. Der Nachteil dieses Ansatzes ist, dass man doppelt so viele Tabelleneinträge vorhalten muss. Zur Steigerung der Effizienz werden auch hier aktuelle Bäume zwischengespeichert und bei Gruppenänderungen lediglich modifiziert. Im Gegensatz dazu, konzentriert sich der Ansatz in dieser Diplomarbeit darauf, möglichst optimale Multicastbäume im Datencenter nutzen zu können. Während der grundlegende Ansatz in [KSS12] auf der Redundanz und der inkrementellen Modifizierung von mehreren Multicastbäumen beruht, wird bei einem Ausfall in dieser Arbeit die Neuberechnung von quellenbasierten Bäumen oder Shared-Trees bevorzugt und so kein zusätzlicher Tabellenplatz benötigt.

Dürr [Dür12] präsentiert ein Netzwerkparadigma mit dem Namen Cloud-assisted SDN (CaSDN), das die Vorteile des Cloud Computings mit der SDN Architektur kombiniert. Am Beispiel einer OpenFlow-basierten Systemarchitektur für einen Multicastdienst wird aufgezeigt, wie die komplexen Verwaltungsfunktionen von der Rechenleistung und Speicherkapazitäten in der Cloud profitieren können. Controllerfunktionen wie die Routenberechnung oder Gruppenverwaltung können ausgelagert werden und die Datenschicht von einem Rechenzentrum aus beeinflussen.

Das optimale Routing-Ergebnis eines Multicast wird durch einen minimalen Steinerbaum beschrieben [Pro02]. Heuristiken wie [KMB81] sind in der Lage, ein gutes Approximationsergebnis in Polynomialzeit zu berechnen. In [KPP93] und [CW98] werden zusätzlich Verzögerungs-Constraints betrachtet.

In dieser Diplomarbeit wird ein OpenFlow-basierter IP-Multicast-Dienst für Datenzentren konzipiert und implementiert. Die angestrebten Ziele sind eine einfache Integration in vorhandene OpenFlow-fähige Netze, effizientes und möglichst global-optimales Routing sowie eine gut skalierbare Netz- und Gruppenverwaltung. Dabei soll die Infrastruktur in modernen Rechencenternetzen nach Möglichkeit optimal ausgenutzt werden.

3 Anforderungen und Systemmodell

In diesem Kapitel wird in Abschnitt 3.1 die Problemstellung präzisiert und danach in Abschnitt 3.2 der Systemaufbau besprochen. Im Anschluss werden in Abschnitt 3.3 die Anforderungen eines Multicastdienstes im Datencenter dargestellt.

3.1 Problemstellung

Der rapide Anstieg von Speicher und Rechenkapazitäten in modernen Datencentern führt auch zu höheren Anforderungen an die Netzwerke. Heutige Anwendungen greifen auf verschiedene Datenbanken und Server, die auf unterschiedlichen Maschinen liegen, zu und aggregieren daraus ein Ergebnis. Darüber hinaus werden immer größere Datenmengen parallel verarbeitet. Dies macht eine effiziente Multicast-Kommunikation in Datenzentren unabdingbar, um den bestehenden Kommunikations-Overhead zu reduzieren.

Die aufwändige Netzadministration erschwert jedoch zunehmend die Anpassung vorhandener Netze an neue Anforderungen. Durch Virtualisierung muss eine immer größer werdende Anzahl von Rechnersystemen verwaltet werden. Dazu kommt die höhere Dynamik durch *Live-Migration*. Virtuelle Maschinen können beliebig zwischen verschiedenen Hostrechnern wechseln, was hohe Ansprüche an eine Netzwerkverwaltung stellt. Die heutigen statischen Netzwerke stehen im Gegensatz zu diesen Anforderungen. Sie können nicht dynamisch auf Ausfälle, Datenverkehrsänderungen oder auf Bedürfnisse der Benutzer bzw. der Anwendungen eingehen. Durch die fehlende Zentralität kann die vorhandene Infrastruktur nicht optimal ausgenutzt werden und es muss sich mit weniger optimalen Routingergebnissen zufrieden gegeben werden. SDN reduziert die Komplexität durch klare Trennung zwischen Daten- und Kontrollschicht und stellt eine globale Netzwerksicht zur Verfügung. Dies bietet die nötige Flexibilität um den dynamischen Anforderungen gerecht zu werden.

In dieser Diplomarbeit wird ein OpenFlow-basierter IP-Multicast-Dienst konzipiert, um die Vorteile von SDN in moderenen Datencentern für Multicast-Anwendungen verfügbar zu machen (Kapitel 4). Die Ziele dabei sind: Eine einfache Integration in vorhandene OpenFlow-fähige Netze, effizientes und möglichst global-optimales Routing sowie eine gut skalierbare Gruppen- und Netzverwaltung. Dabei soll die Infrastruktur eines Datencenternetzes nach Möglichkeit optimal ausgenutzt werden. Eine Implementierung, die diesen Anforderungen gerecht wird, wird in Kapitel 5 beschrieben und im Anschluss auf einer Datencenter-Topologie, in Kapitel 6, evaluiert.

3.2 Systemmodell

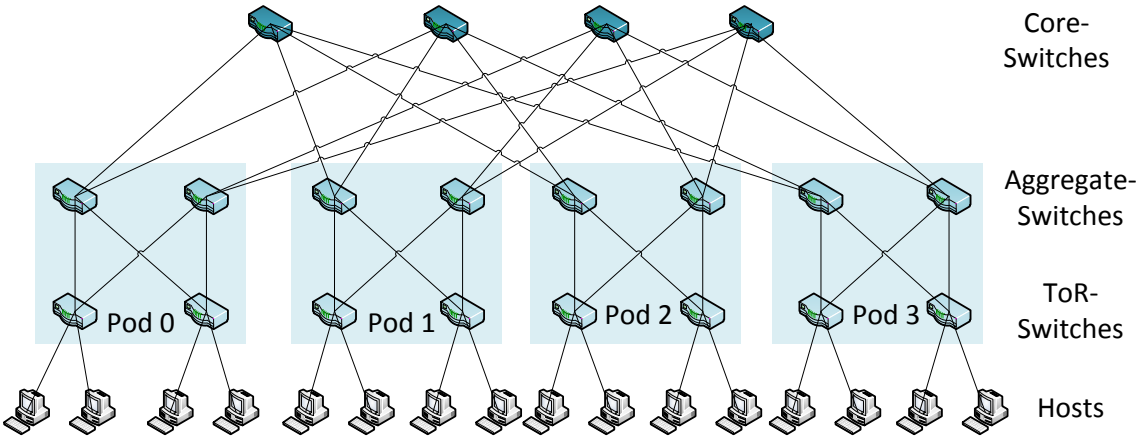

Abbildung 3.1: Fat-Tree Topologie nach [AFLV08] für $k = 4$

Typische Datencenternetzwerke weisen eine Baumstruktur, einen sog. *Fat-Tree* [AFLV08] auf. Die Fat-Tree Topologie ist eine spezielle Instanz eines Clos-Netzwerkes [Clo53], die aus handelsüblichen Switches zusammengesetzt ist (Abbildung 3.1). Sie sollte nicht mit einem Baum nach [Lei85] verwechselt werden, dessen Grundidee in [AFLV08] aufgegriffen wurde und nun ebenfalls Fat-Tree genannt wird. Ein Fat-Tree, im Sinne von [AFLV08], weist eine gute Konnektivität auf und hat das Ziel die Effizienz, Skalierbarkeit und Fehlertoleranz von Datencenternetzwerken zu erhöhen [BBAL+11]. Außerdem stellt diese Topologie einen Kostenvorteil dar, da gängige Standardhardware verbaut werden kann. Ein Beispiel für ein Netzwerkschema, das einen Fat-Tree nutzt, ist *Portland* [NMPF+09]. Das Switching geschieht dort mit Hilfe von Pseudo-MAC Adressen, die jedem Host zugeordnet werden. In diesen Adressen ist die Position des Hosts im Baum codiert, was beim Routing ausgenutzt werden kann. Eine ausführliche Übersicht über weitere Netzwerktopologien in Datenzentren findet sich in [CHZ+11].

Ein Fat-Tree ist durch einen Wert k bestimmt. Er gibt die Anzahl der *Pods* und die Portanzahl der Switches an. Auf der untersten Ebene finden sich jeweils $k/2$ physikalische oder virtuelle Hostmaschinen. Sie sind über die *Top-of-Rack-Switches* (ToR-Switches) miteinander verbunden. Die restlichen $k/2$ Ports verknüpfen die ToR-Switches mit den darüberlegenden *Aggregate-Switches* zu einem Verbund, der als *Pod* bezeichnet wird. Die *Aggregate-Switches* wiederum verbinden die *Pods* auf der höchsten Ebene mit allen $(k/2)^2$ *Core-Switches*. Bild 3.1 zeigt einen Fat-Tree für $k = 4$.

Zum Testen und Evaluieren dieser Diplomarbeit wird beispielhaft eine Hälfte der Fat-Tree Topologie aus Abbildung 3.1 verwendet. Allerdings ist das beschriebene Verfahren grundsätzlich unabhängig von der Topologie. Als Switches werden durchgehend OpenFlow-fähige Multilayer-Switches eingesetzt. Das sind Geräte die sowohl auf der Sicherungsschicht, als auch als Router in der Vermittlungsschicht agieren können. Weiterhin wird ein IP-Netz zu Grunde gelegt, mit dem Ziel, IP-Multicast umzusetzen. Dazu gillt die Annahme, dass die Hostgeräte standardkonform zu IP-Multicast das *Internet Group Management Protocol* [CDK+02] (IGMP) nutzen, um ihre Gruppenmitgliedschaften bekannt zu machen.

Somit ist keine Modifikation der Hostgeräte vorgesehen. Die Switches selbst implementieren OpenFlow und werden von einem zentralen Controller verwaltet.

Wichtige Bewertungskriterien in einem Netzwerk sind *Grad*, *Durchmesser*, *Kantenkonnektivität* und die *Bisektionsbandbreite* [TMJ12]. Der Grad jedes inneren Knoten in einem Fat-Tree ist k. Die längste Distanz zwischen zwei Knoten im Netz wird als Durchmesser bezeichnet. In Bild 3.1 beträgt der Durchmesser sechs Längeneinheiten. Dagegen gibt die Kantenkonnektivität ein Maß für die vorhandenen Leitungsredundanzen an. Sie ist definiert durch die minimale Anzahl von Kanten, die entfernt werden müssen, um das Netz zu unterbrechen. Im Beispiel 3.1 beträgt dieser Wert zwei Längeneinheiten und wird durch den Fall bestimmt, wenn genau ein ToR-Switch abgetrennt wird. Die Bisektionsbandbreite wiederum ist ein Maß für die *worst-case* Kapazität in einem Netz. Je höher die Bisektionsbandbreite, desto seltener treten Blockierungen auf. Sie wird durch die minimale Anzahl von Kanten bestimmt, die entfernt werden müssen, um das Netz in zwei gleichgroße Teilnetze zu zerlegen. Die Summe der Bandbreiten aller geschnittenen Verbindungsleitungen ist die Bisektionsbandbreite. Ein Fat-Tree ist deshalb für Datencenternetze eine geeignete Wahl, da die volle Bisektionsbandbreite erreicht werden kann [AFLV08]. Das hat zur Folge, dass Knoten paarweise zwischen den Hälften mit voller Geschwindigkeit kommunizieren können.

Zur Optimierung des Routings sollten die Mehrfachpfade eines Fat-Trees ausgenutzt werden, um die verfügbare Bandbreite gleichmäßig verteilen zu können (*Multipathing*). Die Aufteilung kann pro Verbindung stattfinden, d.h. pro Multicastbaum, oder auf Paketbasis. Letzteres würde bedeuten, dass die Multicastpakete von einem Sender zu der gleichen Gruppe über mehrere redundante Bäume geteilt werden. Es gibt mehrere Methoden, die in Rechenzentren für ein *Multipathing* zum Einsatz kommen können. Standardverfahren wie ECMP [TH00] basieren auf einfachem Hashing, haben aber den Nachteil dass die Netzlast nicht betrachtet wird. In [AFRR$^+$10] übernimmt ein zentraler Scheduler mit globalem Wissen die Platzierung der Flows. In dieser Diplomarbeit wiederum wird OpenFlow verwendet, um die Bandbreite beim Routing mit einzubeziehen.

3.3 Anforderungen an einen OpenFlow-basierten Multicast-Dienst im Datencenter

Die Hauptaufgaben des Multicast-Dienstes bestehen aus *Gruppenverwaltung*, *Routenverwaltung*, Bereitstellen einer *Netzrepräsentation* und der *Routenberechnung*. Daraus, und aus der Problemstellung in Abschnitt 3.1, lassen sich folgende Anforderungen für einen OpenFlow-basierten Multicastdienst in einem Datencenter ableiten:

Globale Netzrepräsentation Der OpenFlow-Controller muss stets den aktuellen Zustand des Netzwerkes bereitstellen. Dies beinhaltet neben der Netztopologie auch Statistiken, um adaptives Routing aufgrund von aktuellen Netzzuständen zu ermöglichen. Dazu muss eine Netzwerkrepräsentation als kantengewichteter Graph erstellt werden. Wichtig ist, dass ein effizienter Zugriff auf diese Informationen erfolgen kann.

Skalierbarkeit Ein wichtiges Designkriterium für einen Multicastdienst ist die Skalierbarkeit in Gruppengröße und Gruppenanzahl [Wit99]. Auf der einen Seite betrifft das den Controller, der innerhalb kürzester Zeit auf Gruppenänderungen reagieren muss, um keinen Flaschenhals zu bilden. Außerdem muss er in der Lage sein, die Gruppen effektiv zu verwalten und einen schnellen Datenzugriff ermöglichen. Auf der anderen Seite stehen die Beschränkungen bezüglich der Tabellengröße in den Switches. Ein drohender Überlauf in einer Flow-Table sollte erkannt und darauf entsprechend reagiert werden. Dies kann durch Umstellen des Routingprotokolls, Zusammenfassen mehrerer Tabelleneinträge (*Flow Aggregation*) [MSG+12] oder Umleitung des Verkehrs über andere Router erfolgen. Zusätzlich könnte der Controller verteilt implementiert werden. Das verhindert einen *Single-Point-of-Failure* und erhöht die Skalierbarkeit. Außerdem sollten in den Hostrechnern ausreichend Rechen- und Speicherressourcen zur Verfügung stehen.

Global-optimales Routing Die Rechenressourcen eines Datencenters und die Zentralität von SDN erlauben es, komplexere Routingalgorithmen zu verwenden. Ziel ist es, einen möglichst minimalen Multicastbaum in polynomieller Zeit zu berechnen. Dieser sollte mehrere Kriterien gleichzeitig erfüllen können und eine möglichst optimale Lösung finden. *Traffic Engineering*, nach Kapitel 2.3, ist für eine bessere Ausnutzung redundanter Pfade in der Netzwerkstruktur aus Abschnitt 3.2 nötig. Ein Mechanismus dafür ist *Flow Balancing*, bei dem der Datenverkehr bevorzugt über weniger ausgelastete Verbindungsstücke geroutet wird. Das hat zum Ziel, eine möglichst optimale Ausnutzung der Bandbreite zu erreichen [Kos10].

Effizienz Routenberechnung, Flow-Updates, Verwaltung und Weiterleitung sollten effizient ablaufen und ungenutzte Ressourcen, wie innaktive Flow-Table Einträge explizit freigemacht werden. Dazu müssen die installierten Routen im Controller gespeichert werden, um bei Änderungen ein zeitaufwändiges Anfragen in der Datenschicht zu vermeiden. Die Zeit für Routenberechnung und Flow-Updates, kann durch proaktives Vorinstallieren der benötigten Datenpfade reduziert werden. Das Ziel ist, auf Datenebene eine unterbrechungsfreie Weiterleitung in *Line-Rate* zu ermöglichen. Alternativ kann ein reaktives Verfahren zur Flow-Erstellung genutzt werden. Dabei sollten Gruppenänderungen aber nur zu inkrementellen Änderungen an den Tabelleneinträgen führen, damit eine komplette Neuberechnung verhindert wird.

Robustheit Durch das adaptive Umleiten des Datenverkehrs auf redundante Pfade werden Paketverluste aufgrund von Leitungs- und Switch-Ausfällen reduziert. Diese Art von Topologieänderungen muss am Controller registriert und in die aktuelle Routenberechnung mit einbezogen werden. Ein Weiteres Problem ist, dass ein *Daten-Burst* einen Switch mit Paketen überfluten kann, wenn erst durch den OpenFlow-Controller ein passender Tabelleneintrag eingerichtet werden muss. Dauert dieser Prozess zu lange, läuft die Eingangswarteschlange im Switch voll und Pakete werden verworfen.

Integration Der Multicastdienst soll möglichst einfach in ein vorhandenes OpenFlow-fähiges Netzwerk integrierbar sein. Die Grundvoraussetzung dafür ist, dass die Switches OpenFlow-fähig sein müssen. Zusätzlich muss ein Controller und ein Controllernetzwerk existieren. Jedoch sollen keine Änderungen an den Endsystemen nötig sein. Das hat zur Folge, dass die Kommunikation mit den Hosts standardkonform zu IP-Multicast geschehen muss.

4 Konzeption eines OpenFlow-basierten Multicast-Dienstes

In diesem Kapitel wird die Lösungskonzeption für einen OpenFlow-basierten Multicast-Dienst vorgestellt, die den gestellten Anforderungen aus Kapitel 3.3 gerecht werden soll. Zunächst wird in Abschnitt 4.1 eine Übersicht über die erarbeitete Architektur der Controller-Software gegeben sowie die dazu nötigen Softwareprozesse identifiziert. Anschließend werden die einzelnen Prozesse sowie die zugehörigen Verbindungen und Schnittstellen beschrieben (Abschnitte 4.2 bis 4.7).

4.1 Systemarchitektur

Die allgemeine Systemarchitektur des OpenFlow-basierten Multicast-Dienstes ist in Abbildung 4.1 dargestellt. Die Datenschicht weist eine Datencenter-Topologie auf und sämtliche Switches sind Multilayer- und OpenFlow-fähig. Die Kommunikation zwischen Datenschicht und Controller geschieht über OpenFlow. Die Kantenbeschriftungen geben die Art der Informationen bzw. die Nachrichtentypen an, die intern zwischen den Prozessen ausgetauscht werden. Folgende Controller-Prozesse lassen sich aus Abbildung 4.1 identifizieren:

- **OpenFlow:** Southbound-API, zuständig für die Kommunikation mit der Datenschicht und die Modifikation der Flow-Tables
- **Nachrichtenfilter:** Empfängt alle Nachrichten, die aus der Datenschicht an den Controller gesendet werden und leitet sie abhängig vom Nachrichtentyp an den jeweiligen Prozess weiter.
- **Netzstrukturverwaltung:** Die Aufgaben bestehen aus dem Bereitstellen, Überwachen und Speichern des aktuellen Netzzustandes. Das betrifft die Topologie sowie die Bestimmung von Switch-Informationen wie Portnamen oder IP- und MAC-Adressen. Für die Topologiebestimmung wird das *Link Layer Discovery Protocol* (LLDP) [lld05] verwendet.
- **Netzzustandsverwaltung:** Überwacht den Netzzustand aufgrund von Statistikinformationen aus der Datenschicht. Diese Informationen werden mit der Netztopologie kombiniert und stellen der Routenberechnung eine kantengewichtete Netzrepräsentation zur Verfügung. Außerdem werden Ausfälle, sowie drohende Überläufe in den Flow-Tabellen erkannt und eine Behandlung dieser Probleme in der Routenberechnung initiiert.

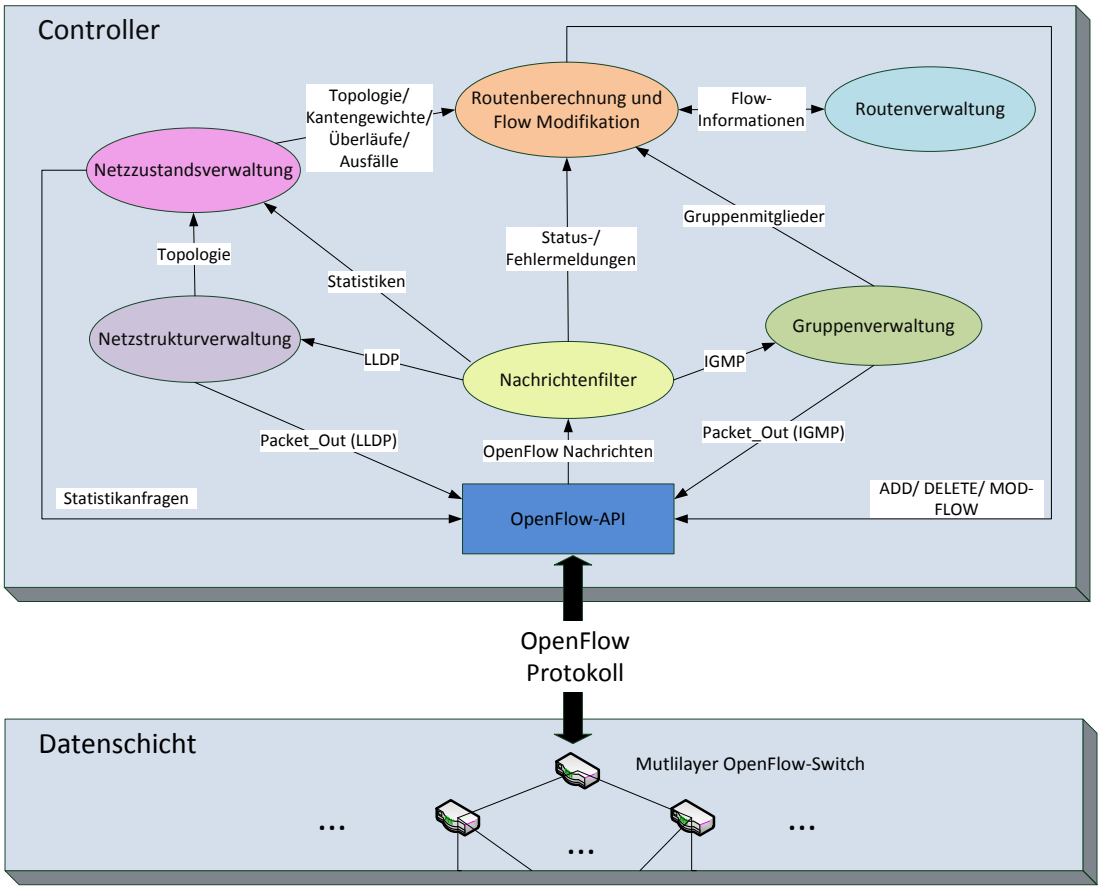

Abbildung 4.1: Systemarchitektur für einen OpenFlow-basierten Multicastdienst

- **Gruppenverwaltung:** Speichert die aktuellen Gruppen und deren Zusammensetzung. Außerdem ist sie dafür zuständig, Gruppenänderungen aus dem Netz anzufragen. Dies geschieht indem das Gruppenverwaltungsprotokoll IGMP implementiert wird.

- **Routenverwaltung:** Hält Informationen über die aktuell implementierten Flows bereit, um bei Gruppenänderungen die nötigen Anpassungen im Netz vornehmen zu können. Nach der Ausführung einer Routenberechnung werden die Daten der Routenverwaltung entsprechend angepasst, sofern die Änderungen durch entsprechende Statusmeldungen von der Datenschicht quittiert wurden und kein Fehler aufgetreten ist.

- **Routenberechnung und Flow Modifikation:** Ist für die Berechnung der Multicastbäume zuständig. Dazu werden Daten aus der Netzzustandsverwaltung, Gruppenverwaltung und der Routenverwaltung verwendet. Treten Änderungen in der Gruppenzusammensetzung oder in der Topologie auf, wird die Routenberechnung von der Gruppenverwaltung bzw. der Netzzustandsverwaltung angestoßen. Mit Hilfe der Informationen aus der Routenverwaltung werden im Anschluss an die Berechnung die nötigen Änderungen über die OpenFlow-API vorgenommen.

Im Nachfolgenden werden, in den Abschnitten 4.2 bis 4.7, die einzelnen Aufgaben der Prozesse aus Abbildung 4.1 sowie die zugehörigen Verbindungen und Schnittstellen ausführlich beschrieben. Diese Beschreibung erfolgt unabhängig von einer konkreten Controller-Implementierung auf OpenFlow-Basis.

4.2 Kommunikation mit der Datenschicht und Nachrichtenfilterung

Die Kommunikation mit der Datenschicht geschieht gemäß dem OpenFlow-Protokoll über die OpenFlow-API. Die vollständige Spezifikation aller OpenFlow-Nachrichtentypen findet sich in [P+12]. Über TCP mit TLS-Verschlüsselung (*Transport Layer Security*) [Die08] wird, durch einen *Handshake* zwischen dem Controller und den OpenFlow-Switches, eine Verbindung über ein externes Controllernetz aufgebaut. Um mit den Switches kommunizieren zu können, werden dementsprechend TCP-Sockets mit ihren Bezeichnern, IP-Adressen und Ports im Controller verwaltet. Da der Controller zu jedem Switch nur jeweils eine TCP-Verbindung aufbaut, können diese eindeutig über ihre IP-Adresse identifiziert werden. Alternativ kann zu diesem Zweck eine Controller-interne Switch-ID zugewiesen werden. Sie erleichtert das Speichern von Gruppen- und Flow-Informationen im Controller. Die OpenFlow Dokumentation [P+12] spezifiziert eine solche ID als 64-Bit Bezeichner, wovon die letzten 48-Bit die MAC-Adresse darstellen. Die oberen 16 Bit sind frei belegbar und hängen so von der jeweiligen Implementierung ab. Wenn im Nachfolgenden vom Senden oder Empfangen einer Nachricht durch den Controller die Rede ist, beinhaltet das implizit auch immer eine Zuordnung der Switch-ID zum jeweiligen TCP-Socket durch IP-Adresse und Port. Im Nachfolgenden wird zur Beschreibung des Datenaustausches gemäß Bild 4.1 zwischen vom Controller ausgehenden und am Controller ankommenden Nachrichten unterschieden.

Ausgehende Nachrichten Es müssen Nachrichtenpakete verschiedener Art zwischen dem Controller und der Datenebene ausgetauscht werden. Die Netzstrukturverwaltung und die Gruppenverwaltung benutzen die OpenFlow-Nachricht *Packet_Out*, um IGMP- oder LLDP-Anfragen an einen Switch zu schicken. Mit einer *Packet_Out*-Nachricht ist es möglich einen beliebigen Ethernet-Rahmen mit einem OpenFlow-Header zu versehen und an die Datenschicht zu schicken. Für die Modifikation von Flow- und Group-Tables schickt die Routenberechnung Änderungsanfragen in Form einer *Flow Table Modification*-Nachricht. Man unterscheidet dabei die Nachrichtentypen *Add*, *Delete* und *Mod* für das Hinzufügen, Löschen oder Ändern von Tabelleneinträgen. Zusätzlich fragt die Netzzustandsverwaltung über eine sog. *Multipart_Request*-Nachricht aktuelle Statistikwerte bezogen auf die Flow-, Table- oder Gruppenzähler eines Switches ab.

Ankommende Nachrichten und Verteilung durch den Nachrichtenfilter Der Nachrichtenfilter aus Abbildung 4.1 ist dafür zuständig, alle am Controller ankommenden Open-Flow Nachrichten entsprechend weiterzuleiten. Dieser erhält über die OpenFlow-API *Packet_In*-, Statistik- sowie Fehler- und Statusnachrichten. Die Zuordnung nach den entsprechenden Headerfeldern ist in Abbildung 4.2 zu sehen.

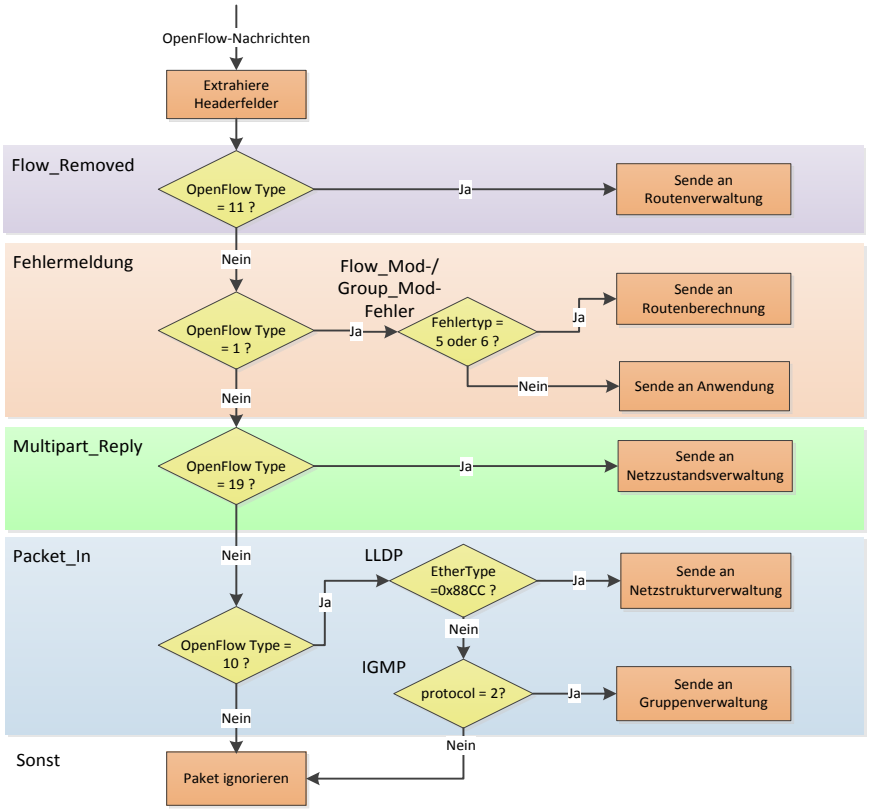

Abbildung 4.2: Flussdiagramm des Nachrichtenfilters

Über die OpenFlow-Nachricht *Packet_In* kann ein beliebiger Switch ein Paket an den Controller weiterleiten. Eine *Packet_In*-Nachricht wird über dem Wert 10 im OpenFlow-Headerfeld *Type* erkannt. Im Nachrichtenfilter wird dann zwischen LLDP- und IGMP-Paketen unterschieden. Am Controller ankommende LLDP-Nachrichten sind Antworten der Switches auf vorhergegangene LLDP-Anfragen, die von der Netzstrukturverwaltung ausgegangen sind. Analog wurden IGMP-Nachrichten von der Gruppenverwaltung gesendet und müssen nun vom Nachrichtenfilter wieder dem anfragenden Prozess zugeordnet werden. Für die Unterscheidung der beiden Nachrichtentypen werden die Headerfelder auf der Sicherungs- und Vermittlungsschicht inspiziert. Ein *EtherType* von $0x88CC$ zeigt einen LLDP-Rahmen an, während ein IP-Header mit einem Nachfolgeprotokollwert von 2 auf eine IGMP Nachricht hinweist (*Protocol*-Header).

