Algorithmen zum effizienten Deployment virtueller Netzwerkservices

Michael Till Hermann Beck

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität München

vorgelegt von
Michael Till Hermann Beck

München, den 28. November 2016

Algorithmen zum effizienten Deployment virtueller Netzwerkservices

Michael Till Hermann Beck

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität München

vorgelegt von
Michael Till Hermann Beck

1. Gutachter:	Prof. Dr. Claudia Linnhoff-Popien
2. Gutachter:	Prof. Dr. Burkhard Stiller
Tag der Einreichung:	28. November 2016
Tag der Disputation:	7. Februar 2017

Eidesstattliche Versicherung

(siehe Promotionsordnung vom 12. Juli 2011, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbständig, ohne unerlaubte Beihilfe angefertigt ist.

München, 08.02.2017 Michael Till Hermann Beck

Zusammenfassung

Die Virtualisierung von Netzfunktionen (NFV, Network Function Virtualization) ist ein zentrales Konzept zukünftiger Mobilfunknetze: Statt wie in klassischen Netzen rein auf Hardwarekomponenten zu setzen, deren Logik untrennbar mit der eigentlichen Hardware verwoben ist, wird die Funktionalität in NFV-Netzen innerhalb virtueller Netzwerkfunktionen gekapselt und von der eigentlichen physischen Hardware separiert. Hochspezialisierte Hardwareboxen werden durch viel flexiblere Standardhardware ersetzt, auf der nun unterschiedliche Netzwerkfunktionen installiert werden können.
Ein Kernkonzept dabei ist die Integration von Cloud Computing-Technologien innerhalb der Mobilfunk-Kerninfrastruktur: Dies ermöglicht es dem Mobilfunkprovider, die Konfiguration des Netzes viel dynamischer an die sich ständig verändernden Anforderungen des Marktes anzupassen. Sollen neue Netzwerkservices installiert werden, kann dazu ein Großteil der bereits vorhandenen physischen Infrastruktur wiederverwendet werden; die vorhandene Hardware muss nicht komplett ausgetauscht werden. Die Integration neuer Services erfolgt stattdessen durch den wesentlich kosteneffizienteren Austausch von (virtuellen) Netzwerkfunktionen – und nicht durch Austausch von Hardware. Mobilfunkprovider werden in Zukunft in der Lage sein, viel einfacher und effizienter zusätzliche Netzfunktionen dort zuzuschalten, wo sie gebraucht werden, ohne dass jedes Mal Änderungen an der eigentlichen Hardware-Konfiguration erforderlich werden. Darüber hinaus können Netzfunktionen flexibel auf andere Komponenten migriert werden, wenn Hardwarekomponenten aus wartungstechnischen Gründen temporär oder dauerhaft außer Betrieb genommen werden.

Die vorliegende Arbeit befasst sich mit der Thematik, wie sich derartige virtuelle Netzwerkservices innerhalb des physischen Netzwerks der Provider einbetten lassen. Im Mittelpunkt steht die Frage, auf welchen Hardwarekomponenten die verschiedenen (virtuellen) Netzwerkfunktionen installiert werden sollen. Aus theoretischer Sicht ist die optimale Berechnung eines solchen Deployments ein $\mathcal{NP}$-hartes Optimierungsproblem. Optimale Algorithmen zur Lösung dieses Problems sind daher nur in sehr kleinen Szenarien anwendbar. Für die effiziente Lösung im Zusammenhang mit Szenarien realer Größenordnung kommen aus diesem Grund nur heuristische Ansätze in Betracht, die für die Bestimmung eines guten, aber nicht zwingenderweise optimalen Deployments entworfen werden. Die vorliegende Arbeit befasst sich mit der effizienten, heuristischen Lösung dieses $\mathcal{NP}$-harten Deployment-Problems.

Es wird zunächst eine Simulationsumgebung beschrieben, die die umfassende Evaluation von Deploymentalgorithmen ermöglicht. Anders als bisherige Simulationstools lässt sich die hier beschriebene Umgebung sehr einfach um neue Funktionen erweitern.

Daran anschließend wird ein verteilter Deployment-Algorithmus vorgestellt, der virtuelle Netze innerhalb von Cloud-Infrastrukturen effizient einbetten kann (DPVNE, *Distributed and Parallel Virtual Network Embedding*). Kernidee hinter diesem Ansatz ist die Aufteilung der physischen Cloud-Infrastruktur in hierarchisch organisierte Netzwerkpartitionen. Dies ermöglicht die parallele Einbettung virtueller Netze. Durch die Verteilung des Berechnungsaufwands auf mehrere Knoten lässt sich das Deployment-Problem auch in Szenarien mit sehr großen Netzwerkinfrastrukturen lösen.

Darüber hinaus wird ein Backtracking-basierter Algorithmus vorgestellt, mit dem das Deployment virtueller Netzwerkservices in NFV-Szenarien durchgeführt werden kann (CoordVNF, *Coordinated deployment of Virtual Network Functions*). In NFV-Szenarien liegt der Fokus auf der Betrachtung der Netzwerkservices, die zur Verarbeitung von Datenströmen innerhalb der Infrastruktur des Mobilfunkproviders installiert werden. Jeder Netzwerkservice besteht dabei aus mehreren (virtuellen) Netzwerkfunktionen, die verschiedene Operationen auf empfangene Daten anwenden und diese dann zur Weiterverarbeitung an andere Netzwerkfunktionen weitergeben. Die genaue Reihenfolge, in der die Datenströme durch die einzelnen Netzwerkfunktionen geroutet werden, ist dabei nicht eindeutig vorgegeben. Anders als in Cloud-Szenarien ist die Struktur der einzubettenden virtuellen Netze also in Teilen flexibel, was zu interessanten neuen, theoretischen Aspekten bzgl. des Deployment-Problems führt. Der CoordVNF-Algorithmus ist als einer der ersten Ansätze in der Lage, solche flexiblen NFV-Netzwerkservices effizient innerhalb der Infrastruktur des Mobilfunkproviders zu platzieren. Im Gegensatz zu bisherigen Verfahren kann CoordVNF auch im Zusammenhang mit größeren Infrastrukturen verwendet werden.

Abschließend wird das Deployment ausfallsicherer Netzwerkservices diskutiert. In diesem Kontext wird beschrieben, wie sich die Robustheit eingebetteter NFV-Services durch Reservierung zusätzlicher Backup-Ressourcen erhöhen lässt. Aufbauend auf CoordVNF wird dann ein Deploymentalgorithmus vorgestellt, der in der Lage ist, Einbettungen gegenüber Ausfällen abzusichern (SVNF, *Survivable deployment of Virtual Network Functions*).

Abstract

Network Function Virtualization (NFV) is being considered as an emerging key technology for future mobile network infrastructures. In classical networks, network functions are tightly bound to specific hardware boxes. In contrast, in NFV networks, (software) functionality is separated from hardware components. Highly specific hardware boxes are being replaced by commodity computing, networking, and storage equipment, offering resources for hosting and running more than just one specific type of network function.
One of the key concepts of NFV is the integration of cloud computing technology into the network core: This enables virtual network functions to be installed and deployed where they are needed; additional resources can be dynamically provided in times of high demand, whereas virtual functions can also be consolidated on a smaller hardware setting if demand decreases. NFV enables operators to manage network functions in a much more flexible way, without having the need of instructing technicians to manually reconfigure hardware equipment on-site – instead, network functions can be deployed and managed remotely. Additionally, for reliability reasons, virtual network functions can be easily migrated to backup resources in case of hardware or software failures.

This thesis discusses the question on how those virtual network functions can be efficiently deployed within the physical network infrastructure. From a theoretical perspective, finding the *optimal* deployment of virtual network services (e.g., in terms of embedding cost) is known as a $\mathcal{NP}$-hard optimization problem. In this context, the thesis introduces heuristic approaches for solving the deployment problem.

To this end, first, an extensible simulation framework is discussed which enables researchers to thoroughly evaluate both existing and novel deployment algorithms.

Second, a distributed algorithm (DPVNE) is presented for embedding virtual networks into a shared physical cloud infrastructure. Here, the main idea is to partition the physical network into several smaller, non-overlapping network regions. Embeddings in those network partitions can then be performed in parallel: Computational efforts can be spread to multiple distributed nodes. This way, solving this $\mathcal{NP}$-hard optimization problem becomes feasible even in large-scale network scenarios where virtual network deployment requests arrive continuously.

Third, a backtracking-based algorithm (CoordVNF) is presented in this thesis for the deployment of virtual network functions in NFV scenarios. In contrast to cloud scenarios, in NFV scenarios, the exact chaining of network functions is not always predefined: In fact, the same network service can be provided by several chainings of network functions. The first embedding algorithm presented here aims to deploy those flexible virtual network services in a cost- and time-efficient way, even in large-scale scenarios. Finally, the thesis discusses the deployment of resilient NFV services; in this context, an extension of the CoordVNF algorithm is presented that allocates additional backup resources for protecting network services from failures.

Inhaltsverzeichnis

Danksagung

Die vorliegende Dissertation ist unter der inspirierenden Betreuung von Frau Prof. Dr. Claudia Linnhoff-Popien entstanden. Ihr möchte ich an allererster Stelle für die Möglichkeit danken, an ihrem Lehrstuhl arbeiten und promovieren zu können. Ganz besonders hervorheben möchte ich ihre begleitende Unterstützung und die stete Motivation, die meine Arbeit am Lehrstuhl sowie die Dissertationsschrift förderten.

Ebenso danke ich Herrn Prof. Dr. Burkhard Stiller, der mir die Gelegenheit gab, mit ihm und seinem Lehrstuhlteam ausführlich über meine Arbeit zu sprechen. Seine Kommentare und Anmerkungen waren bei der Ausarbeitung sehr hilfreich.

Meinen Lehrstuhlkollegen aus München und aus Passau danke ich für das außerordentlich gute, produktive und freundschaftliche Arbeitsklima.

Andreas Fischer danke ich für die zahlreichen fachlichen Diskussionen und für die gemeinsame Arbeit am Alevin-Simulationsframework. Kai Samelin, Lenz Belzner, Juan-Felipe Botero, Josef Plankl und Andreas Stahlbauer danke ich ganz herzlich für die vielen Gespräche und den regen wissenschaftlichen Austausch.

Schließlich möchte ich auch meiner Frau Natalie und meinen Eltern für all die Unterstützung danken.

1 Einführung

Die Infrastruktur von Mobilfunknetzen verändert sich ständig. Anforderungen, die an heutige Netze gestellt werden, sind inzwischen ganz andere als noch vor wenigen Jahren. Standen früher noch Telefonie- und SMS-Dienste im Vordergrund, verzeichnen Mobilfunkbetreiber heute einen dramatischen Zuwachs datenbasierter Anwendungen. Tatsächlich verdoppelt sich das weltweit in Mobilfunknetzen transportierte Datenvolumen jährlich. Es wird davon ausgegangen, dass sich dieser Trend auch in den kommenden Jahren ungebrochen fortsetzt [45].
Um mit dieser Entwicklung Schritt halten zu können, sind die Mobilfunkprovider allein schon aus Wettbewerbsgründen gezwungen, ihre Netze fortlaufend um neue Technologien zu erweitern. Netzbetreiber müssen in die Pflege und Erweiterung ihrer Netzes investieren. Zum einen muss der Ausbau bestehender Technologien vorangetrieben werden: An Stellen, an denen die vorhandenen Hardwarekapazitäten nicht mehr zur Bewältigung des aktuellen Bedarfs ausreichen, müssen zusätzliche Ressourcen installiert werden. Neben der Pflege der aktuellen Technologie hat ein Netzbetreiber zum anderen auch dafür Sorge zu tragen, dass die eigene Infrastruktur mit aktuellen technologischen Entwicklungen mithalten kann. Neue Entwicklungen und Technologien wie LTE-Advanced oder 5G-Bestrebungen sollen möglichst schnell integriert werden. Dabei müssen nicht nur die Randnetze mitsamt der Basisstationen aktualisiert und erweitert werden, sondern insbesondere auch die eigentliche Kernnetzinfrastruktur. Gleichzeitig müssen ältere Technologien weiterhin zur Verfügung stehen, da viele Endgeräte, die über das Mobilfunknetz kommunizieren, nach wie vor auf diese Technologien angewiesen sind.
Die vorhandenen Netzkomponenten müssen also fortlaufend an die sich verändernden Erfordernisse angepasst und rekonfiguriert werden. Das Aufrüsten der Infrastruktur und deren Betrieb ist daher mit enormem finanziellen Aufwand verbunden.