Tritt bei der Änderung eines Eintrages in einer Flow-Tabelle ein Fehler auf, antwortet der Switch mit einer Fehlermeldung, was durch einen Wert $type = 1$ im OpenFlow-Header erkannt wird. Fehlermeldungen werden als Antwort auf eine *Mod/Add/Delete*-Nachricht geschickt und müssen deshalb an die Routenberechnung geleitet werden. Der Fehlertyp 5 gibt einen Fehler bei der Flow-Modifikation an, während der Fehlertyp 6 einen Fehler bei der Gruppenmodifikation anzeigt. Für den Multicastdienst nicht direkt relevante Fehlertypen werden durch eine allgemeine Fehlerbehandlung in OpenFlow oder von der Anwendung verarbeitet und sind hier nicht weiter aufgeführt.

Weiterhin werden in Abbildung 4.2 *Flow_Removed*- und *Multipart_Reply*-Nachrichten unterschieden. Eine *Flow_Removed*-Nachricht ist eine Statusnachricht, die das Löschen eines Flows bestätigt. Sie wird durch den Wert 11 im OpenFlow-Headerfeld *type* erkannt. Da der Multicastdienst den *Soft-State*-Mechanismus von OpenFlow für Flow-Table Einträge nicht verwendet, kann eine solche Löschmeldung nur als Antwort auf eine *Delete*- Anweisung gesendet worden sein. Sie wird deshalb an die Routenberechnung weitergeleitet. Die Antwort auf eine vorangegangene Statistikanfrage der Netzzustandsverwaltung geschieht in Form einer *Multipart_Reply*-Nachricht. Der zugehörige OpenFlow *type*-Wert beträgt 19. Sämtliche Nachrichten dieser Art werden zurück zur Netzzustandsverwaltung geleitet. Alle anderen Pakete, die keine der beschriebenen Headerinformationen aufweisen, sind für den Multicastdienst nicht von direkter Bedeutung und werden ignoriert. Die Verarbeitung solcher Pakete erfolgt außerhalb der Sichtweise der beschriebenen Architektur.

4.3 Netzstrukturverwaltung

Um eine Repräsentation der physikalischen Netzstruktur für die Routenberechnung zu erhalten, führt die Netzstrukturverwaltung das *Link Layer Discovery Protocol* (LLDP) [lld05] aus. Außerdem dienen diese Informationen als Grundlage für das Erkennen von Änderungen und Ausfällen durch die Netzzustandsverwaltung. Mit LLDP kann ermittelt werden, welche Netzwerkgeräte mit welchen Nachbargeräten verbunden sind. Außerdem können System- und Portnamen, Netzadressen sowie die Funktionen der einzelnen Geräte bestimmt werden. Aus diesen Informationen wird ein Graph in Form einer Adjazenzliste konstruiert.

In einem herkömmlichen Netzwerk initiieren die Router eine LLDP Kommunikation, indem sie Informationen über sich selbst an ihre Nachbarn versenden. Empfangene Informationen werden in einer *Information-Base* im Router gespeichert und können dann z. B. über das SNMP-Protokoll [CMPS02] abgefragt werden. Im OpenFlow-Netzwerk wird diese Funktionalität von den Switches in den Controller verlagert.

LLDP arbeitet auf der Sicherungsschicht. Ein entsprechender Rahmen wird mit dem *EtherType* 0x88cc versehen und periodisch über eine *Packet_Out*-Nachricht, vom Controller an alle OpenFlow-fähige Switches geschickt. LLDP-Rahmen sollen von den empfangenden Switches über alle Ports weitergeleitet werden, um benachbarte Netzwerkverbindungen zu erkennen. Ein LLDP-Paket muss von einem Switch als solches erkannt und daraufhin eine entsprechende *Output:All* Aktion ausgeführt werden. Der logische *All*-Port steht dabei stellvertretend für alle Ports eines Switches. Handelt es sich bei einem Nachbargerät um einen OpenFlow-Switch, leitet dieser den LLDP-Rahmen über eine *Packet_In*-Nachricht an den Controller zurück. Dies geschieht entweder aufgrund der Switch-Konfiguration, falls kein passender Eintrag in der Flow-Table gefunden wird, oder alternativ, durch eine explizit eingerichtete Aktion zur Weiterleitung an den Controllerport (*Output:Controller*). Im Controller leitet der Nachrichtenfilter den LLDP-Rahmen an die Netzstrukturverwaltung weiter, die dann daraus schließen kann, dass zwischen zwei Switches eine Verbindung besteht.

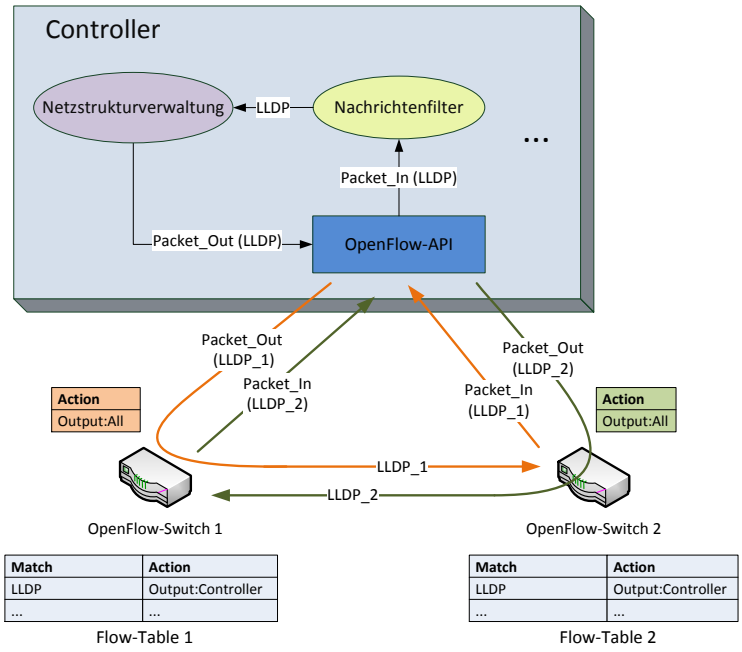

Abbildung 4.3: LLDP-Kommunikation zwischen Netzstrukturverwaltung und zwei benachbarten OpenFlow-Switches

In Abbildung 4.3 ist der beschriebene Informationsaustausch für zwei benachbarte OpenFlow-Switches dargestellt. Außerdem sind die dafür nötigen Aktionen und die Abgleichkriterien auf Datenebene zu sehen. Der Eintrag in der Flow-Table muss vor der eigentliche Kommunikation eingerichtet werden und darf danach nicht mehr verworfen werden. Das heisst es darf insbesondere kein Timeout-Wert gesetzt sein. Die Aktion *Output:All* für ausgehende LLDP-Rahmen wird direkt in der *Packet_Out*-Nachricht definiert. Nur Pakete, die im Zuge dieser *Packet_Out*-Nachricht vom Controller kommen, werden über diese Aktion behandelt. Der Eintrag in der Flow-Table des Switches gleicht, unabhängig vom Eingangsport, nach dem LLDP *EtherType* ab. Das heißt sämtliche LLDP-Rahmen werden vom empfangenden Switch direkt an den Controller weitergegeben.

Wichtig ist, dass ein Switch einen LLDP-Rahmen, den er von einem Nachbarn erhalten hat, nicht auf der Datenschicht weitersendet. Die LLDP Spezifikation gibt als spezielle Empfängeradresse für LLDP-Rahmen die Multicastadresse „01:80:c2:00:00:0e“ vor. Rahmen mit dieser Adresse dürfen von einem Switch nicht weitergeleitet werden, so dass nur Informationen zwischen Nachbarn ausgetauscht werden. In dem Fall, dass zwischen zwei OpenFlow-Switches keine Direktverbindung besteht, sondern ein zusätzlicher, nicht OpenFlow-fähiger Switch dazwischengeschaltet ist, führt das zu einem Problem. Der Switch würde den LLDP-Rahmen konsumieren und nicht beim nächsten OpenFlow-Switch ankommen. Als Alternative kann man auf die Broadcastadresse „ff:ff:ff:ff:ff:ff“ zurückgreifen und manuell durch einen extra Eintrag in der Flow-Table sicherstellen, dass der nächste OpenFlow-Switch den Rahmen nicht mehr weiterleitet.

4.4 Netzzustandsverwaltung

Die Netzzustandsverwaltung ist dafür zuständig, den aktuellen Zustand der Switches und der Verbindungen auf Datenebene zu ermitteln. Dazu gehören die verfügbare Restbandbreite aller Leitungen und die Überwachung der Flow-Tables. Die Restbandbreite wird für die adaptive Wegberechnung in der Routenberechnung benötigt. Die Netzzustandsverwaltung errechnet für die gegebene Netztopologie eine gewichtete Summe. Diese besteht aus den Distanzen, den Leitungskapazitäten und der verfügbaren Restbandbreite. Das Ergebnis ist ein zusammengesetztes Kantengewicht, dass die Leitungsauslastung im Datencenter in die Routenberechnung miteinbezieht. Die Informationen über die Topologie, Distanzen und Kapazitäten stammen von der Netzstrukturverwaltung.

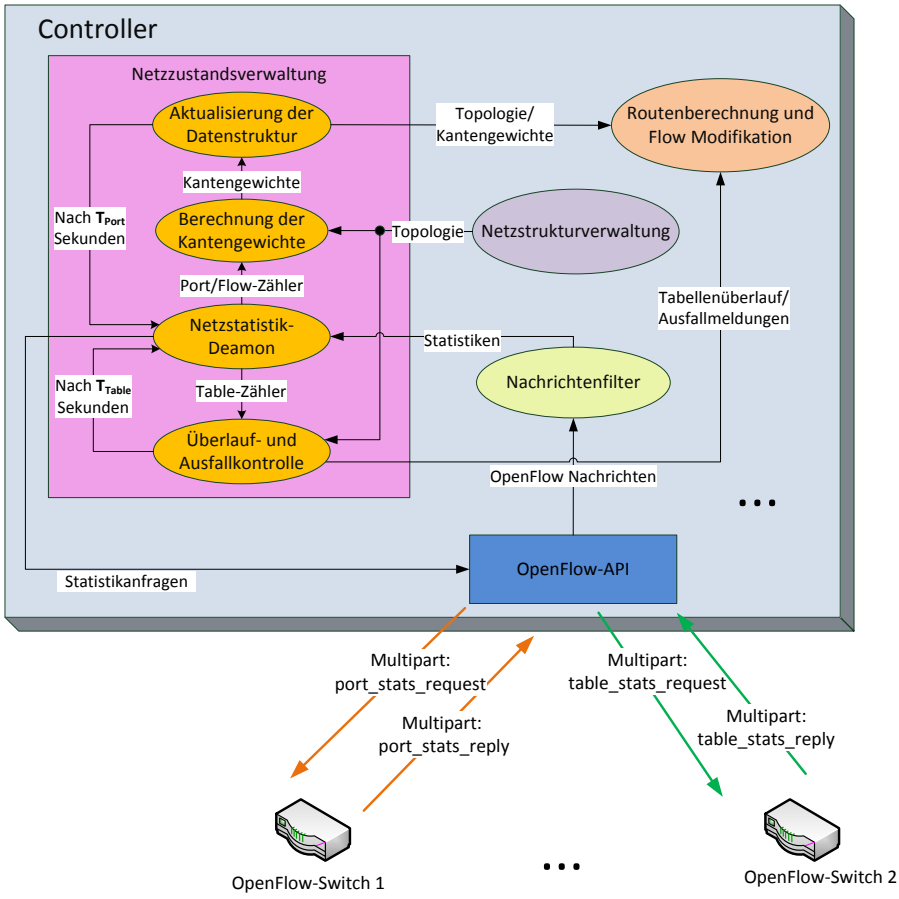

Abbildung 4.4: Aufgaben der Netzzustandsverwaltung

Eine Übersicht über den Ablauf der einzelnen Aufgaben der Netzzustandsverwaltung ist in Abbildung 4.4 dargestellt. Statistik- und Topologieinformationen werden an die Netzzustandsverwaltung übergeben. Eine kantengewichtete Netzrepräsentation, sowie die Benachrichtigung über einen drohenden Tabellenüberlauf stellen die Ausgangsinformationen an die Routenberechnung dar. Zusätzlich wird die Routenberechnung über Ausfälle in der Netzstruktur informiert. Die in Abbildung 4.4 definierten Aufgaben werden nachfolgend in den Abschnitten 4.4.1 bis 4.4.4 einzeln beschrieben.

4.4.1 Ermitteln der Netzstatistiken

OpenFlow bietet über eine *Multipart*-Nachricht die Möglichkeit, Statistikinformationen an den Switches abzufragen. Das geschieht hier periodisch über den Netzstatistik-Deamon. Im Zeitabständen von T_{Port}-Sekunden fragt er Port- und Flow-Statistiken ab. Im Abstand von T_{Table}-Sekunden werden Statistiken über Flow- und Group-Tables ermittelt. Dabei soll $T_{Port} \neq T_{Table}$ gelten.

Ein OpenFlow-Switch implementiert nach der OpenFlow-Spezifikation mehrere Zähler, die die Zeit, die Anzahl der Pakete und die Anzahl der Bytes auf verschiedenen Granularitätsebenen vorhalten. Ein *Multipart_Request* weist den Switch an, die aktuellen Zählerwerte an den Controller zurückzuschicken. Für die Anfrage muss im Header lediglich der gewünschten Port, die Tabelle oder ein Flow vorgeben werden. Aus dem Body der *Multipart*-Antwort kann die Netzzustandsverwaltung die gewünschten Zähler auswerten.

Hier sind die Bytezähler der einzelnen Ports und Flows von Interesse, um daraus die aktuelle Netzlast zu errechnen. Für die Port-Statistik wird an alle Switches ein *Multipart Request* vom Typ *Port_Stats* versendet. Im Header wird der *Any*-Port vorgegeben, woraufhin der Controller die Zähler für alle auf dem Switch vorhandenen Ports zurückerhält. Zusätzlich werden für jeden aktiven Multicastbaum die Flow-Statistikwerte benötigt, um feststellen zu können mit welcher Bandbreite dieser zur Gesamtauslastung beiträgt. Dies geschieht, gleichzeitig zur Portanfrage, über einen *Multipart Requests* vom Typ *Flow_Stats*, der an die jeweils zur Gruppe gehörenden Switches gesendet wird. Diese Information ist wichtig, damit bei einer Neuberechnung für eine Gruppe nicht der eigen produzierte Traffic zu einer möglicherweise schlechteren Route führt.

Sei $C_{out}(t, p)$ der Zähler, der die Anzahl der Bytes für einen Switch zurückgibt, die bis zum Zeitpunkt t über den Port p herausgeschickt wurden. Durch periodisches Abfragen der aktuellen Zählerwerte in einem Zeitintervall T_{Port}, kann die aktuelle Bandbreitenauslastung $L_i(e)$ eines Ausgangsports (bzw. Kante) e im Netzgraph $G(V, E)$ folgendermaßen errechnet werden:

$$L_i(e) := \frac{C_{out}(t_i, p) - C_{out}(t_{i-1}, p)}{T_{Port}} \tag{4.1}$$

Damit bestimmt $L_i(e)$ die Anzahl der Bytes pro Sekunde, die im Zeitraum zwischen t_i und t_{i-1} über diese Kante geflossen sind. Im Systemmodell wird von Vollduplex-Leitungen ausgegangen, so dass in beide Richtungen, unabhängig voneinander, die volle Bandbreite zur Verfügung steht. Analog zu $L_i(e)$ bezeichnet $L_{(s,K)_i}(e)$ die Bandbreite, die über einen Flow-Eintrag einer bestimmten Gruppe $K \subseteq V$ mit zugehörigem Sender $s \in V$ momentan genutzt wird. Sie wird über die angefragten Bytezähler der Gruppen-Flows ermittelt. $L_{(s,K)_i}(e)$ ist 0, wenn diese Gruppe zuvor nicht existiert hat oder in der letzten Zeitperiode T_{Port} keine Übertragung stattgefunden hat.

Um die Historie in die aktuelle Berechnung mit einzubeziehen, wird der exponentiell geglättete Mittelwert (*exponential moving average*) $L^*_{i-1}(e)$ aus den vorangegangenen Messungen herangezogen. Der initiale Bandbreitenwert $L^*_0(e)$ ist dabei mit 0 vorgegeben.

Die aktuelle Auslastung unter Berücksichtigung der Historie wird mit $L_i^*(e)$ bezeichnet und errechnet sich durch:

$$L_i^*(e) := \beta L_i(e) + (1 - \beta) L_{i-1}^*(e) \tag{4.2}$$

Die rekursive Formel (4.2) bezieht, neben der aktuellen Auslastung, die vorangegangenen Mittelwerte mit absteigender Gewichtung mit ein. Je weiter ein Wert in der Vergangenheit liegt, desto geringer ist sein Einfluss. Der Faktor $0 \leq \beta \leq 1$ bestimmt dabei die Gewichtung zwischen der aktuellen Bandbreite und dem vorangegangenen Mittelwert. Die Betrachtung der Historie ist sinnvoll, da die zukünftige Auslastung einer Kante im Allgemeinen nicht bekannt ist und die Kantenauslastung, während der aktuellen Routenberechnung im Zeitintervall T_{Port}, stark von den vorherigen Werten abweichen kann. So ist es beispielsweise denkbar, dass eine Kante in größeren Abständen durch bestimmte Multicastübertragungen stark ausgelastet wird, dies aber in der aktuellen Berechnung, innerhalb des im Vergleich kurzen Intervalls T_{Port}, nicht erkannt wird. Durch die Einberechnung vergangener Werte ist so eine realistischere Prognose zukünftiger Übertragungen möglich. Die selbe Berechnung, wie in Gleichung 4.2, geschieht analog für ${L^*}_{(s,K)_i}(e)$ für jeden vorhandenen Multicastbaum. Dieser Wert gibt den Einfluss des Baumes (s, K) bezüglich der Auslastung auf Kante e, inklusive der Historie, an.

Die Bytezähler, die für die Berechnung aus Gleichung 4.1 benötigt werden, sind in der OpenFlow Spezifikation als optional gekennzeichnet. Es muss daher bei der Auswahl der Hardware darauf geachtet werden, dass diese auch tatsächlich implementiert sind. Zusätzlich kann es sich bei den Zählern um ungenauere Softwarezähler handeln, die lediglich durch das periodische Anfragen von Hardwarezählern implementiert sind [P+12]. Ungenauigkeiten können aber auch durch die Verzögerung der *Multipart*-Nachrichten an der Schnittstelle, im Netz und in der Eingangswarteschlange der Switches auftreten. Die Kommunikation im Controllernetzwerk geschieht typischerweise über TCP, was zusätzlich schwer vorhersehbare Verzögerungszeiten mit sich bringen kann.

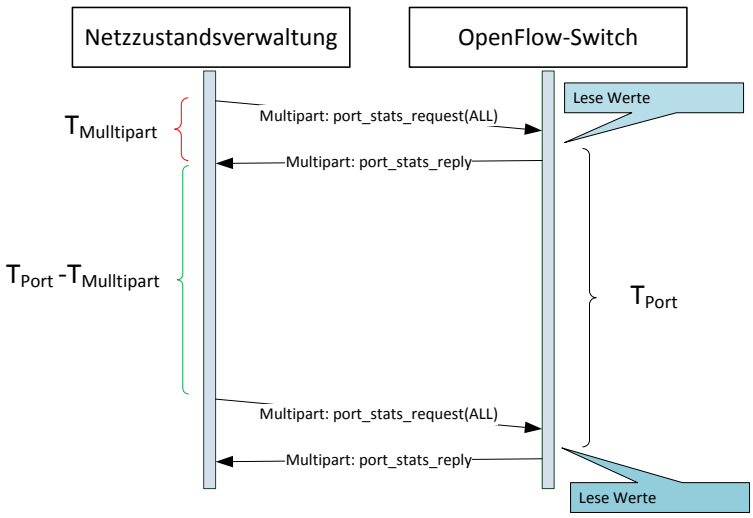

Abbildung 4.5: Anpassung der Intervallzeit T_{Port}

Durch die hier verwendete Trennung zwischen Controllernetz und Datenschicht, können diese Probleme zum Teil reduziert werden, sofern das Controllernetz eine geringe Auslastung aufweist. Zusätzlich wird zur Verbesserung der Genauigkeit die letzte Verzögerungszeit zwischen Anfrage und Antwort der *Multipart*-Nachricht gemessen und das Zeitintervall T_{Port} bis zur nächsten Anfrage entsprechend angepasst (siehe Abbildung 4.5). Das basiert auf der Annahme, dass in einem gering ausgelasteten Netz die Übertragungszeit der Anfrage und die der Antwort etwa gleich lang ist. In diesem Fall kann man die Zeit, die vergangen ist, seit dem die Zählerwerte tatsächlich gelesen wurden, mit $T_{Multipart}/2$ annähern. Indem man nun die Wartezeit T_{Port} bis zum Senden der nächsten Anfragenachricht um $T_{Multipart}$ reduziert, erreicht man im Idealfall einen Zeitabstand von T_{Port} zwischen zwei Lesevorgängen.

4.4.2 Überlauf- und Ausfallkontrolle

Neben den Byte-Statistiken ist der Netzstatistik-Deamon auch für die Ermittlung der Einträge für die Flow- und Group-Tables zuständig. Diese Informationen werden in der Überlauf- und Ausfallkontrolle verarbeitet. Desweiteren findet nach jeder Neubestimmung der Topologie eine Überprüfung nach Switch- und Leitungsausfällen statt.

Überlaufkontrolle Mit dem *Multipart-Type: Flow_Stats_Request* kann man, analog zu Flows und Ports, die Zähler einer Flow-Table abfragen. Dazu muss in der Anfrage die Tabellen-ID angegeben werden. So erhält man die Anzahl aller aktiven Flow-Einträge sowie die Anzahl der *Lookups* und der erfolgreichen *Matchings* zurück. Die maximale Anzahl der Tabelleneinträge hängt vom Switch-Modell ab und muss vorab bekannt sein. Die Überlaufkontrolle bestimmt daraus, zu wie viel Prozent die Tabellen gefüllt sind. Überschreitet eine Tabelle den definierten Schwellenwert K_{max}, folgt eine Signalisierung an die Routenverwaltung. Daraufhin werden nachfolgende Flow-Einträge in Form eines Shared-Trees eingerichtet, um Tabelleneinträge zu sparen. Fällt die Anzahl der maximalen Einträge aller Tabellen anschließend erneut unter einen bestimmten unteren Schwellwert K_{min}, wird der Routenberechnung signalisiert, dass wieder auf quellenbasierte Bäume umgestellt werden kann. Die Abfrage geschieht ebenfalls periodisch mit der Zykluszeit T_{Table}. Allerdings nicht im gleichen Zyklus wie die Portstatistik-Abfrage, um die Netzlast im Controller-Netzwerk möglichst gering zu halten.

Ausfallkontrolle Die Überlauf- und Ausfallkontrolle speichert eine Kopie der aktuellen Netztopologie. Nach jeder periodischen Neubestimmung der Netzstrukturverwaltung, geschieht ein Abgleich zwischen der aktuellen und der alten Topologie, durch einen Knoten- und Kantenvergleich. Dadurch werden Änderungen, wie der Ausfall eines Switches oder einer Leitung erkannt und an die Routenberechnung gemeldet. Dabei werden 2 Mengen übergeben. Die Menge S der ausgefallenen Switches und die Menge E aller ausgefallenen Leitungen, die nicht zu einem Switch in S gehören. Die Routenberechnung führt mit Hilfe dieser Informationen eine sofortige Neuberechnung für die betroffenen Routen durch, um Paketverluste zu reduzieren. Um nicht auf den nächsten Aktualisierungszyklus der Netzstrukturverwaltung warten zu müssen, werden die ausgefallenen Kanten sowie alle Kanten eines ausgefallenen Switches vorübergehend mit unendlich markiert. Es sei anzumerken, dass bei Topologieänderungen

lediglich ein Ausfall einer gesonderten Behandlung bedarf. Neu hinzugekommene Switches und Verbindungen werden dagegen von der Routenberechnung für aktuelle Berechnungen aktualisiert. Vorhandene Routen passen sich, abhängig von der Gruppendynamik, an eine neue Topologie automatisch an und erfordern deshalb keine extra Betrachtung.

4.4.3 Berechnung der Kantengewichte

Wie in Kapitel 2.3.2 diskutiert, gibt es verschiedene Möglichkeiten, ein Routing mit mehreren Metriken zu realisieren. Hier soll das Ziel sein, *Multipathing* auf Baumebene auszunutzen, um eine gleichmäßige Lastverteilung (*Flow-Balancing*) zu erreichen. Im Hinblick auf eine möglichst performante Routenberechnung wird eine gewichtete Summe verwendet, um die Kantenauslastung in die Routenberechnung mit einzubeziehen. Das hat den großen Vorteil, dass der Routingalgorithmus lediglich nach einem Wert optimieren muss. Im Vergleich zu komplexeren Optimierungsverfahren, bringt dies für das Routing keine höhere Laufzeit mit sich.

Gegeben ist die Netztopologie in Form eines Graphen $G(V, E, d(e))$. Die Funktion $d(e)$ ordnet jeder Kante $e \in E$ eine fixe Distanz zu. Die Gleichung 4.3 bestimmt das zusammengesetzte Kantengewicht $w(e)$, das die Distanz und die aktuell zur Verfügung stehende Bandbreite, inklusive der Historie, miteinander verrechnet. $B(e)$ ist die initiale Bandbreite (Kapazität) der Leitung und ändert sich nicht, während $L(e)$ die aktuelle Netzlast der Kante nach Gleichung 4.1 angibt. $L_i^*(e)$ betrachtet zusätzlich den Mittelwert vorangegangener Werte nach Gleichung 4.2. Die maximale Kantendistanz $d(e)$ aller Kanten im Graph wird mit d_{max} bezeichnet.

$$w(e) := \begin{cases} \left(\alpha \frac{d(e)}{d_{max}} + (1-\alpha)\frac{L_i^*(e)}{B(e)}\right) r & \text{für } \frac{L_i(e)}{B(e)} \leq L \\ \text{"}\infty\text{"} & \text{für } \frac{L_i(e)}{B(e)} > L \end{cases} \tag{4.3}$$

Überschreitet die aktuelle Netzlast einen bestimmten konstanten Schwellenwert $0 \leq L \leq 1$, wird die Kante als ausgelastet markiert. Was zur Folge hat, dass ausschließlich alternative Pfade genutzt werden. Das verhindert Paketverluste durch Leitungsüberlastung (*Congestion*). Setzt man $L = 1$ wird dieser Fall ignoriert und es wird niemals eine Kante mit unendlich markiert. Bei Einrichtung einer Route ist im Allgemeinen nicht bekannt, wie hoch die verbrauchte Bandbreite sein wird. Deshalb sollte K so gewählt werden, dass ein ausreichender Puffer nach oben gegeben ist. Im gegebenen Systemmodell stellt 0,9 ein sinnvoller Wert dar. Das bedeutet, dass einer Kante die zu 90% oder mehr ausgelastet ist, keine neuen Routen mehr zugewiesen werden. Für den Fall, dass die Bandbreite kleiner als K ist, normalisiert der erste Summand in Gleichung 4.3 die Kantenlänge. Der zweite normalisiert analog dazu die exponentiell geglättete Netzlast in Bezug auf die Kantenkapazität. Im konkreten Fall gilt hier $\frac{d(e)}{d_{max}} = 1 \forall e$, da als Distanz die Anzahl der Hops verwendet wird. $0 \leq \alpha \leq 1$ bestimmt das Verhältnis zwischen Distanz und Bandbreite. Der Extremfall $\alpha = 1$ würde eine Pfadauswahl nach kürzesten Wegen unabhängig von der Bandbreite bedeuten, während man bei $\alpha = 0$ lediglich die Bandbreite beachten würde. Im Fall $\alpha = 0,5$ wären beide Teile genau gleich gewichtet. Der Faktor r skaliert das Ergebnis auf beliebige Größe, so dass der Routingalgorithmus wahlweise mit Ganzzahlen oder Gleitkommazahlen rechnen kann.

Analog dazu werden die Gewichte $w_{(s,K)}$ nach 4.4 ermittelt. Diese Werte werden für jeden Multicastbaum abgespeichert und erst bei einer Neuberechnung des Baumes jeweils vom aktuellen Gesamtgewicht $w(e)$ abgezogen. Der Wert $w_{(s,K)}$ spiegelt dabei den Anteil der Bandbreite wieder, den der Multicastbaum (s, K) selbst verbraucht. Er soll bei auftretenden Gruppenänderungen und den darauf folgenden Routenberechnungen für diesen Baum ignoriert werden.

$$w_{(s,K)} := \begin{cases} \left((1-\alpha)\frac{L^*_{(s,K)_i}(e)}{B(e)}\right)r & \text{für } \frac{L_{(s,K)_i}(e)}{B(e)} \leq L \\ \text{"}\infty\text{"} & \text{für } \frac{L_{(s,K)_i}(e)}{B(e)} > L \end{cases} \tag{4.4}$$

Bild 4.6 zeigt ein Beispiel für einen Graphen mit berechneten Kantengewichten $w(e)$ und die zugehörige Datenstruktur. Durch den Vollduplex-Betrieb werden für die beiden Richtungen eigene Gewichte berechnet, die unabhängig von der jeweiligen Gegenrichtung sind. Die Auslastung der Kanten ist in Prozent angegeben, zugehörige Kantengewichte sind rot, bzw. violett hervorgehoben. Die Kante zwischen Knoten 2 und 3 ist zu 50% ausgelastet. Der Routingalgorithmus würde, um Knoten 1 mit Knoten 5 zu verbinden, die Kante über Knoten 4 bevorzugen. Die Gesamtkosten für diesen Weg betragen nur $(13, 2)$, während der Weg über Knoten 3 Kosten von $(16, 2)$ aufweist. Die Gewichtungen der Kanten in Gegenrichtung (grau dargestellt), haben durchgehend den Wert 4 und sind für dieses Beispiel nicht relevant. Die Berechnungsvorschrift für das Beispiel lautet $w(e) = \left(0.4 + 0.6\frac{L^*_i(e)}{B(e)}\right)10$ nach folgender Wertebelegung für Gleichung 4.3:

α	0.4
$d(e)$	1
d_{max}	1
r	10
L	1

Tabelle 4.1: Wertebelegung für Abbildung 4.6

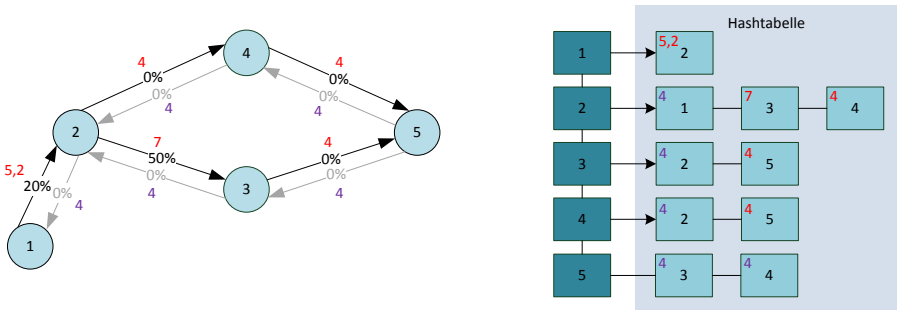

Abbildung 4.6: Beispielnetz als kantengewichteter Graph und Adjazenzliste

4.4.4 Datenstruktur

Der Netzgraph wird von der Netzstrukturverwaltung in Form einer Adjazenzliste zur Verfügung gestellt. Das hat den Vorteil, dass die Datenstruktur jederzeit erweiterbar ist, einen geringen Platzbedarf hat und schnell initialisiert werden kann [Leh]. Aus Effizienzgründen kommt dazu keine verkettete Liste, sondern eine Hashtabelle zum Einsatz. Einem Knoten wird über seine Knoten-ID die zugehörige Kantenmenge zugewiesen. Die Netzzustandsverwaltung erweitert die Datenstruktur um die berechneten Kantengewichte, indem für jede Kante, über eine weitere Hashtabelle, ein Kantengewicht zugeordnet wird (Bild 4.6). Dadurch ist die Datenstruktur für die Gewichtsbelegung von der Topologie entkoppelt und kann unabhängig vom Zyklus der Netzzustandsverwaltung aktualisiert oder nach Belieben ausgetauscht werden. Dieser Vorgang geschieht periodisch, da sich die Auslastung der Kanten ständig ändern kann und der Routingalgorithmus adaptiv darauf reagieren soll. Neben dem eigentlichen Netzgraph gibt es eine weitere Hashtabelle, die die aktuelle Auslastung bezogen auf einen Multicastbaum abspeichert. Sie wird analog zu der in Bild 4.6 gezeigten Hashtabelle angelegt. Diese Informationen werden erst dann benötigt, wenn für den betroffenen Baum tatsächlich eine Neuberechnung stattfindet und sind deshalb erst in der Routenberechnung relevant. Nachdem T_{Port}-Sekunden seit der letzten Aktualisierung vergangen sind, wird der Netzstatistik-Deamon nach Bild 4.4 erneut gestartet und der geschilderte Ablauf beginnt mit der Ermittlung der Netzstatistiken von vorne.

4.5 Gruppenverwaltung

Die Gruppenverwaltung ist für das Ermitteln, Speichern und Weitergeben von Mitgliederinformationen der Multicastgruppen zuständig. Abbildung 4.7 zeigt eine Übersicht über diese Aufgaben. Die Gruppeninformationen werden in der Routenberechnung benötigt und können über eine Schnittstelle abgefragt werden. Der IGMP-Deamon sendet periodisch IGMP-Anfragen an die Datenschicht. Die Antworten werden gespeichert und die Routenberechnung über Änderungen informiert. Im Nachfolgenden sind die einzelnen Aufgaben ausführlicher beschrieben.

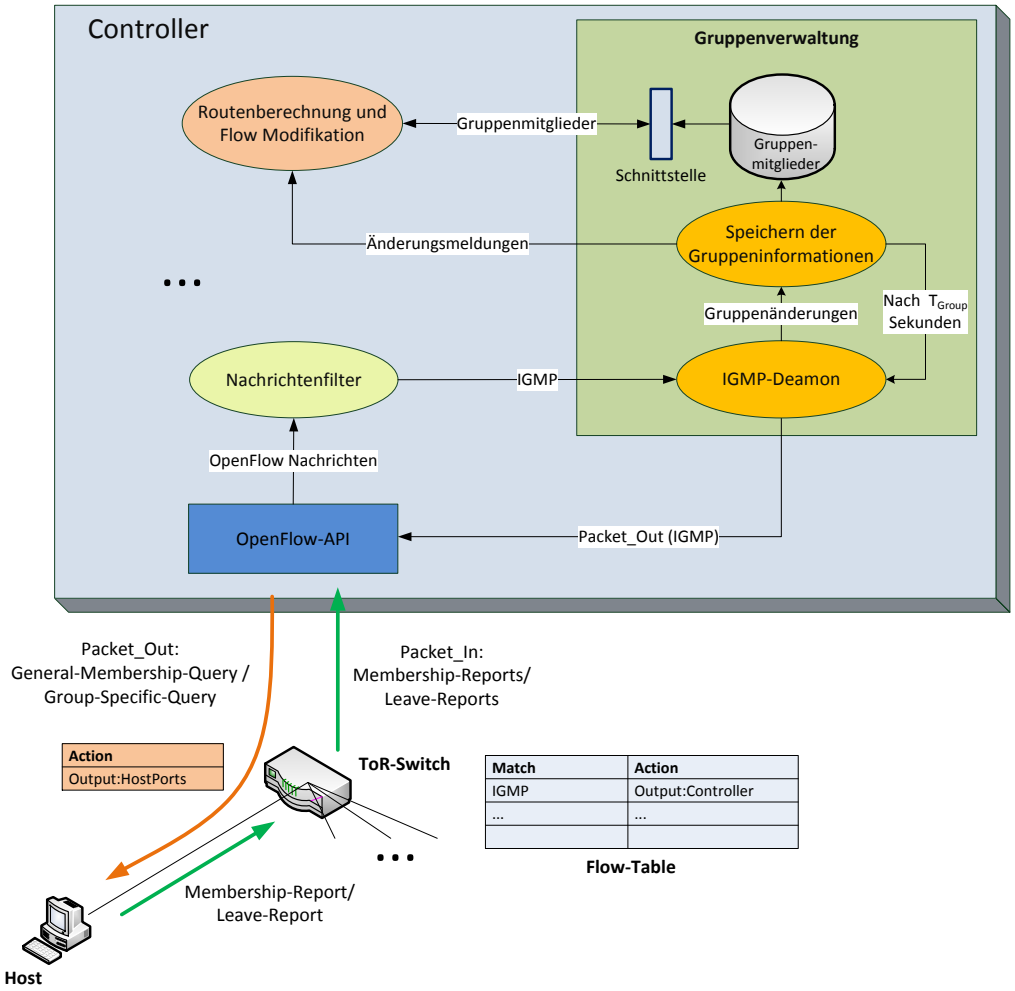

Abbildung 4.7: Aufgaben der Gruppenverwaltung

4.5.1 IGMP über OpenFlow

Einrichtung der Flow-Tables Initial müssen die Flow-Tables in den Switches so konfiguriert werden, dass IGMP-Nachrichten weitergeleitet werden können. Nach dem hier betrachteten Systemmodell, betrifft das lediglich die ToR-Switches, da alle Hostmaschinen als Blattknoten im Fat-Tree angeschlossen sind. In allgemeinen Netztopologien müsste man Flows in allen vorhandenen Switches einrichten.

Bild 4.7 zeigt schematisch die Flow-Table eines ToR-Switches. Das Hinzufügen der Flows geschieht initial per OpenFlow über eine *Flow_Mod*-Nachricht vom Typ *ADD* mit deaktivierten Timeout-Zählern. Ein IGMP-Paket wird über eine *Packet_Out*-Nachricht zu den Hosts geleitet. Dafür wird die auszuführende Aktion, hier das Weiterleiten an alle Host-Ports, direkt in der *Packet_Out*-Nachricht spezifiziert. Abgeglichen wird dabei der Eingangsport und das Feld *nw_proto*, welches das Nachfolgeprotokoll des IP-Headers bestimmt. Eine Protokollnummer von 2 identifiziert IGMP-Pakete. Eine alternative Lösung wäre, einen extra Tabelleneintrag für ausgehende IGMP-Pakete in den Switches einzurichten und die Aktion *Packet_Out*-Nachricht mit *Table* zu markieren. Dies hat zur Folge, dass dieses Paket standardmäßig über die Flow-Table des Empfängerswitches abgeglichen wird. Die erste Variante spart jedoch Tabellenplatz und wird deshalb hier bevorzugt. Ein Eintrag in der Flow-Table leitet schließlich die IGMP-Antworten zurück zum Controller, indem unabhängig vom Eingangsport nach IGMP-Paketen gefiltert wird. Da sich im Systemmodell nur OpenFlow-Switches befinden, ist der von IGMP spezifizierte TTL (*Time to Live*) von 1 nicht mehr von Bedeutung. Durch die Einträge in den Flow-Tables wird eine Weiterleitung über das lokale Netz hinaus implizit ausgeschlossen. Bei einer Mischung zwischen OpenFlow-Switches und Netzgeräten, die nicht OpenFlow fähig sind, muss der TTL jedoch dementsprechend angepasst werden, um das ungewollte Konsumieren einer IGMP Nachricht zu verhindern.

Implementierung von IGMP Die Gruppenverwaltung implementiert IGMP, um eine nach IP standardkonforme Kommunikation mit den Hostgeräten zu ermöglichen. Der IGMP-Deamon erzeugt periodisch, in Zeitabständen von QI-Sekunden (*Query Interval*), *General-Membership-Queries*. Der Standardwert für QI ist 125 Sekunden [CDK+02]. Über *Packet_Out*-Nachrichten werden die IGMP-Anfragen an alle ToR-Switches geschickt. Dabei ist der Controller in der Lage, beliebig IGMP-Anfragen der Version 1, 2 oder 3 zu verschicken. Normalerweise einigen sich die Netzgeräte dann auf die neuste IGMP-Version, die alle Kommunikationsteilnehmer (hier Controller und Host) unterstützen. Als Zieladresse im IP-Header wird vom Controller „224.0.0.1“ vorgegeben. Dies entspricht einer vordefinierten Multicastadresse, die automatisch alle Geräte im Netz beinhaltet. Dadurch wird sichergestellt, dass auch alle Endgeräte die IGMP-Nachricht annehmen und verarbeiten. Die Hosts antworten dann wiederum per *General-Membership-Report* mit ihren aktuellen Gruppenmitgliedschaften.

IGMP definiert neben den *General-Membership-Queries* auch noch gruppenspezifische Anfragen. Damit wird bei den Hosts konkret nach einer im Header definierten Gruppe gefragt. Das Versenden einer solchen *Group-Specific-Query* geschieht nicht periodisch, sondern nur als Antwort auf einen explizit gesendeten *Leave-Report*. Ein Host schickt einen *Leave-Report* gemäß IGMPv2/3 ohne vorangegangene Anfrage. Mit der *Group-Specific-Query* wird geprüft,

ob noch weitere Hosts, die an den entsprechenden Gruppennachrichten interessiert sind, an diesem Switch angeschlossen sind. Analog dazu, kann ein Host, der nicht auf die nächste *General-Membership-Query* warten möchte, explizit per *Membership-Report* einer Gruppe beitreten. Sämtliche Änderungen in der Gruppenzusammensetzung führen zu einer Änderung des Gruppenspeichers.

4.5.2 Datenstruktur

Lookup- und Update-Operationen auf den gespeicherten Gruppeninformationen sollen effizient und skalierbar sein. Aus diesem Grund werden die aktuellen Gruppenzusammensetzungen im Hauptspeicher vorgehalten. Dadurch wird ein schneller Zugriff auf die Gruppeninformationen für die Routenberechnung und die Gruppenverwaltung sichergestellt. Abgespeichert wird eine Menge von ToR-Switches, die entsprechende Multicast-Nachrichten erhalten wollen. Die letztendliche Auslieferung an die Endgeräte geschieht vom Tor-Switch aus per Broadcast. Für eine noch genauere Zustellung könnte man auch einzelne Hosts speichern. Das würde die Datenstruktur jedoch unnötig vergrößern und im betrachteten Systemmodell zu keinem nennenswerten Vorteil führen.

Zur Speicherung der Daten wird eine Hashtabelle verwendet. Das bringt den Vorteil einer, im Idealfall, konstanten Zugriffszeit bei geringem Speicherplatz. Die dort gespeicherten Gruppendaten müssen stets an die aktuelle Situation im Netz angepasst werden. Der Routenberechnung soll so eine aktuelle Sichtweise der Gruppen und deren Mitglieder zur Verfügung gestellt werden. Jegliche Änderungen, signalisiert durch ankommende IGMP-Reports, führen zu entsprechenden Anpassungen in der Hashtabelle. Ein IGMP-Report enthält stets eine Multicastadresse für die Gruppe, zu der ein Switch beitreten möchte. Anhand des Eingangsports im Controller ist es außerdem möglich, die ID des Switches zu bestimmen, der den IGMP-Report gesendet hat. Diese beiden Informationen sind für die Gruppenverwaltung ausreichend und werden in folgender Datenstruktur gespeichert:

$$\begin{aligned} &< Multicastadresse, \{Switch_1, Switch_2, ...\} > \\ &mit\ Switch_k := (SwitchID_k, Timeout\_Zähler_k) \end{aligned} \tag{4.5}$$

Die Multicastadresse identifiziert eine Gruppe eindeutig und bildet den Schlüssel der Hashtabelle. Jeder Multicastgruppe ist über ihren Schlüssel eine Liste von Switches zugeordnet, die die Gruppenmitglieder darstellen. Ein Switch besteht aus einer Switch-ID, die das Gruppenmitglied eindeutig identifiziert. Außerdem ist jedem Switch ein Zähler zugeordnet, der den *Soft-State*-Mechanismus von IGMP implementiert. Der Zähler wird periodisch bis auf 0 heruntergezählt, woraufhin der Switch-Eintrag automatisch aus der Tabelle gelöscht wird. Das hat eine sofortige Benachrichtigung an die Routenberechnung zur Folge. Sobald keine Einträge mehr in einer Switch-Liste vorhanden sind, wird die gesamte Multicastgruppe, nach einer gewissen Verzögerungszeit, automatisch aus der Hashtabelle entfernt. In Abbildung 4.8 ist die Datenstruktur mit der Zuordnung der Switches zu einer Multicastadresse zu sehen.

Der initiale Timeout-Wert für $Timeout\_Zähler_k$ berechnet sich nach der IGMP Spezifikation [CDK+02] mit:

$$Timeout\_Zähler_k := R \times QI + QRI \tag{4.6}$$

R ist die *Robustness Variable* und wird abhängig von der Zuverlässigkeit des Netzes gewählt. In einem verlustreichen Netzwerk sollte R einen höheren Wert aufweisen, der Standartwert beträgt 2. QI (*Query Interval*) ist die Zeitperiode zwischen zwei Anfragen. QRI (*Query Response Interval*) steht für die maximale Antwortzeit, die einem Host für eine *General-Membership-Query* zur Verfügung steht. Der Standardwert für QRI beträgt 10 Sekunden.

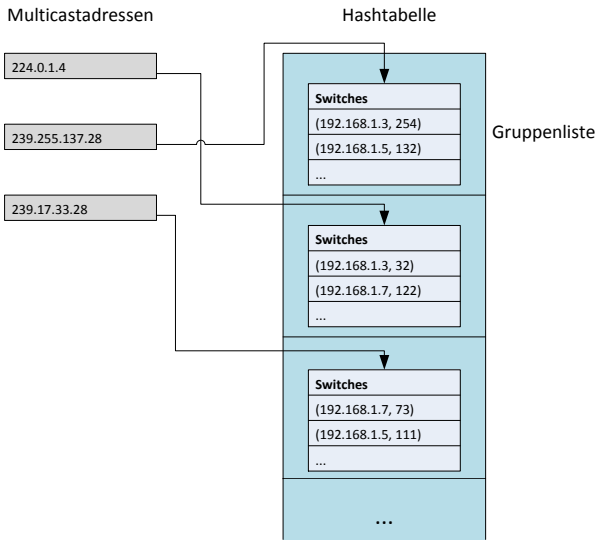

Abbildung 4.8: Zuordnung der Gruppenmitglieder über eine Hashtabelle

4.5.3 Verarbeitung von IGMP-Nachrichten im Controller

Periodisch werden dem IGMP-Deamon Gruppenänderungen in Form von *IGMP-Membership-Reports* gemeldet, oder ein explizites Verlassen durch *Leave-Reports* signalisiert. Bild 4.9 zeigt die Verarbeitung der Report-Nachrichten nach dem Empfang durch den IGMP-Deamon. Die linke Seite (hellblauer Hintergrund) filtert die Nachrichten nach ihrem Typ. Die rechte Seite zeigt die Verarbeitungsschritte, die für die einzelnen Typen durchgeführt werden (roter Hintergrund: *Leave-Report*, lila Hintergrund: *Membership-Report*). Die dazu benötigten Informationen sind die Multicastadresse und der IGMP-Nachrichtentyp aus dem Nachrichtenheader, sowie die Switch-ID des Senders.

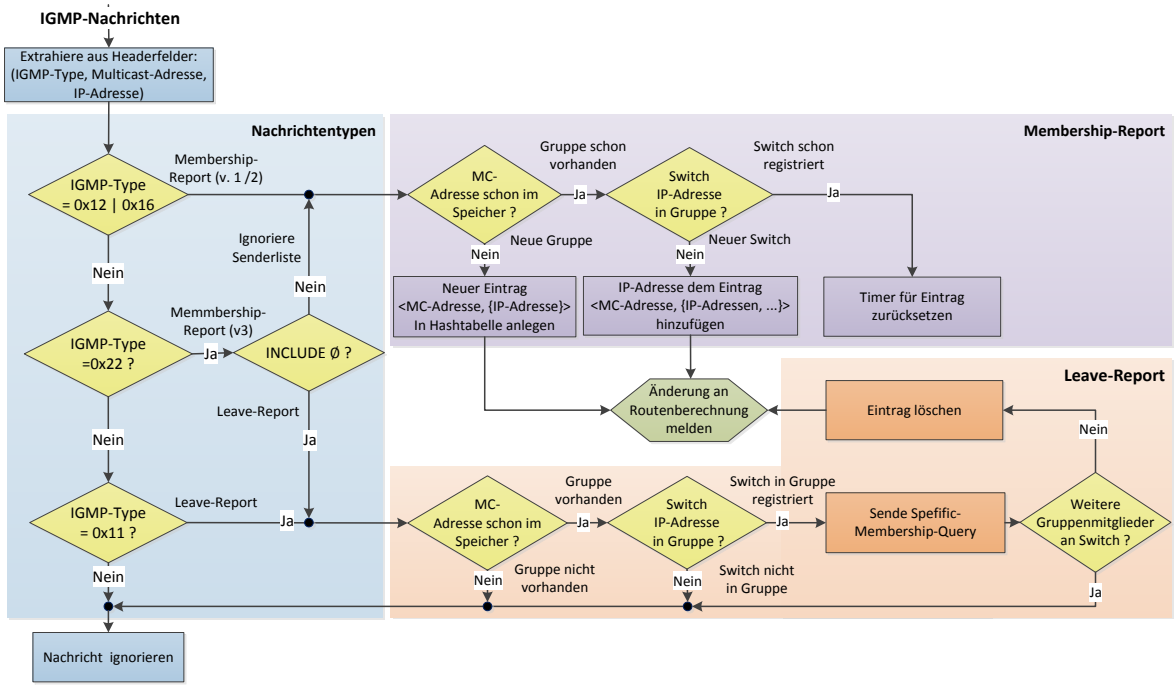

Abbildung 4.9: Verarbeitung ankommender IGMP-Nachrichten in der Gruppenverwaltung

Die *Membership-Reports* der IGMP-Versionen unterscheiden sich im *IGMP-Type*-Header. Die Reports der Version 1 und 2 werden gemeinsam verarbeitet, da der Unterschied zwischen diesen lediglich aus verschiedenen Typnummern besteht. Aus Flexibilitätsgründen wird auch IGMPv3 unterstützt. Jedoch ignoriert die Gruppenverwaltung aufgrund der Einfachheit solche IGMPv3-Reports, die bestimmte Sendergruppen ausschließen wollen. Das Verarbeiten solcher Nachrichten würde erheblichen Zusatzaufwand bedeuten, da diese Informationen für jeden Host verschieden sein können. Der hier beschriebene Controller verwaltet allerdings lediglich ToR-Switches und keine einzelnen Hosts. Einzig der Fall, wenn alle Sender einer Gruppe ausgeschlossen sind, wird einzeln behandelt. Dies würde nämlich einem *Leave-Report* gleichkommen. In allen anderen Fällen wird der Switch analog zu IGMPv1- und IGMPv2-Reports, der im Header spezifizierten Multicastgruppe, hinzugefügt. Dennoch hat die Verwendung von IGMPv3 den Vorteil, dass die gesamte Netzlast reduziert wird. Die Gruppenmitgliedschaften eines Hosts werden, im Gegensatz zu den vorherigen Versionen, bei Version 3 gemeinsam in einer einzigen Nachricht versendet. Falls trotzdem eine Filterung nach einzelnen Senderquellen gewünscht ist, muss dies der Host selbst übernehmen. Existiert bei Ankuft eines Reports noch kein Eintrag zu der im Reportheader empfangenen Multicastadresse, wird zuerst eine neue Gruppe erstellt. Dies geschieht, indem die Multicastadresse in die Hashtabelle hinzugefügt wird. Danach wird der Switch mit seiner ID in die Liste der neu erstellten Gruppe hinzugefügt. Falls eine entsprechende Gruppe bereits existiert, der Switch aber noch kein Mitglied ist, wird er in die vorhandene Gruppenliste ergänzt. Für den Fall, dass der Host bereits in der Mitgliederliste enthalten ist, wird der IGMP-Zähler zurückgesetzt. Das Verhindert ein Timeout, da der Switch weiterhin Interesse an der Multicastgruppe bekundet hat.

Die andere Art von Nachricht, die der IGMP-Deamon empfängt, sind *Leave-Reports*. Sollte die darin spezifizierte Gruppe nicht existieren oder die Switch-Adresse in der entsprechenden Gruppenliste gar nicht vorhanden sein, wird die Nachricht einfach ignoriert. Anderenfalls muss vor dem Löschen sichergestellt werden, dass kein anderer Host die Gruppennachrichten weiterhin erhalten möchte. Deswegen wird per *Specific-Membership-Query* nach weiteren Interessenten gefragt. Erst im Anschluss darf der Eintrag gelöscht werden. Zum Schluss müssen jegliche Gruppenänderungen der Routenberechnung gemeldet werden, damit die neuen Multicastbäume berechnet und in den Switches eingerichtet werden können.

4.6 Routenverwaltung

Bild 4.10 gibt einen Überblick über die Routenverwaltung und dessen Interaktion mit der Routenberechnung. Die Routenverwaltung speichert alle Multicastbäume, die aktuell in Form von Flow-Table Einträgen auf der Datenebene implementiert sind. Somit stellt sie die Datenbasis für die eingerichteten Flows auf Datenschicht zur Verfügung. Dazu existiert eine Datenstruktur im Hauptspeicher, die nach jeder bestätigten Änderung in der Flow-Table entsprechend aktualisiert wird. Über eine Schnittstelle erfolgt sowohl der Zugriff sowie die Aktualisierung der Daten durch die Routenberechnung. Abhängig von der Arbeitsweise des Routingalgorithmus, können die Flow-Informationen z. B. für die konsistente Aktualisierung in der Datenschicht oder für eine inkrementelle Baumaktualisierung genutzt werden.

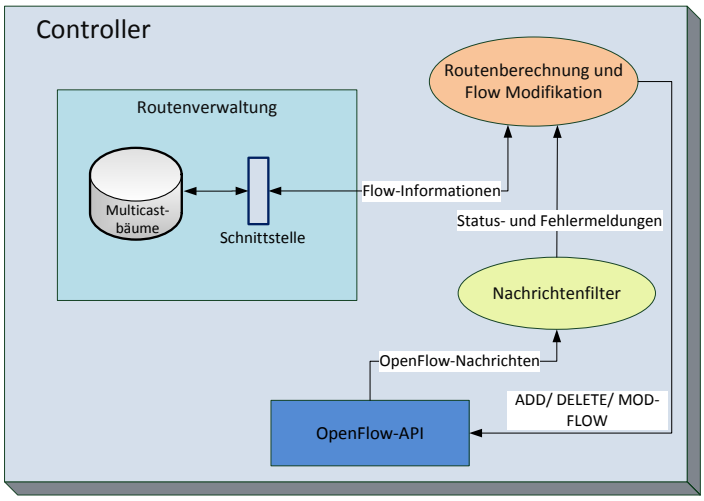

Abbildung 4.10: Übersicht über die Routenverwaltung und die Interaktion mit der Routenberechnung

Weiterhin werden diese Daten bei der Flow-Einrichtung benötigt. Bevor eine neue Route auf Datenschicht installiert werden kann, muss der alte Baum durch die Flow-Modifikation per OpenFlow-Nachrichten vom Typ *Delete* gelöscht werden. Die Informationen, wer die

Empfängerswitches dieser Nachrichten sein müssen, stammen aus der Routenverwaltung. Die Teilstrukturen des neuen Baumes, die sich nicht unterscheiden, werden nicht gelöscht und können direkt übernommen werden. Es sei angemerkt, dass OpenFlow einen *Soft-State*-Mechanismus für Flow-Table-Einträge unterstützt, der veraltete Einträge automatisch löschen würde. Aufgrund der proaktiven Berechnung der Multicastbäume kann, davon hier aber nicht Gebrauch gemacht werden und es muss auf ein explizites Löschen zurückgegriffen werden (siehe Abschnitt 4.7.1). Arbeitet ein Routingalgorithmus inkrementell, kann er die Daten der Routenverwaltung als Ausgangsdaten für die Berechnung nutzen. Auf Basis des alten Baumes werden neue Kanten dann demenstprechend durch die Flow-Modifikation entfernt oder hinzugefügt.

4.6.1 Datenstruktur

Sofern eine neue Flow-Aktualisierung von der Datenschicht bestätigt wurde, bzw. kein Fehler gemeldet wurde, müssen die Änderungen an die Routenverwaltung zurückgemeldet werden. Dadurch wird sichergestellt, dass die Datenbasis stets eine konsistente Sicht aufweist. Diese Datenbasis besteht nach Definition 4.7 aus einer Hashtabelle, die sowohl Multicastbäume als auch Unicast-Routen speichern kann. Eine Hashtabelle ermöglicht dabei eine besonders effiziente Zugriffsmöglichkeit. Multicastbäume werden in Form von Adjazenzlisten vorgehalten. Unterschiede zwischen zwei Bäumen können so durch einen einfachen Kantenvergleich erkannt werden. Für eine Unicast-Route ist das Speichern als Knotenliste ohne Kantenverweise ausreichend.

$$< (SenderID + GruppenID), Multicastbaum/Route > \tag{4.7}$$

Der Schlüssel der Hashtabelle besteht aus einer Konkatenation von Sender-ID und Gruppen-ID. Es sei angemerkt, dass im Systemmodell aus Abschnitt 3.2, die Senderrolle nur durch einen ToR-Switch eingenommen werden kann. Als Sender-ID wird die Switch-ID oder die IP-Adresse des Controller-Netzes verwendet, während die Gruppen-ID aus der Multicastadresse besteht. Einem jeden (Sender-ID, Gruppen-ID)-Paar wird über die Liste ein Multicastbaum zugeordnet. Damit ist die Datenstruktur für das Speichern von quellenbasierten Bäumen geeignet. Wahlweise können auch Shared-Trees gespeichert werden, indem der gemeinsame Baum unter dem Schlüssel der Gruppen-ID angelegt wird. Jeder Eintrag in der Hashliste verweist dann lediglich auf diesen Baum, indem die zugehörige Unicast-Route dort gespeichert wird. Sie dient dazu, den Sender-Switch mit dem Shared-Tree zu verbinden. Zur einfacheren Unterscheidung existiert in jedem Eintrag ein *Flag*, das entweder eine Unicast-Route oder einen quellenbasierten Baum anzeigt. Diese Speichermethode ermöglicht ein beliebiges Umschalten zwischen den beiden Baumarten, ohne Einfluss auf den Aufbau der Datenstruktur. Das bedeutet, es wird sichergestellt, dass zu einem (Sender-ID, Gruppen-ID)-Paar entweder eine Unicast-Route, oder ein quellenbasierter Baum existiert. Zwischen verschiedenen (Sender-ID, Gruppen-ID)-Paaren, insbesondere auch innerhalb derselben Gruppe, ist aber durchaus eine Mischung beider Methoden möglich.

Bild 4.11 zeigt dazu ein Beispiel. Der linke Hashtabellen-Ausschnitt zeigt die Zuordnung von zwei Senderquellen zu einem Shared-Tree der Multicastgruppe „239.255.137.28“. Die Einträge der Hashtabelle verweisen auf den Shared-Tree und auf die Knotenliste der Unicast-Route. Im rechten Ausschnitt wird drei verschiedenen Sendern der gleichen Gruppe jeweils ein eigener Multicastbaum zugeordnet. Die Sender-ID ist die IP-Adresse.

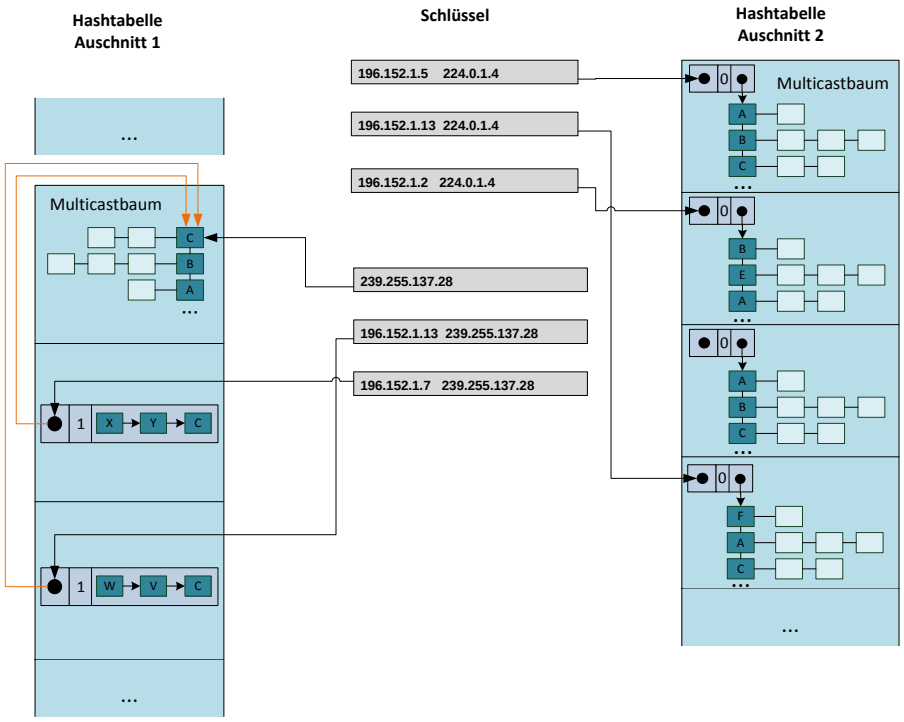

Abbildung 4.11: Datenstruktur der Routenverwaltung

4.7 Routenberechnung und Flow-Modifikation

Die Routenberechnung ist eine Kernaufgabe des Multicastdienstes. Die Daten aus Netzzustandsverwaltung, Gruppenverwaltung und Routenverwaltung werden für die Berechnung der Multicastbäume verwendet. Das Resultat ist eine Menge von Flow-Aktualisierungen, die in Form von OpenFlow *Modification*-Nachrichten an die Switches gesendet werden. Die Einrichtung geschieht über die Group-Table mit zugehörigen Flow-Table-Einträgen, die auf eine Gruppe verweisen. Rückmeldungen durch Status- und Fehlermeldungen geben Auskunft darüber, ob die Modifikationen erfolgreich waren.

In Abbildung 4.12 ist eine Übersicht der Aufgaben und der Datenflüsse in der Routenberechnung zu sehen. In der *Routingkontrolle* laufen alle benötigen Eingangsdaten zusammen. Daraufhin erfolgen die Auswahl des Routingalgorithmus und die Weitergabe des Ergebnisses an die *Flow-Modifikation*. Die Flow-Modifikation ist schließlich für die Implementierung der Multicastbäume auf der Datenschicht zuständig.