Ein ganz aktueller Trend, an dessen Umsetzung sowohl Mobilfunkprovider als auch Netzwerkausrüster arbeiten, ist die Network Function Virtualization (NFV). Ziel der NFV-Bewegung ist es, die Grundlagen für eine ganz neue Mobilfunk-Infrastruktur zu schaffen, die wesentlich flexibler als die der heutigen Netze ist. Kernidee ist dabei die Entkopplung der logischen Funktionalität von der eigentlichen physischen Hardware durch den Einsatz von Virtualisierungslösungen. Auf dieses Konzept wird im Folgenden näher eingegangen.
Insgesamt ist das NFV-Konzept noch sehr neu. So wurde von der ETSI (European Telecommunications Standards Institute) erst 2016 eine Arbeitsgruppe

zur Spezifizierung von NFV-Softwarekomponenten gegründet [51]. Mehr als 270 Unternehmen aus dem Mobilfunksektor haben sich inzwischen über das NFV-Standardisierungsgremium der ETSI zusammengeschlossen, um an der gemeinsamen Umsetzung der NFV-Idee zu arbeiten [47].

1.1 Network Function Virtualization

In klassischen Mobilfunknetzen kommen hochspezialisierte Hardwarebausteine zum Einsatz, deren Funktionalität fest und untrennbar mit der eigentlichen Hardware gekoppelt ist. Hersteller solch klassischer Netzwerkkomponenten entwickeln also eine Zusammenstellung aufeinander abgestimmter Hardwarebausteine, auf die jeweils maßgeschneiderte, speziell auf diese Hardwarekonfigurationen angepasste Softwarelösungen installiert werden. Vorteil einer solchen Vorgehensweise ist, dass die eingesetzten Systeme „aus einer Hand“ stammen und damit optimal aufeinander abgestimmt sind. In der Vergangenheit führte dies vermehrt dazu, dass immer mehr proprietäre Lösungen entstanden, die nicht mehr vollständig kompatibel zu den Produkten anderer Hersteller sind (dieser Abhängigkeitseffekt wird in der Literatur häufig als *Vendor Lock-In* bezeichnet).
Das Verwobensein von Hardware und Software macht es zudem schwierig und kostspielig, neue Netzwerktechnologien zu integrieren: Soll eine Komponente auf ein neues Netzwerkprotokoll umgestellt werden, müssen entsprechende Aktualisierungen – also neue, hochspezialisierte Hardwarebausteine – eingekauft und integriert werden, was tiefgreifende, langwierige und kostspielige Änderungen der Netzwerkkonfiguration nach sich zieht.

NFV ist ein Konzept, welches darauf abzielt, die Kerninfrastruktur von Mobilfunknetzen flexibler, energieeffizienter und wartbarer zu machen. Die Idee hinter NFV ist, die eigentliche Funktionalität von der Hardware zu entkoppeln und innerhalb von virtuellen Umgebungen – den sogenannten virtuellen Netzwerkfunktionen (VNFs) – zu kapseln (vgl. Abbildung 1.1).
NFV-fähige Netzwerke bestehen aus High-End Standardhardware-Komponenten, die dank hardwaregestützter Virtualisierungsfähigkeiten in der Lage sind, unterschiedliche Netzwerkfunktionen auszuführen. Eine NFV-Hardwarekomponente lässt sich dann nicht mehr wie in klassischen Netzen nur für eine einzige Funktion (wie zum Beispiel für eine Firewall-Funktion) verwenden. Stattdessen kann bei Bedarf eine ganz andere Netzwerkfunktion installiert werden (zum Beispiel eine Monitoring-Funktion oder eine Deep Packet Inspection-Funktion), sodass sich die Funktionalität der Netzkomponenten flexibel austauschen lässt. Ein weiterer Vorteil neben der erhöhten Flexibilität des Netzes ist, dass Standardhardware preislich günstiger ist als hochspezialisierte Komponenten.
Innerhalb der physischen Infrastruktur sind drei unterschiedliche Arten von Standardhardware-Komponenten vorgesehen [39]:

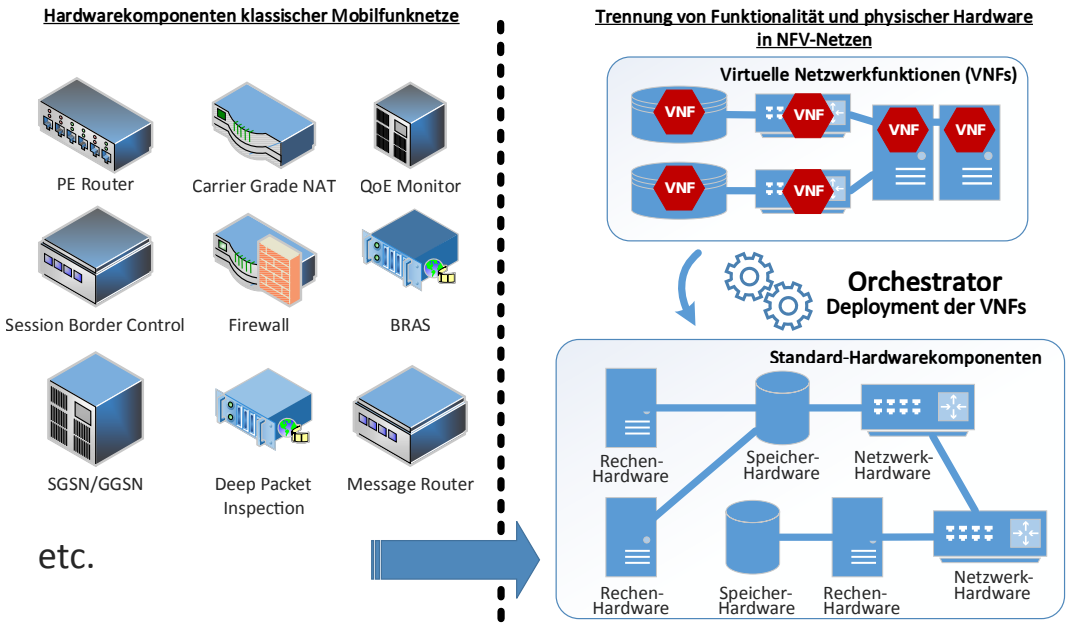


Abb. 1.1: Virtualisierung von Netzwerkfunktionen (basierend auf [39])

I) **Server-Hardware:** Standard-Hardware, die über hohe Rechenleistung verfügt und auf die besonders rechenintensive VNFs installiert werden

II) **Netzwerk-Hardware:** Hardware mit erweiterter Netzwerk-Funktionalität, welche insbesondere für die Ausführung von VNFs, die große Datenmengen weiterleiten, prädestiniert ist

III) **Speicher-Hardware:** Hardwarekomponenten mit erweiterten Speicherkapazitäten; hier können u.a. Datenbank-ähnliche VNFs platziert werden, die vorwiegend Daten abspeichern und verwalten.

Je nach Einsatzzweck sollen Netzwerkfunktionen auf der Kategorie von Hardware installiert werden, die am besten zum jeweiligen Einsatzzweck passt. Beispielsweise sind einige der Funktionen für die Durchführung rechenintensiver Aufgaben zuständig, während andere auf die Filterung oder Aggregation von Datenflüssen spezialisiert sind. Erstere sollten vorzugsweise auf Servern installiert werden, die über hohe CPU-Kapazitäten verfügen, während letztere auf Netzwerkkomponenten betrieben werden sollten. Auf diese Deployment-Frage wird im Folgenden näher eingegangen.

1.2 Deployment virtueller Netzwerkservices

Die Virtualisierung von Netzwerkressourcen führt zu der Frage, auf welchen Standardkomponenten des physischen Netzwerkes die virtuellen Komponenten installiert werden sollen. Jede virtuelle Netzwerkfunktion muss dabei einer

physischen Komponente zugewiesen werden, und für jede virtuelle Kommunikationsverbindung müssen entsprechende Ressourcen des physischen Netzes reserviert werden. Dabei müssen die verschiedenen Hardwareanforderungen der Netzwerkservices berücksichtigt werden. Beispielsweise kann eine virtuelle Netzwerkfunktion bestimmte Mindestanforderungen an die CPU-Kapazitäten der physischen Rechner stellen, denen sie zugewiesen werden kann. Zudem sollen Netzwerkfunktionen, die untereinander Daten austauschen, möglichst so platziert werden, dass sie effizient miteinander kommunizieren können.
Hierfür ist der **Orchestrator** verantwortlich. Er ist das Herzstück der NFV-Infrastruktur, siehe Abbildung 1.1: Seine Aufgabe ist es, Netzwerkservices effizient innerhalb der Netzinfrastruktur zu platzieren. Beim Deployment muss der Orchestrator die unterschiedlichen Anforderungen der Netzwerkservices berücksichtigen, damit sichergestellt ist, dass alle eingebetteten Services wie vorgesehen funktionieren. Genau hier liegt auch der Fokus der vorliegenden Arbeit: Es werden Algorithmen vorgestellt, die vom Orchestrator verwendet werden, um effiziente Deployment-Entscheidungen zu treffen.

1.3 NFV, Cloud Computing und Software Defined Networking – Eine Einordnung

Die Idee, Netzwerke flexibler, kostengünstiger und effizienter zu machen, ist an sich nicht neu und wird von der Industrie seit jeher auf verschiedene Art und Weise verfolgt. Zwei Technologien, die mit der Virtualisierung von Netzwerkfunktionen eng verwandt sind, sind Cloud Computing und Software Defined Networking (SDN) [87]:

Cloud Computing: Cloud Computing ermöglicht den parallelen Betrieb virtueller Maschinen (VMs) auf einer gemeinsam verwendeten physischen Infrastruktur.
Benötigte Cloud-Hardwareressourcen lassen sich mit minimalem Aufwand reservieren und auch während des Betriebs hinzufügen, ohne dass dazu eine Rekonfiguration der physischen Infrastruktur notwendig wird. Diese Elastizität des Cloud Computings wird durch die Isolation der Softwarekomponenten innerhalb virtueller Maschinen oder Software-Containern erreicht. Das klassische Verständnis vom Serviceanbieter, der die von ihm angebotenen Dienste innerhalb seines eigenen Rechenzentrums installiert und verwaltet, wird beim Cloud Computing aufgehoben durch eine Teilung dieser Zuständigkeiten in zwei voneinander getrennte Rollen: Die des Infrastrukturproviders (ISP) und die der Serviceprovider. Der ISP stellt Hardwarekapazitäten, möglicherweise gegen eine stundenbasierte Vergütung, zur Verfügung und kümmert sich um den Betrieb und die Instandhaltung der Hardware. Darüber hinaus stellt der ISP Schnittstellen zur Verfügung, über die sich Hardwareressourcen anmieten lassen. Angemietet werden Ressourcen durch die Serviceprovider, die VMs

bzw. Software-Container konfigurieren und den ISP über die bereit gestellten Schnittstellen mit dem eigentlichen Deployment beauftragen.
Beim Public Cloud Computing vermietet ein ISP die Rechen- und Speicherressourcen seines Netzwerkes an externe Kunden, die erforderliche Hardwareressourcen für einen bestimmten Zeitraum anmieten. Beispiele für solche Cloud-Systeme sind Amazon EC2 [2], Microsoft Azure [6] und die Fujitsu Public Cloud [3]. Unter dem Begriff Private Cloud Computing versteht man den Einsatz von Cloud-Techniken innerhalb der unternehmenseigenen Infrastruktur. Beim Hybrid Cloud Computing werden beide Ansätze miteinander kombiniert; die Private Cloud wird also um zusätzlich gemietete Public Cloud-Ressourcen erweitert [83].
Durch die Kapselung der Services innerhalb von VMs oder Software-Containern lassen sich mehrere Services isoliert voneinander auf derselben physischen Hardware betreiben. Cloud Computing ermöglicht das schnelle, elastische Skalieren von Diensten: zusätzliche Ressourcen können für einen Dienst innerhalb von wenigen Sekunden zugebucht und – umgekehrt – im Anschluss wieder freigegeben werden. Darüber hinaus lassen sich mehrere Dienste auf derselben physischen Hardware konsolidieren, was ein effizienteres Deployment der zu betreibenden Dienste ermöglicht. Ferner können Hardwareausfälle durch temporäre Migration von Diensten überbrückt werden, sodass Reparaturmaßnahmen schnell und möglichst transparent für den Serviceprovider durchgeführt werden können.
Die grundlegende Idee hinter NFV ist es, die eigentliche Funktionalität der Netzwerkkomponenten von der Hardware zu entkoppeln und die Funktionen in virtuellen Maschinen zu kapseln, die auf Standard-Hardwarekomponenten ausführbar sind. Das Deployment dieser virtuellen Netzwerkfunktionen kann dann anhand von Cloud-Technologien erfolgen; die Virtualisierung ermöglicht die flexible Integration bzw. Rekonfiguration von Netzwerkfunktionen innerhalb der physischen Infrastruktur. Die VNFs werden dabei in erster Linie innerhalb einer Private Cloud-Umgebung installiert. Grundsätzlich kann auch ein Hybrid Cloud- oder gar ein Public Cloud-Ansatz gewählt werden; in diesem Fall müssen Service Level Agreements (SLAs) zwischen dem Service- und dem Infrastrukturprovider vereinbart werden, in denen Absprachen über die Dienstgüte getroffen und vertraglich geregelt werden [50].

Software Defined Networking: Idee hinter dem SDN-Konzept ist die Abstraktion der Kontrollfunktionalität von Netzwerkelementen, also der Logik, die dafür zuständig ist, zu entscheiden, wohin Daten weitergereicht werden sollen, von der eigentlichen Datenweiterleitungsfunktionalität. Die Trennung dieser beiden Funktionen ermöglicht die einfache, programmierbare und zentrale Konfiguration des gesamten Netzwerkes, ohne dass einzelne Geräte manuell rekonfiguriert werden müssen. Die Kontrollfunktionalität einzelner Netzwerkgeräte kann gebündelt und softwaregesteuert von einer oder von mehreren zentralen Stellen durchgeführt werden, wodurch sich

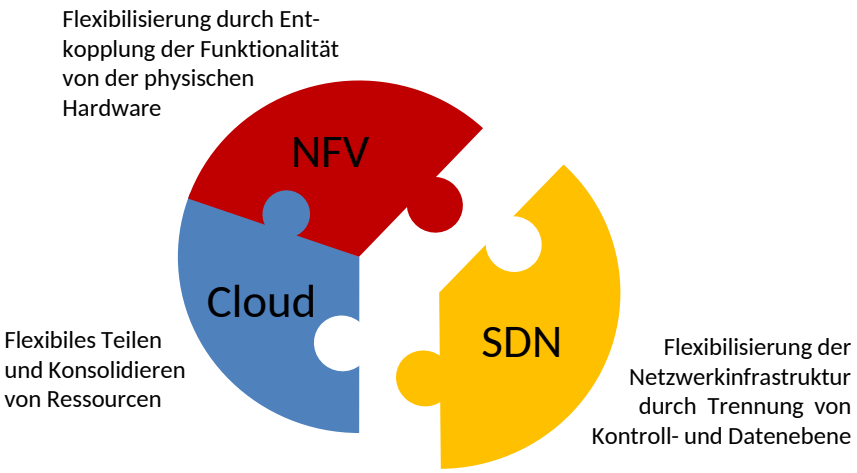


Abb. 1.2: NFV, Cloud Computing und SDN

das Netzwerkmanagement vereinfacht und die Rekonfigurierbarkeit der Infrastruktur flexibler wird. Dies ist insbesondere im Zusammenhang mit dem Betrieb virtueller Services in Cloud Computing oder NFV-Szenarien vorteilhaft, da eine schnelle, softwaregestützte Rekonfigurationsmöglichkeit der Netzwerkstrukturen gerade in diesen Bereichen wichtige Voraussetzung für die Flexibilität der Dienste ist. Aus diesem Grund wird SDN als Technologie gesehen, die eine zentrale Rolle bei der Umsetzung flexibler NFV-basierter Mobilfunknetze einnimmt.

Abbildung 1.2 veranschaulicht den Zusammenhang zwischen NFV, Cloud Computing und SDN. Ein Kernkonzept von NFV ist die Integration von Cloud Computing-Technologien in die Mobilfunkinfrastruktur; aus diesem Grund sind die beiden Konzepte eng miteinander verwoben. SDN ist ein technisch unabhängiges, aber als komplementär zu NFV zu betrachtendes Konzept, das ein flexibles und skalierbares Management der Netzwerkkonfiguration möglich macht [40].

Die NFV-Technik ermöglicht also die flexible Platzierung virtueller Netzwerkfunktionen, während SDN eine schnelle Rekonfiguration des Netzwerks erlaubt. Die Routen der Datenströme, die zwischen den NFV-Netzwerkfunktionen ausgetauscht werden, lassen sich mit Hilfe der SDN-Technologie sehr flexibel festlegen. Die in dieser Arbeit vorgestellten Deployment-Algorithmen betrachten dabei beide Aspekte: Die Reservierung von Hardwareressourcen für den Betrieb der (NFV-)Netzwerkfunktionen sowie die Reservierung geeigneter (SDN-) Kommunikationspfade.

1.4 Fokus und Aufbau der vorliegenden Arbeit

Der Fokus dieser Arbeit liegt auf der Entwicklung effizienter, skalierbarer Algorithmen zum Deployment virtueller Netze in Cloud- und NFV-Szenarien. Die Begriffe *Deployment*, *Platzierung* und *Einbettung* werden im Folgenden synonym verwendet.

Kapitel 2 beschreibt, inwiefern es sich bei der Platzierung solcher Netze um ein $\mathcal{NP}$-hartes Optimierungsproblem handelt und welche Platzierungsstrategien in diesem Zusammenhang verfolgt werden können.

In Kapitel 3 wird eine Simulationsumgebung vorgestellt, die es ermöglicht, Einbettungsalgorithmen umfassend zu evaluieren und miteinander zu vergleichen. In Abgrenzung zu anderen Simulationstools ist die hier vorgestellte Umgebung nicht nur wesentlich umfangreicher, sondern auch deutlich flexibler, da sie sehr einfach um neue Funktionen erweitert werden kann.

In Kapitel 4 wird zunächst auf das aus dem Cloud-Kontext bekannte Virtual Network Embedding-Optimierungsproblem eingegangen, bei dem statische virtuelle Netzwerkstrukturen effizient in ein physisches Netzwerk eingebettet werden sollen. Es wird untersucht, wie sich dieses komplexe Deployment-Problem auch im Zusammenhang mit sehr großen Netzwerkinfrastrukturen effizient lösen lässt. Dazu wird ein verteilter Einbettungsalgorithmus vorgestellt, der den Berechnungsaufwand, der mit dem Deployment virtueller Netze verbunden ist, auf gleich mehrere Rechner verteilen kann. Gerade in Szenarien, in denen kontinuierlich viele Einbettungen vorgenommen werden müssen, führt ein solch verteiltes Vorgehen zu einer deutlich schnelleren Platzierung der virtuellen Netze, als dies mit bisherigen Verfahren möglich ist. Die Infrastruktur des physischen Netzes wird dazu in hierarchisch organisierte Partitionen eingeteilt. Die Einbettungsberechnungen werden dann parallel innerhalb dieser Partitionshierarchie von mehreren verteilt arbeitenden Agenten durchgeführt.

Kapitel 5 befasst sich mit dem Deployment von NFV-Netzwerkservices. Die Bestimmung eines optimalen Deployments solcher Services ist ein verwandtes, ebenfalls $\mathcal{NP}$-hartes Optimierungsproblem. Ein wesentlicher Unterschied zum Virtual Network Embedding-Problem ist, dass die Struktur des einzubettenden virtuellen Netzes bei der Spezifikation eines Netzwerkservices nicht von vornherein fest und unveränderlich vorgeschrieben ist. Die Anordnung bzw. Verkettung der Netzwerkfunktionen eines NFV-Netzwerkservices lässt sich also in einem gewissen Rahmen und unter Berücksichtigung der spezifizierten Vorgaben und Abhängigkeiten flexibel wählen. Die NFV-Netzwerkfunktionen müssen also nicht nur, wie beim Virtual Network Embedding, in geeigneter Weise innerhalb der physischen Netzwerkinfrastruktur eingebettet werden,

sondern darüber hinaus zu validen Netzwerkservices verkettet werden. Allein der Einbettungsschritt stellt für sich genommen schon ein $\mathcal{NP}$-hartes Optimierungsproblem dar. Diese Verkettung muss entsprechend der Vorgaben des Serviceproviders erfolgen. In Kapitel 5 wird ein Backtracking-basierter Algorithmus vorgestellt, der diese beiden Teilprobleme (die Verkettung und die Einbettung) in einem gemeinsamen, koordinierten Schritt löst. Dieser heuristische Ansatz ist in der Lage, gute Einbettungsergebnisse bei akzeptablem Laufzeitverhalten zu liefern und skaliert auch in großen Netzwerken. Der hier vorgestellte Ansatz ist wesentlich schneller als bisherige Algorithmen.

In Kapitel 6 wird diskutiert, wie sich NFV-Netzwerkservices gegen Hardwareausfälle absichern lassen. Dazu wird ein Verfahren vorgestellt, welches zusätzliche Backupressourcen reserviert, auf die nach Ausfall einzelner Hardwarekomponenten als Fallback zurückgegriffen werden kann. Auf diese Weise kann der Betrieb der Netzwerkservices im Fehlerfall innerhalb kürzester Zeit wieder aufgenommen werden. In der Literatur finden sich bislang keine vergleichbaren Ansätze, die das Deployment von ausfallsicheren NFV-Netzwerkservices diskutieren.

1.5 Vorarbeiten

Teile dieser Arbeit wurden bereits auf mehreren internationalen Konferenzen, in Form von Buchbeiträgen sowie in Journalen veröffentlicht. Üblicherweise entstehen solche Publikationen im Austausch mit Projektpartnern, mit Kollegen sowie mit anderen, externen Wissenschaftlern. Im Folgenden wird im Einzelnen aufgeschlüsselt, was der Beitrag des Autors der vorliegenden Arbeit (im Folgenden nur noch als „Autor“ bezeichnet) an diesen relevanten Vorpublikationen ist:

Kapitel 2 (Hintergrund: Deployment virtueller Netzwerkstrukturen): Dieses Hintergrund-Kapitel enthält eine kurze Abhandlung zweier Beiträge, die bereits vorveröffentlicht wurden: Die Klassifizierung von Mobile Edge Computing-Anwendungen sowie die Alevin-Simulationsumgebung.
Die in Abschnitt 2.1 beschriebene Klassifizierung von Mobile Edge Computing-Anwendungen baut auf Ergebnissen auf, die in Zusammenarbeit mit Kollegen und Projektpartnern von Nokia Networks diskutiert und vom Autor kategorisiert und in [25] veröffentlicht wurden. Diese Kategorisierung ist ebenfalls Teil der vorliegenden Arbeit. Martin Werner und Sebastian Feld waren insbesondere bei der Bewertung der einzelnen Use Cases bezüglich verschiedener Metriken beteiligt, was nicht Teil der vorliegenden Arbeit ist. Thomas Schimper gab Feedback und trug zur Finalisierung des Papers bei.

Kapitel 3 (Simulationsbasierte Evaluation von Einbettungsalgorithmen): In Kapitel 3 wird das Alevin-Simulationsframework vorgestellt, mit dem

Deployment-Algorithmen evaluiert werden können. Der Funktionsumfang des Frameworks ist sehr umfangreich und wurde vom Autor in Kooperation mit internationalen Wissenschaftlern nach und nach ausgebaut und um zusätzliche Implementierungen von Deployment-Algorithmen erweitert. Die Entwicklung der Alevin-Simulationsumgebung in der Version 2.0 wurde federführend von Michael Beck in Zusammenarbeit mit Andreas Fischer und Fabian Kokot vorangetrieben und die während dieses Prozesses zusammen mit Prof. Claudia Linnhoff-Popien und Prof. Hermann De Meer gewonnenen Erkenntnisse in [23] publiziert. Insbesondere wurden hier die Features sowie die Architektur des Alevin-Simulationsframeworks beschrieben. Ferner wurden Messergebnisse, die während der Durchführung einzelner Simulationen aufgezeichnet wurden, ausgewertet. Die Architektur der Alevin-Umgebung wurde in Zusammenarbeit mit Fabian Kokot und Andreas Fischer visualisiert – eine überarbeitete Version der in [23] publizierten Grafik ist in Abbildung 3.2 gezeigt. Andreas Fischer war zudem am „Lessons Learned"-Abschnitt des Papers beteiligt, der nicht in die vorliegende Arbeit einfloss. Prof. Claudia Linnhoff-Popien und Prof. Hermann De Meer steuerten hilfreiche Anregungen und Feedback bei. Der Evaluationsabschnitt des Papers, aus dem Teile in Kapitel 3 einflossen, ist vollständig dem Autor zuzuschreiben, der die Experimente umsetzte und die Ergebnisse auswertete.
Im Rahmen der vorliegenden Arbeit wurde insbesondere an der Umsetzung einer erweiterten Simulations-Logik für Alevin gearbeitet, mit der sich Deployment-Algorithmen automatisiert in vielen unterschiedlichen Szenarien evaluieren lassen. Insbesondere wurde dabei die Konzeption und Implementation eines Online-Simulationsmoduls für Alevin vorangetrieben, die in vollem Umfang dem Autor zuzurechnen ist, vgl. Abschnitt 3.3.2. Darüber hinaus wurde Alevin vom Autor um völlig neue Funktionen erweitert, mit denen es möglich ist, NFV-Deploymentalgorithmen zu evaluieren; ursprünglich war Alevin lediglich als Simulationstool für VNE-Algorithmen vorgesehen.