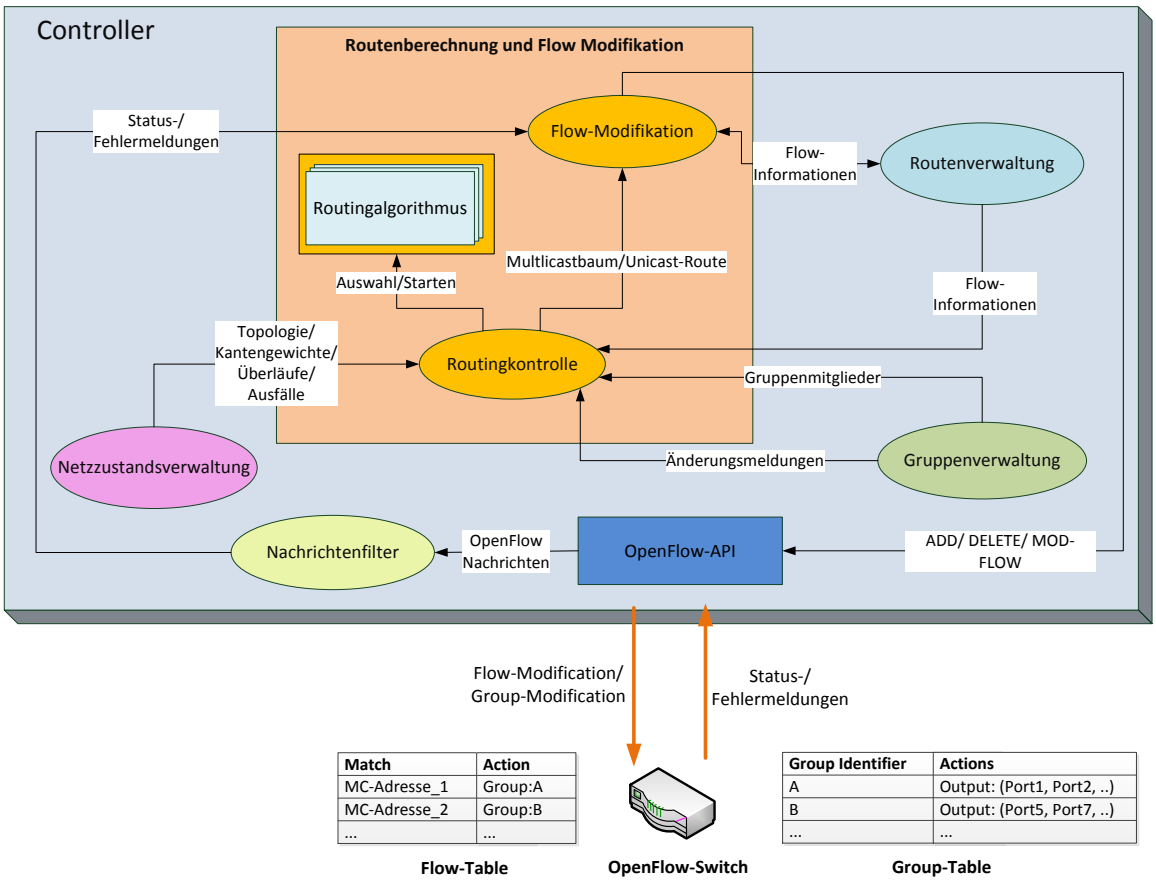

Abbildung 4.12: Übersicht zur Routenberechnung und Flow-Modifikation

4.7.1 Proaktives vs. reaktives Routing

Eine grundsätzliche Designentscheidung ist die Wahl zwischen proaktiver und reaktiver Routenberechnung.

Beim reaktiven Routing werden die Multicastbäume erst dann bestimmt, wenn tatsächlich eine Übertragung vom jeweiligen Sender stattfindet. Das hat zur Folge, dass Datenpakete auf die Einrichtung der Routen warten müssen, was im Vergleich zur einer direkten Weiterleitung eine erhebliche Verzögerung bedeutet. Angenommen ein Host startet erstmalig eine Übertragung an eine Multicastgruppe. Im Zuge dessen beginnt er einen *Daten-Burst* mit voller Bandbreite an seinen ToR-Switch zu senden. Betrachtet man heutiges 10-Gigabit Ethernet im Vergleich zu den verfügbaren Zwischenspeicherkapazitäten in den Netzwerkgeräten, kann ein Sender in der Zeit, die für die Aktualisierung der Flow-Table benötigt wird, die Eingangswarteschlange des ToR-Switches überflutet haben. Das hat dann neben erhöhten Verzögerungszeiten auch Paketverluste zur Folge und stellt ein grundsätzliches Problem dar.

Deshalb fällt die Wahl auf eine proaktive Routenberechnung. Diese Entscheidung ist unabhängig vom gewählten Routingalgorithmus und wird von der Routingkontrolle umgesetzt. Die Multicastbäume werden für jede vorhandene Gruppe und für jeden Sender initial auf Basis der aktuell vorliegenden Gruppeninformationen bestimmt. Tritt danach ein weiteres Ereignis, wie eine Gruppenänderung oder die Anfrage für eine neue Multicastgruppe auf, werden alle zur Gruppe gehörigen Bäume erneut vorberechnet und entsprechende Routen direkt in den Switches eingerichtet. Das erlaubt im Idealfall eine Weiterleitung in *Line-Rate* und stellt die Hauptmotivation für die Wahl dieses Ansatzs dar. Die Nachteile sind ein erhöhter Rechen- und Flow-Modifikationsaufwand, der direkt in Relation zu der Menge der Sender, Gruppengröße und der Anzahl der Änderungen steht. Bezogen auf das Systemmodell wird allerdings von einer vorher bekannten, statischen Anzahl von Hosts sowie von einem eher seltenen Auftreten von Gruppenänderungen ausgegangen. Die tatsächlich gemessenen Auswirkungen sind, abhängig von der Gruppendynamik, in der Evaluierung (Kapitel 6) zu sehen.

4.7.2 Routingkontrolle

Die *Routingkontrolle* ist ein Prozess, der den Ablauf in der Routenberechnung koordiniert und zusammen mit einem Routingalgorithmus die Kernlogik des Multicastdienstes implementiert. Abhängig von der aktuellen Netzsituation wird einer der drei vorhandenen Routingalgorithmen gewählt und auf den Eingangsdaten ausgeführt. Dabei kann sich die Routingkontrolle in zwei verschiedenen Modi befinden: Standard- oder Überlaufmodus. Sie signalisieren, ob genügend Flow-Table-Einträge zur Verfügung stehen, um optimale Bäume zu berechnen. Stehen genügend freie Einträge zur Verfügung, ist der Standardmodus aktiv und es wird eine monolithische Steinerheuristik gewählt, die quellenbasierte Multicastbäume berechnet. Für Test- und Evaluationszwecke, ist außerdem ein inkrementeller Algorithmus implementiert, auf den beliebig während des laufenden Betriebs umgeschaltet werden kann. Befindet sich die Routingkontrolle im Überlaufmodus, wird auf Shared-Trees gewechselt, um Tabelleneinträge zu sparen und einem Überlauf der Flow-Table vorzubeugen. Ein Umschalten zwischen den Modi kann jederzeit durch das Auftreten einer Überlaufwarnung geschehen und hat keinen Einfluss auf zuvor eingerichtete Routen. Überlaufwarnungen werden direkt von der Netzzustandsverwaltung gemeldet. Sie überwacht die Auslastung der Flow-Tables und setzt den Modus, abhängig von bestimmten Schwellwerten auf „Überlauf“, oder zurück auf „Standard“. Quellenbasierte- und geteilte Bäume (*Shared-Trees*) können in beliebiger Mischung koexistieren, sofern für jeden Sender nur eine Baumart zur gleichen Zeit genutzt wird. Findet sich sowohl ein Shared-Tree als auch ein quellenbasierter Baum auf Datenebene, wird letzterer bevorzugt verwendet.

Abhängig von den Eingangsmeldungen und dem Modus der Routingkontrolle gibt es folgende Fälle, die zu einem Anstoßen der jeweiligen Routingalgorithmen führen:

Gruppenänderung Tritt eine Gruppenänderung in Gruppe K auf, meldet die Gruppenverwaltung dies an die Routingkontrolle und verarbeitet die Änderung nach Abbildung 4.13. Wenn eine Gruppe aus der Gruppenverwaltung gelöscht wurde, wird die Flow-Modifikation angestoßen, um alle Bäume dieser Gruppe zu entfernen. Ansonsten bestimmt der Überlauf-

modus die Wahl zwischen quellbasiertem Baum und Shared-Tree. Bei quellbasierten Bäumen müssen alle n Senderhosts beachtet werden und so auch n Bäume berechnet werden. Bei einem Shared-Tree müssen zusätzlich zum eigentlichen Baum, Unicast-Routen für jeden ToR-Switch errechnet werden. Sie verbinden den jeweiligen Sender mit dem Shared-Tree. Nach Beendigung einer Berechnung wird das Ergebnis an die Flow-Modifikation gegeben, die zur Einrichtung der Routen dann die entsprechenden OpenFlow-Nachrichten an die beteiligten Switches schickt. Ein Sonderfall tritt bei der ersten Gruppenänderung nach dem Umschalten vom Überlaufmodus in den Standardmodus auf. Der Shared-Tree muss hier explizit gelöscht werden, da dieser extra unter der Multicastadresse als Hash-Schlüssel abgespeichert wurde. Diese Löschoperation wird in diesem Fall direkt von der Routingkontrolle initiiert und an die Flow-Modifikation gemeldet.

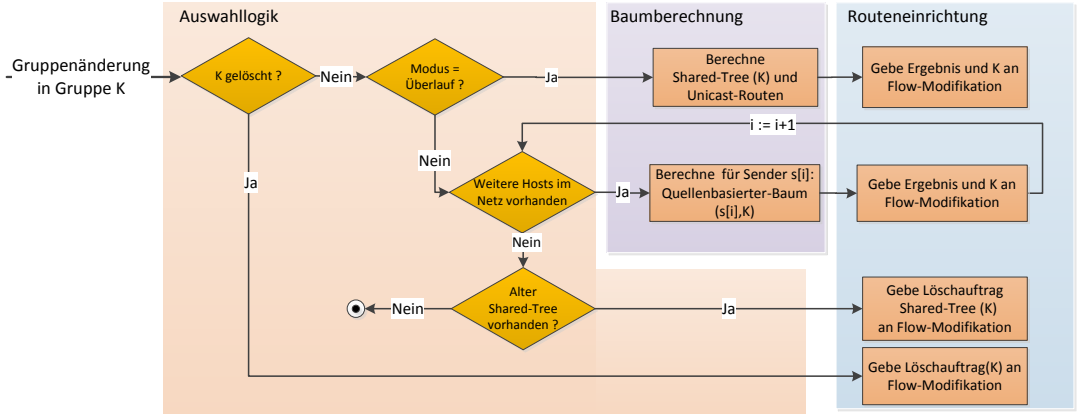

Abbildung 4.13: Verarbeitung von Gruppenänderungen in der Routingkontrolle

Switch- und Leitungsausfälle Das folgende Verhalten ist in Abbildung 4.14 dargestellt. Das Ziel ist, möglichst schnell auf Ausfälle zu reagieren zu können, um den Paketverlust auf einem Minimum zu halten. Ein Ausfall wird durch die Netzzustandsverwaltung gemeldet, indem die ausgefallenen Switches und Leitungen als Mengen (S,E) an die Routingkontrolle gemeldet werden. Die Menge E enthält nur ausgefallene Leitungen, die nicht zu einem Switch aus S gehören. Daraufhin wird mit Hilfe der Routenverwaltung bestimmt, welche Multicastbäume und Unicast-Routen vom Ausfall betroffen sind. Dazu müssen alle Adjazenzlisten geprüft werden, ob einer der ausgefallenen Knoten oder Kanten darin enthalten ist. Die Schnittstelle der Routenverwaltung stellt für diesen Zweck eine entsprechende Operation zur Verfügung. Sie gibt alle betroffenen Multicastbäume in Form ihrer eindeutigen Schlüssel (Sender-ID, Gruppen-ID) bzw. (Gruppen-ID) zurück. Mit der Gruppen-ID werden die zugehörigen Mitglieder von der Gruppenverwaltung abgefragt. Abhängig vom aktuellen Modus stößt die Routingkontrolle einen Routingalgorithmus an, um für die vom Ausfall betroffene Menge an Bäumen eine Alternativroute zu berechnen. Die ausgefallenen Kanten werden zuvor im Netzgraph mit unendlich markiert, um nicht auf den nächsten Aktualisierungszyklus der Netzstrukturverwaltung warten zu müssen. Nach der Neuberechnung werden die Änderungen über die Flow-Modifikation zur Datenebene gemeldet.

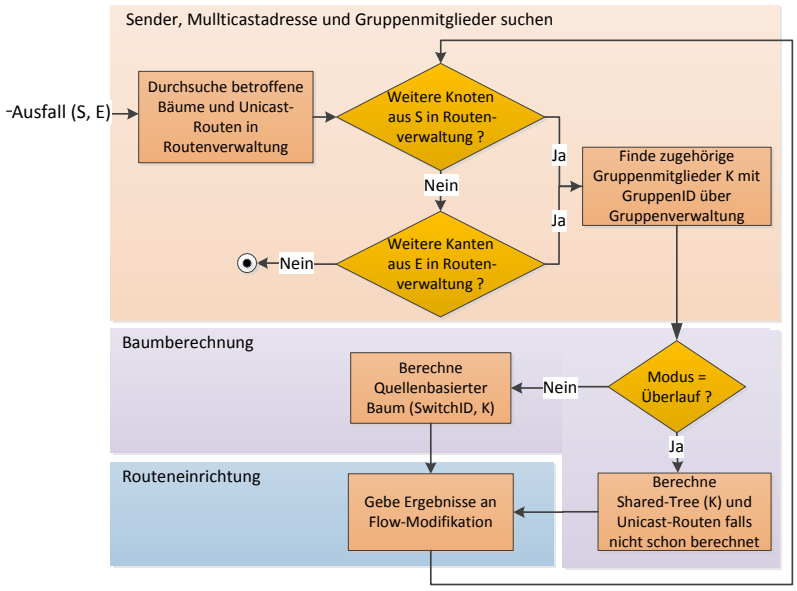

Abbildung 4.14: Verarbeitung von Switch- und Leitungsausfällen in der Routingkontrolle

4.7.3 Routing-Algorithmen

Die Aufgabe eines Routingalgorithmus ist es, auf dem Netzgraph G einen Multicastbaum S_K für eine Gruppe K zu berechnen. Die optimale Lösung dafür ist durch das Steinerbaumproblem nach Abschnitt 2.3.1 definiert. Eine Instanz des Problems ist mit $(G(V, E, w), K)$ gegeben, mit dem Ziel, die Kosten des Multicastbaumes S_K bezüglich w zu minimieren. Vor jeder Berechnung wird der kantengewichtete Graph der Netzzustandsverwaltung mit den aktuellen Gewichten $w_{(s,K)}$ des betroffenen Multicastbaumes verrechnet, um die eigene Bandbreite vom Gesamtgewicht abzuziehen. Somit steht für jeden Baum ein Graph $G(V, E, w)$ mit individuellen Kantengewichten w zur Verfügung.

In dieser Arbeit werden drei Algorithmen implementiert, zwischen diesen im laufenden Betrieb beliebig umgeschaltet werden kann. Die KMB-Steinerheuristik [KMB81], ein Algorithmus für Shared-Trees und ein inkrementeller Greedy-Algorithmus (iGA). KMB ist eine Näherung für den minimalen Steinerbaum mit polynomiellen Zeitaufwand und findet hier für die Berechnung der quellenbasierten Bäume Anwendung. Im Vergleich ermittelt KMB die hier kostenoptimalste Lösung. Für die Berechnung der Shared-Trees wird ein Rendezvous-Knoten verwendet. Damit muss nur ein Multicastbaum pro Gruppe berechnet werden. Der Baum ist jedoch lediglich für den Rendezvous-Knoten optimiert und nicht für jede Quelle. Außerdem sind Unicast-Routen nötig, die die Sender über eine effiziente Route mit dem Shared-Tree verbinden. Während die ersten beiden Algorithmen monolithisch arbeiten, bietet der Greedy-Algorithmus (iGA) eine Vergleichsmöglichkeit zu einem quellbasierten inkrementellen Lösungsansatz. Dieser Ansatz spart eine komplexe Neueinrichtung sämtlicher Routen nach aufgetretener Gruppenänderung. Jedoch divergiert der Baum nach einer gewissen Anzahl von Änderungen immer mehr von der optimalen Lösung.

Alle drei vorgestellten Algorithmen haben eine polynomielle Laufzeit. Abhängig von der Optimalität des Algorithmus werden die Kosten entsprechend der in 4.4.3 eingeführten Metrik minimiert und dadurch ein *Flow-Balancing* erreicht. Folgende Eingangsdaten stehen den Routingalgorithmen dafür zur Verfügung:

- Repräsentation des aktuellen Netzes als kantengewichteter Graph $G(V, E, w(e))$, inklusive aller ToR-Switches $s \in V$ als mögliche Senderknoten und die zugehörigen Gewichte $w_{(s,K)}(e)$ für alle bereits vorhanden Multicastbäume.

- Gruppenmitglieder aus der Gruppenverwaltung als Terminalmenge T. Die Gesamtmenge bezüglich des Senderknotens s wird mit $K := T \cup \{s\}$ bezeichnet. Im Fall eines Shared-Tree, ist s der Rendezvous-Knoten.

- Flow-Informationen aus der Routenverwaltung, die die Menge der im Moment aktuellen Multicastbäume darstellen.

KMB Steinerheuristik Der KMB-Algorithmus von Kou et al. [KMB81] ist eine Steinerheuristik und basiert auf minimalen Spannbäumen. Sie ist in Algorithmus 1 dargestellt. KMB wird hier im Standardmodus der Routenberechnung verwendet, um minimale quellenbasierte Multicastbäume zu approximieren. Die optimale Lösung wird dabei 2-approximiert, d.h. die Kosten, hier Länge und Auslastung, des berechneten Steinerbaumes sind maximal doppelt so hoch wie der minimale Steinerbaum. Die Zeitkomplexität beträgt im schlechtesten Fall $O(|K||V|^2)$.

Algorithmus 1: KMB: 2-Approximationsalgorithmus nach [KMB81]

Eingabe : $(G(V, E, w), K)$
Ausgabe : Steinerbaum S_K von G

$G^*(K, E_D, w_D) := SUB(G(V, E, w))$;
$S^* := \text{MST}(G^*)$;
$G_{S^*} := RSUB(S^*)$;
$S_K^+ := MST(G_{S^*})$;
while S_K^+ *hat Blattknoten* $v \notin K$ **do**
  entferne v aus S_K^+
end
return S_K^+

Zuerst wird der Distanzgraph G^* über die SUB Funktion berechnet. Dabei werden die Kanten zwischen allen Knotenpaaren aus K durch ihren kürzesten Pfad subsituiert. Die Kanten E_D stellen aggregierte Kanten der kürzesten Pfade aus G mit summierten Gewichtswerten w_D dar. Im Distanzgraph wird nun ein minimaler Spannbaum berechnet (MST-Funktion), der dann anschließend wieder in einen Teilgraph mit den Ursprungskanten aufgelöst wird (RSUB). Eine Kante in S^* entspricht einem kürzesten Pfad in G, d.h. nach der Rücksubstitution, ist der neu entstandene Teilgraph G_{S^*} im Allgemeinen kein Baum mehr. Auf diesem Teilgraph wird nun erneut ein minimaler Spannbaum berechnet. Anschließend werden überflüssige Blattknoten entfernt, so dass nur noch die Blätter aus K vorhanden sind (PRUNE).

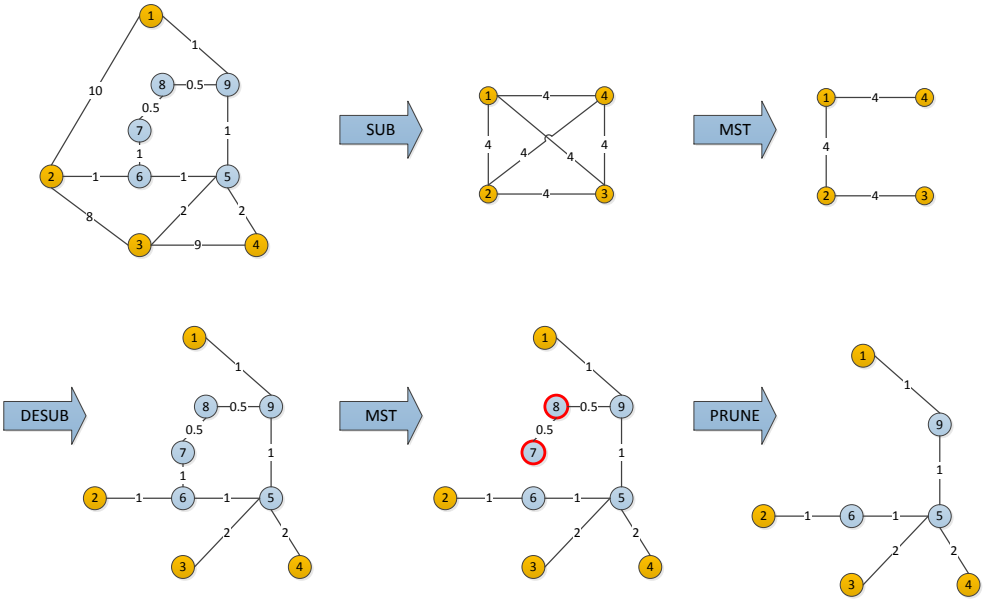

Abbildung 4.15: Beispiel für den KMB-Algorithmus aus [KMB81]

Abbildung 4.15 zeigt in einem kantengewichteten Beispielgraphen die einzelnen Schritte aus Algorithmus 1. Dabei besteht die Knotenmenge aus neun durchnummerierten Knoten und es gilt $K = \{1, 2, 3, 4\}$. Im Bild sind die Zielknoten gelb und alle anderen blau dargestellt. Eine rote Umrandung steht für überflüssige Baumzweige, die im letzten Schritt gestutzt werden können. Für die Berechnung der kürzesten Pfade in KMB kommt Dijkstra [Dij59] zum Einsatz. Um den minimalen Spannbaum zu finden, wird der Algorithmus von Tarjan [Tar06] verwendet. Dabei handelt es sich um eine effizientere Implementierung von Edmond's Algorithmus [Edm67] für optimale Kantenverzweigungen (*Branching*) in gerichteten Graphen. Technisch gesehen handelt es sich bei dem betrachteten Systemmodell um einen gerichteten Graphen, da der Vollduplex-Betrieb zwischen zwei Switches auch zwei unabhängige gerichtete Kanten im Graphen impliziert, die insbesondere auch verschiedene Kantengewichte haben. Das minimale Spannbaumproblem wird im gerichteten Fall durch einen gewurzelten Baum (*rooted Tree*) gelöst. Die Wurzel ist in diesem Fall der Senderknoten. Tarjan's Algorithmus hat eine Zeitkomplexität von $O(|E|log|V|)$ für dünn besiedelte Graphen (sonst $O(|V|^2)$) und erreicht damit die gleiche Performance wie der bekannte Algorithmus von Prim [Pri57] für ungerichtete Graphen. Für eine verbessertes Zeitverhalten kann die Laufzeit von Dijkstra's Algorithmus durch die Implementierung mit *Fibonacci-Heaps*, von ursprünglich $O(|V|^2 + |E|)$ auf $O(|E| + |V|log|V|)$, noch weiter reduziert werden [FT87].

Der KMB-Algorithmus gibt folgende Garantie zwischen der Gesamtpfadlänge von S_K und dem minimalen Steinerbaum S_{opt}.

$$|S_K| \leq 2(1 - \tfrac{1}{|K|})|S_{opt}|, \quad mit\ |S| = \sum_{e\ aus\ S} w(e) \tag{4.8}$$

In der Praxis kommt der schlechteste Fall selten vor, so dass im Schnitt das Ergebnis nur um 5% vom Optimum abweicht [DL93]. Es gibt derzeit keinen Polynomialzeit-Algorithmus, der das Steinerbaumproblem besser als 2-Approximiert (im *worst case*) [Kos10]. Da die Berechnung für jeden Sender pro Multicastgruppe passieren muss, ist der Gesamtaufwand im betrachteten Systemmodell für eine Neuberechnung nach oben durch $k^2/2 \times O(|K||V|^2)$ begrenzt. Dabei ist k der Fat-Tree Parameter, der die Anzahl der Pods im Systemmodell angibt. Pro Pod gibt es $k/2$ Tor-Switches, die jeweils eine Senderrolle einnehmen können.

Shared-Tree Algorithmus mit Rendezvous-Knoten Der Shared-Tree Algorithmus wird angewandt, wenn die Flow- bzw. Group-Tables zu viele Einträge haben und ein Überlauf droht. Im Vergleich zu quellenbasierten Bäumen, ist ein Shared-Tree für die Weiterleitung aller Pakete sämtlicher Sender (ToR-Switches) für eine Multicastgruppe zuständig. Nachdem die Routingkontrolle in den Überlaufmodus wechselt, findet automatisch ein Wechsel des Routingalgorithmus statt. Alle danach auftretenden Gruppen-Ereignisse werden anschließend als Shared-Tree behandelt.

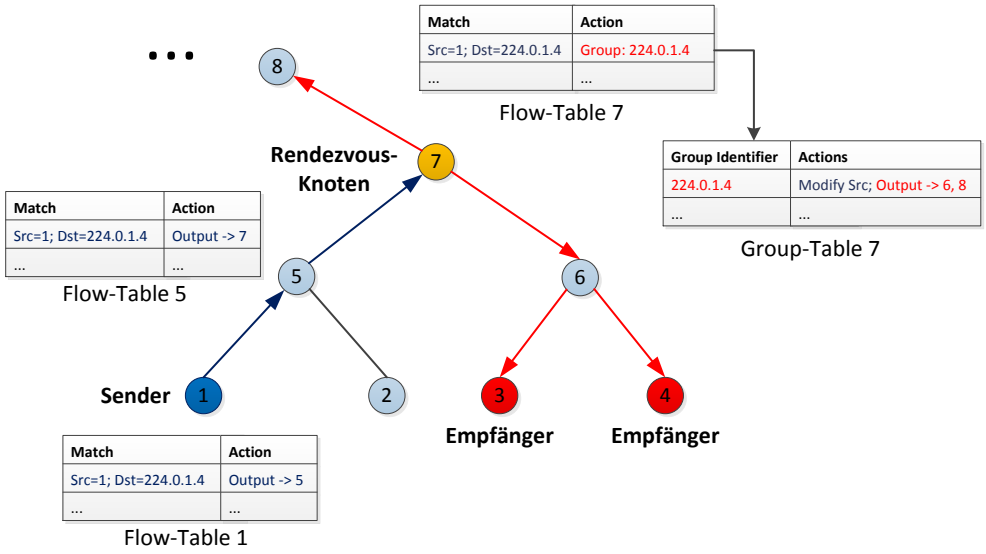

Abbildung 4.16: Shared-Tree mit zugehörigen Flow-Tables

Tritt nach dem Wechsel in den Überlaufmodus eine neue Gruppe oder eine Gruppenänderung auf, kommt zuerst KMB zum Einsatz, um einen neuen Baum zu berechnen. Als Senderquelle wird ein fester Rendezvous-Knoten vorgegeben, der vorab ermittelt wurde. Nach der erfolgreichen Berechnung des Shared-Trees müssen die Routen ermittelt werden, die die Sender mit dem Baum verbinden. Von jedem ToR-Switch wird dazu proaktiv eine kürzeste Unicast-Route zum Rendezvous-Knoten mit Dijkstra's-Algorithmus [Dij59] errechnet.

Die errechneten Unicast-Routen bilden auf der Datenschicht die Wege, die alle Multicastpakete einer Gruppe an den Rendezvous-Knoten weiterleiten. Erreichen die Pakete den Rendezvous-Knoten, startet die eigentliche Auslieferung an die Empfänger. Errechnete Unicast-Routen werden zusammen mit dem Shared-Tree an die Flow-Modifikation gegeben und auf Datenschicht eingerichtet. In Bild 4.16 ist ein Ausschnitt eines Shared-Trees mit zugehörigen

Flow-Tables gezeigt. Der Sender schickt seine Pakete entlang der blauen Unicast-Route in Richtung Shared-Tree, bis er auf den Rendezvous-Knoten trifft. Im Bild entspricht das Knoten 7, der Multicastpakete von Sender 1 mit der Zieladresse „224.0.1.4“ empfängt. Von dort aus startet die Auslieferung entlang des Baumes (in rot). Dabei ist der Baum über die Group-Table definiert, auf die ein Flow-Table-Eintrag verweist. Der Senderknoten ist dunkelblau dargestellt, die Gruppenmitglieder in rot.

Eine wichtige Fragestellung betrifft die Auswahl des Rendezvous-Knoten. Dies hat direkten Einfluss auf die Verzögerungszeit und Bandbreitenausnutzung im Netz. Eine schlechte Knotenwahl kann unnötig lange und ungünstige Pfade zur Folge haben. Eine Möglichkeit ist die zufällige Platzierung. Dadurch wird zwar ab einer gewissen Anzahl von Mullticastgruppen eine Gleichverteilung erreicht, die einzelnen Routen sind jedoch nicht optimal gewählt. Für eine Optimierung ist das offensichtliche Ziel, den Rendezvous-Knoten in der Mitte des Netzwerks zu platzieren, so dass die maximalen Kosten für jeden potentiellen Sender und Empfänger minimiert werden.

Für jede Multicastgruppe wird ein eigener Rendezvous-Knoten berechnet. Der Switch, zu dem alle Empfängerknoten den kleinsten Abstand bezüglich der in Abschnitt 4.3 vorgestellten Metrik haben, wird als die Mitte dieser Knoten bezeichnet. Um die kürzesten Pfade zwischen allen Switches und der Empfängergruppe zu bestimmen, kommt eine n-malige Ausführung des Dijkstra-Algorithmus zum Einsatz. Danach wird für jeden Knoten der größte Abstand zu einem Gruppenmitglied verglichen. Es muss nur noch der Knoten im Netz ausgewählt werden, bei dem dieser maximale Abstand für alle Empfänger am kleinsten ist. Sollte dies auf mehrere Kandidaten zutreffen, wird der nächstbeste als die neue Graphmitte für die Multicastgruppe ausgewählt. Der Zeitaufwand wird durch das Berechnen der kürzesten Pfade bestimmt und beträgt $O(n^3)$, wobei n die Anzahl der Switches im Netz darstellt. Senderknoten werden bei der Berechnung des Rendezvous-Knoten nicht extra behandelt, da jeder ToR-Switch zu jedem Zeitpunkt eine Senderrolle für eine beliebige Gruppe annehmen kann.

Inkrementeller Greedy-Algorithmus (iGA) Im dynamischen Fall wird das Steinerbaumproblem aus Kapitel 2.3.1, um ein $R := \{(v_i, o_i) | v_i \in V, o_i \in \{add, leave\}\}$ erweitert. R besteht aus einer Menge von Änderungsanfragen, bei denen Knoten der Terminalgruppe beitreten oder sie verlassen können. Das Ziel ist nach jeder Änderung (v_i, o_i) einen Steinerbaum S_i zu berechnen, der die veränderte Terminalmenge K_i umspannt. Die Änderungsanfragen aus der Menge R treten iterativ und plötzlich auf (online), und sind einem (Online-)Algorithmus, der dieses Problem lösen soll, zuvor nicht bekannt. Im Gegensatz zu KMB oder Shared-Tree, arbeitet ein solcher Lösungsansatz stets auf den aktuellen Bauminformationen aus der Routenverwaltung.

Das kommt einem Algorithmus gleich, der inkrementell arbeitet. Das heißt der aktuell berechnete Baum wird bei einer Änderung aktualisiert, anstatt ihn neu zu berechnen. Die Grundvoraussetzung ist, dass bereits ein durch KMB berechneter initialer Multicastbaum für jede Quelle existiert und die Routenkontrolle sich im Standard-Modus befindet. Findet sich über die Routenverwaltung kein quellenbasierter Multicastbaum, wird ein solcher erst errechnet.

Algorithmus 2 beschreibt die iterative Berechnungsvorschrift zur Anpassung eines bestehenden quellenbasierten Baumes bei aufgetretener Gruppenänderung.

Algorithmus 2: Online Greedy-Algorithmus

Eingabe : $(G(V, E, w), S_i(K_i), (v_i, o_i)$
Ausgabe : Steinerbaum $S_{i+1}(K_{i+1})$ der K_{i+1} umspannt
if $o_i = add$ **then**
| Verbinde v_i mit S_i auf kürzestem Pfad
else
| Markiere v_i als gelöscht. Stutze ausgehend von v_i alle unnötigen Kanten in S_{i+1}
end

Gruppenänderungen (v_i, o_i) werden durch die Gruppenverwaltung nacheinander einzeln gemeldet, der zugehörige Baum $S_{(i-1)}$ findet sich in der Routenverwaltung. Der Online-Greedy-Algorithmus (Algorithmus 2) verbindet bei einer Änderung (v_i, add) den hinzugekommenen Knoten v_i mit der Terminalmenge K_{i-1} auf dem kürzesten Pfad zu S_{i-1}. D.h. mit dem Knoten aus $S_{(i-1)}$, der am nächsten zu v_i liegt. Beim Verlassen eines Knotens wird der Zweig einfach gestutzt (*Pruning*).

Der Vorteil dieses Algorithmus ist, dass er wenige Änderungen auf Datenschicht zur Folge hat. Da dies ein wesentlicher Zeitfaktor ist, wurde Algorithmus 2 in dieser Diplomarbeit implementiert, um den tatsächlichen Zusatzaufwand mit der Optimalität des Routings vergleichen zu können. Imaze und Waxmann [IW91] haben gezeigt, dass das *Competitive Ratio* eines solchen Algorithmus bei $log_2 N$ liegt. Das *Competitive Ratio* ist das Kostenmaximum eines Online-Algorithmus aller Änderungsanfragen zum Verhältnis zu den Kosten für einen optimalen offline Baum, bei dem die Änderungen schon im Voraus bekannt sind.

4.7.4 Flow-Modifikation

Die *Flow-Modifikation* ist für die Implementierung der Routingergebnisse auf Datenebene und für die Aktualisierung der Routenverwaltung zuständig. Ziel ist es, möglichst wenig Änderungen auf Datenebene durchführen zu müssen. Das ist insbesondere für den Fall relevant, wenn nach einer Neuberechnung alte, aber immer noch gültige Tabelleneinträge beibehalten werden können. Das bedeutet, dass nach Änderungen ein Vergleich zwischen dem neuen und dem bisherigen Baum stattfinden muss, um die Unterschiede festzustellen.

Zur Einrichtung eines neuen Multicastbaumes werden OpenFlow *Modification*-Nachrichten vom Typ *Add*, *Delete* oder *Mod* an die entsprechenden Switches gesendet. An jedem Switch sind ein Gruppeneintrag in der Group-Table, sowie ein zugehöriger Verweis in der Flow-Table nötig. Das Deployment wird als erfolgreich angesehen, wenn nach einer ausreichend langen Wartezeit keine Fehlermeldung zurückgemeldet wurde. Erst dann darf die Flow-Modifikation die Datenbasis der Routenverwaltung aus Abschnitt 4.6.1 aktualisieren. Zu beachten ist, dass die eingerichteten Flows keine gültigen Timeoutzeiten aufweisen dürfen. Das würde im Widerspruch zum proaktiven Berechnungsansatz stehen. Somit wird die Flow-Modifikation gezwungen, vor der Einrichtung der neuen Routen, nunmehr veraltete Einträge explizit zu löschen.

Der Ablauf in der Flow-Modifikation wird in *Lösch-*, *Deployment-* und *Aktualisierungsphase* gegliedert. Nachfolgend werden die einzelnen Ereignisse unterschieden, die zu einem Aufruf der Flow-Modifikation führen:

- **Neuer quellenbasierter Multicastbaum**: Nach der Ausführung einer der beiden Routingalgorithmen für quellenbasierte Multicastbäume wird das Ergebnis, zusammen mit der Gruppen-ID und der Sender-ID, als Adjazenzliste an die Flow-Modifikation übergeben. Mit dem Schlüssel (Sender-ID+Gruppen-ID) kann der vorherige Multicastbaum aus der Routenverwaltung gesucht werden. Im Anschluss folgt die Lösch- und Deployment-Phase für Multicastbäume.

- **Neuer Shared-Tree**: Nach der Berechnung eines neuen Shared-Trees wird der Baum als Adjazenzliste zusammen mit der Gruppen-ID weitergegeben. Der passende Eintrag in der Routenverwaltung wird dann über den Schlüssel (Gruppen-ID) bestimmt. Anschließend wird ebenfalls mit der Lösch- und Deployment-Phase für Multicastbäume fortgefahren.

- **Neuer Sender für Shared-Tree**: Für jeden im Graph vorhandenen Sender (Sender-ID) wird, nach der Berechnung eines neuen Shared-Trees, die zugehörige Knotenliste für eine Unicast-Route zum Rendezvous-Knoten übergeben. Außerdem ist die Gruppen-ID von Bedeutung. Der Lookup in der Routenverwaltung erfolgt über (Sender-ID+Gruppen-ID) um die bisherige Verbindungsroute zu erhalten. Anschließend wird die Lösch- und Deployment-Phase für Unicast-Routen ausgeführt

- **Gruppe löschen**: Die Gruppenverwaltung meldet das Löschen einer Gruppe mit der Gruppen-ID an die Routingkontrolle. Da ein solches Ereignis nicht zu einer Neuberechnung führt, wird die Flow-Modifikation direkt benachrichtigt. Alle zugehörigen Einträge auf der Datenebene werden daraufhin gelöscht und die Routenverwaltung aktualisiert.

Lösch- und Deployment-Phase für Multicastbäume Der Ablauf nach der Übergabe eines neuen Multicastbaumes an die Flow-Modifikation ist mit den einzelnen Phasen in Abbildung 4.17 zu sehen. Ergibt der Lookup über die Routenverwaltung kein Ergebnis, kann die Löschphase übersprungen und direkt mit der Deployment-Phase für Multicastbäume fortgefahren werden. Ansonsten unterscheidet ein *Flag* in der Datenstruktur, ob eine Unicast-Route oder ein Multicastbaum gefunden wurde. Bei einer Unicast-Route werden alle zugehörigen Tabelleneinträge gelöscht. Alle beteiligten Switches erhalten dazu eine *Flow-Table Modification*-Nachricht mit *Delete*-Anweisung. Anschließend kann direkt mit der der Deployment-Phase für Multicastbäume fortgefahren werden.

Ist durch den Lookup in der Routenverwaltung ein Multicastbaum erkannt worden, findet ein Vergleich zwischen dem neu berechneten und dem aktuellen Baum statt. Zuerst werden all diejenigen Knoten bestimmt, die im alten Baum, aber nicht im neuen vorhanden sind. Sie werden mit V_a bezeichnet. Dadurch werden die Switches identifiziert, bei denen der gesamte Gruppeneintrag gelöscht werden kann. Sie erhalten jeweils eine *Group-Table Modification*-Nachricht mit der *Delete*-Anweisung für den passenden Tabelleneintrag,

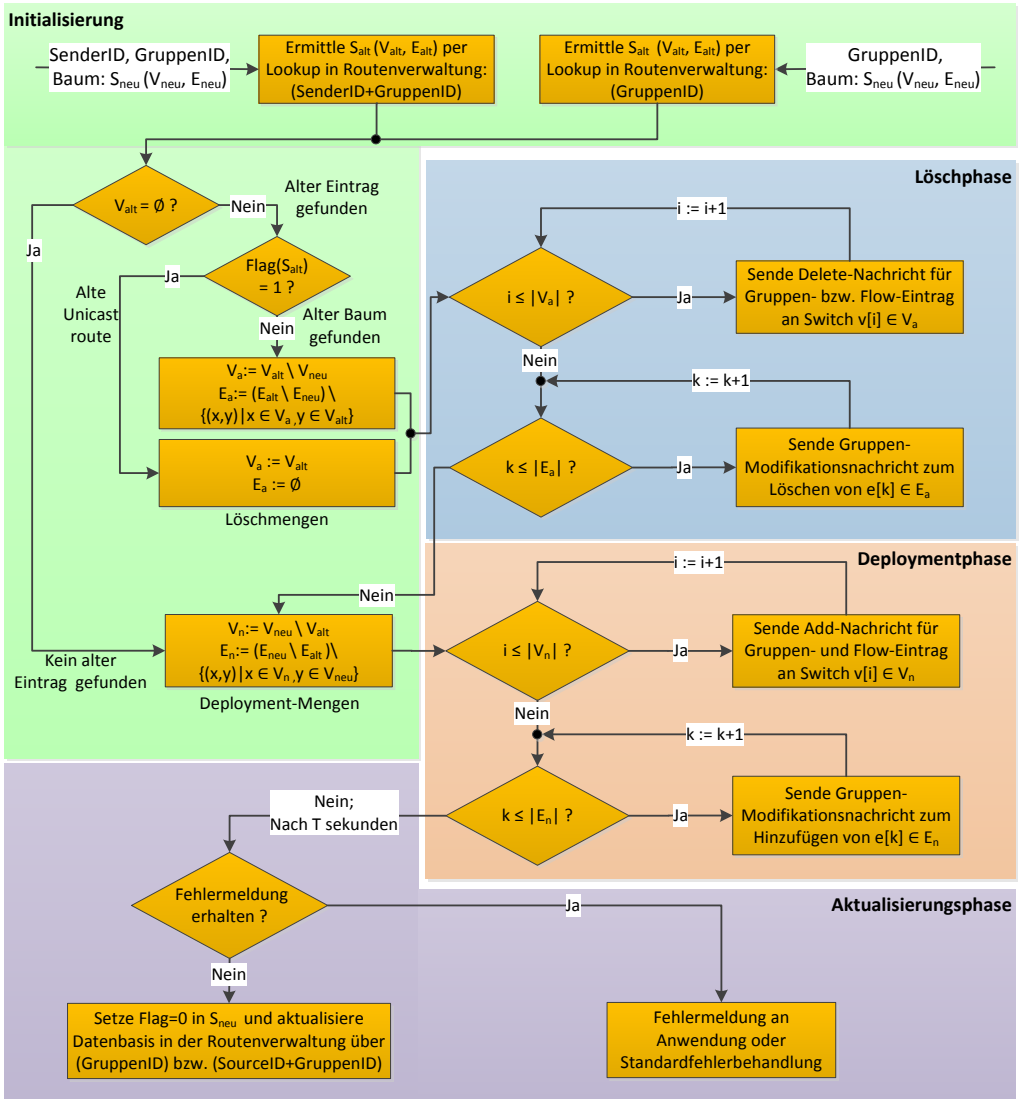

Abbildung 4.17: Ablauf und Phasen der Flow-Modifikation

der über die Multicastadresse identifiziert wird. Anschließend muss der zugehörige Flow-Table Eintrag entfernt werden, der auf die nun gelöschte Gruppe verweist. Nach erfolgreicher Löschmeldung, geschieht dieselbe Betrachtung für hinfällige Kanten, wobei nur noch solche Kanten beachtet werden, die nicht bereits zuvor mit einem Knoten gelöscht wurden (E_a). Eine nichtleere Kantenmenge E_a repräsentiert den Fall, dass der neue Baum teilweise über die gleichen Switches verläuft wie bisher, jedoch nicht mehr über dieselben Ein- und Ausgangskanten. Alle Kanten, die im neuen Baum nicht mehr vorkommen, werden über eine *Group-Table Modification*-Nachricht mit der *Modify*-Anweisung gelöscht. Das erfolgt durch Entfernen der passenden *Output:Port*-Aktion im Gruppeneintrag. Anschließend wird mit der Deployment-Phase für Multicastbäume fortgefahren.

Das Deployment läuft analog zur Löschphase ab. Unabhängig vom vorliegenden Baum, werden zuerst alle Knoten bestimmt, die neu hinzugekommen sind. Sie werden mit V_n bezeichnet. Die Menge der Kanten zu einem dieser Knoten definiert am zugehörigen Switch die Erstellung eines neuen Gruppeneintrags. Die passende *Group-Table-Modification*-Nachricht beinhaltet eine *ADD*-Anweisung, sowie die nötigen Informationen zur Gruppen-ID und den Kanten. In der Group-Table wird daraufhin ein neuer Eintrag erstellt, dem so viele *Action Buckets* zugeordnet werden, wie Kanten existieren. Alle Kanten werden auf die passenden Ports abgebildet und jedem *Bucket* eine *Output:Port*-Aktion zugeordnet. Der Gruppentyp muss auf *all* gesetzt werden, um sicherzustellen, dass eine Nachricht über jeden spezifizierten Port weitergeleitet wird. Der Eintrag der Flow-Table, der über die Group-Table behandelt werden soll, wird mit einer *Group-Action* versehen. Eine in der Group-Table definierte Gruppe kann über einen Verweis in der *Group-Action* mit einem 32-bit *Group Identifier* identifiziert werden. Dafür wird für quellenbasierte Bäume ein Hashwert aus Senderquelle und Multicastadresse verwendet, Shared Trees nutzen einfach die Multicastadresse. Zum Schluss muss per *Flow-Table Modification* über *ADD* ein neuer Eintrag in der Flow-Table erstellt werden. Der Eintrag hat die Aufgabe alle Pakete, die an die Multicastadresse geschickt werden, an den eben erstellten Gruppeneintrag zu verweisen. Nachdem alle neuen Knoten verarbeitet wurden, erfolgt die Einrichtung der neu hinzugekommenen Kantenmenge E_n. Dabei werden nur Kanten betrachtet, die nicht bereits im Zuge eines neuen Knoten hinzugefügt werden. Sie werden durch Hinzufügen einer entsprechenden *Output:Port*-Aktion für einen vorhandenen Gruppeneintrag per *Modify*-Anweisung behandelt. Der Eintrag in der Flow-Table muss nicht verändert werden.

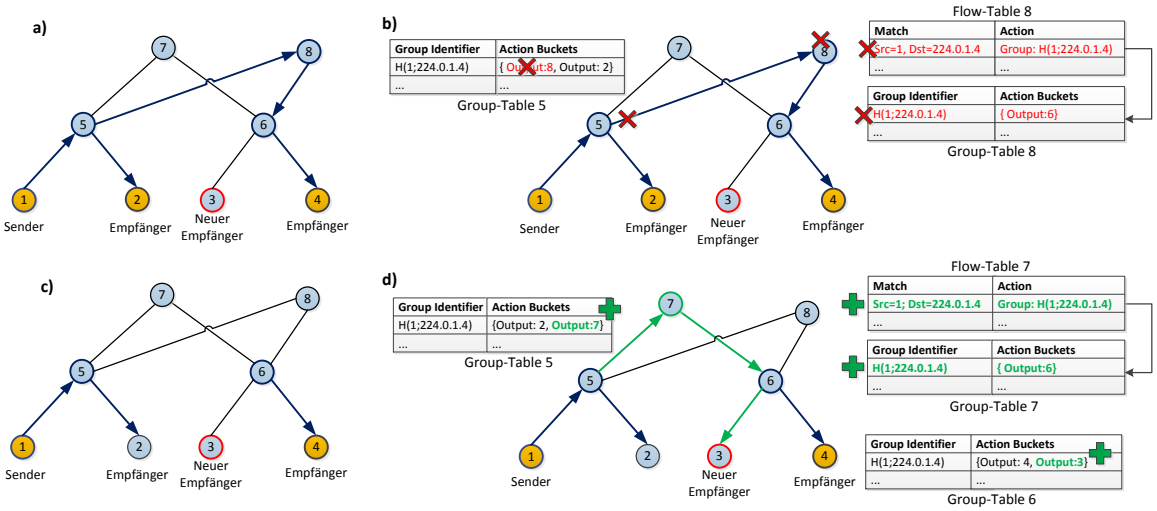

Abbildung 4.18: Lösch- und Deployment-Phase für einen quellenbasierten Multicastbaum

Abbildung 4.18 zeigt die Abfolge der Lösch- und Deployment-Phase auf Datenschicht am Beispiel eines quellenbasierten Multicastbaumes. Knoten 3 ist neu hinzugekommen, was eine Neuberechnung zur Folge hat. Der alte Baum in 4.18.a beinhaltet Knoten 8, während der neue Baum über Knoten 7 verläuft (4.18.d). Das hat das Löschen von Knoten 8 sowie der Kante von 5 zur Folge (4.18.b). Dazu müssen die gezeigten Tabelleneinträge (in rot) gelöscht bzw. modifiziert werden. In 4.18.c ist der Baum nach erfolgreicher Löschphase dargestellt. In Teilbild 4.18.d wird im Zuge der Deployment-Phase Knoten 7 hinzugefügt, sowie die neuen

Kanten von Knoten 5 und 6. Dafür müssen die gezeigten Tabelleneinträge (in grün) eingerichtet bzw. modifiziert werden. Die Funktion H stellt eine beliebige Hashfunktion dar, deren einzige Aufgabe es ist, eine Kombination aus Sender-ID und Multicastadresse auf einen 32-bit langen Identifier abzubilden.

Die Priorität sämtlicher Flow-Einträge wird mit einem festen Wert vorgegeben. Dabei gilt die Konvention, dass eine Baumkante höher priorisiert wird, als eine Unicast-Route. Außerdem erhalten quellenbasierte Baumkanten eine höhere Priorität als ein Shared-Tree, für den Fall dass beide Baumarten zur gleichen Zeit vorhanden sind. Es ist jedoch nicht erlaubt, dass für den gleichen Sender zu einem Zeitpunkt sowohl eine Unicast-Route als auch ein quellenbasierter Baum existiert. Somit bedarf dieser Fall keiner gesonderten Prioritätsbehandlung. Der Zeitaufwand für die Vergleichsoperationen beim Löschen und Deployment kann jeweils durch $O(|V_a|^2 + |E_a|^2)$ bzw. $O(|V_n|^2 + |E_n|^2)$ nach oben abgeschätzt werden. Dabei entsprechen die Mengen der Definition aus Abbildung 4.17 für die jeweilige Phase. Es muss allerdings angemerkt werden, dass E_a bzw. E_n bereits keine Kanten mehr aus V_a bzw. V_n enthalten und die Anzahl der Kanten pro Knoten durch k im Systemmodell beschränkt ist (Kapitel 3.2). Außerdem beziehen sich die Ursprungsmengen V und E auf einen Baum S_T und nicht auf den gesamten Graph.

Jeder Flow, der in einem Switch eingerichtet wird, muss über einen Bezeichner eindeutig identifizierbar sein. Außerdem dürfen keine uneindeutigen Überschneidungen bei den Abgleichkriterien entstehen. Bei quellenbasierten Bäumen können sich die einzelnen Bäume einer Gruppe überlappen, was bedeutet, dass ein Abgleich auf die Multicastadresse in dem Fall nicht ausreichend ist. Für eine Eindeutige Unterscheidung der Flows zwischen den jeweiligen Absendern, muss zusätzlich die Senderadresse abgeglichen werden. Der Controller selbst verwaltet jedoch keine Hosts, d.h. der Host übergibt nach dem ersten Hop die Senderrolle an seinen ToR-Switch. Eine OpenFlow Action vom Typ *Modify-Field* überschreibt die MAC-Adresse des Senders mit der des Switches. Dies findet immer dann statt, wenn ein ToR-Switch eine Multicastnachricht über einen Host-Port empfängt. Diese Lösung erreicht eine Unabhängigkeit von den eigentlichen Hosts und spart Tabelleneinträge in der Hinsicht, dass nicht für jede einzelne Sendermaschine eigene Flows benötigt werden. Dafür ist jedoch beim Empfang auf einem ToR-Switch ein zusätzlicher Flow nötig, der die Multicastpakete letztendlich über alle Host-Ports ausliefert. Bei einem Shared-Tree ist dafür außerdem eine zusätzliche Unterscheidung am Rendezvous-Knoten nötig, da die ToR-Switches hier im Allgemeinen nicht dem Sender-Switch des Baumes entsprechen. Zuerst werden die Pakete über Unicast-Routen zum Rendezvous-Knoten geschickt. Bis zum Erreichen des Baumes werden sie über ihre ToR-Senderadresse abgeglichen. Anschließend erfolgt am Rendezvous-Knoten ein erneutes Umschreiben der Addresse in die Adresse des Shared-Trees (Rendevous-Knoten-ID + Multicastadresse) wie in Bild 4.16 gezeigt.

Lösch- und Deployment-Phase für Unicast-Routen Die Lösch- und Deployment-Phase für Unicast-Routen erfolgt analog zu den Phasen für Multicastbäume. Wenn sich in der Routenverwaltung noch kein Eintrag findet wird die Löschphase übersprungen und die neue Route direkt eingerichtet. Anderenfalls wird zwischen einem quellenbasierten Baum oder einer Unicast-Route unterschieden. Findet sich ein Baum, wird dieser komplett gelöscht und danach direkt mit der Deployment-Phase für Unicast-Routen fortgefahren. Wurde eine

Unicast-Route erkannt, werden die Knoten und Kanten verglichen, alte Tabelleneinträge gelöscht und anschließend neue hinzugefügt. Kanten, die sowohl in der alten als auch in der neuen Route vorhanden waren, werden beibehalten. Der einzige Unterschied zur Lösch- und Deployment-Phase für Multicastbäume ist, dass die Routen über die Flow-Table eingerichtet werden, während die Bäume über die Group-Table definiert werden. Die entsprechenden OpenFlow-Nachrichten sind: *Flow-Table Modification*-Nachrichten mit den jeweiligen Header-Anweisungen: *Delete*, *Add* und *Modify*. An den ToR-Switches erfolgt derselbe Rollentausch durch das Umschreiben der Senderadresse, wie das bei quellenbasierten-Bäumen der Fall ist. Eine zusätzliche *Modify-Field*-Action ist beim Wechsel zwischen Unicast-Route und Multicastbaum am Rendezvous-Knoten notwendig. Ein Beispiel dafür ist in Bild 4.16 gezeigt.

Aktualisierungsphase Das erfolgreiche Löschen eines Flows wird in OpenFlow durch eine *Flow-Removed*-Statusmeldung angezeigt. Tritt bei der Modifikation einer Flow-Table ein Fehler auf, meldet der Switch eine entsprechende Fehlermeldung zurück. In diesem Fall darf die Routenverwaltung nicht aktualisiert werden, um keinen inkonsistenten Zustand zur Datenschicht hervorzurufen. Um sicherzugehen, dass keine Fehler aufgetreten sind, muss eine ausreichend lange Zeitspanne abgewartet werden. Diese Wartezeit bezieht sich jedoch lediglich auf die Aktualisierung der internen Datenstruktur in der Routenverwaltung und nicht auf die Einrichtung der Routen auf Datenebene. Sind in dieser Zeitspanne keine Fehler zurückgemeldet worden, wird der neuberechnete Multicastbaum oder die Unicast-Route in der Routenverwaltung aktualisiert. Dazu wird, falls vorhanden, der alte Eintrag in der Hashtabelle verworfen und durch den neuen ersetzt. Anschließend muss durch ein *Flag* gekennzeichnet werden, ob es sich um einen Baum (Flag=0) oder eine Unicast-Route (Flag=1) handelt. Der Zugriff erfolgt dabei über eine entsprechende API, die von der Routenverwaltung zur Verfügung gestellt wird. Ein Beispiel zur Datenstruktur findet sich in Abschnitt 4.6.1, Bild 4.11. Es muss beachtet werden, dass die verschiedenen Routen zum richtigen Schlüssel abgespeichert werden. Ein Shared-Tree wird unter dem Hashtabelleneintrag der Multicastadresse abgespeichert. Die Unicast-Routen werden als Knotenliste, zusammen mit einem Zeiger auf den Shared-Tree mit dem konkatenierten Schlüssel zwischen Sender-ID und Multicastadresse abgelegt. Die Reihenfolge in der Knotenliste entspricht dem Weg der Unicast-Route vom Sender bis zum Rendezvous-Knoten des Shared-Tree. Ein quellenbasierter Baum wird unter dem gleichen Schlüssel gespeichert, da von einem Sender zur gleichen Zeit entweder eine Unicast-Route zum Shared-Tree oder ein quellenbasierter Baum benötigt wird.

5 Multicast-Implementierung in Floodlight

In diesem Kapitel soll die in Kapitel 4 erarbeitete Konzeption für einen Multicastdienst in Java implementiert werden. Zuerst wird in Abschnitt 5.1 der OpenFlow-Controller *Floodlight* [flo] vorgestellt und anschließend in Abschnitt 5.2 eine darauf aufbauende Multicastimplementierung beschrieben.

5.1 Floodlight OpenFlow-Controller

Floodlight [flo] ist ein in Java implementierter OpenFlow-Controller, der aus einer frühen Version des Beacon Projekts [Uni] hervorgegangen ist. Entwickelt wurde er von Big Switch Networks und diente als Basis für die im November 2012 erschienene kommerzielle Version mit dem Namen *Big Network Controller* [Net]. Floodlight selbst ist Open-Source und wird unter Apache Lizenz vertrieben. Anders als Beacon basiert Floodlight nicht mehr auf OSGi und bietet ein eigenes, einfach zu erweiterndes Modulsystem. Die wachsende Verbreitung und die ständige Weiterentwicklung durch Big Switch Networks, sowie die Plattformunabhängigkeit und die einfache Erweiterungsmöglichkeiten in Java waren die ausschlaggebenden Kriterien für die Wahl von Floodlight in dieser Diplomarbeit.

Abbildung 5.1 zeigt die Controller-Architektur. Neben der Umsetzung des OpenFlow Protokolls existieren mehrere parallel laufende Module. Sie implementieren zusammen jeweils eine Menge von verschiedenen Funktionalitäten, um ein OpenFlow-Netzwerk in Java ansprechen und verwalten zu können. So repräsentiert beispielsweise ein Objekt der Klasse `IOFSwitch` einen Switch auf Datenebene, der über einen eindeutigen Bezeichner namens *Datapath-ID* (DPID) identifiziert wird. Dessen Attribute können über entsprechende Methoden beliebig ausgelesen oder manipuliert werden. Die Umsetzung in die zugehörigen OpenFlow-Nachrichten, sowie die eigentliche Kommunikation gemäß des OpenFlow-Protokolls werden intern über den *FloodlightProvider* abgewickelt und so vor dem Entwickler verborgen. Ein *Modulmanager* ist für die Verwaltung der aktiven Module zuständig. Neue Module müssen dabei zuerst in einer Konfigurationsdatei bekannt gemacht werden, bevor der *Modulmanager* sie automatisch mit Floodlight starten und verwalten kann. Eine Klasse wird als Modul behandelt, wenn sie das Interface `IFloodlightModule` implementiert.

Jedes Modul kann mehrere Services bereitstellen, die als Java-Schnittstelle zwischen den internen Modulen einer Anwendung dienen. Der *Device Manager* verwaltet Hostmaschinen im Netzwerk und stellt dafür einen Service zur Verfügung. *LinkDiscovery* ist für das Erkennen von Verbindungen zwischen zwei Switches zuständig und bietet über den `LinkDiscoveryService` eine abstrahierte Sichtweise auf das Netzwerk an. Der *TopologyManager* verfolgt diese

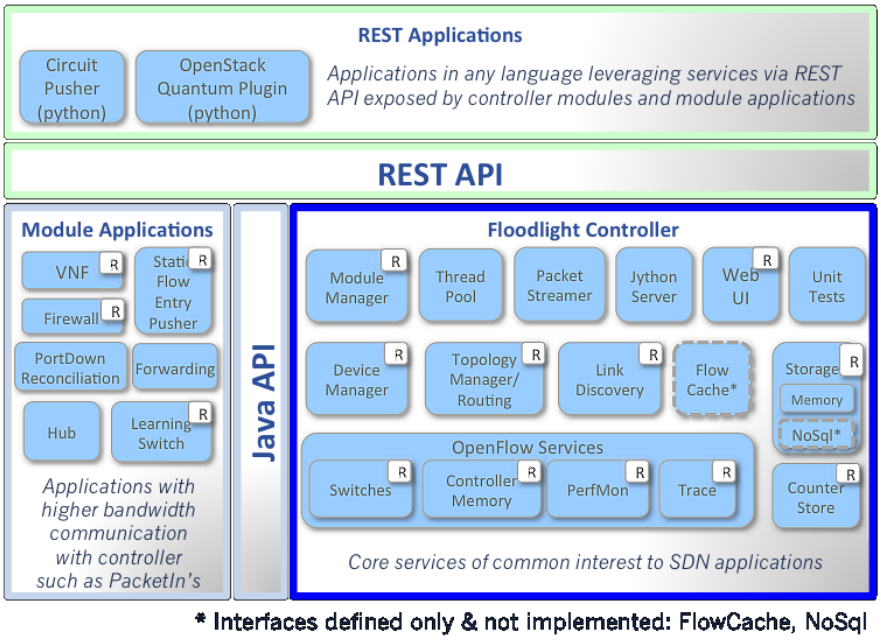

Abbildung 5.1: Architektur von Floodlight nach [flo]

Informationen und stellt unter anderem einen Listener für geänderte Netzwerktopologie zur Verfügung. Für das Empfangen von OpenFlow-Nachrichten wie *Packet_In*-Pakete sowie das Senden von *Packet_Out*-Nachrichten stehen ebenfalls entsprechende Listener und Schnittstellen zur Verfügung. Außerdem existieren ein *Routing*- und ein *Forwarding-Modul*, die ein Unicast-Routing realisieren. Standardmäßig wird über eine einfache Entfernungsmetrik ein kürzester Pfad zwischen einem Sender und Empfänger berechnet und reaktiv, also bei Empfang eines *Packet_In*-Pakets, auf Datenebene eingerichtet. Eine Multicastbehandlung ist zum Zeitpunkt dieser Diplomarbeit nicht realisiert.

Zusätzlich zum *Forwarding* beinhaltet Floodlight neben den Basisservices noch einige weitere Anwendungen. So existiert unter anderem ein *Firewall-Modul*, ein Modul für die Behandlung von *Port-Down Events* und ein *Static-Flow-Entry-Pusher*, der es erlaubt, über einen Service beliebige Flows auf Datenebene einzurichten. Jedes dieser Anwendungsmodule kann nach Bedarf geladen oder deaktiviert werden. Eine eigene Anwendung kann ebenso in Form von Modulen erweitert werden, die wiederum eigene Services zur Verfügung stellen können, während auf die Grundfunktionen über die erläuterten Basis-Services zugegriffen werden kann.

Alle Module können zusätzlich eine REST Schnittstelle zur Verfügung stellen. Sie sind durch ein „R“ in Bild 5.1 gekennzeichnet. Die meisten Basis- und Anwendungsmodule bieten Entwicklern so die Möglichkeit, aufbauende REST-Anwendungen unabhängig von der Programmiersprache zu entwickeln. Ein Beispiel dafür zeigt das *Web-UI Modul*, mit dem man jederzeit Topologieinformationen und Statistiken über eine Weboberfläche im Browser kontrollieren kann.