Kapitel 4 (Verteiltes und skalierbares Einbetten virtueller Netze): Kapitel 4 basiert auf den in [16, 21] publizierten Inhalten. Der hier beschriebene Ansatz wurde vom Autor konzipiert, implementiert und evaluiert. Sowohl das Konzept der Partitionierung der physischen Infrastruktur, das Kommunikationsprotokoll, die Funktionsweise der verteilten Einbettungsknoten bei der Berechnung von Deployments, als auch die Formulierung der Einbettungsheuristik wurden vom Autor erdacht und umgesetzt. Andreas Fischer und Juan-Felipe Botero beteiligten sich im Rahmen zahlreicher Telefonkonferenzen und Diskussionen an der Strukturierung der Publikation; ferner halfen sie bei der Diskussion der Evaluationsergebnisse sowie bei der Recherche nach verwandten Arbeiten. Prof. Claudia Linnhoff-Popien und Prof. Hermann De Meer sei für die hilfreichen Kommentare und das umfassende Feedback gedankt.
Abbildung 4.16 zeigt das Ergebnis einer Messung, die im Rahmen der Bachelorarbeit von Stephan Holzner durchgeführt wurde [63]. Dies ist an entsprechender

Stelle noch einmal explizit vermerkt.

Kapitel 5 (Koordinierte Verkettung und Einbettung virtueller Netzwerkfunktionen): Kapitel 5 beruht auf einigen in [14] und [15] publizierten Ideen. Diese Publikationen entstanden in Zusammenarbeit mit Juan-Felipe Botero. Der hier beschriebene Ansatz wurde vom Autor konzipiert, implementiert und evaluiert: Die Idee hinter dem koordinierten Vorgehen des CoordVNF-Algorithmus, die konkrete Formulierung des Backtracking-basierten CoordVNF-Algorithmus, die Implementation und die Durchführung der Evaluation sowie die Beschreibung der Evaluationsergebnisse sind Michael Beck zuzuordnen. Bezüglich der beiden Publikationen beteiligte sich Juan-Felipe Botero bei der Recherche nach verwandten Ansätzen sowie bei der Diskussion relevanter Evaluationsmetriken. Zudem trug er zur Überarbeitung des englischsprachigen Textes bei und gab inhaltliches Feedback.

Kapitel 6 (Ausfallsichere Einbettung virtueller Netzwerkservices): Kapitel 6 beschreibt einige Ideen, die in [17] publiziert wurden. Die Konzeption und Implementation des SVNF-Algorithmus mitsamt der unterschiedlichen Absicherungsstrategien, mit denen virtuelle Netzwerkservices gegen Ausfälle abgesichert werden können, wurde vom Autor vorgenommen. Die Evaluation sowie die Beschreibung der Evaluationsergebnisse sind ebenfalls dem Autor zuzurechnen. Für die Publikation übernahm Juan-Felipe Botero die Recherche nach verwandten Ansätzen. Zudem beteiligte er sich, aufbauend auf [14], an der Beschreibung des zugrunde liegenden, generellen NFV-RA-Deployment-Problems und trug zur sprachlichen Überarbeitung der Publikation bei.

2 Hintergrund: Deployment virtueller Netzwerkstrukturen

Dieses Kapitel beschreibt zunächst, wie sich virtuelle *Netzwerkfunktionen* durch Einsatz der Mobile Edge Computing-Technologie am Rand des Mobilfunknetzes und damit in nächster Nähe zum Endanwender nutzen lassen.
Anschließend wird das Deployment virtueller *Netzwerkservices* innerhalb der physischen Netzwerkinfrastruktur beschrieben. Ein Netzwerkservice besteht aus gleich mehreren Netzwerkfunktionen, die untereinander Daten austauschen. Im Zusammenhang mit dem Deployment solcher Services wird zunächst auf das Virtual Network Embedding (VNE)-Problem eingegangen, welches für die vorliegende Arbeit von ganz zentraler Bedeutung ist. In VNE-Szenarien sollen mehrere miteinander vernetzte virtuelle Maschinen in eine physische Cloud-Infrastruktur eingebettet werden. Dabei wird davon ausgegangen, dass ein solches virtuelles Netz durch eine statische, fest vorgegebene Netzstruktur definiert ist.
Darauf aufbauend wird das Deployment von NFV-Netzwerkservices beschrieben. In NFV-Netzen werden Datenströme, die innerhalb des Mobilfunknetzes verarbeitet werden müssen, durch virtuelle Netzwerkfunktionen geroutet. Jede dieser Funktionen wendet eine oder mehrere Verarbeitungsoperationen auf die an sie gerouteten Datenströme an und leitet das Ergebnis an weitere Netzwerkfunktionen weiter, die ebenfalls Teil des Netzwerkservices sind. Im Gegensatz zu den in VNE-Szenarien betrachteten virtuellen Maschinen können virtuelle Netzwerkfunktionen mit einem gewissen Flexibilitätsgrad innerhalb der einzubettenden virtuellen Infrastruktur angeordnet werden. Das Deployment solcher flexibler virtueller Netzwerkservices lässt sich auf das *Virtual Network Function Resource Allocation*-Problem zurückführen. In beiden Fällen handelt es sich um $\mathcal{NP}$-harte Optimierungsprobleme.
Neben der Beschreibung dieser beiden Optimierungsprobleme wird im vorliegenden Kapitel die Frage erörtert, was im Allgemeinen ein „gutes“ Deployment von virtuellen Netzen und Netzwerkservices ausmacht und anhand welcher Metriken sich solche Einbettungen numerisch bewerten lassen.

2.1 Virtuelle Netzwerkfunktionen am Rand des Mobilfunknetzes

Bei Mobile Edge Computing (MEC) handelt es sich um eine neuartige Mobilfunk-Technologie, bei der in unmittelbarer Nähe zu den Basisstationen

Server mit zusätzlichen CPU- und Speicherkapazitäten installiert werden. Solche Recheneinheiten sind in örtlicher Nähe zu einzelnen Basisstationen aufgestellt und mit diesen unmittelbar oder über sehr kurze Kommunikationsverbindungen verbunden, vgl. Abbildung 2.1. Die Idee, derartige lokale Cloud-Einheiten am Rand des Mobilfunknetzes zu installieren, wird derzeit aktiv von Seiten mehrerer im Mobilfunksegment vertretenen Unternehmen propagiert [69, 67, 77, 8, 66] und über das European Telecommunications Standards Institute (ETSI) standardisiert [46]. Im Folgenden werden diese Cloud-Einheiten als Mobile Edge Computing Server bezeichnet, kurz MEC-Server.
MEC-Server eignen sich aufgrund ihrer örtlichen Nähe zu den mobilen Endgeräten für Netzwerkservices mit besonders hohen Latenzanforderungen. Solche latenzsensitiven Dienste profitieren insbesondere dann vom Einsatz der MEC-Technologie, wenn, wie in vielen ländlichen Regionen, die Anbindung der Basisstation an die Kerninfrastruktur für den entsprechenden Service nur unzureichend ausgelegt ist und sich in Folge dessen Verzögerungen bei der Übertragung bemerkbar machen.
Dieser Abschnitt beschreibt eine Taxonomie für solche MEC-gestützten, virtuellen Netzwerkservices. Im Folgenden werden zunächst unterschiedliche Integrationsmöglichkeiten für die Installation von MEC-Server innerhalb der physischen Infrastruktur des Mobilfunknetzes diskutiert. Im Anschluss werden verschiedene Einsatzmöglichkeiten analysiert, klassifiziert und bewertet.

2.1.1 Physisches Deployment von Mobile Edge Computing-Servern

Kernidee des MEC-Konzeptes ist es, Netzwerkdienste und -funktionen in unmittelbarer Nähe zum Mobilfunknutzer zu platzieren. Aus diesem Grund werden MEC-Server in örtlicher Nähe zur Basisstation installiert – etwa direkt am Mobilfunkmasten oder im ersten Aggregationspunkt des Access-Netzes (vgl. Abbildung 2.1). Das ETSI-Standardisierungsgremium sieht dabei mehrere Möglichkeiten für die technische Integration von MEC-Servern in die Mobilfunkinfrastruktur vor [49]. Auf welche dieser Deployment-Optionen der Mobilfunkprovider zurückgreift, kann von verschiedenen, im Folgenden näher diskutieren Faktoren abhängig gemacht werden:

I) *Co-Hosting der MEC-Server direkt an der LTE/4G-Basisstation*
Der Server ist Teil des LTE-Netzes und wird als solcher direkt an die Mobilfunkstation angeschlossen. In der LTE-Terminologie wird die Mobilfunkstation eNodeB genannt. Dieses eNodeB-nahe Deployment hat den Vorteil, dass sich Dienste und Daten in unmittelbarer Nähe zum Endnutzer bereitstellen lassen und damit sehr geringe Latenzzeiten erreicht werden können. Ein Nachteil ist, dass auf einen so installierten MEC-Server nicht von anderen Basisstationen zugegriffen werden kann und es aufgrund der technischen Integration des Servers in das LTE-Netz auch

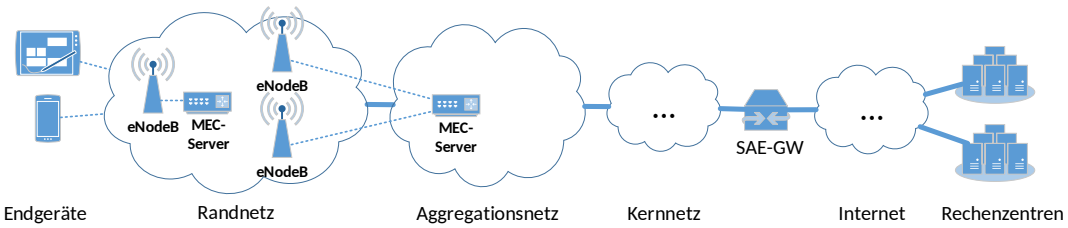


Abb. 2.1: Möglichkeiten der Integration von Mobile Edge Computing Einheiten direkt an den Basisstationen oder innerhalb des Aggregationsnetzes

nicht möglich ist, über andere Mobilfunktechnologien (wie aus den älteren 2G- oder 3G-Netzen) auf die MEC-Services zuzugreifen.

II) *Integration der MEC-Server innerhalb des 3G-Radio Network Controllers*
Dieses Deployment hat den Vorteil, dass gleich mehrere Basisstationen auf die vom MEC-Server angebotenen Dienste zugreifen können. Denn in 3G-Mobilfunknetz sind in der Regel mehrere Basisstationen an einen solchen Radio Network Controller angebunden. Allerdings ist ein Zugriff über andere Mobilfunktechnologien wie LTE/4G nicht möglich.

III) *Integration der MEC-Server innerhalb des Aggregationsnetzes*
Diese Deployment-Option erlaubt den gemeinsamen Zugriff auf einen MEC-Server von mehreren Basisstationen aus. Dadurch, dass der Server nicht direkt mit der eNodeB-Einheit des LTE-Netzes verbunden ist, lässt sich der Server zudem netzübergreifend verwenden: Mobilfunknutzer, die nicht über LTE/4G, sondern über eine andere Funktechnologie verbunden sind, können dann ebenfalls auf die Services des MEC-Servers zurückgreifen. Ein Nachteil dieser Deployment-Option ist die etwas höhere Latenz zwischen mobilem Endgerät und den MEC-Services.

Die von den MEC-Servern auszuführenden Services werden entweder vom Mobilfunkprovider selbst oder von externen Serviceanbietern installiert und gewartet. Bei Mobilfunknetzen handelt es sich um sicherheitskritische Infrastrukturen. Daher werden die Dienste externer Anbieter innerhalb virtueller Maschinen ausgeführt; auf diese Weise kann der Infrastrukturprovider sicherstellen, dass die Dienste sowohl untereinander als auch von der kritischen Mobilfunkkerninfrastruktur isoliert sind. Die virtuellen Maschinen werden dazu in einer kontrollierten Umgebung ausgeführt, in der entsprechende Sicherheitsmaßnahmen umgesetzt werden können.

2.1.2 Klassifizierung von Anwendungen und Use-Cases

MEC-Anwendungen können in die folgenden, in Abbildung 2.2 gezeigten Kategorien eingeteilt werden. Teile dieser Kategorisierung wurden in [25, 19]

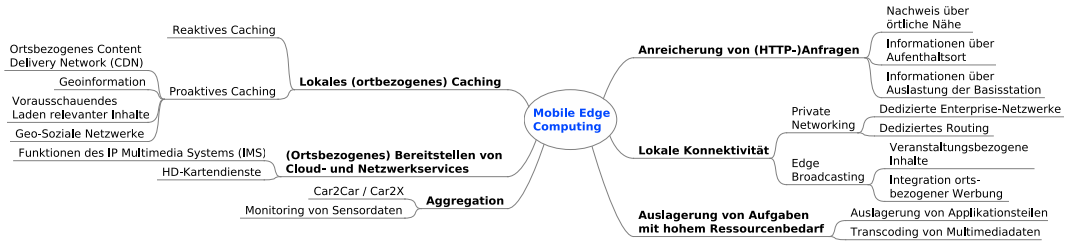


Abb. 2.2: Klassifizierung von Mobile Edge Computing-Anwendungen

publiziert:

Lokales (ortsbezogenes) Caching von Daten: Eine der vielleicht naheliegendsten Einsatzmöglichkeiten der MEC-Technologie ist die lokale Bereitstellung von Daten, die möglichst verzögerungsfrei beim Endnutzer ankommen sollen. Mit MEC lassen sich Inhalte noch viel näher am Endnutzer platzieren, als dies über heutige Content Delivery Networks (CDNs) möglich wäre.
Das MEC-gestützte Caching kann sowohl reaktiv als auch proaktiv durchgeführt werden. Beim *reaktiven Caching* werden populäre Inhalte auf den MEC-Servern vorgehalten, wenn sich herausstellt, dass gewisse Inhalte verstärkt innerhalb des vom MEC-Server abgedeckten Bereichs angefordert werden.
Im Unterschied dazu werden die vorgelagerten Inhalte beim *proaktiven Caching* bereits zu einem früheren Zeitpunkt auf den MEC-Servern abgelegt. Das Caching kann in diesem Fall aktiv von Seiten der Dienstanbieter angestoßen werden. Ein Anwendungsszenario ist hierbei der Roll-Out von Softwareaktualisierungen, die bereits dann auf die örtlichen MEC-Server verteilt werden, bevor sie im großen Stil von den Endgeräten heruntergeladen werden.
Ein Vorteil des MEC-gestützten Cachings ist, dass ein möglicher Ortsbezug der vorzuhaltenden Daten berücksichtigt werden kann. Geo-soziale Netzwerke (GSNs) wie Google Latitude oder Yelp, aber auch Museen oder Veranstalter können HD-Inhalte an genau den Orten bereitstellen, an denen sie Verwendung finden.
(Ortsbezogene) Bereitstellung von Cloud- und Netzwerkservices: Ähnlich wie bei der lokalen Bereitstellung von Daten können auch bestimmte Netzwerkservices auf MEC-Servern installiert werden. Dies können reguläre Cloud- oder spezifische Netzwerkdienste wie beispielsweise Funktionen des IP Multimedia Subsystems (IMS) sein, deren Funktionalität aus dem Kernnetz entweder ganz oder teilweise auf die MEC-Server übertragen wird. Ein aktuelles Thema in diesem Kontext ist das Vorhalten von HD-Kartenmaterial, welches für viele verkehrssicherheitsrelevante Anwendungen stets aktuell und quasi in Echtzeit vorliegen muss [60]. Insbesondere werden solche Echtzeitdaten als wesentliche Voraussetzung für den Einsatz autonomer Fahrzeuge gesehen. Diese Dienste müssen dann nicht mehr von entfernten Rechenzentren angefordert und bezogen werden.
Aggregation: Daten, die von unterschiedlichen Teilnehmern aufgezeichnet

und an einen zentralen Dienst übermittelt werden, können bereits vor der Übertragung aggregiert und vollständig oder zum Teil ausgewertet werden. Dies kann insbesondere im Car2Car/Car2X-Umfeld sinnvoll sein, in dem viele zeitkritische Sensordaten berücksichtigt werden müssen.
Anreicherung von HTTP-Anfragen: Die HTTP-Anfragen, die von mobilen Endgeräten zum Abruf entfernter Inhalte versendet werden, können vom MEC-Server um Zusatzinformationen angereichert werden. Dies kann insbesondere zur Verbesserung der Quality of Experience geschehen: Hierzu können detaillierte Informationen über die aktuelle Auslastung des Downlinks der Basisstation (also der Funk-Verbindung von der Basisstation zum mobilen Endgerät) mitgeliefert werden, sodass der entfernte Clouddienst bereits vor Beginn der Auslieferung der angeforderten Daten in der Lage ist, die zu übertragenden Inhalte entsprechend der verfügbaren Übertragungskapazitäten zu skalieren.
Lokale Konnektivität: Endgeräte können ohne Umweg über das Kernnetz direkt miteinander kommunizieren. Die Kommunikation erfolgt dann lediglich über das Aggregationsnetz oder, je nach Deployment des örtlichen MEC-Servers, sogar ausschließlich über den MEC-Server. Diese Möglichkeit kann insbesondere für Unternehmen interessant sein, die ein lokales privates Netzwerk innerhalb des Unternehmenscampus aufspannen.
Auslagerung von Aufgaben mit hohem Ressourcenbedarf: Virtual Reality-Anwendungen stellen hohe Anforderungen an die Rechenkapazitäten der Endgeräte. Hier bietet sich die Auslagerung der ressourcenintensiven Berechnungen an externe Clouddienste an. Da es sich hierbei jedoch um zeitkritische Berechnungen handelt, können Virtual Reality-Anwendungen von der Verfügbarkeit lokaler MEC-Rechenressourcen profitieren. Zudem können zeitkritische Aufgaben, die zu einem hohen Energieverbrauch auf dem Endgerät führen, ausgelagert werden [41].

Eine ausführliche Diskussion dieser sowie weiterer Einsatzszenarien und Use-Cases wurde in Zusammenarbeit mit Nokia Networks in [25], [20] und [19] publiziert. Die Umsetzung eines konkreten MEC-Services wurde darüber hinaus in [18] beschrieben.

2.2 Deployment virtueller Netze: Das VNE-Problem

Der vorherige Abschnitt befasste sich mit dem Einsatz virtueller Maschinen bzw. virtueller Netzwerkfunktionen im Aggregationsnetz der Mobilfunkinfrastruktur. Dabei lag der Fokus auf der isolierten Betrachtung hochperformanter Dienste, die auf MEC-Servern installiert werden können.

Die nachfolgenden Abschnitte befassen sich nun mit der eigentlichen Frage des Deployments *vernetzter* virtueller Netzwerkfunktionen, die nicht nur auf

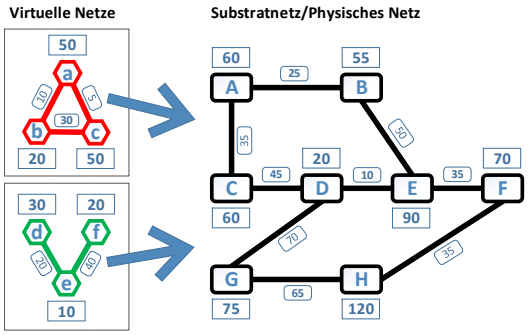
MEC-Servern, sondern auch innerhalb der Mobilfunk-Kerninfrastruktur betrieben werden können. Virtuelle Netzwerkfunktionen werden im Folgenden nicht mehr isoliert voneinander betrachtet, sondern im Zusammenspiel mit anderen Netzwerkfunktionen des Mobilfunknetzes.

Beim Deployment virtualisierter Netzwerkservices muss auf die infrastrukturellen Gegebenheiten und die Anforderungen der zu installierenden Dienste Rücksicht genommen werden: Zum einen müssen dazu die Ressourcenanforderungen der einzelnen Netzwerkfunktionen und deren Hardwarekompatibilität berücksichtigt werden. Zum anderen sollten die verschiedenen Funktionen eines Netzwerkservices so platziert werden, dass insgesamt ein effizienter Betrieb der Konfiguration möglich ist.

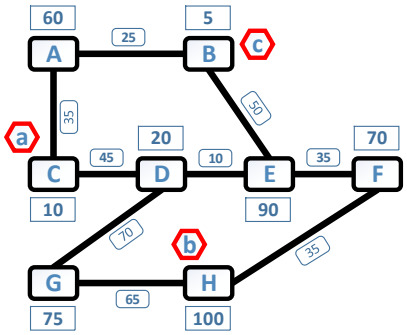
Die Netzwerkservices sollen also nicht „irgendwie“ innerhalb der physischen Netzinfrastruktur installiert werden – wobei selbst ein derart „chaotisches“ Deployment nicht trivial ist, da sowohl die Abhängigkeiten der einzelnen Netzwerkfunktionen untereinander als auch deren Ressourcenanforderungen bei der Platzierung zu berücksichtigen sind. Der Entwurf eines validen *und* effizienten Deployments ist eine sehr komplexe Aufgabe, die eine sorgsame und wohldurchdachte Planung erforderlich macht. Im Folgenden wird dieses Deployment-Problem, mit dessen Lösung sich die vorliegende Arbeit befasst, aus algorithmischer Sicht betrachtet.

Ein insbesondere im Zusammenhang mit Cloud-Infrastrukturen relevantes Optimierungsproblem ist das sogenannte Virtual Network Embedding-Problem (kurz VNE-Problem). Beim VNE-Problem sollen mehrere virtuelle Netze in ein gemeinsames physisches Netz eingebettet werden [38, 11]. Das physische Netz besteht aus Rechnern, die über begrenzte Hardwareressourcen verfügen, z.B. in Form von CPU-Ressourcen. Die Rechner sind über Kommunikationsverbindungen miteinander verbunden, wobei diese Verbindungen ebenfalls über nur begrenzte Kapazitäten verfügen, z.B. in Form von Bandbreiten-Ressourcen. Ein virtuelles Netz besteht analog aus virtuellen Maschinen (VMs), die über virtuelle Kommunikationskanäle miteinander verbunden sind. Die Struktur der virtuellen Netze ist über die Topologie des Netzes fest vorgegeben und nicht variabel. Für jede VM sind die Hardwareanforderungen spezifiziert, die zum Betrieb benötigt werden. Ebenso ist für jede virtuelle Verbindung der Bandbreiten-Bedarf der Kommunikationsverbindung spezifiziert. Aus komplexitätstheoretischer Sicht ist es egal, ob nun von CPU- und Bandbreitenressourcen gesprochen wird, die für die einzubettenden VMs und deren Kommunikationslinks reserviert werden, oder von anderen zu reservierenden Ressourcen (wie beispielsweise CPU-Kerne und Frequenzspektren). In der VNE-Literatur wird im Allgemeinen angenommen, dass physische Rechner CPU-Ressourcen zur Verfügung stellen, während physische Kommunikationsverbindungen über Bandbreitenressourcen bereitstellen, was im Folgenden übernommen wird.

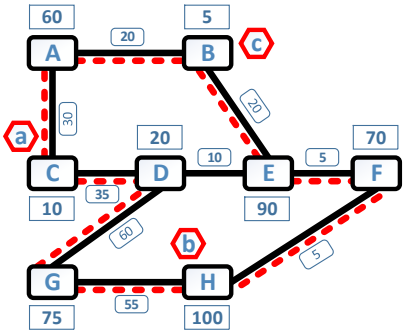
Aufgrund der fixen Struktur der virtuellen Netze lassen sich sowohl das physische Netz als auch die virtuellen Netze als gewichtete Graphen abstrahieren.



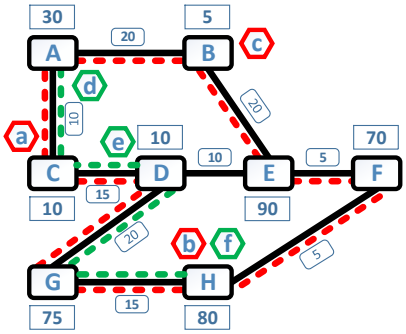
(a) Eintreffen zweier Einbettungsanfragen



(b) Schritt 1: Einbettung der Knoten des ersten virtuellen Netzes



(c) Schritt 2: Die eingebetteten virtuellen Knoten müssen innerhalb des physischen Netzes verbunden werden



(d) Schritt 3: Analog werden die virtuellen Knoten und Links des zweiten virtuellen Netzes eingebettet

Abb. 2.3: Zwei virtuelle Netze sollen innerhalb einer Netzinfrastruktur eingebettet werden – als Knoten- und Kantengewicht sind die nach jedem Schritt verbleibenden CPU- und Bandbreitenressourcen angegeben.

Abbildung 2.3 veranschaulicht das Deployment virtueller Netze in ein gemeinsam verwendetes physisches Netz. Konkret sind hier zwei Einbettungsanfragen dargestellt (rot und grün), die jeweils ein virtuelles Netz spezifizieren. Diese virtuellen Netze sollen in die Infrastruktur des physischen Netzes (schwarz) eingebettet werden, vgl. Abbildung 2.3a. Virtuelle Maschinen sind als Knoten des virtuellen Netzes repräsentiert und deren virtuelle Kommunikationsverbindungen als Links innerhalb des Netzwerkgraphen. Die Kantengewichte der virtuellen Links repräsentieren die Bandbreiten-Anforderungen der Verbindungen. Auch die Knoten der virtuellen Netze sind gewichtet; über das Knotengewicht ist der CPU-Bedarf der VMs spezifiziert. Analog dazu ist das physische Netz aufgebaut, welches aus physischen Knoten, die über CPU-Ressourcen verfügen, und physischen Links mit Bandbreitenressourcen konstruiert ist. Die in jedem Schritt verfügbaren CPU- und Bandbreitenressourcen sind innerhalb der Abbildung als Gewichte der Knoten bzw. Kanten angegeben.