5.2 Erweiterung von Floodlight um einen Multicastdienst

Eine Umsetzung der OpenFlow-Version 1.3 in Floodlight ist bereits angekündigt, zum Zeitpunkt dieser Diplomarbeit wird jedoch lediglich die Version 1.0 unterstützt. Dadurch existieren in der Implementierung einige Einschränkungen, insbesondere das Fehlen der Group-Tables. Obwohl nicht explizit in der OpenFlow Spezifikation [Spe09] dokumentiert, ist das Einfügen mehrerer OpenFlow-Aktionen für einen einzigen Flow auch ohne Group-Table über die Flow-Table möglich. Die Abarbeitungsreihenfolge geschieht sequentiell in der Reihenfolge, in der die Aktionen definiert wurden. So ermöglichen mehrere *Ouput-Actions* auch das Beschreiben mehrerer Ausgangsports für einen Flow und so eine Möglichkeit zur Abbildung von Multicastbäumen. Es muss aber darauf hingewiesen werden, dass diese Möglichkeit herstellerabhängig ist und so möglicherweise nicht von allen OpenFlow-Switches in der Version 1.0 unterstützt wird.

Für die Implementierung des Multicastdienstes ergibt sich die Wahl zwischen einer REST-Anwendung und einer direkten in Java integrierten Lösung. Zum Zeitpunkt dieser Diplomarbeit können zukünftige Änderungen an den Schnittstellen aber nicht ausgeschlossen werden [flo]. Aufgrund der starken Abhängigkeit zu Topologie und Statistikwerten, sowie der häufigen Kommunikation mit der Datenschicht, fällt die Wahl auf Anwendungsmodule, die direkt in Floodlight integriert werden. Dabei können die herausgearbeiteten Prozesse aus Kapitel 4 fast eins zu eins in Floodlight-Module umgesetzt werden.

Das Paket `net.floodlightcontroller.multicast` enthält alle für den Multicastdienst umgesetzten Module und Klassen. Folgende Multicast-Module werden beim Start des Controllers geladen: `NetworkStateManagement.java`, `GroupManagement.java`, `RoutingControl.java`, `FlowMod.java` und `RouteManagement.java`. Im Folgenden wird die Umsetzung der einzelnen Prozesse mit Hilfe dieser Module besprochen.

Der Informationsaustausch zwischen den Modulen geschieht über Services, die den anderen Multicast-Modulen als Schnittstelle zur Verfügung stehen. Nach Konvention werden Schnittstellen stets mit „I{*Modulname*}Service.java“ bezeichnet. Dabei steht der Modulname für die Klasse, die das Interface implementiert. Für die Weitergabe von Ereignissen wird das *Observer-Pattern* verwendet. Ein Observer erlaubt es, ein auftretendes Ereignis von einem Modul an alle zuvor angemeldeten Empfängermodule weiterzugeben. Damit eine Klasse als Empfänger agieren kann, muss dieser das Interface `Observer` implementieren.

Nachrichtenfilter Floodlight bietet über den `IOFMessageListener` bereits die Möglichkeit, angekommene Nachrichten zu erhalten und über Abfragen nach Headerfeldern zu filtern. Somit besteht die Umsetzung des Nachrichtenfilters aus der Implementierung des Listeners durch die jeweiligen Empfängermodule. `GroupManagement.java` filtert diese Nachrichten nach der IGMP Protokollnummer. Das Modul `NetworkStateManagement.java` fragt Port-Statistiken direkt ab und benötigt somit überhaupt keine Behandlung der Statistiknachrichten auf OpenFlow Ebene.

Netzstruktur- und Zustandsverwaltung Die Netzzustandsverwaltung wird durch das Modul `NetworkStateManagement.java` umgesetzt. Das Abfragen von Verbindungen über LLDP, sowie das Bereitstellen der Topologie als (Switch, Link)-Objekte geschieht bereits im *LinkDisvoceryManager* von Floodlight und muss deshalb nicht neu implementiert werden. Die Topologieinformationen werden über den `LinkDiscoveryService` und `TopologyService` abgefragt und anschließend vom *NetworkStateManagement-Modul* weiterverarbeitet. Ein Java-Thread fragt periodisch alle Switches nach Portstatistiken ab und errechnet daraus die aktuellen Gewichte, die als *ConcurrentHashMap* gespeichert werden. Diese Datenstruktur garantiert vollständig parallelen Zugriff mehrerer Threads ohne Blockierungen. So können die Gewichte bei Bedarf durch andere Prozesse gelesen werden, ohne die periodische Aktualisierung zu stören. Das *NetworkStateManagement-Modul* ist außerdem für die Überwachung der Flow-Einträge zuständig. Ein drohender Überlauf wird durch einen periodisch ablaufenden Thread erkannt und über den *Overflow-Observer* sofort an das *RoutingControl-Modul* gemeldet.

Zusätzlich wird die Floodlight-Klasse `IOFSwitch` um das Attribut *isToR* erweitert. Mit Hilfe des `TopologyService` werden alle Switches periodisch geprüft, ob sie in direkter Nachbarschaft mit einer Hostmaschine verbunden sind. Falls dies der Fall ist, werden sie als ToR-Switches markiert. Diese Kennzeichnung ist für ein gezieltes Versenden von IGMP-Nachrichten und die Einrichtung initialer Flows im vorliegenden Systemmodell relevant. Für die Weitergabe der Netzzustands-Daten stellt das *NetworkStateManagement-Modul* im Zuge des Multicastdienstes einen Service `IStateManagementService.java` zur Verfügung. Er ermöglicht dem *Routing-Modul* Zugriff auf die aufbereitete Topologie und die zugehörigen Kantengewichte.

Gruppenverwaltung Die Gruppenverwaltung wird durch das Modul `GroupManagement.java` implementiert. Hier können auch die jeweiligen IGMP Antwort- und Sendezeiten belegt und geändert werden. Die Datenstruktur für aktuelle Gruppeninformationen unterstützt ebenfalls vollständige Parallelität durch eine *ConcurrentHashMap*. Ein Objekt der Klasse `GroupElement.java` repräsentiert einen Eintrag in der *HashMap* und beinhaltet neben der Multicastadresse und der Switch-ID auch den aktuellen Zählerwert für die Umsetzung des IGMP-Protokolls. Die Zähler aller Einträge werden periodisch von einem Java-Thread heruntergezählt. Ein weiterer Thread versendet in definierten Zeitabständen die IGMP-*Membership-Requests*. Für die Verarbeitung von IGMP-Paketen ist es nötig, eine Ergänzung in den Floodlight Datenstrukturen für die Nachrichtenkommunikation vorzunehmen. Die neue Klasse `IGMP.java` erweitert dazu die Floodlight Klasse `BasePacket.java` um die spezifischen Eigenschaften eines IGMP Paketes. Das sind insbesondere die Headerfelder *Type*, *MaxResponseTime*, *Checksum*, *GroupAddress*. Um auch IGMPv3 Nachrichten verarbeiten zu können, wird `IGMP.java` wiederum von einer weitere Klasse `IGMPv3Report.java` erweitert. Zur Laufzeit werden alle ankommenden IGMP-Nachrichten zuerst in ein Objekt der Klasse `IGMP` deserialisiert und anschließend aufgrund des Typ-Feldes bestimmt, ob eine Deserialisierung in ein *IGMPv3Report-Objekt* stattfinden muss. Um ankommende IP-Pakete mit IGMP-Inhalt richtig zuordnen und deserialisieren zu können, muss außerdem die IGMP Protokollnummer in `IPv4.java` ergänzt und registriert werden.

Hat der Controller einen ankommenden IGMP-*Membership-Report* richtig zugeordnet und die zugehörigen Gruppenadressen ausgelesen, hat dies einen Methodenaufruf im

GroupManagement-Modul zur Folge. Daraufhin werden Gruppenzähler entweder erneuert oder Änderungen in der Gruppenzusammensetzung vorgenommen. Um solche Änderungen möglichst schnell bekannt zu machen, registriert das *GroupManagement-Modul* den *GroupObserver*, der aktuelle Gruppenänderungen sofort an das *Routing-Modul* meldet. Außerdem läuft ein Java-Thread, der periodisch IGMP-Anfragen erstellt und diese über das Floodlight Switch-Objekt des jeweiligen Empfängerswitches verschickt. Die aktuellen Gruppeninformationen stehen über den Service `IGroupManagementService.java` für andere Module zur Verfügung.

Routenberechnung Das Modul `RoutingControl.java` bildet die Funktionalität der Routenberechnung ab. Wichtig hierfür ist die Listener-Methode `topologyChanged()` sowie die Observer-Methode `Update()`, über die die inneren Klassen `GroupObserver.java` und `Overflowobserver.java` benachrichtigt werden. Diese Methoden werden bei Topologie-, Gruppen- und Überlaufänderungen aufgerufen und filtern die Ereignisse nach Kriterien wie *Delete-*, *Add* oder *New-Events*.

Eine Gruppenänderung stößt den Routingalgorithmus an. Das ist, abhängig vom Überlaufmodus, entweder KMB, Shared-Tree oder iGA. Einem Routingalgorithmus stehen die Basisalgorithmen in den Klassen `Dijkstra.java`, `Tarjan.java` und `Kruskal.java` zur Verfügung. `Tarjan` wird für KMB benötigt und errechnet einen minimal gewurzelten Baum (minimaler gerichteter Spannbaum), `Dijkstra` ermittelt den kürzesten Weg zwischen einer Quelle und einem Ziel. Außerdem implementiert das *RoutingControl-Modul* die für KMB relevanten Methoden *SUB*, *RESUB* und *PRUNE*. Für die Shared-Tree Berechnung steht eine Methode `calculateCenter()` bereit, die den Rendezvous-Knoten für eine Gruppe errechnet. Der Shared-Tree Algorithmus verwendet anschließend die KMB-Methode, um von dem Rendezvous-Knoten einen quellenbasierten Baum zu allen Gruppenmitgliedern zu berechnen. Mit Hilfe des Algorithmus von Dijkstra und KMB wird außerdem iGA umgesetzt.

Weitere Routingalgorithmen können beliebig erweitert werden und bei Bedarf die erwähnten Basisalgorithmen nutzen. Zur Repräsentation des aktuellen Netzgraphen und der jeweiligen End- und Zwischenergebnisse existieren die Klassen: `RoutingEdge.java`, `RoutingNode.java`, `CompleteGraph.java`, `DijkstraPath.java` und `MulticastTree.java`, sowie etliche Hilfs- und Konvertierungskonstrukte. Nach Ausführung einer Berechnung nutzt das *RoutingControl-Modul* den *RouteManagement-Service*, um die neuen Multicastbäume oder Änderungen weiterzugeben und abzuspeichern.

Um Topologieänderungen zu erkennen, kann ein beliebiges Modul den von Floodlight bereitgestellten `ITopologyListener` implementieren. Daraufhin werden alle Änderungen über den Methodenaufruf erkannt und es kann nach Ereignissen wie Switch-Ausfall oder Link-Ausfall gefiltert werden. Ein solcher Aufruf hat einen sofortigen Datenabgleich mit dem *NetworkStateManagement-* und *RouteManagement-Modul* sowie eine Neuberechnung der betroffenen Multicastbäume zur Folge.

Das *NetworkStateManagement-Modul* meldet über ein Observer-Ereignis einen drohenden Überlauf der Routingtabelle. `RoutingControl` implementiert diesen Observer in einer inneren Klasse mit dem Namen `OverflowObserver.java`. Daraufhin wird der interne Routingmodus

„OverflowMode=true“ gesetzt. Diese Maßnahme ersetzt für zukünftige Gruppenänderungen KMB durch den Aufruf des Shared-Tree Algorithmus. Ein Zurücksetzen des Overflow-Modus erfolgt durch ein analoges Ereignis am Observer.

Routenverwaltung Über das Modul `RouteManagement.java` wird die Funktionalität der Routenverwaltung abgedeckt. Das Modul ist für die Speicherung der Objekte vom Typ `MulticastTree.java` und `DijkstraPath.java` zuständig. Dafür wird eine *HashMap* genutzt und die nötigen Methoden für das Anfragen, Speichern, Vergleichen oder Ändern von Multicastbäumen über den `IRouteManagementService.java` bereitgestellt. Ein *MulticastTree-Objekt* enthält eine *HashMap*, die aus Knoten und zugeordneten Kantenlisten besteht. Die Datenbasis für *DijkstraPath-Objekte* besteht aus einer einfachen Knotenliste. Für die eindeutige Identifizierung von quellenbasierten Bäumen und Unicast-Pfaden wird die von Floodlight vergebene Switch-DPID des Senders in Kombination mit der Multicastadresse genutzt. Der Schlüssel eines Shared-Trees besteht aus der Multicastadresse ohne Senderangabe.

Flow-Modifikation Das Modul `FlowMod.java` ist für die Einrichtung der Flows auf Datenebene zuständig. Aufgerufen wird es über den Service `IFlowModService.java`, der die Methoden für das Hinzufügen, Ändern und Löschen von Multicastbäumen, IGMP-Flows und *ModifyField*-Flows bereitstellt. Jeder Flow muss auch auf Datenebene einen eindeutigen Bezeichner tragen. Hier gilt die Konvention, dass sämtliche Bäume und Routen die DPID der Senderquelle in Verbindung mit der Multicastadresse tragen. Eine Quelle stellt dabei entweder einen ToR-Switch oder einen Rendezvous-Knoten dar.

Das *FlowMod-Modul* kümmert sich zuerst um die Einrichtung initialer Flows. Dazu zählen *ModifyField*-Actions, die jeder ToR-Switch beim Erhalt eines Paketes über einen Host-Port benötigt, um die Senderadresse von einem Host in die Senderadresse in die DPID des Switches umzuschreiben. Zusätzlich benötigt jeder ToR-Switch einen IGMP-Flow für die korrekte Weiterleitung von IGMP-Antworten an den Controller-Port. Ändert sich die Topologie, insbesondere die Menge der ToR-Switches, muss die Einrichtung erneut geschehen. Ansonsten bleiben diese Flows während der gesamten Laufzeit des Controllers statisch.

Änderungen in Flows, die Multicastbäume betreffen, werden vom *RoutingControl-Modul* initiiert und treten unter anderem bei jeder Gruppenänderung auf. Der aktuelle Multicastbaum wird mit dem neuen verglichen und im *FlowMod-Modul* in Einzeloperationen wie `addFlowForLink()` oder `deleteFlowFromGroup()` umgesetzt. Analoge Operationen existieren bei Shared-Trees für die Einrichtung von *DijkstraPath-Objekten* als Unicast-Routen. Eine Sonderbehandlung ist für den Rendezvous-Knoten nötig. Für am Baum ankommende Unicast-Routen muss ein entsprechender *ModifyField*-Flow eingerichtet werden, um die Quelladresse vom Sender-Switch in die des Rendezvous-Knoten umzuschreiben. Nachdem alle Flows so eingerichtet sind, dass sämtliche Ziel-Switches der Gruppenmitglieder die Multicastpakete empfangen können, fehlt noch die eigentliche Auslieferung an die Hosts. Dafür werden pro Gruppe alle Host-Ports ermittelt und zusätzliche Aktionen für die Auslieferung an die Endgeräte eingerichtet.

Bei Gruppenänderungen muss stets zwischen ToR-Switches und inneren Knoten unterschieden werden, da sämtliche Aktionen, inklusive den *ModifyField*-Actions und den *Output*-Actions für die Auslieferung zu den Hosts, in einem einzigen Flow definiert werden müssen. Aufgrund der sequentiellen Abarbeitung der Aktionen muss darauf geachtet werden, dass die *ModifyField-Actions* vor allen *Output-Actions* stehen.

Bild 5.2 zeigt die Beispiel-Flows für einen ToR-Switch mit der DPID *„00:00:00:00:00:00:00:09“* für einen quellenbasierten Multicastbaum auf Datenebene. Der obere Flow leitet ankommende Pakete mit der Multicastadresse „200.200.200.200“, die vom quellenbasierten Baum mit der Sender-DPID *„00:00:00:00:00:00:00:0b“* kommen, über die Host-Ports 1 und 2 weiter. Der Flow in der Mitte ist für das Zurückschicken einer IGMP-Antwort an den Controller-Port −3 zuständig. Der untere Flow nimmt Pakete entgegen, die von dem Host, der über Port 1 angeschlossen ist, gesendet wurden. Zuerst wird dazu die MAC-Adresse über eine *ModifyField*-Action in *„00:00:00:00:00:09“* umgeschrieben und anschließend gemäß des Multicastbaumes über mehrere Ausgangsports verteilt.

Die Entscheidung, für diese Implementierung das *FlowMod-Modul* von der RoutingControl zu entkoppeln, hat den praktischen Grund, dass alle Manipulationen der Datenschicht an zentraler Stelle vorgenommen werden. Alternativ hätte die initiale Einrichtung der IGMP und *ModifyField*-Flows aber auch in den jeweiligen Modulen `NetworkStateManagement` und `GroupManagement` geschehen können und das *FlowMod-Modul* in die Routenberechnung integriert werden können.

Flows (9)

Cookie	Priority	Match	Action	Packets	Bytes	Age	Timeout
45035996273704960	30	src=00:00:00:00:00:0b, ethertype=0x0800, dest=200.200.200.200	output 2, output 1	0	0	5 s	0 s
45035996273704960	5	ethertype=0x0800, proto=2, IP src port=0, IP dest port=0	output -3	0	0	101 s	0 s
45035996273704960	0	port=1, ethertype=0x0800, dest=255.255.255.255	src=00:00:00:00:00:09, output 2, output 1, output 3	0	0	5 s	0 s

Abbildung 5.2: Beispiel-Flows für einen ToR-Switch auf der Floodlight Weboberfläche

Weitere Klassen und Erweiterungen Die Klasse `REST.java` erlaubt das Absetzen eines REST-Aufrufs aus dem Java Code heraus und so die Verwendung der REST-Schnittstelle von Floodlight. Dies kann als Evaluierungs- und Testmöglichkeit genutzt werden oder bei Bedarf für Einzelaufgaben, wie das Abfragen von Statistiken, von Nutzen sein. Zu Vergleichs- und Testzwecken kann das Routing jederzeit durch ein Flag im *RoutingControl-Modul* in den *simple-Mode* versetzt werden. Daraufhin wird bei der Einrichtung neuer Flows auf Vergleiche mit vorherigen Berechnungen verzichtet. Alte Flows müssen dann zuerst komplett gelöscht werden, bevor neue eingerichtet werden können. Zusätzlich kann über das Flag *force_iga* auf den inkrementellen Algorithmus (iGA) umgeschaltet werden.

6 Evaluierung

In diesem Kapitel wird die in Kapitel 5 vorgestellte Implementierung getestet und ausgewertet. In Abschnitt 6.1 wird der hier verwendete Versuchsaufbau für das Testbed beschrieben. Im Anschluss werden in Abschnitt 6.2 die Messergebnisse vorgestellt.

6.1 Versuchsaufbau

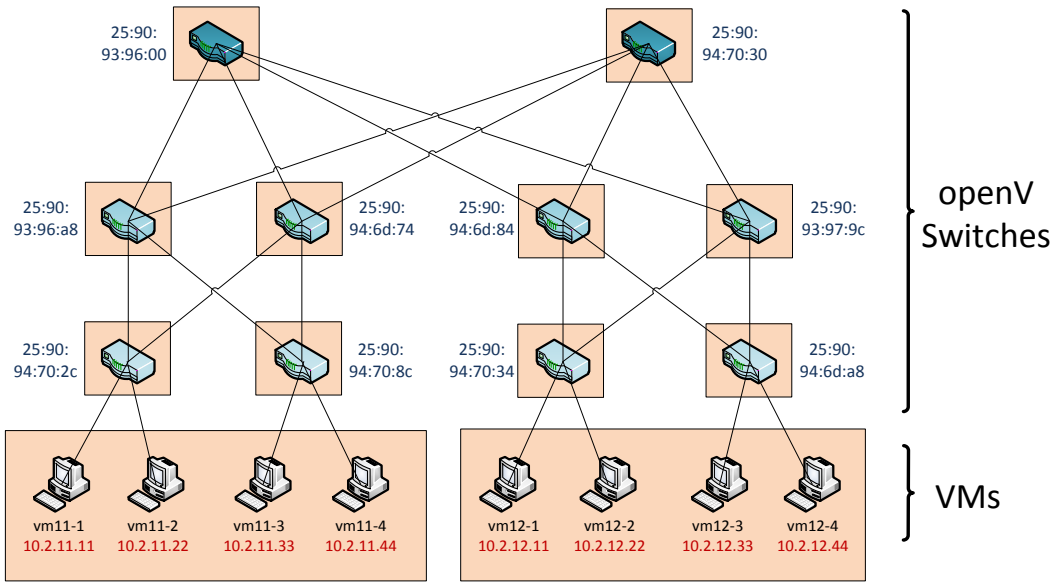

Abbildung 6.1: Topologie des Testaufbaus mit den IP-Adressen der VMs und den Switch-IDs

Der Test und die Evaluierung des in Floodlight implementierten Multicast-Dienstes findet auf dem Testbed der Abteilung Verteilte Systeme der Universität Stuttgart statt. Der Aufbau spiegelt eine Hälfte der Fat-Tree Topologie aus Kapitel 3.2, Bild 3.1, für k = 4, wieder. Abbildung 6.1 zeigt die Topologie dieses Testaufbaus mit den zugewiesenen IP-Adressen und den Switch-IDs. Die ersten 24 Bit einer Switch-ID sind in diesem Fall null und werden aus Übersichtsgründen weggelassen. Die Topologie ist durch insgesamt zwölf pyhsikalische Rechnerknoten realisiert. Es existieren acht virtuelle Maschinen, die als Sender oder Empfänger für Multicastnachrichten agieren können, wobei sich jeweils vier VMs einen physikalischen Hostrechner teilen. Dabei steht jeder VM 10Gb HD und 4 Gb RAM zur Verfügung. Im Netz werden *Open-vSwitches* eingesetzt, die alle auf einem eigenen PC mit jeweils

vier physikalischen Datenports laufen. Ein weiterer Port steht für die Controllerverbindung zur Verfügung (on-board NIC). Die Netzwerkgeschwindigkeit beträgt 1 Gbit/s im full duplex Betrieb, d.h. die volle Bandbreite steht sowohl in Hin- als auch in Gegenrichtung zur Verfügung. Durch schnelle CPU und I/O Performance auf den Switch-Rechnern wird eine Weiterleitung in *Line-Rate* angestrebt. Sämtliche virtuellen Switches unterstützen zum Zeitpunkt dieser Diplomarbeit den OpenFlow 1.0 Standard.

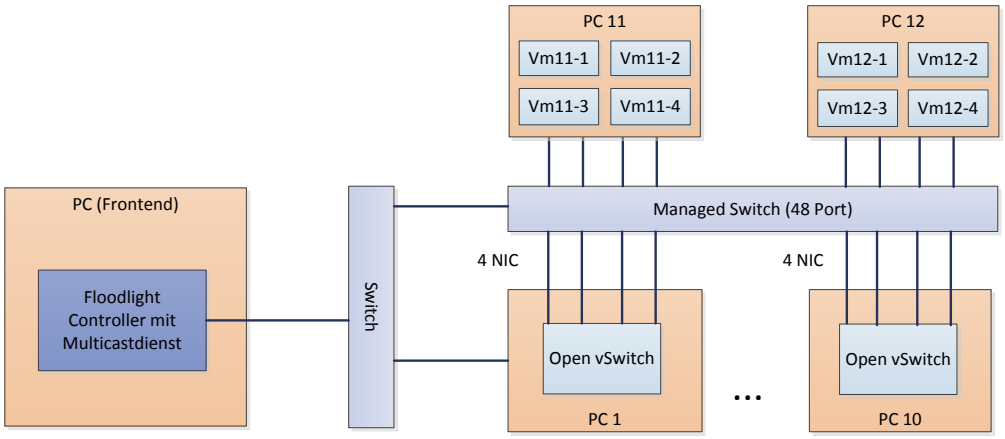

Abbildung 6.2: Realisierung des Testaufbaus über Hardware-Switches

Um einfache Topologieänderungen vornehmen zu können, werden die gesamten Verbindungen durch einen 48-Port großen *Managed-Switch* programmiert. Dabei handelt es sich um einen 1 Gbps Hardware-Switch, der die Punkt-zu-Punkt Verbindungen zwischen den Rechnern durch einfaches switching entsprechend vordefinierten Regeln konfiguriert. So müssen bei Topologieänderungen keine physikalischen Kabel umgesteckt werden. Neben dem *Managed-Switch* steht ein weiterer 1 Gbps Hardware-Switch für das Controllernetzwerk zur Verfügung. Der Floodlight Controller mit der Multicast-Erweiterung läuft auf einer Frontend-Maschine, die mit dem Controllernetzwerk verbunden ist. Über diese Maschine können auch das Netzwerk und die virtuellen Maschinen konfiguriert werden. Bild 6.2 zeigt die Realisierung über den Managed-Switch sowie das Controllernetzwerk.

6.2 Testverfahren

Im Vordergrund des Testens steht der Vergleich zwischen den einzelnen Routingalgorithmen, sowie die allgemeine Performance eines OpenFlow-Netzes in Bezug auf die Multicast-Kommunikation. Für die nachfolgenden Durchsatzmessungen wurde das Kommandozeilen-Werkzeug *iperf* benutzt, um 1470 Byte große UDP-Datagramme mit einer bestimmten Bandbreite über einen fest definierten Zeitraum an eine Multicastadresse zu schicken. Die Auswertung am Empfänger übernimmt ein einfaches Java-Programm, das der Gruppe beitritt und die Anzahl der angekommenen Pakete innerhalb dieses Zeitraumes ermittelt. Somit berechnet sich der Durchsatz aus dem Quotienten zwischen den erhaltenen und der Anzahl der gesendeten Pakete. Die theoretische Leitungskapazität liegt bei 1 Gbit/s. In den nachfolgenden Messungen erreichten die Sender mit *iperf* eine maximale Bandbreite von etwa 800 MBit/s, was 100 MB/s entspricht.

Alle aufgeführten Übertragungszeiten sind in Mikrosekunden gemessen. Dazu existiert jeweils ein weiteres Java-Programm für den Sender und den Empfänger. Aufgrund der Ungenauigkeit zwischen den Uhren auf den Endsystemen wird die Paketumlaufzeit (*Round Trip Time*) gemessen und die Übertragungszeit angenähert, indem die Rücksendezeit abgezogen wird. Dazu schickt das Sender-Programm ein Datagramm über einen UDP-Socket an eine Multicastadresse. Das Gegenstück am Empfänger tritt zuvor einer über die Kommandozeile definierten Multicastadresse bei und schickt das Paket unverzüglich an den Sender zurück. Dieser kann nun wiederum die Zeitdifferenz für diesen Weg messen. Da bei Multicast die proaktiv berechneten Routen jeweils nur vom Sender zu den Empfängern gültig sind, wird die Antwort außerhalb des SDN-Testbeds zurückgeschickt. Aus diesem Grund können sich die Pfade für Hin- und Rückweg stark unterscheiden. Deshalb wurde die Paketumlaufzeit des Rückweges ebenfalls gemessen und die zugehörige Übertragungszeit mit der Hälfte angenähert. Übertragungen außerhalb des Testbeds an die selbe physikalische Maschine wiesen eine Verzögerung von durchschnittlich $420\mu s$ auf, während die Datagramme an die jeweils andere Maschine etwa $600\mu s$ benötigten. Die Datagrammgröße wird analog zu den Durchsatzmessungen mit 1470 Byte gewählt. Die nachfolgend aufgelisteten Übertragungszeitmessungen sind Mittelwerte aus mehreren hundert Messungen. Der erste Messwert wurde dabei verworfen, da er einen stark erhöhten Wert bei der Rücksendung aufgewiesen hat.

6.2.1 Auswertung für quellenbasierte Multicastbäume (KMB)

Zu Beginn soll das Netz im lastfreien Fall untersucht werden. Abbildung 6.3 (oben) zeigt die Übertragungszeiten von quellenbasierten Multicastbäumen abhängig von der Gruppengröße, die mittels KMB berechnet wurden. Im unteren Diagramm sind die zugehörigen Durchsatzmessungen zu sehen. Die Maschine vm11-1 agiert hierbei als Sender für eine ansteigende Zahl vom Empfängern. Der betrachtete Empfänger ist vm12-4, d.h. es wird die längst mögliche Strecke zwischen 2 Maschinen untersucht. Dabei muss beachtet werden, dass, obwohl hier 8 Hosts unterstützt werden, die effektive Anzahl an Sendern und Empfänger nur 4 beträgt. Der Grund hierfür ist, dass der Controller lediglich ToR-Switches verwaltet und deshalb jeweils zwei Hosts gemeinsam behandelt. Somit fallen hier stets zwei VMs auf einen gemeinsamen Multicastbaum zusammen.

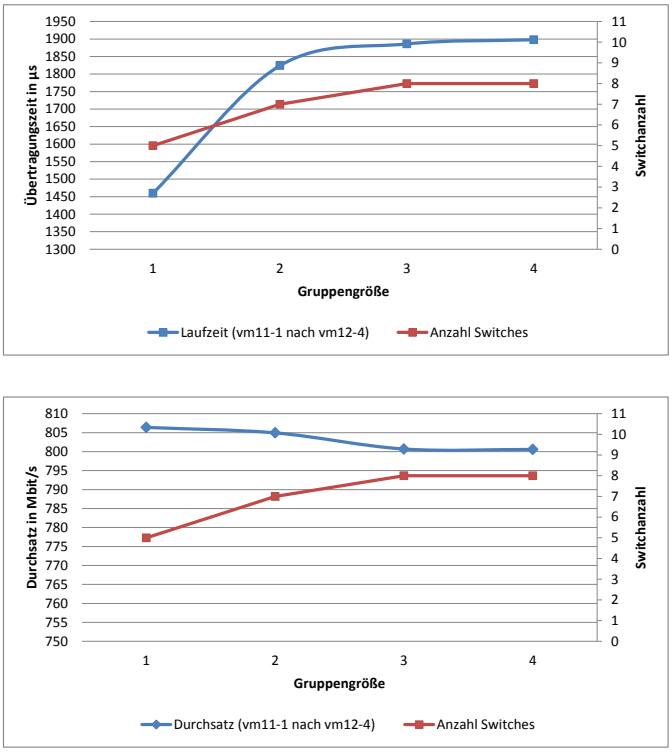

Abbildung 6.3: Latenz und Durchsatz eines mit KMB berechneten Baumes abhängig von der Gruppengröße

Die Messwerte aus Abbildung 6.3 (oben) zeigen eine Übertragungszeit von 1460μs für eine Gruppengröße von 1. In diesem Fall degeneriert der Multicastbaum zu einer Unicast-Route, die den kürzesten Pfad nach Dijkstra darstellt. Auf diesem Pfad liegen 5 *Open-vSwitches* (rote Kurve), was eine Verzögerungszeit von etwa 300μs pro Switch nahelegt. Zusätzlich dürfte durch das Umschreiben der Senderadresse die Verzögerungszeit an den ToR-Switches etwas höher liegen, als bei einem Switch in der Mitte des Netzes. Bei einer Gruppengröße von 2 wählt der Algorithmus für den Empfänger vm12-4 einen weniger optimalen Pfad, was sich in einer verlängerten Übertragungszeit wiederspiegelt. Das gleiche gilt auch für die Gruppengrößen 3 und 4. Der Durchsatz aus Bild 6.3 (unten) ist dabei weitestgehend konstant. Da keine weitere Last auf dem Netz liegt, kann die volle Bandbreite von rund 800MBit/s erreicht werden.