Beim VNE-Problem soll nun jede VM/jeder virtuelle Knoten einem Rechner des physischen Netzes/einem physischen Knoten zugewiesen werden. Gleichzeitig muss die Konnektivität der in das physische Netz eingebetteten virtuellen Knoten gewährleistet sein. Zu diesem Zweck muss für jeden virtuellen Link ein geeigneter Kommunikationspfad innerhalb des physischen Netzes gefunden und reserviert werden.
Zunächst werden physische Knoten mit ausreichend verfügbaren CPU-Ressourcen ausgewählt, denen die Knoten des ersten virtuellen Netzes zugewiesen werden können, vgl. Abbildung 2.3b. Anschließend werden die virtuellen Kommunikationsverbindungen zwischen den platzierten Knoten reserviert (vlg. Abbildung 2.3c): In Abhängigkeit von der Struktur des virtuellen Netzes müssen dazu die eingebetteten virtuellen Knoten innerhalb des physischen Netzes mit anderen virtuellen Knoten verbunden werden. Dazu müssen Bandbreitenressourcen auf entsprechenden Kommunikationspfaden reserviert werden.
Abbildung 2.3d veranschaulicht das Deployment des zweiten virtuellen Netzes, dessen Platzierung analog erfolgt.

Algorithmen, welche die Einbettung virtueller Netze berechnen, müssen gleich zwei Teilprobleme lösen:

I) **Einbettung virtueller Knoten**
Jeder virtuelle Knoten mit Knotengewicht x kann nur auf einem physischen Knoten mit einem Gewicht $\geq x$ eingebettet werden. Generell können einem physischen Knoten mehrere virtuelle Knoten zugewiesen werden, solange dieser physische Knoten über die entsprechenden CPU-Kapazitäten verfügt.

II) **Einbettung virtueller Links**
Das Deployment der virtuellen Knoten allein ist für den Betrieb des virtuellen Netzes nicht ausreichend, da die Konnektivität der virtuellen Instanzen nicht gewährleistet ist. Für die virtuellen Links werden daher Bandbreitenressourcen auf Pfaden des physischen Netzes reserviert, wobei sich ein solcher Pfad über mehrere physische Links erstrecken kann. Auf jedem einzelnen Link eines physischen Pfades werden die Bandbreitenressourcen reserviert, die für den Betrieb der jeweiligen virtuellen Links erforderlich sind.

Die Einbettung der virtuellen Knoten und Links wurde hier zur Veranschaulichung in zwei separaten Schritten beschrieben. Tatsächlich lösen einige der ersten VNE-Algorithmen, die in der Literatur vorgestellt wurden, das Problem ebenfalls in zwei voneinander getrennten, unkoordinierten Schritten, beispielsweise [109, 105, 95, 91, 76, 27]. Aufgrund der Tatsache, dass die Zuweisungsmöglichkeiten der Link-Einbettungsphase durch die vorherige Zuweisung der virtuellen Knoten stark eingeschränkt werden, resultiert eine

derart unkoordinierte Vorgehensweise in der Regel in ineffizienten Deployment-Ergebnissen – unter Umständen ist die Einbettung der virtuellen Links in Folge einer ungünstigen Platzierung der virtuellen Knoten überhaupt nicht mehr möglich. Aus diesem Grund wählen viele der neueren VNE-Algorithmen eine koordinierte Vorgehensweise, in der diese beiden Teilprobleme, nämlich die Zuweisung der virtuellen Knoten und die Zuweisung der virtuellen Links, zusammen genommen betrachtet und koordiniert angegangen werden, beispielsweise [86, 99, 98, 97, 71].

Bei der Berechnung der Einbettung virtueller Netze lassen sich unterschiedliche Optimierungsziele verfolgen. Die meisten VNE-Verfahren versuchen, für jedes virtuelle Netz möglichst wenig CPU- und Bandbreitenressourcen zu reservieren, damit nach der Platzierung eines virtuellen Netzes noch möglichst viele Ressourcen für die Einbettung zukünftiger Netze zur Verfügung stehen [55]. Weitere Optimierungskriterien werden in Abschnitt 2.4 beschrieben.

Es sei an dieser Stelle angemerkt, dass virtuelle Netzwerkkomponenten nicht nur physischen, sondern wiederum ebenfalls virtuellen Komponenten zugewiesen werden können. Dieses Prinzip wird verschachtelte Virtualisierung (*nested virtualization*) genannt [55]. In einem solchen Szenario gibt es mehrere Ebenen virtueller Netze, von denen nur die unterste Ebene aus einem physischen Netz besteht. Statt von physischen Netzwerken, auf die virtuelle Netze verteilt werden, wird daher im Folgenden verallgemeinernd vom Substratnetz beziehungsweise von den Substratknoten und Substratlinks gesprochen.

2.2.1 Komplexität des VNE-Problems

Das VNE-Problem *optimal* zu lösen ist $\mathcal{NP}$-hart, da zwei $\mathcal{NP}$-harte Teilprobleme gelöst werden müssen: Zum einen muss eine optimale Verteilung der virtuellen Knoten auf die Substratknoten gefunden werden. Dieses Teilproblem lässt sich auf das Bin Packing-Problem reduzieren, welches schon für sich genommen ein kombinatorisches, $\mathcal{NP}$-hartes Optimierungsproblem ist. Darüber hinaus ist die optimale Zuweisung der virtuellen Links auf einzelne Substratpfade ebenfalls $\mathcal{NP}$-hart: Selbst wenn die optimale Verteilung der virtuellen Knoten bekannt wäre, müsste noch eine optimale Zuweisung der Kommunikationsverbindungen gefunden werden. Dieses Link-Zuweisungsproblem kann auf das ebenfalls $\mathcal{NP}$-harte *unsplittable flow problem* reduziert werden [55, 32, 9].

Optimale Verfahren sind also nur in sehr kleinen Szenarien praktikabel anwendbar. Sehr viele VNE-Ansätze lösen das Optimierungsproblem daher heuristisch; in der Tat handelt es sich bei den meisten VNE-Algorithmen um Heuristiken (u.a. [109, 105, 30, 53, 93, 34, 29]) oder Meta-Heuristiken (u.a. [52, 107, 33, 106]).

Parameter	Beschreibung
	Substratnetz
$G = (N, L)$	Das Substratnetz G ist modelliert als gerichteter Graph G mit den Substratknoten N und Substratlinks L, wobei $L \subseteq N^2$
$\text{res}_{\text{cpu}} : N \to \mathbb{Z}_0$	Die zur Verfügung stehenden CPU-Kapazitäten eines Substratknotens, die für das Deployment virtueller Knoten reserviert werden können
$\text{res}_{\text{bw}} : L \to \mathbb{Z}_0$	Zur Verfügung stehende Bandbreitenressourcen, die von den Substratlinks zum Betreiben virtueller Links bereit gestellt werden
	Virtuelle Netze
$\text{VNR}^i = (N^i, L^i)$	Das virtuelle Netz der i-ten Einbettungsanfrage mit den virtuellen Knoten N^i und den virtuellen Links L^i
$\text{dem}_{\text{cpu}} : N^i \to \mathbb{Z}_0$	CPU-Ressourcenanforderungen der virtuellen Knoten N^i (dem_{cpu} für englisch *CPU-demands*)
$\text{dem}_{\text{bw}} : L^i \to \mathbb{Z}_0$	Bandbreiten-Ressourcenanforderungen der virtuellen Links L^i (dem_{bw} für englisch *bandwidth demands*)
	Deployment-Funktionen
$\text{map}_{\text{node}} : N^i \to N$	Funktion, die die Platzierung der virtuellen Knoten auf Knoten des Substratnetzes beschreibt.
$\text{map}_{\text{link}} : L^i \to 2^L$	Funktion, die die Platzierung der virtuellen Links auf *Pfade* des Substratnetzes beschreibt.

Tabelle 2.1: Modell für das Virtual Network Embedding Problem

Ein ausführlicher Survey über verschiedene VNE-Einbettungsstrategien, -Optimierungsziele und -Metriken wurde in [55] publiziert.

2.2.2 Ein Modell für das VNE-Problem

Dieser Abschnitt beschreibt ein Modell für das VNE-Problem, auf welches im Rahmen dieser Arbeit Bezug genommen wird.
Sowohl das Substratnetz als auch die virtuellen Netze sind als Graphen modelliert, vgl. Tabelle 2.1. Ein Substratnetz $G = (N, L)$ ist ein Graph G mit den Substratknoten N und den Substratlinks L. Substratknoten und -links stellen unterschiedliche Ressourcen zur Verfügung. Für Substratknoten können dies beispielsweise CPU-Ressourcen $\text{res}_{\text{cpu}} : N \to \mathbb{Z}_0$, für Substratlinks Bandbreiten-Ressourcen $\text{res}_{\text{bw}} : N \to \mathbb{Z}_0$ sein.
In Analogie dazu wird ein virtuelles Netz VNR^i als ein Graph mit den virtuellen Knoten N^i und den virtuellen Links L^i modelliert. CPU-

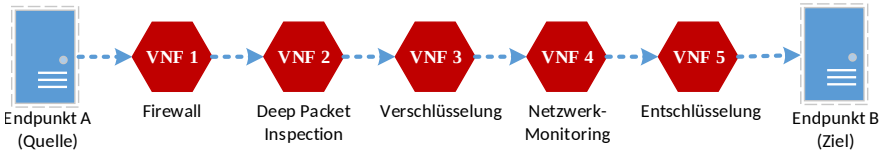


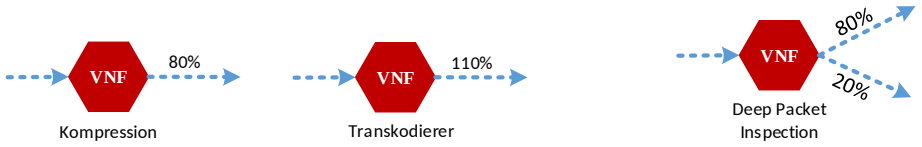
Abb. 2.4: Mehrere zu einem Netzwerkservice verkettete Netzwerkfunktionen, die einen Datenstrom zwischen den Endpunkten A und B verarbeiten

Ressourcenanforderungen eines virtuellen Knotens $n^i \in N^i$ sind gegeben über die Funktion $\text{dem}_{\text{cpu}}(n^i)$ und Bandbreiten-Ressourcenanforderungen eines virtuellen Links $l^i \in L^i$ über $\text{dem}_{\text{bw}}(l^i)$.
Wie oben diskutiert, kann das VNE-Einbettungsproblem als zweiteiliger Prozess formuliert werden: Zum einen muss jeder virtuelle Knoten einem Substratknoten zugewiesen werden, der über genügend CPU-Kapazitäten verfügt; dieser Schritt lässt sich über die Zuordnungsfunktion $\text{map}_{\text{node}} : N^i \to N$ beschreiben. Zum anderen muss jeder virtuelle Link einem Substratpfad mit ausreichenden Bandbreiten-Ressourcen zugeordnet werden, was über die Funktion $\text{map}_{\text{link}} : L^i \to 2^L$ ausgedrückt wird.

2.3 Deployment von NFV-Netzwerkservices: Das NFV-RA-Problem

Im Folgenden wird beschrieben, inwiefern das VNE-Problem mit dem Deployment virtueller Netzwerkservices zusammenhängt. Wie in Kapitel 1 beschrieben, ist der Grundgedanke hinter NFV-fähigen Mobilfunknetzen die logische Funktionalität der Funktionseinheiten von der eigentlichen physischen Hardware zu entkoppeln. In NFV-Netzen liegt der Fokus, anders als in Cloud-Szenarien, nicht mehr primär auf der Betrachtung der zu betreibenden virtuellen Maschinen, sondern noch viel stärker auf der Betrachtung der zu verarbeitenden Datenströme, die durch die Netzwerkinfrastruktur geroutet werden. Diese Datenströme werden von mehreren miteinander vernetzen virtuellen Netzwerkfunktionen (VNFs) verarbeitet. Kooperierende virtuelle Netzwerk*funktionen* bilden dabei einen Netzwerk*service*. Ein solcher Netzwerkservice ist in Abbildung 2.4 veranschaulicht: Die zwischen den hier gezeigten Endpunkten A und B ausgetauschten Daten durchlaufen verschiedene VNFs wie eine Firewall, eine Netzwerk-Monitoringeinheit und eine Kryptographiefunktion.

Die Netzwerkfunktionen wenden unterschiedliche Verarbeitungsoperationen auf die durch sie geleiteten Datenströme an; sie sind in der Lage, die Daten zu filtern, zu modifizieren und zum Zwecke der Weiterverarbeitung an eine oder mehrere Nachfolgefunktionen weiterzureichen. Jeder Datenverarbeitungsschritt kann dabei Auswirkungen auf das Volumen des Datenstroms ha-



(a) Netzwerkfunktionen mit unterschiedlichen Ausgangsdatenraten

(b) Eine Netzwerkfunktion, die den Datenfluss in zwei Teilflüsse splittet

Abb. 2.5: Typen von virtuellen Netzwerkfunktionen

ben: Beispielsweise reduziert eine Kompressionsfunktion die Größe des Datenstroms; im Gegensatz dazu kann eine Transkodierfunktion, die das Encoding von Multimediadaten ändert, das Datenaufkommen steigern. Dies ist in Abbildung 2.5 veranschaulicht: Der Ausgabedatenstrom der gezeigten Kompressionsfunktion ist um 20% kleiner als der Eingabedatenstrom, der der Transkodierfunktion um 10% größer. Netzwerkfunktionen können den Datenfluss in mehrere Teilflüsse aufsplitten oder mehrere Datenflüsse wieder zusammenführen; eine Deep Packet Inspection-Einheit kann beispielsweise TCP-Daten von nicht-TCP-Daten trennen, vgl. Abbildung 2.5b, während eine Multimedia-Encodingfunktion Audio- und Videodaten zu einem einzelnen Multimediastream vereinen kann.

2.3.1 Verkettung virtueller Netzwerkfunktionen

In welcher Reihenfolge die einzelnen VNFs miteinander verkettet werden dürfen, wird innerhalb von Einbettungsanfragen, sogenannten VNFRs (für *Virtual Network Function embedding Requests*), spezifiziert. In einem VNFR ist also festgelegt, aus welchen VNFs der Netzwerkservice zusammengesetzt werden soll. Je nachdem, wie die VNFs angeordnet werden, ergeben sich unterschiedliche Ressourcenanforderungen, die vom Substratnetz erfüllt werden müssen. Dies ist in Abbildung 2.6 gezeigt: Hier soll der in der Einbettungsanfrage spezifizierte Videooptimierer mit einer Deep Packet Inspection-VNF verknüpft werden. Dafür kann in dem gewählten Beispiel entweder die Instanz der Videooptimier-VNF zuvorderst platziert werden oder die Deep Packet Inspection-Instanz. Vgl. hierzu die im Bild rechts gezeigten virtuellen Netze, die sich aus *Instanzen* der VNFs des VNFR zusammensetzen; im Zusammenhang mit NFV werden solche virtuellen Netze in der Literatur üblicherweise auch als *Virtual Network Function Forwarding Graph*, VNF-FG, bezeichnet. Im ersten Fall ist der Bandbreitenbedarf um 100 MBps höher. Im zweiten Fall müssen dafür insgesamt drei VNF-Instanzen anstelle von nur zwei wie im ersten Fall erzeugt werden. Je nach gewählter Variante fallen die Topologien der erzeugten VNF-FGs also unterschiedlich aus.

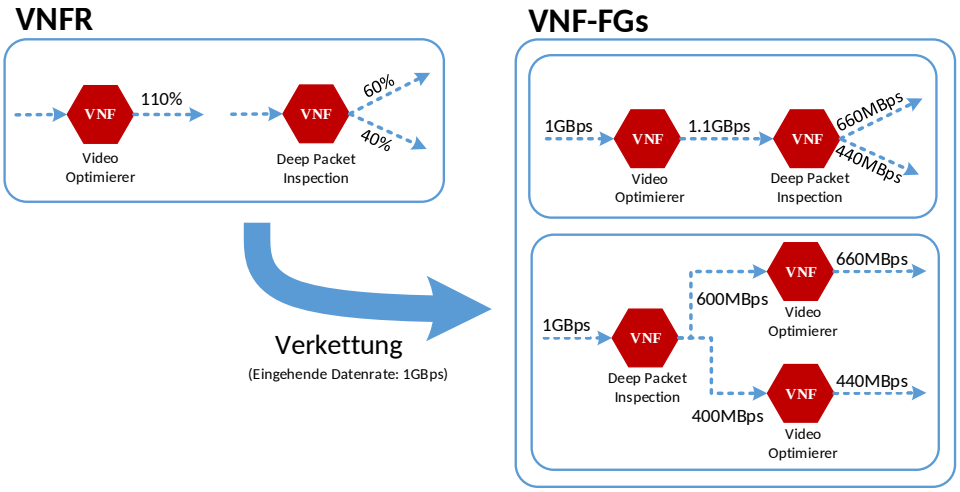


Abb. 2.6: Mehrere Möglichkeiten der Verkettung von virtuellen Netzwerkfunktionen zu einem Netzwerkservice, der sich aus *Instanzen* dieser Netzwerkfunktionen zusammensetzt

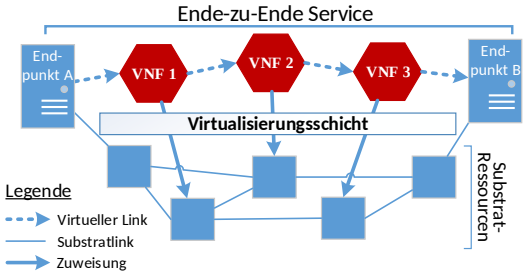


Abb. 2.7: Einbettung eines NFV-Netzwerkservices (Abbildung in Anlehnung an [48])

2.3.2 Einbettung verketteter Netzwerkfunktionen

Beim Deployment virtueller Netzwerkservices geht es darum, den einzelnen Servern die für den Betrieb benötigten Netzwerkfunktionen zuzuweisen. Abbildung 2.7 veranschaulicht die Einbettung eines Netzwerkservices in das Substratnetz. Für jeden Netzwerkservice muss eine laut den Spezifikationen des VNFR zulässige Verkettungsmöglichkeit gewählt werden, die sich in die Infrastruktur des Substratnetzes einbetten lässt. Es muss also *ein* VNF-FG ausgewählt werden, dessen Einbettung berechnet wird.
Ähnlich wie bei VNE-Szenarien geht es bei der Einbettung solcher Netzwerkservices darum, die virtuellen Funktionseinheiten innerhalb des Substratnetzes zu platzieren. Dieses Zuweisungsproblem wird in der Literatur als das *Network Function Virtualization-Resource Allocation*-Problem (NFV-RA) bezeichnet.
Zusammengefasst kann das Deployment virtueller Netzwerkservices in die folgenden zwei Teilaufgaben eingeteilt werden:

1. Verkettung der VNFs (*Chaining*) zu einer Service Chain

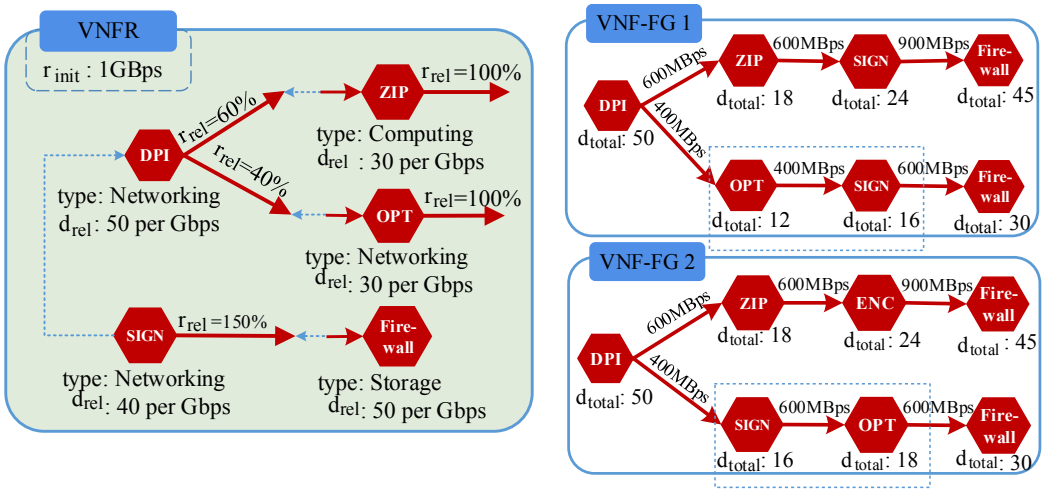
Dies erfolgt entsprechend der Spezifikation des Netzwerkservices, in der die Abhängigkeiten der einzelnen Netzwerkfunktionen aufgeführt sind.

2. Zuweisung der VNFs auf Ressourcen des Substratnetzes (*VNF Forwarding Graph Embedding*, VNF-FGE)

Ein Deployment-Algorithmus muss daher zwei Teilaufgaben lösen: Zum einen ist eine gültige und günstige Verkettung der VNFs auszuwählen, zum anderen muss, ähnlich wie beim VNE-Problem, eine effiziente Einbettung einer dieser Verkettungsoptionen gefunden werden. Dieses kombinierte Optimierungsproblem wird als NFV-RA-Problem bezeichnet. Die Platzierung der VNFs ist $\mathcal{NP}$-hart, und zwar aus denselben Gründen, aus denen auch das VNE-Problem $\mathcal{NP}$-hart ist.

2.3.3 Abgrenzung zum Virtual Network Embedding-Problem

Das NFV-RA-Problem ist verwandt mit dem in Abschnitt 2.2 beschriebenen VNE-Problem. Auch hier müssen virtuelle Maschinen auf Substratknoten verteilt werden, deren Ressourcen beschränkt sind. Anders als das VNE-Problem handelt es sich in einem NFV-Szenario nicht um statische virtuelle Netztopologien, die innerhalb des Substratnetzes zu platzieren sind, sondern um Netze, deren Topologie flexibel ist. Hier geht es um die Verkettung virtueller Netzwerkfunktionen, die die Daten aus verschiedenen Datenströmen verarbeiten. Die Bearbeitungsreihenfolge, in der Netzwerkfunktionen Datenflüsse verarbeiten und in der Daten untereinander weitergereicht werden, ist, anders als in VNE-Szenarien, nicht eindeutig vorgegeben. VNFs können flexibel miteinander verkettet werden; die Verkettung der VNFs muss dabei nach den Regeln erfolgen, die innerhalb der Einbettungsanfrage spezifiziert sind. Je nachdem, in welcher Reihenfolge die Funktionen verkettet werden, ändert sich auch der jeweilige Bedarf an Netzwerkressourcen, die für die Einbettung in das Substratnetz benötigt werden. Beispielsweise sind die Rechenanforderungen, die an eine Transkodierfunktion gestellt werden, welche die Enkodierung von Multimediadaten verändert, abhängig von der Menge der zu transcodierenden Daten.
Zudem ändert sich der Bandbreitenbedarf einer Kette je nach Anordnung der Netzwerkfunktionen. Ein Zuweisungsalgorithmus, der zum Ziel hat, den Ressourcenbedarf der von ihm eingebetteten Funktionsketten zu minimieren, wird tendenziell solche Funktionen zuvorderst platzieren, welche den Bandbreitenbedarf eines Netzwerkflusses reduzieren.
Abgesehen von den Unterschieden, die sich aus der flexiblen Verkettung der Netzwerkfunktionen ergeben, unterscheidet sich das NFV-RA-Problem vom VNE-Problem auch in der Art und Weise, wie die Netzwerkfunktionen den Substratknoten zugewiesen werden. Beim VNE-Problem repräsentieren virtuelle Knoten separate virtuelle Maschinen, die untereinander vollständig voneinander isoliert bleiben. In einem NFV-RA-Szenario können virtuelle Maschinen



(a) NFV-Einbettungsanfrage (VNFR)

(b) Verkettung von Netzwerkfunktionen

Abb. 2.8: In einer Einbettungsanfrage sind die Verkettungsmöglichkeiten der einzelnen Netzwerkfunktionen spezifiziert.

„geteilt“ und von mehreren VNF-Instanzen des gleichen Typs gemeinsam genutzt werden: Zwei Firewall-Funktionen benötigen nicht zwingend separate virtuelle Maschinen, wenn sie demselben Substratknoten zugewiesen werden.

2.3.4 Ein Modell für das NFV-RA-Problem

Im Folgenden wird das NFV-RA-Problem modelliert. Das Modell setzt sich aus drei Teilen zusammen: dem Modell für das Substratnetz, vgl. Tabelle 2.2, dem Modell für Einbettungsanfragen, vgl. Tabelle 2.3, sowie dem Modell für verkettete Netzwerkfunktionen, vgl. Tabelle 2.4.

Das Substratnetz ist als gerichteter Graph modelliert, dessen Knoten und Kanten Ressourcen zugewiesen sind. Jedem Knoten ist ein bestimmter Hardware-Typ zugeordnet, wobei ein Knoten entweder vom Typ Netzwerk-, Rechen- oder Speicherhardware ist. Ein Substratknoten kann nur VNFs desselben Typs hosten. Das heißt, dass ein Netzwerk-Substratknoten auch nur Instanzen von Netzwerk-VNFs hosten kann, aber keine Instanzen von Rechen-VNFs. Dessen Ressourcen sind allgemein als Kapazitäten modelliert, die von den VNF-Instanzen reserviert werden können.