Die Messwerte für die restlichen Empfänger sind in Tabelle 6.1 gezeigt. Es handelt sich um Übertragungszeiten bei voller Gruppengröße für den Baum aus 6.1 (rechts). Auffällig ist der hohe Wert für den Empfänger vm11-2, da zwischen dem Sender und dem Empfänger faktisch nur ein Switch liegt. Eine Erklärung hierfür könnte die Tatsache sein, dass der ToR-Switch hier sowohl als Sender- und Empfänger-Switch agiert. Dies bringt zusätzlichen Overhead am Switch mit sich. Außerdem könnten die Einflüsse der Virtualisierung beim Senden und Empfangen zu einer höheren Verzögerungzeit beitragen. Allerdings konnte dieses Phänomen im Rahmen dieser Diplomarbeit nicht eindeutig erklärt werden.

Empfänger	Laufzeit	Switches
vm12-4	1886μs	7
vm12-1	1404μs	5
vm11-3	1114μs	3
vm11-2	692μs	1

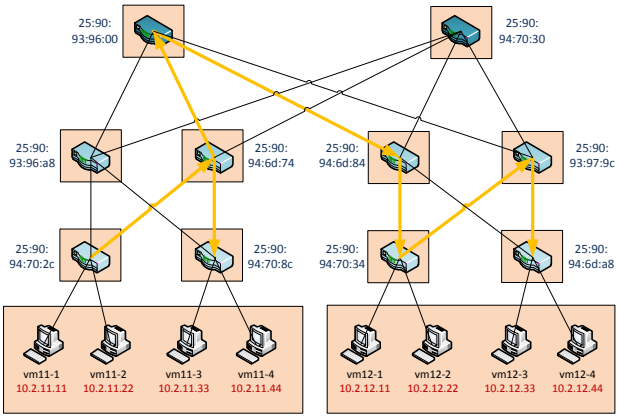

Tabelle 6.1: Übertragungszeiten aller Empfänger für einen quellenbasierten Baum mit dem Sender vm11-1

Im Fall eines belasteten Netzwerkes sind vor allem das *Load-Balancing* und der direkt davon abhängige Durchsatz von Relevanz. Diese Faktoren werden in Relation zu der Anzahl aktiver Multicastgruppen untersucht. Abhängig davon wie viele Kanten im Netz belastet oder ausgelastet sind, wird bei einer Gruppenänderung mit KMB ein Multicastbaum berechnet, der eine möglichst unbelastete, als auch kurze Route bevorzugt. Der Faktor α, aus Kapitel 4 Gleichung 4.3, der das Verhältnis zwischen Bandbreite und Entfernung bei der Kantengewichtsberechnung bestimmt, wird für die nachfolgenden Tests mit $\alpha = 0.5$ vorbelegt. Somit werden von KMB sowohl kürzere Wege als auch gering belastete Kanten im Verhältnis eins zu eins mit einbezogen.

Abbildung 6.4 zeigt die durch KMB berechneten Multicastbäume (in Rot) bei einer steigenden Anzahl von Gruppen im Netz. Diese Gruppen besitzen jeweils einen Empfänger und senden gleichzeitig einen konstanten Datenstrom von 370 Mbit/s (schwarze Pfeile). Die Gruppenkommunikation beschränkt sich hierbei auf die Maschinen vm11-3, vm11-4, vm12-2, vm12-3, um die Testergebnisse nicht durch Überlastung eines Sender- oder Empfängerhosts zu beeinflussen. Der rot gefärbte Multicastbaum soll nun Daten von vm11-1 nach vm12-4 senden. Er wird in den Bildern 6.4 stets zuletzt berechnet und soll so die Wegwahl des Routingalgorithmus bei steigender Vorbelastung der Kanten demonstrieren. Die Berechnungsreihenfolge der anderen Bäume entspricht der Reihenfolge von Bild 6.4.a bis 6.4.d, wobei pro Bild ein weiterer Baum hinzukommt. Bis zu einer Gruppenanzahl von mindestens 3 weiteren Gruppen (Teilbilder 6.4.a bis 6.4.c) ist KMB in der Lage, für den roten Baum eine Route zu finden,

die nur unbelastete Kanten nutzt. Sobald 4 verschiedene Gruppen gleichzeitig senden, ist dies nicht mehr möglich und zwei Kanten überlappen sich mit dem gelb gekennzeichneten Baum in 6.4.d.

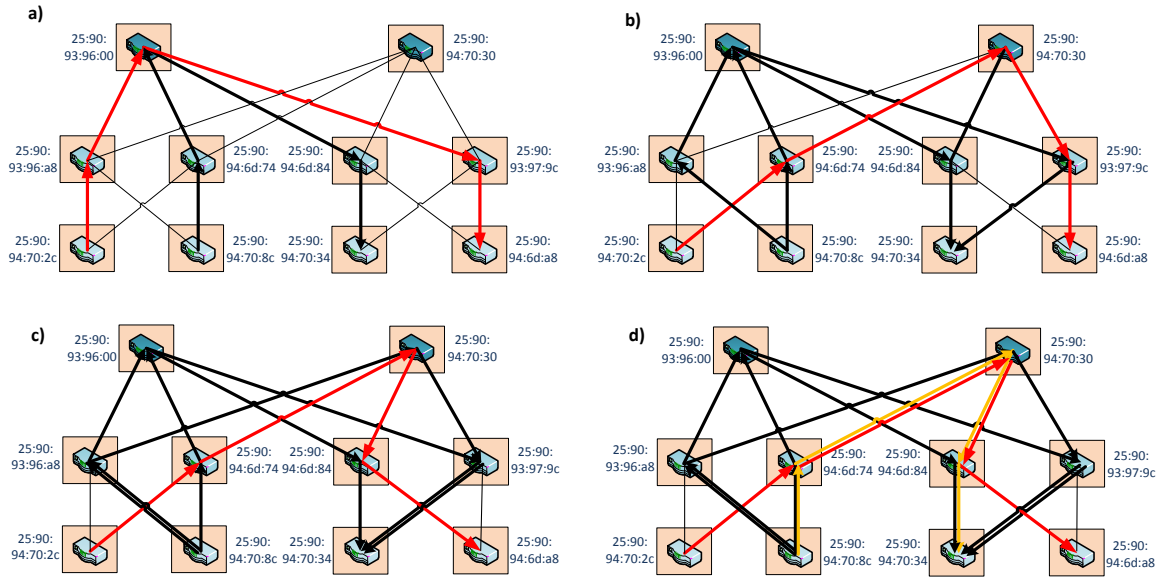

Abbildung 6.4: Load-Balancing bei quellenbasierten Bäumen in Abhängigkeit der Gruppenanzahl unter Last. Jeder schwarze und gelbe Pfeil symbolisiert eine Auslastung von 370Mbit/s für die zugehörige Kante.

Für den in Teilbild 6.4.d gezeigten Fall, mit insgesamt 4 aktiven Multicastgruppen, wird nun der Durchsatz des roten Baumes untersucht. Dieser sendet dabei mit der vollen Bandbreite von 800 Mbit/s. Die Ergebnisse sind in Abbildung 6.5 (unten) zu sehen. Alle schwarz gekennzeichneten Bäume senden dabei einmal nicht (0 Mbit/s) sowie einmal mit 370 Mbit/s, während der gelbe Baum nun eine variable Bandbreite aufweist. Bis zu einer Senderate von 100 MBit/s bzw. 200 MBit/s ist für beide Kurven kein nennenswerter Abfall des Durchsatzes zu verzeichnen. Durch die Überlappungen zwischen den zwei betrachteten Bäumen tritt jedoch bei höheren Senderaten eine Überlast auf, woraufhin Pakete verloren gehen. Während in der Kurve für 0 Mbit/s für geringe Senderaten noch die volle Bandbreite erreicht wird, ist diese für die Messungen mit 370 Mbit/s zu Beginn bereits deutlich reduziert. Dies könnte auf die Virtualisierung zurückzuführen sein, da die einzelnen Hosts lediglich durch 2 physikalische Rechner realisiert sind. Bei einer Sendeleistung von 500 MBit/s ist der Durchsatz um 62% bzw. 71% abgefallen und hält sich anschließend bei einem Wert von 475 Mbit/s. Die Übertragungszeiten 6.5 (oben) zeigen einen leicht ansteigenden Trend bei höherer Leistungsauslastung. Der Maximalwert für die 370 Mbit/s Kurve liegt mit 1728μs insgesamt 180μs über dem Startwert. Die Werte für die 0 Mbit/s Kurve liegen im Schnitt 358μs darunter.

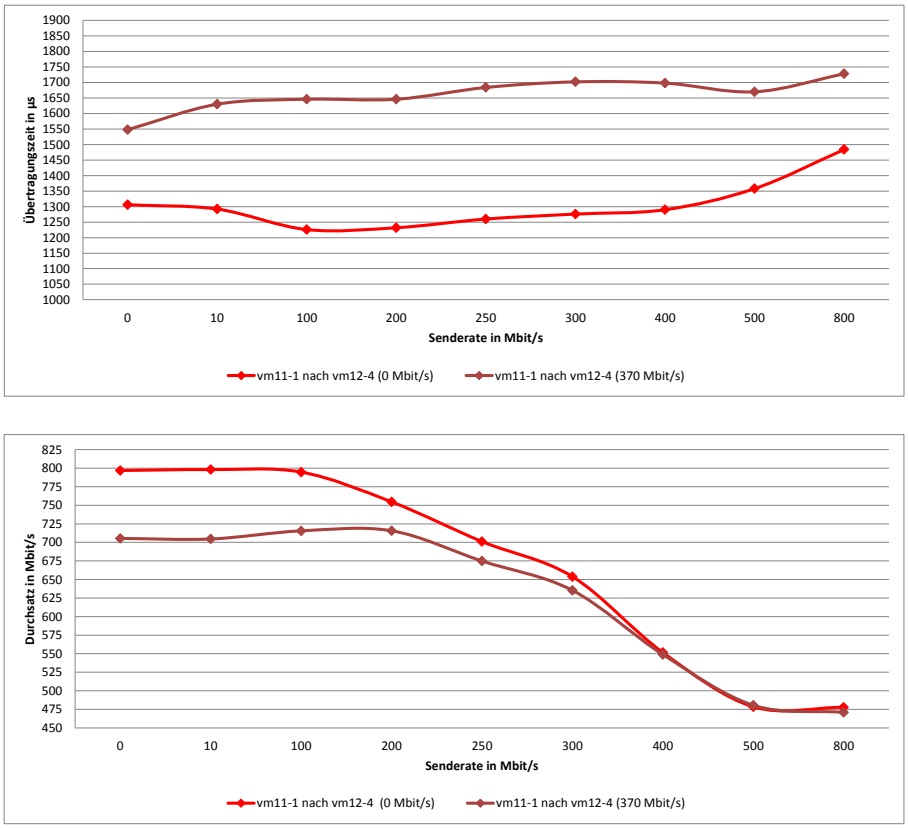

Abbildung 6.5: Latenz und Durchsatz des roten Baumes in Abhängigkeit der Senderate des gelben Baumes aus Abbildung 6.4.d

Für die Messung der Übertragungszeiten zu allen anderen Empfängern wird der rote Baum aus Abbildung 6.4.d zur vollen Gruppengröße, die alle 4 Empfänger umfasst, erweitert. Abbildung 6.6 zeigt den Vergleich zwischen einem komplett unbelasteten Netz sowie für vorbelastete Netze mit Senderaten von 370Mbit/s und 800Mbit/s für sämtliche in Abbildung 6.4.d vorhandenen Multicastbäume. Die Messung erfolgt wieder zwischen vm11-1 als Sender und jeweils 4 Empfänger-VMs, verteilt über die vorhandenen ToR-Switches. Im unbelasteten Fall hängt die Verzögerungszeit vor allem von der Länge des Pfades ab, wofür pro Switch etwa 300μs benötigt werden. Senden alle 4 parallel existierenden Bäume mit einer Rate von 370 Mbit/s oder 800 Mbit/s, ist eine erhöhte Übertragungszeit aufgrund von insgesamt drei Kantenüberlappungen zu den Empfängern vm12-1 und vm12-4 festzustellen. Der im Vergleich stark erhöhte Wert für vm12-1 ist aufgrund der hohen Überlast zwischen ToR-Switch und Host zu erklären, da diese Maschine der Empfänger für alle vier aktiven Multicastbäume darstellt. Somit tritt dort bereits bei einer Rate von 370 Mbit/s eine starke Überlast mit Paketverlust auf, während sich dies für vm12-4 erst bei einer Senderate von 800Mbit/s auswirkt.

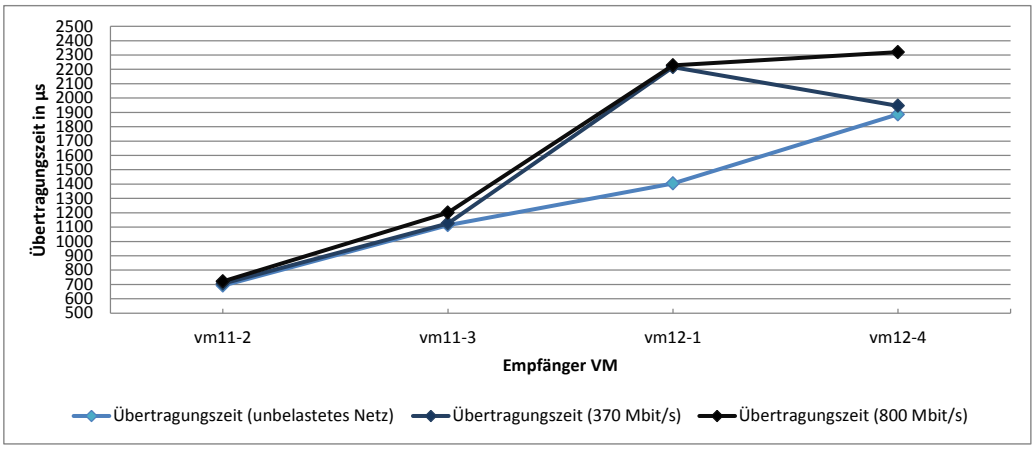

Abbildung 6.6: Verzögerungszeiten für verschiedene Sender im un- und vorbelasteten Netzwerk

6.2.2 Auswertung für Shared-Trees

Um Datagramme über einen Shared-Tree verschicken zu können, werden die Pakete zuerst über eine Unicast-Route an den Rendezvous-Knoten geschickt. Die Pfadlänge und die daraus resultierenden Übertragungszeiten hängen vom Sender ab, da der Rendezvous-Knoten nur bezüglich der Empfängergruppe und nicht nach den Sendern optimiert wird.

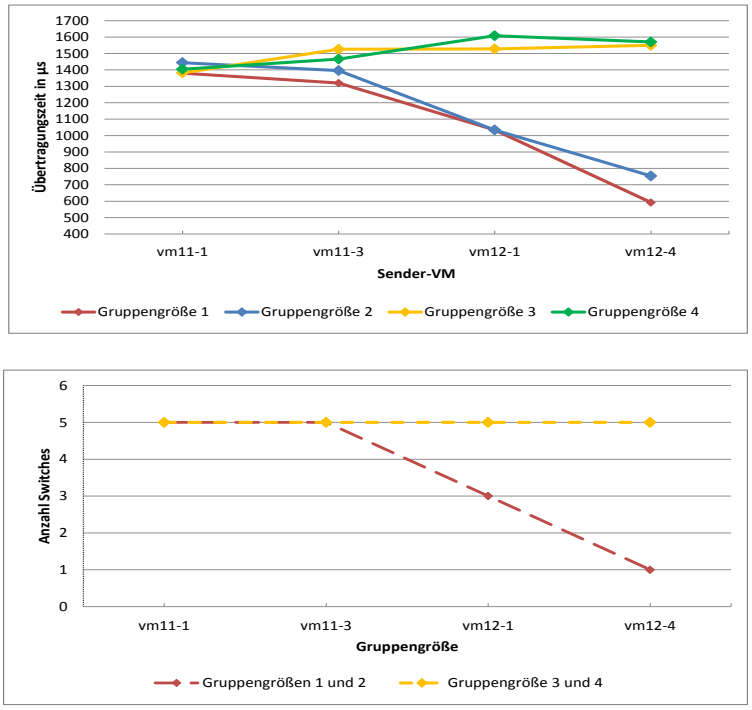

Abbildung 6.7: Übertragungszeit und Anzahl der Switches eines Shared-Trees, inklusive den Unicast-Routen, in Abhängigkeit vom Senderknoten und der Gruppengröße.

Abbildung 6.7 (oben) beleuchtet die Übertragungszeiten in einem Shared-Tree an den festen Empfänger vm12-3 in Abhängigkeit des Senders und der Gruppengröße. Bei einer Größe von 1 degeneriert der Multicastbaum zur Unicast-Kommunikation und das Ergebnis unterscheidet sich nicht von KMB aus Abschnitt 6.2.1. In diesem Fall weist eine Kommunikation ausgehend von vm11-1 die höchste Übertragungszeit auf, da der Pfad mit 5 Switches am längsten ist. Ein Datagramm von vm12-1 oder vm12-4 muss dagegen nur einen Switch durchqueren. Ab einer Gruppengröße von 3 wird einer der beiden Core-Switches als Rendezvous-Knoten gewählt. Dadurch verlängert sich auch der Pfad zu vm12-1 bzw. vm12-4 auf jeweils 5 Zwischenschritte, was so auch eine verlängerte Übertragungszeit mit sich bringt. Die Längen aller Pfade abhängig der Gruppengröße sind in 6.7 (unten) zu sehen. Sie beinhalten sowohl die Unicast-Route zum Rendezvous-Knoten als auch die eigentliche Verteilung entlang des Baumes.

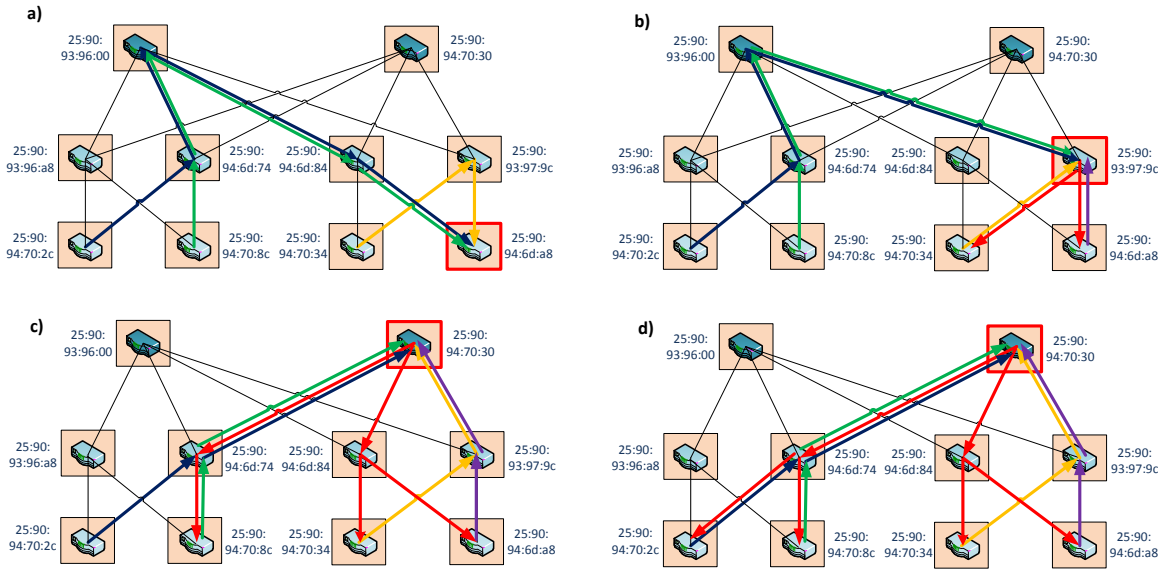

Abbildung 6.8: Wahl des Rendezvous-Knoten bei ansteigender Gruppengröße eines Shared-Trees für die Messungen aus Abbildung 6.7.

Die Bäume der verschiedenen Größen sind in Abbildung 6.8 gezeigt. Dabei ist der Multicastbaum und der Rendezvous-Knoten stets rot und die Unicast-Routen, abhängig vom Sender, jeweils in einer anderer Farbe gekennzeichnet. Abbildung 6.8.a zeigt, dass der Rendezvous-Knoten bei einer Gruppengröße von 1 mit dem Empfängerknoten zusammenfällt. Sind 2 Empfänger vorhanden, wird der nächstliegende Zwischenknoten als Rendezvous-Knoten gewählt, was hier dem Switch *25:90:93:97:9c* entspricht. Ab einer Größe von 3 findet sich der Rendezvous-Knoten dann auf der Ebene der Core-Switches.

Nachfolgend wird das Netz im belasteten Zustand betrachtet. Abbildung 6.9 zeigt die Übertragungszeit und den Durchsatz von vm11-1 an alle Empfänger in Abhängigkeit der Gruppenanzahl. Diese Gruppen haben eine feste Größe von 2 Empfängern und senden ausgehend von vm11-4 parallel mit einer Datenrate von 370 Mbit/s. In den Bildern 6.10.a bis 6.10.d sind die zugehörigen Bäume in aufsteigender Reihenfolge zur Gruppenanzahl zu sehen. Der rote Baum sendet mit voller Bandbreite. Er entspricht dabei immer der Gruppe, für die die Messung zu Grunde liegt. Eine dazu gehörige Unicast-Route zum Rendezvous-Knoten ist orange

dargestellt. Die Belastung einer Kante von 370 Mbit/s durch einen anderen Multicastbaum oder dessen zugehörige Unicast-Route ist durch einen schwarzen Pfeil gekennzeichnet. Um das *Load-Balancing* zu demonstrieren, wurde der rote Baum stets zuletzt berechnet.

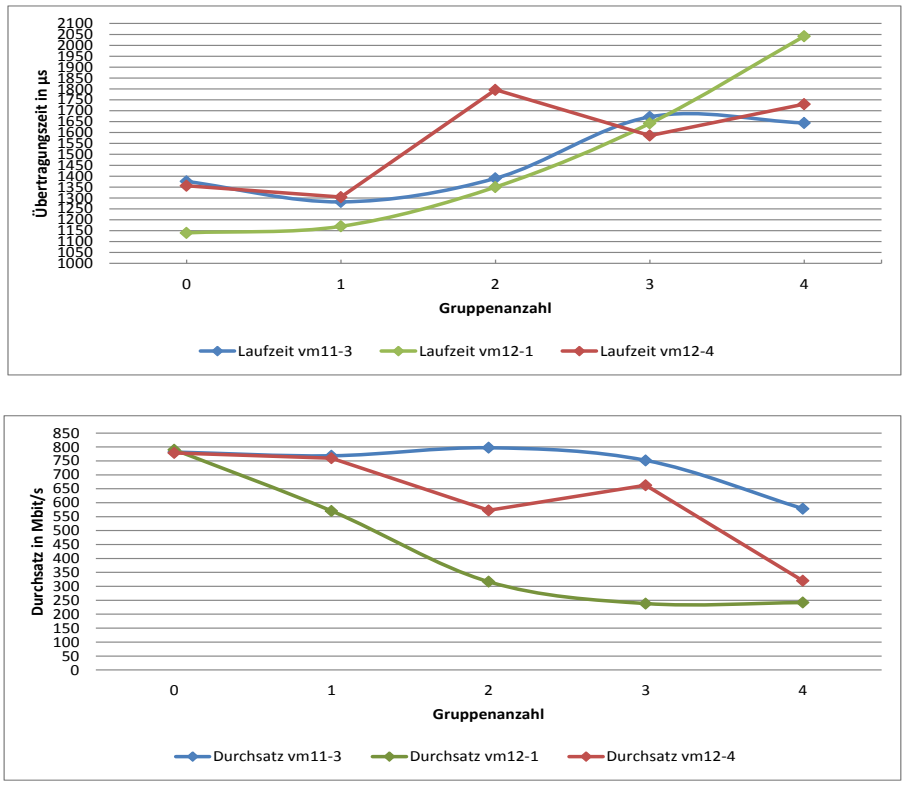

Abbildung 6.9: Latenz und Durchsatz eines Shared-Trees bei steigender Gruppenanzahl.

Die Ergebnisse nach 6.9 (unten) zeigen, dass mit steigender Gruppenanzahl die Wahrscheinlichkeit, eine vorbelastete Kante zu belegen, stark ansteigt. Für den Empfänger vm11-3 kann ab einer Anzahl von 4 Gruppen eine Kantenüberlappung nicht mehr vermieden werden. Für vm12-2 und vm12-4 findet bereits bei einer Gruppenanzahl von 2 eine Doppelbelegung statt. Bei einer Größe von 4 müssen diese sich bereits eine Leitung mit 2 anderen Sendern teilen, wodurch der Durchsatz um mehr als ein Drittel einbricht. Betrachtet man vm12-1, dann fällt der Durchsatz schon bei einer Gruppenanzahl von 1 stark ab. Grund hierfür ist, dass vm12-1 neben vm11-1 als Empfänger für alle im Moment sendenden Multicastgruppen agiert. Das hat zur Folge, dass eine Überlast auf der Leitung zwischen ToR-Switch und Host auftritt. So ist bereits bei einer Gruppengröße von 2 diese Kante komplett ausgelastet und der Durchsatz bricht um mehr als die Hälfte ein. Allgemein ist zu beobachten, dass durch die längeren Wege eines Shared-Trees, im Gegensatz zu quellenbasierten Bäumen, auch mehr Kantenüberlappungen auftreten, was sich durch einen schlechteren Durchsatz äußert. Die Übertragungszeiten aus 6.9 (oben) zeigen das ausgehend von der Durchsatzbetrachtung erwartete Verhalten. Steigende Last führt zu höheren Zeiten sowie einer Abweichung von vm12-1 ab einer Gruppenanzahl von 4 aufgrund der Überlast zwischen ToR-Switch und Host.

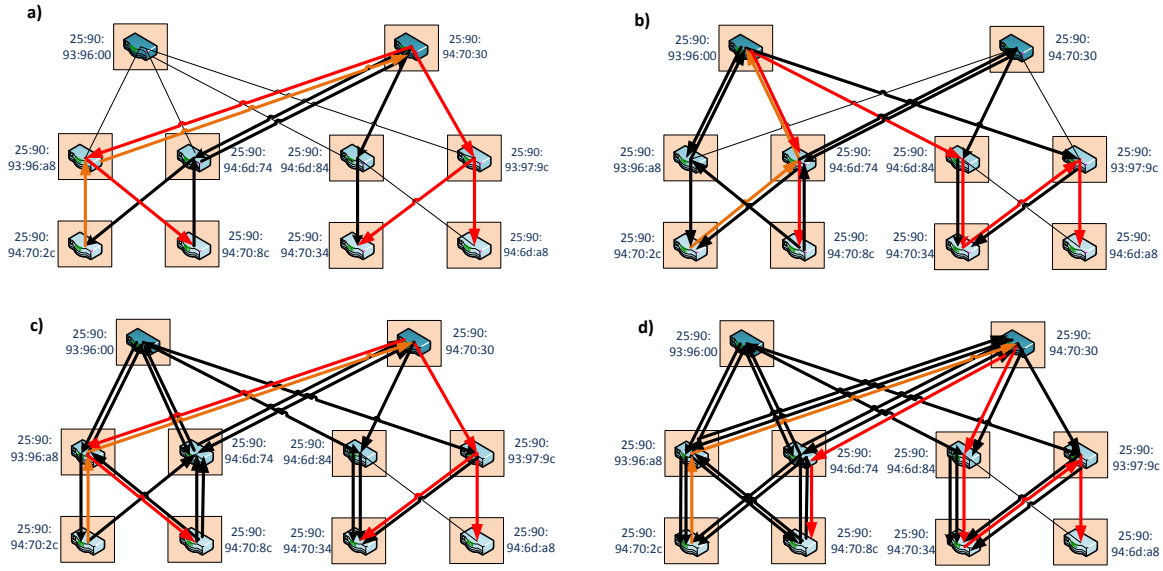

Abbildung 6.10: Load-Balancing in Abhängigkeit der Gruppenanzahl unter Last für die Messungen aus Abbildung 6.9

6.2.3 Vergleich der Routingalgorithmen für dynamische Gruppen

Während in den vorangegangenen Messungen keine Änderungs-Events zur Laufzeit betrachtet wurden, soll nun die Performanz der Routingalgorithmen bei plötzlich auftretenden Gruppenänderungen untersucht werden. Dafür dient ein Baum mit 2 Empfängern als Ausgangsgruppe. Gemessen wird der Durchsatz zwischen vm11-1 und vm12-4 während der Neueinrichtung der Routen. Abbildung 6.11 zeigt die Ergebnisse in Abhängigkeit des Routingalgorithmus und den Änderungen pro Sekunde. Änderungen treten innerhalb einer Zeitspanne von 10 Sekunden in einer festen Reihenfolge auf, wobei vm12-4 der einzige statische Empfänger im Baum darstellt. Zuerst treten die restlichen Empfänger der Gruppe bei, woraufhin 2 weitere diese wieder verlassen. Dieser Vorgang wird dann mit verschiedenen VMs wiederholt.

Neben KMB und Shared-Tree werden auch KMB im *simple-Mode* und iGA als Vergleichsmöglichkeiten herangezogen. Im *simple-Mode* werden die berechneten Routen ohne Abgleich mit dem vorher existierenden Baum auf Datenebene installiert. Existierende Kanten können somit nicht wiederverwendet werden. Dafür ist der Rechenaufwand aber um einiges geringer. Weiterhin soll iGA als Stellvertreter für inkrementelle Algorithmen dienen, bei denen keine Neuberechnung stattfindet, sondern ein vorher existierender Baum modifiziert wird. Dadurch werden die Flow-Änderungen auf Datenebene minimiert.

Da sämtliche Algorithmen zu Beginn auf KMB beruhen, zeigt sich in Abbildung 6.11 bei 0.1 Änderungen/s für alle Kurven ein leichter Paketverlust, der auf der initialen Neueinrichtung der Routen beruht. Signifikante Unterschiede sind erst ab einer Änderungsrate von 0.3 zu erkennen. Die Möglichkeit alte Routen bei KMB beizubehalten, erweist sich bei steigender Änderungszahl gegenüber des reduzierten Rechenaufwandes von KMB im *simple-Mode* als

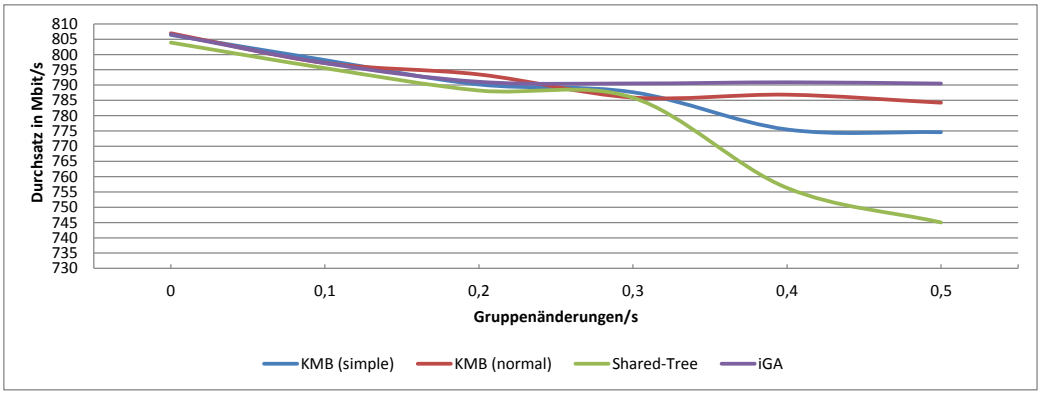

Abbildung 6.11: Vergleich der Routingalgorithmen abhängig von der Anzahl der Gruppenänderungen

signifikanter Vorteil. Daraus lässt sich schließen, dass die Einrichtung der Routen auf Datenebene zeitkritischer ist, als eine erhöhte Rechenzeit im Controller. Die Neueinrichtung eines Shared-Trees hingegen hat im Test die größten Paketverluste zu verzeichnen, was sich in einer Reduzierung des Durchsatzes wiederspiegelt. Der Grund hierfür ist vor allem auf die komplette Neueinrichtung der Unicast-Routen beim Wechsel des Rendezvous-Knoten zurückzuführen. Das beste Ergebnis liefert erwartungsgemäß iGA. Jedoch fällt der Unterschied zu KMB relativ gering aus. Es überwiegen deshalb klar die Vorteile einer besseren Routenwahl durch KMB, weshalb der inkrementelle Ansatz in dieser Arbeit nicht weiter verfolgt wurde.

6.2.4 Overhead bezogen auf die Flow-Tabellen-Größe

Während ein Shared-Tree in der Routenwahl und der Performance bei Gruppenänderungen eindeutig schlechtere Ergebnisse liefert, liegt der Vorteil in der reduzierten Flowanzahl auf Datenebene. Bei einem drohenden Tabellenüberlauf findet deshalb ein Wechsel des Routingalgorithmus statt. Der Anstieg der Tabelleneinträge in Abhängigkeit der Gruppenanzahl ist in Abbildung 6.12 dargestellt. Die Kurven zeigen die durchschnittliche sowie die maximale Anzahl aller Flows über sämtliche Switches im Netz bei der Verwendung von KMB (rote Kurven) und Shared-Tree (grüne Kurven). Im Vergleich benötigen quellenbasierte Bäume, für dieselbe Multicastgruppe, das $1,6$-fache an Flow-Einträgen als ein entsprechender Shared-Tree. Die maximale Anzahl von Flows in einem Switch wächst für KMB im Test um 5 Einträge pro Multicastgruppe, die durchschnittliche Anzahl um etwa 2.4 Einträge pro Switch. Im Vergleich dazu wächst ein Shared-Tree nur um 3 Einträge in der Maximums-Kurve sowie um durchschnittlich 1.7 Einträge pro Switch. Diese Werte Verhalten sich jedoch nicht immer linear. Die Flowanzahl ist abhängig von der Wegwahl des Routingalgorithmus. Längere Wege führen demnach auch zu mehr Flow-Einträgen. Für die fünfte Multicastgruppe in diesem Beispiel wählt KMB aufgrund der Kantenauslastung einen leicht verlängerten Weg, was die durchschnittliche Anzahl an Tabelleneinträge von den erwarteten 13 auf 13.3 erhöht. Das erklärt die leichte Krümmung der Kurve aus Abbildung 6.12.

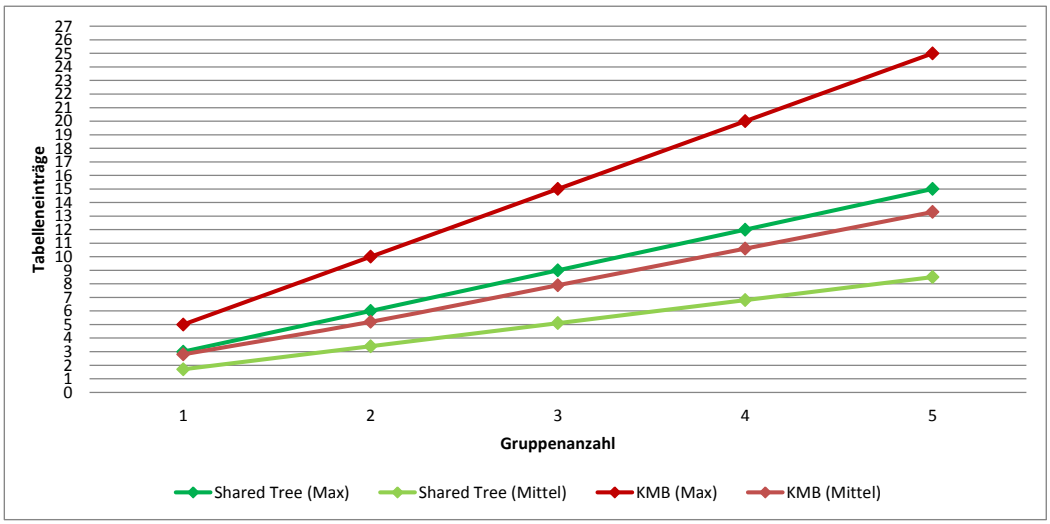

Abbildung 6.12: Anzahl der maximalen und durchschnittlichen Tabelleneinträge auf Datenebene abhängig von der Gruppenanzahl

6.2.5 Ausfallbetrachtung

Zuletzt wird die Robustheit des Systems untersucht. Falls ein Switch- oder Leitungsausfall auftritt, wird dieses Ereignis an die Routingkontrolle gemeldet. Diese führt eine Neuberechnung der betroffenen Multicastbäume oder Unicast-Routen aus. Der Durchsatz ist dabei durch die Zeit, die das Controllerprogramm benötigt, um den Ausfall zu erkennen sowie durch die Berechnung und Einrichtung der neuen Routen bestimmt. Letzteres wurde bereits im Rahmen von Gruppenänderungen untersucht. Die Verzögerungszeit, vom Auftritt eines Ausfalles bis hin zur eigentlichen Neuberechnung, wurde mit durchschnittlich 9.67 Sekunden gemessen. Innerhalb dieser Zeitspanne treten Paketverluste für die betroffenen und im Moment aktiven Multicastgruppen auf.

Für diesen Test wird ein Multicastbaum eingerichtet, der über den Switch mit der DPID „25:90:94:6d:74“ verläuft. Anschließend wird eine Firewall-Regel aktiviert, mit der man Verbindungen zum Controller für diesen Switch blockiert. Gleichzeitig werden alle zugehörigen Einträge in der Flow-Table gelöscht. Der *TopologyListener* meldet daraufhin den Ausfall und die Ausfallbehandlung des Multicastdienstes bestimmt alle in Folge dessen ausgefallenen Leitungen. Anschließend werden sämtliche eingerichteten Bäume nach Überlappungen mit einer ausgefallenen Kante durchsucht und an die Routenberechnung weitergegeben. Die Kantengewichte werden zuvor auf unendlich gesetzt, um eine Neubelegung dieser Kanten zu verhindern und dabei nicht auf den nächsten Aktualisierungszyklus der Netzzustandsverwaltung warten zu müssen.

Die gemessene Zeit von 9.67 Sekunden wird hauptsächlich durch die Zykluszeit der LLDP-Anfragen bestimmt. Bei Floodlight beträgt sie initial 15 Sekunden. Erhält der Controller für eine Verbindungsleitung keine LLDP-Nachricht mehr, resultiert dies in einem *Link-Down*-Ereignis. Aufgrund der Größenordnung dieser Erkennungszeit ist die Zeit für die Neube-

rechnung und Einrichtung im betrachteten Testumfeld vernachlässigbar gering. Daraus kann geschlossen werden, dass diese Zeitspanne von 15 Sekunden, unabhängig vom gewählten Routingalgorithmus, eine obere Schranke für die Ausfallerkennung darstellt. Durch das Herabsetzen der Zykluszeit wäre eine schnellere Reaktion bei Ausfällen zu erwarten. Jedoch führt dies zu einer Erhöhung der *Packet_In*-Nachrichten am Controller. Die zusätzliche Controllerlast ist dann stark von der Größe des vorliegenden Netzes abhängig. Falls der Ausfall einer Verbindung zu einem *Port-Down*-Ereignis führt, muss Floodlight nicht auf die LLDP-Antworten warten und registriert den Ausfall, ohne Verzögerung, sofort. In diesem Fall gleicht die Durchsatzbetrachtung der, für die Gruppenänderungen.

6.2.6 Fazit

Die Ergebnisse aus den vorangegangenen Abschnitten zeigen, dass die Verzögerungszeiten eines Shared-Trees aufgrund längerer Pfade im Mittel höher sind als für einen quellbasierten Baum. Die Länge hängt stark von der Wahl des Rendezvous-Knoten und somit von der Anzahl und Position der Empfänger ab. Allgemein konnte eine Verzögerungszeit von $300\mu s$ pro Switch festgestellt werden. Unabhängig vom Routingalgorithmus beträgt die gemessene Verzögerungszeit bei einem Ausfall vertretbare 9.67 Sekunden.

Der Durchsatz wird vor allem durch die Lastverteilung bestimmt. Auch hier hat KMB den Vorteil, bessere Wege zu finden und reduziert dadurch die Wahrscheinlichkeit einer Überlast auf den Kanten. Bei Shared-Trees führen Umwege über den Rendezvous-Knoten und der höhere Änderungsaufwand für die Unicastrouten zu verschlechtertem Durchsatzverhalten. Die Vorteile liegen wiederum in der geringeren Anzahl der Tabelleneinträge, die in der Flow-Table für die Einrichtung nötig sind. Somit eignet sich ein Shared-Tree besonders dann, wenn der Tabellenplatz knapp wird und ein Überlauf droht.

Die im Zuge der Evaluation durchgeführten Tests implizieren, dass der Multicastdienst durchaus in der Praxis Anwendung finden kann. Die ermittelten Verzögerungszeiten zeigen, dass selbst mit virtualisierter Hardware ein performanter Datenaustausch in OpenFlow-basierten Netzen möglich ist. Außerdem spiegeln die im Zuge der Durchsatzmessungen dargelegten *Load Balancing* Eigenschaften eine gute Ausnutzung heutiger Datencenter-Topologien wieder. Dies, in Zusammenhang mit der hohen Flexibilität und der Optimierung der Routingalgorithmen, könnten in Zukunft ein auschlaggebendes Argument für die Integration von SDN-basierten Diensten in Rechnernetze darstellen.

7 Zusammenfassung und Ausblick

In diesem Kapitel werden in 7.1 die vorherigen Abschnitte zusammengefasst und das Erreichen der Ziele dieser Diplomarbeit bewertet. Anschließend wird in Abschnitt 7.2 ein Ausblick auf zukünftige Arbeiten gegeben.

7.1 Zusammenfassung

Das Ziel dieser Arbeit war die Konzeption und Implementierung eines OpenFlow-basierten IP-Multicast-Dienstes für Datenzentren. Dieser sollte den Kriterien aus Kapitel 3.3 bezüglich der Skalierbarkeit, Effizienz, Robustheit und Integration genügen und ein möglichst optimales Routing auf einer globalen Netzwerksicht realisieren. Waren bisherige Multicastlösungen fast ausschließlich verteilt implementiert, bietet das Software defined Networking (SDN) einen zentralisierten Ansatz, der für ein neuartiges Multicast-Routing ausgenutzt werden kann. In Kapitel 4 wurden im Zuge einer Konzeption konkrete Prozesse herausgearbeitet, die für einen zentralen Multicastdienst umgesetzt werden müssen. Eine Untersuchung des Steinerbaumproblems resultierte in der Auswahl der Heuristik KMB zur Realisierung eines verbesserten aber trotzdem skalierbaren Routings. Für eine gleichmäßige Lastverteilung im Datencenternetzwerk wurde eine Routingmetrik, bestehend aus einer gewichteten Summe aus Bandbreite und Entfernung, bevorzugt. Die Auswertung hat gezeigt, dass so eine gute Lastverteilung erreicht wird, ohne einen komplexeren Routingalgorithmus implementieren zu müssen. Dabei ist es durch proaktive Einrichtung der Routen, sowie durch den Vergleich bestehender Bäume bei Gruppenänderungen gelungen, eine effiziente Routeneinrichtung zu realisieren und die Controllerverzögerung als potentiellen Flaschenhals zu reduzieren. Zusätzlich ermöglichen effiziente Datenstrukturen basierend auf Hash-Tabellen einen schnellen Zugriff auf die verschiedenen Controllerinformationen wie Multicastgruppen, eingerichtete Routen, IGMP-Zählerwerte und die kantengewichtete Netztopologie. Skalierbarkeitsprobleme bezüglich der Flow-Einträge in den Switches werden durch einen Shared-Tree Algorithmus, auf den während der Laufzeit gewechselt werden kann, vorgebeugt. Der Multicastdienst ist außerdem in der Lage, Topologieänderungen und insbesondere Leitungs- und Switchausfälle zu erkennen und die eingerichteten Routen automatisch umzuleiten. Sowohl bei Gruppen- als auch Topologieänderungen zeigten die Testergebnisse Paketverluste in vertretbarem Umfang. Für die Verwaltung der Gruppen im Controllerprogramm wird IGMP implementiert. Dabei ist der Multicastdienst in der Lage sowohl IGMPv1, IGMPv2, als auch IGMPv3 Nachrichten zu verarbeiten und kann somit ohne weiteres in bestehende OpenFlow-Netze integriert werden.

Die erarbeitete Konzeption wurde anschließend in Kapitel 5 als Erweiterung für den SDN-Controller Floodlight in Java implementiert. Prozesse, die im Zuge der Konzeption herausgearbeitet wurden, konnten eins zu eins in Floodlight-Module übernommen werden. Als wichtigste Datenstruktur kam die *ConcurrentHashMap* zum Einsatz. Sie ermöglicht einen schnellen und uneingeschränkten parallelen Zugriff auf die gespeicherten Controllerinformationen. Damit plötzlich auftretende Ereignisse wie Topologie- oder Gruppenänderungen schnell an die Routenberechnung gemeldet werden können, wurde für die Kommunikation zwischen den Modulen auf das *Observer-Pattern* zurückgegriffen.

Schließlich wurde der Dienst auf dem in Kapitel 6.1 vorgestellten Testbed evaluiert. Die Testumgebung spiegelt einen Ausschnitt aus einer realistischen *Fat-Tree* Topologie wieder. Dabei wurden *Open-vSwitches* verwendet. Die Ergebnisse haben gezeigt, dass die Verzögerungszeiten einer Route bei etwa $300\mu s$ pro Switch liegen und der Durchsatz in einem lastfreien Netz annähernd die maximale Bandbreite erreicht. Wobei beachtet werden muss, dass es sich bei diesen Werten um virtuelle Switches handelt. Eine Evaluierung auf realer Hardware war zum Zeitpunkt dieser Diplomarbeit nicht möglich. Trotzdem kann geschlussfolgert werden, dass der hier beleuchtete Ansatz für einen Multicastdienst auf OpenFlow-Basis eine gute und vor allem flexible Alternative zu bestehenden Multicastunterstützungen in Datencenternetzen gesehen werden kann.

7.2 Zukünfige Arbeiten

Zum Zeitpunkt dieser Arbeit unterstützen der Floodlight Controller und die hier verwendete virtualisierte Hardware nur den OpenFlow Standard 1.0. Für eine saubere Umsetzung von Multicasting auf einem OpenFlow-Switch, sollte dieser jedoch mindestens Version 1.1.0 unterstützen. Der Grund hierfür liegt in der fehlenden Umsetzung der Group-Tables, die es erlauben würden ein Paket über mehrere Ports zu senden. Aus diesem Grund beruht die Konzeption dieser Arbeit bereits auf Version 1.3, obwohl die beschriebene Implementierung in Kapitel 5 auf Version 1.0 basiert. Eine Implementierung und Test mit Group-Table Unterstützung war während dieser Diplomarbeit nicht möglich und sollte Gegenstand zukünftiger Untersuchungen sein.

Zusätzlich sind Verbesserungen des hier erarbeiteten Multicastdienstes denkbar, die auf einer feineren Unterscheidung der Hostgeräte basieren und somit eine zielgenauere Auslieferung bieten können. Während die hier vorgestellte Lösung darauf verzichtet, einzelne Hosts zu verwalten, könnte dies jedoch einen zusätzlichen Gewinn an Performance auf Datenschicht mit sich bringen. Außerdem wäre eine Behandlung von IGMPv3-Reports, bei denen der Empfänger selbst bestimmen kann von welchen Sendern Multicastnachrichten ausgeliefert werden sollen, möglich.

Als Erweiterung ist es außerdem vorstellbar, IGMP durch eine entsprechende auf OpenFlow zugeschnitte Lösung zu ersetzen. Entsprechende Vorschlänge wurden von Marcondes et al. [MSG+12] für IPTV-Anwendungen aufgegriffen. Die Autoren kommen zu dem Schluss, dass IGMP bei hoher Gruppendynamik einen großen Einfluss auf die Verzögerung hat. Der Nachteil ist allerdings, dass eine entsprechende Anpassung der Hosts nötig sein wird. Damit kann der Dienst nicht mehr so einfach in ein bestehendes OpenFlow-Netzwerk integriert werden.

Zusätzlich muss angemerkt werden, dass sich die hier vorgestellte Lösung auf einen SDN-Controller reduziert. Um die Verfügbarkeit zu erhöhen, könnten zukünftige Arbeiten ein verteiltes Controllerprogramm anstreben. Das würde das Risiko, dass der Controller ein *Single-Point-of-Failure* darstellt reduzieren und könnte die Controller-Last in großen Netzwerken noch weiter verringern.

Literaturverzeichnis

[AFLV08] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74. ACM, 2008.

[AFM92] S. Armstrong, A. Freier, and K. Marzullo. Multicast transport protocol. Technical report, RFC 1301, Internet Engineering Task Force, February 1992. Available from http://www. rfc-editor. org/rfc/rfc1301. txt, 1992.

[AFRR+10] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 19–19, 2010.

[ANS+05] A. Adams, J. Nicholas, W. Siadak, et al. Protocol independent multicast-dense mode (PIM-DM): Protocol specification (revised). *IETF, RFC 3973*, 2005.

[BBAL+11] B. Boughzala, R. Ben Ali, M. Lemay, Y. Lemieux, and O. Cherkaoui. Openflow supporting inter-domain virtual machine migration. In *Wireless and Optical Communications Networks (WOCN), 2011 Eighth International Conference on*, pages 1–7. IEEE, 2011.

[BBB07] Z. Begic, M. Bolic, and H. Bajric. Centralized versus distributed replication model for multicast replication. In *ELMAR, 2007*, pages 187–191. IEEE, 2007.

[BC97] P. Berman and C. Coulston. On-line algorithms for Steiner tree problems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 344–353. ACM, 1997.

[BFC93] T. Ballardie, P. Francis, and J. Crowcroft. Core based trees (CBT). *ACM SIGCOMM Computer Communication Review*, 23(4):85–95, 1993.

[BO10a] Marcos LP Bueno and Gina MB Oliveira. Pareto-Based Optimization of Multicast Flows with QoS and Traffic Engineering Requirements. In *Network Computing and Applications (NCA), 2010 9th IEEE International Symposium on*, pages 257–260. IEEE, 2010.

[BO10b] M.L.P. Bueno and G.M.B. Oliveira. Multicast flow routing: Evaluation of heuristics and multiobjective evolutionary algorithms. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[Brü07] Dieter Brümmerhoff. *Finanzwissenschaft* -. Oldenbourg Verlag, München, 9. vollst. überarb. u. erw. edition, 2007.

[CB04] Jorge Crichigno and Benjamín Barán. Multiobjective multicast routing algorithm for traffic engineering. In *Computer Communications and Networks, 2004. ICCCN 2004. Proceedings. 13th International Conference on*, pages 301–306. IEEE, 2004.

[CDK+02] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet group management protocol, version 3. 2002.

[CHZ+11] K. Chen, C. Hu, X. Zhang, K. Zheng, Y. Chen, and A.V. Vasilakos. Survey on routing in data centers: insights and future directions. *Network, IEEE*, 25(4):6–10, 2011.

[CKS06] M.J. Christensen, K. Kimball, and F. Solensky. Considerations for Internet group management protocol (IGMP) and multicast listener discovery (MLD) snooping switches. 2006.

[Clo53] C. Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32(2):406–424, 1953.

[CMPS02] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction and applicability statements for internet standard management framework. *RFC3410, IETF, December*, 2002.

[Coo06] Stephen Cook. The P versus NP problem. *The millennium prize problems*, page 86, 2006.

[CW98] JS Crawford and AG Waters. Heuristics for ATM Multicast Routing. *ATM'98 Sixth IFIP Wokshop on Performance Modelling and Evaluation of ATM Networks. Participants Proceedings: Tutorial Papers*, page 5, 1998.

[DFH99] S. Deering, W. Fenner, and B. Haberman. Multicast listener discovery (MLD) for IPv6. Technical report, RFC 2710, October, 1999.

[Die08] Tim Dierks. The transport layer security (TLS) protocol version 1.2. *IETF, RFC 5246*, 2008.

[Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[DL93] M. Doar and I. Leslie. How bad is naive multicast routing? In *INFOCOM'93. Proceedings. Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future, IEEE*, pages 82–89. IEEE, 1993.

[Dür12] F. Dürr. Towards Cloud-assisted Software-defined Networking. Technical report, Institute for Parallel and Distributed Systems (IPVS), Universität Stuttgart, 2012.

[Edm67] Jack Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71:233–240, 1967.

[Fen97] W.C. Fenner. Internet group management protocol, version 2. *RFC 2236*, 1997.

[FHKH06] B. Fenner, M. Handley, I. Kouvelas, and H. Holbrook. Protocol independent multicast-sparse mode (PIM-SM): protocol specification (revised). 2006.

[flo] Floodlight. `http://floodlight.openflowhub.org/`. [Online; Zugriff 08.01.2013].

[Fou12] Open Networking Foundation. Software-defined Networking: The New Norm for Networks. *ONF*, 2012.

[FT87] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[GC08] John F Gantz and Christopher Chute. The diverse and exploding digital universe: An updated forecast of worldwide information growth through 2011. IDC, 2008.

[GKP+08] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[goo] 100Gbps and beyond: What lies ahead in the world of networking. `http://arstechnica.com/information-technology/2013/02/100gbps-and-beyond-what-lies-ahead-in-the-world-of-networking/2/`. [Online; Zugriff 15.03.2013].

[Gui] I. Guis. The SDN Gold Rush To The Northbound API. `http://www.sdncentral.com/guest-blog-posts/the-sdn-gold-rush-to-the-northbound-api/2012/11/`. [Online; Zugriff 08.01.2013].

[IBM] IBM. Programmable Network Controller. `http://www-03.ibm.com/systems/networking/software/pnc/index.html`. [Online; Zugriff 08.01.2013].

[IW91] M. Imase and B.M. Waxman. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991.

[KMB81] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta informatica*, 15(2):141–145, 1981.

[Kos10] Arie Koster. *Graphs and Algorithms in Communication Networks - Studies in Broadband, Optical, Wireless, and Ad Hoc Networks*. Springer, Berlin, Heidelberg, 2010.

[KP99] S. Keshav and S. Paul. Centralized multicast. In *Network Protocols, 1999.(ICNP'99) Proceedings. Seventh International Conference on*, pages 59–68. IEEE, 1999.

[KPP93] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos. Multicast routing for multimedia communication. *Networking, IEEE/ACM Transactions on*, 1(3):286–292, 1993.

[KRT99] J. Kleinberg, Y. Rabani, and É. Tardos. Fairness in routing and load balancing. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 568–578. IEEE, 1999.

[Kru56] J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[KSIT11] H. Kudou, M. Shimamura, T. Ikenaga, and M. Tsuru. Effects of routing granularity on communication performance in OpenFlow networks. In *Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on*, pages 590–595. IEEE, 2011.

[KSS12] D. Kotani, K. Suzuki, and H. Shimonishi. A design and implementation of OpenFlow Controller handling IP multicast with Fast Tree Switching. In *Applications and the Internet (SAINT), 2012 IEEE/IPSJ 12th International Symposium on*, pages 60–67. IEEE, 2012.

[Kui02] Kuipers, Fernando and Van Mieghem, Piet. Mamcra: a constrained-based multicast routing algorithm. *Comput. Commun.*, 25(8):802–811, May 2002.

[KWE+11] James Kempf, Scott Whyte, Jonathan Ellithorpe, Peyman Kazemian, Mart Haitjema, Neda Beheshti, Stephen Stuart, and Howard Green. OpenFlow MPLS and the open source label switched router. In *Proceedings of the 23rd International Teletraffic Congress*, pages 8–14. ITCP, 2011.

[Leh] Lehrstuhl für Informatik - RWTH AACHEN. Datenstrukturen und Algorithmen. `http://www.stormware-computer.de/script/kapitel_4_1.pdf`. [Online; Zugriff 28.04.2013].

[Lei85] C.E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *Computers, IEEE Transactions on*, 100(10):892–901, 1985.

[lld05] IEEE Standard Local and Metropolitan Area Networks. Station and Media Access Control Connectivity Discovery. *IEEE Std 802.1AB-2005*, pages 0_1 -158, 2005.

[MAB+08] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[Meh88] K. Mehlhorn. A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters*, 27(3):125–128, 1988.

[Moy94] J. Moy. Multicast Extensions to OSPF. *SRI Network Information Center*, 1994.

[MSG+12] C.A.C. Marcondes, T.P.C. Santos, A.P. Godoy, C.C. Viel, and C.A.C. Teixeira. CastFlow: Clean-slate multicast approach using in-advance path processing in programmable networks. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pages 000094–000101. IEEE, 2012.

[MYLSP07] I. Martinez-Yelmo, D. Larrabeiti, I. Soto, and P. Pacyna. Multicast traffic aggregation in MPLS-based VPN networks. *Communications Magazine, IEEE*, 45(10):78–85, 2007.

[Net] Big Switch Networks. Big Network Controller. `http://www.bigswitch.com/products/big-network-controller/`. [Online; Zugriff 08.01.2013].

[NMPF+09] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 39–50. ACM, 2009.

[NNDKB08] D. Nace, L. Nhat Doan, O. Klopfenstein, and A. Bashllari. Max–min fairness in multi-commodity flows. *Computers & Operations Research*, 35(2):557–573, 2008.

[ONF] Open Networking Foundation - ONF. `https://www.opennetworking.org/`. [Online; Zugriff 23.05.2013].

[Onl] Chip Online. OpenFlow als Quasi-Standard fuer SDN. `http://business.chip.de/artikel/Software-definierte-Netze-Der-Weg-aus-der-Netzwerk-Krise-3_59100735.html`. [Online; Zugriff 08.01.2012].

[OP05] C.A.S. Oliveira and P.M. Pardalos. A survey of combinatorial optimization problems in multicast routing. *Computers & Operations Research*, 32(8):1953–1981, 2005.

[P+11] B. Pfaff et al. OpenFlow Switch Specification Version 1.1.0 Implemented (Wire Protocol 0x02), 2011.

[P+12] B. Pfaff et al. OpenFlow Switch Specification Version 1.3.0 Implemented (Wire Protocol 0x04), 2012.

[Pac] Hewlet Packard. Virtual Application Networks SDN Controller. `http://h17007.www1.hp.com/us/en/solutions/technology/van/index.aspx`. [Online; Zugriff 08.01.2013].

[Plu82] D. Plummer. Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48. bit Ethernet address for transmission on Ethernet hardware. 1982.

[Pos80] J. Postel. User datagram protocol. *Isi, RFC 768*, 1980.

[Pos81] J. Postel. Transmission control protocol. *RFC 793*, 1981.

[PPX98] J.C. Pasquale, G.C. Polyzos, and G. Xylomenos. The multimedia multicasting problem. *Multimedia Systems*, 6(1):43–59, 1998.

[Pri57] R.C. Prim. Shortest connection networks and some generalizations. *Bell system technical journal*, 36(6):1389–1401, 1957.

[Pro02] H Proemel. *The Steiner tree problem : a tour through graphs, algorithms, and complexity*. Vieweg, Braunschweig, 2002.

[PZH05] M. Piechowiak, P. Zwierzykowski, and S. Hanczewski. Performance Analysis of Multicast Heuristic Algorithms. In *Third International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks*, page 41. Citeseer, 2005.

[RES06] S. Ratnasamy, A. Ermolinskiy, and S. Shenker. Revisiting IP multicast. *Computer Communication Review*, 36(4):15, 2006.

[Rey02] J. Reynolds. Assigned numbers: RFC 1700 (on-line database). 2002.

[RMSRM99] S. Raghavan, G. Manimaran, and C. Siva Ram Murthy. A rearrangeable algorithm for the construction of delay-constrained dynamic multicast trees. *Networking, IEEE/ACM Transactions on*, 7(4):514–529, 1999.

[SFL+98] T. Speakman, D. Farinacci, S. Lin, A. Tweedly, N. Bhaskar, R. Edmonstone, K. Johnson, R. Sumanasekera, L. Vicisano, J. Gemmell, et al. Pragmatic general multicast. *InternetDraft, August*, 1998.

[Spe09] OpenFlow Switch Specification. OpenFlow Switch Specification Version 1.0.0 Implemented (Wire Protocol 0x01), 2009.

[Tar06] Robert E Tarjan. Finding optimum branchings. *Networks*, 7(1):25–35, 2006.

[TH00] D. Thaler and C. Hopps. Multipath issues in unicast and multicast next-hop selection. Technical report, RFC 2991, November, 2000.

[TMJ12] F. Tichy, T Moschny, and A. Jannesari. Netzwerk-Kenngroessen und -Topologien, 2012. Universität Karlsruhe.

[TW12] Andrew S. Tanenbaum and Prof. David J. Wetherall. *Computernetzwerke*. Pearson Studium, Muenchen, 5. aktualisierte auflage edition, 2012.

[Uni] Stanford University. Beacon. `https://openflow.stanford.edu/display/Beacon/Home`. [Online; Zugriff 01.03.2013].

[VC10] L. Vegoda and M. Cotton. IANA Guidelines for IPv4 Multicast Address Assignments. 2010.

[Wax88] B.M. Waxman. Routing of multipoint connections. *Selected Areas in Communications, IEEE Journal on*, 6(9):1617 –1622, dec 1988.

[WDP88a] D. Waitzman, SE Deering, and C. Partridge. Distance vector multicast routing protocol. 1988.

[WDP88b] David Waitzman, SE Deering, and C Partridge. Distance vector multicast routing protocol. 1988.

[Wik] Wikipedia. Traffic engineering — Wikipedia, the free encyclopedia. [Online; Zugriff 28.04.2013].

[Wit99] Ralph Wittmann. *Multicast: Protokolle und Anwendungen*. Dpunkt, Heidelberg, 1999.

[WVK+01] B. Whetten, L. Vicisano, R. Kermode, M. Handley, S. Floyd, and M. Luby. Reliable multicast transport building blocks for one-to-many bulk-data transfer. *RFC3048, January*, 1947, 2001.

[WY93] J. Westbrook and D. Yan. Greedy algorithms for the on-line Steiner tree and generalized Steiner problems. *Algorithms and Data Structures*, pages 622–633, 1993.

[Yas06] S. Yasukawa. Signaling Requirements for Point-to-Multipoint Traffic-Engineered MPLS Label Switched Paths (LSPs). 2006.

[ZLT01] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm, 2001.

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben. Wörtliche und sinngemäße Übernahmen aus anderen Quellen habe ich nach bestem Wissen und Gewissen als solche kenntlich gemacht.

Stuttgart, den 21. Mai 2013 _____