Für jede Einbettungsanfrage (VNFR) sind die VNFs spezifiziert, die zum Betrieb des Netzwerkservices eingebettet werden müssen, graphisch veranschaulicht in Abbildung 2.8a. Außerdem ist die initiale Rate des Datenflusses angegeben (r_{init}), der durch den Netzwerkservice geleitet werden soll. Jede der VNFs verfügt über eingehende und ausgehende Links. Eine Netzwerkfunktion,

die über mehrere ausgehende Links verfügt, splittet den Netzwerkfluss in mehrere Teilflüsse; eine Funktion mit mehreren eingehenden Links vereint mehrere Teilflüsse wieder. Für jeden ausgehenden Link ist die relative Datenrate angegeben, mit der die VNF eintreffende Daten weiterleitet (in Abbildung 2.8a wurde beispielsweise eine `DPI`-Funktion spezifiziert, die eintreffende Daten in zwei kleinere Datenströme aufsplittet; das Volumen des ersten Datenstroms entspricht 60% des Volumens des Eingangsdatenstroms, das des zweiten 40%). Ferner sind die Abhängigkeiten der VNFs angegeben: In dem in Abbildung 2.8a gezeigten Szenario soll der bei `Firewall`-VNF eintreffende Datenfluss zuvor immer durch die Signierfunktion (`SIGN`) geleitet werden; diese Anforderung ist hier visualisiert als blaue gestrichelte Linie zwischen den Links der VNFs. Das bedeutet, dass Datenströme, die von der VNF auf dem eingehenden Link entgegengenommen werden, zuvor *Instanzen* der geforderten VNFs durchlaufen haben müssen.

Ausgehend vom spezifizierten VNFR lassen sich mögliche VNF-FGs ableiten, die sich hinsichtlich des Bandbreiten- und Kapazitätsbedarfs unterscheiden. Abbildung 2.8b zeigt zwei valide Verkettungsmöglichkeiten des VNFR aus Abbildung 2.8a. Im Vergleich zu VNF-FG 1 ist die Reihenfolge der `OPT`- und der `SIGN`-VNFs vertauscht – dies hat sowohl Auswirkungen auf die Ressourcenanforderungen der VNF-Instanzen als auch auf die Größe des weitergeleiteten Datenstroms.

2.4 Effizientes Einbetten virtueller Netzstrukturen

Die Bestimmung eines optimalen Deployments der virtuellen Netzwerkstrukturen ist sowohl beim VNE-Problem als auch beim NFV-RA-Problem $\mathcal{NP}$-hart. Der Begriff der *Optimalität* kann in diesem Zusammenhang unter verschiedenen Gesichtspunkten betrachtet werden. Im Folgenden wird eine Übersicht über die in der Literatur verbreiteten Einbettungsstrategien gegeben.

Eines der am häufigsten diskutierten Optimierungsziele ist dabei die Reduktion der Betriebskosten des Infrastrukturproviders. Im Vordergrund steht dabei die Frage, wie sich die virtuellen Netze derart einbetten lassen, dass der Ressourcenbedarf für den Betrieb der virtuellen Netze möglichst minimal ausfällt. In der Literatur wird dies unter dem Begriff der Einbettungskosten-Minimierung zusammengefasst. Ein wesentlicher Aspekt für ein solches ressourceneffizientes Deployment ist dabei die durchschnittliche Pfadlänge der Substratpfade, die für die Links des virtuellen Netzes reserviert werden: Mit der Länge der reservierten Pfade steigen auch die Einbettungskosten des Deployments. Das liegt daran, dass auf jedem Link, der Bestandteil eines Substratpfades ist, ent-

Parameter	Beschreibung
$\mathbf{G} = (\mathbf{N}, \mathbf{L})$	Das Substratnetz G ist modelliert als ein gerichteter Graph mit den Substratknoten N und Substratlinks L sowie der Menge aller möglichen Pfade $P \subseteq 2^L$.
$\mathbf{v} : \mathbf{P} \rightarrow \mathbf{2^N}$	Bestimmt die sogenannten *hidden hops* eines Substratpfades $p \in P$. Dies sind solche Knoten, durch die ein Substratpfad führt, die aber weder Start- noch Zielknoten des Pfades sind [28].
$\mathbf{source} : \mathbf{L} \rightarrow \mathbf{N}$ $\mathbf{destination} : \mathbf{L} \rightarrow \mathbf{N}$	Diese Funktionen liefern den Ursprungs- (source) und den Zielknoten (destination) eines Substratlinks $l \in L$.
$\mathbf{host} : \mathbf{N} \rightarrow \mathbb{Z}$	Maximale Anzahl der VMs, die auf einem Substratknoten installiert werden können.
$\mathbf{res_{node}} : \mathbf{N} \rightarrow \mathbb{Z}$	Zur Verfügung stehende Verarbeitungskapazitäten eines Substratknotens; Verarbeitungskapazitäten werden zur Verarbeitung von Rechen-/Speicher-/Netzwerk-Aufgaben für VNF-Instanzen reserviert.
$\mathbf{res_{link}} : \mathbf{L} \rightarrow \mathbb{Z}$	Zur Verfügung stehende Bandbreitenressourcen, die von den Substratlinks zum Hosting von virtuellen Links bereit gestellt werden.
$\mathbf{type}_s : \mathbf{N} \rightarrow \{\mathbf{Computing}, \mathbf{Storage}, \mathbf{Networking}\}$	Diese Funktion bestimmt den Typ eines Substratknotens. Ein Substratknoten kann nur solche VNFs hosten, die vom selben Typ wie der Substratknoten selbst sind (also nur dann, wenn $\text{type}_v(v) = \text{type}_s(n)$ gilt).

Tabelle 2.2: NFV-RA Modell: Das Substratnetz

sprechende Ressourcen reserviert werden müssen. Dieser Zusammenhang ist in Abbildung 2.9 veranschaulicht. Hier soll das erste virtuelle Netz aus dem in Abbildung 2.3 gezeigten Szenario eingebettet werden. Aufgrund der Ressourcenverteilung des Substratnetzes sind dabei mehrere Platzierungsmöglichkeiten durchführbar. Bei dem in Abbildung 2.9a gezeigten Deployment sind die Einbettungskosten recht hoch, da sich die Kommunikationsverbindungen zwischen den virtuellen Knoten über lange Pfade des Substratnetzes erstrecken. Auf jedem Link dieser Pfade müssen entsprechende Ressourcen reserviert werden, die sich für die Platzierung weiterer virtueller Netze nicht mehr nutzen lassen. In Abbildung 2.9b wurde ein ressourceneffizienteres Deployment gewählt.

Je weniger Ressourcen innerhalb des Substratnetzes zur Verfügung stehen, desto unwahrscheinlicher ist es, dass weitere virtuelle Netze in der Substratnetzinfrastruktur erfolgreich untergebracht werden können. Aus diesem Grund versuchen viele Einbettungsverfahren, die Einbettungskosten zu *minimieren*, um die Anzahl der virtuellen Netze, die erfolgreich eingebettet werden können, zu *maximieren*. Entsprechende VNE-Algorithmen sind u.a. die in

Parameter	Beschreibung
$\mathbf{VNFR}^i(\mathbf{V}^i)$	Der i-te VNF-Request (VNFR), zu dem die VNFs V^i gehören.
$\mathbf{type}_v^i : \mathbf{V}^i \rightarrow \{\mathbf{Computing}, \mathbf{Storage}, \mathbf{Networking}\}$	Bestimmt den Typ einer VNF.
$\mathbf{l}_{\mathbf{out}}^i : \mathbf{V}^i \rightarrow \mathbf{L}_{\text{out}}^i$	Bestimmt die ausgehenden Links einer VNF; eine VNF mit mehreren ausgehenden Links splittet den Netzwerkfluss in mehrere Teilflüsse.
$\mathbf{l}_{\mathbf{in}}^i : \mathbf{V}^i \rightarrow \mathbf{L}_{\text{in}}^i$	Bestimmt die eingehenden Links einer VNF. Eine VNF mit mehreren eingehenden Links führt zwei Netzwerkflüsse zusammen.
$\mathbf{r}_{\mathbf{init}}^i : \mathbb{Z}$	Initiale Datenrate eines VNFR.
$\mathbf{d}_{\mathbf{rel}}^i : \mathbf{V}^i \rightarrow \mathbb{R}$	Relative benötigte Verarbeitungskapazität einer VNF.
$\mathbf{r}_{\mathbf{rel}}^i : \mathbf{L}_{\text{out}}^i \rightarrow \mathbb{Z}$	Relative Datenrate eines ausgehenden VNF-Links.
$\mathbf{req}^i : \mathbf{L}_{\text{in}}^i \rightarrow \{\mathbf{L}_{\text{out}}\}$	Abhängigkeiten einer VNF. Diese werden für jeden eingehenden Link definiert und beziehen sich auf von anderen VNFs ausgehende Links. Durch den Bezug auf ausgehende Links anderer VNFs lassen sich VNFs definieren, die nur selektiv auf einigen der Teilflüsse des VNF-FG platziert werden sollen, nicht aber auf allen. Die Zuweisung einer VNF-Instanz ist nur dann gültig, wenn der Netzwerkfluss zuerst durch Instanzen derjenigen VNFs geleitet wurde, von denen die aktuell einzubettende VNF abhängt.

Tabelle 2.3: NFV-RA Modell: Einbettungsanfragen

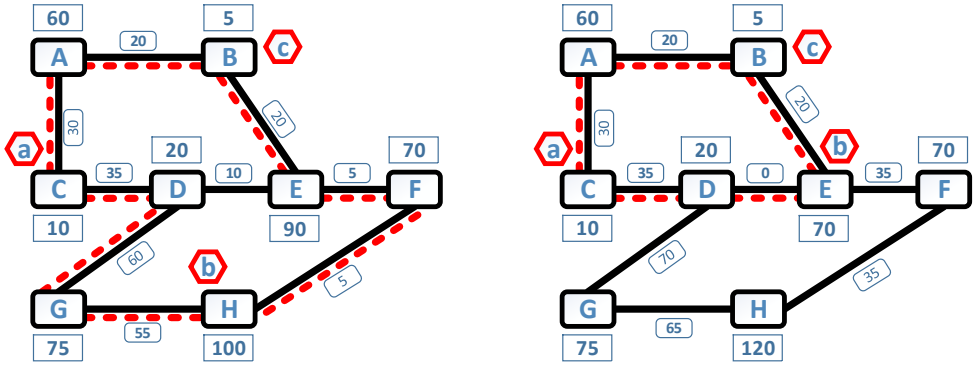
[37, 78, 52, 29] vorgestellten Verfahren, während dies in der NFV-RA-Literatur u.a. in [12, 85, 82, 13, 88, 59] diskutiert wird.
In der VNE-Literatur wurden zudem Verfahren vorgestellt, die versuchen, die virtuellen Netze so einzubetten, dass sie mit minimalem Stromverbrauch betrieben werden können. Hier steht also die Maximierung der Energieeffizienz des Deployments im Vordergrund [54, 26, 92, 79, 84, 43, 108, 44, 61]: Einige Arbeiten aus dem NFV-Umfeld befassen sich ebenfalls mit dem energieeffizienten Deployment [13, 82]. Im Allgemeinen gilt, dass der Stromverbrauch für den Betrieb virtueller Netze umso geringer ausfällt, je weniger Komponenten des Substratnetzes für den Betrieb der virtuellen Netze benötigt werden. Folglich sinken auch die Betriebskosten des Infrastrukturanbieters. Die Einbettung mit den geringsten Einbettungskosten ist dabei nicht automatisch auch die energieeffizienteste Lösung. Beispielsweise ist die Einbettung eines virtuellen Links auf einen langen Substratpfad zwar mit hohen Einbettungskosten ver-

Parameter	Beschreibung
$\mathbf{VNF\text{-}FG}^{i,j}(\mathbf{V}^{i,j}_{\text{inst}}, \mathbf{L}^{i,j}_{\text{inst}})$	Der j-te VNF-FG eines NFV-Requests VNFR^i; ein VNF-FG ist modelliert als ein Graph mit den VNF-Instanzen $V^{i,j}_{\text{inst}}$ und den virtuellen Links $L^{i,j}_{\text{inst}}$. Für jede als Teil eines VNFR definierte VNF kann es mehrere Instanzen innerhalb eines VNF-FG geben (cf. Fig. 2.4). Ebenso kann es mehrere virtuelle Links innerhalb eines VNF-FG geben, die sich auf denselben VNF-Link beziehen.
$\mathbf{d}^{i,j}_{\mathbf{total}} : \mathbf{V}^{i,j}_{\text{inst}} \rightarrow \mathbb{R}$	Bestimmt die absoluten Kapazitätsanforderungen einer VNF-Instanz innerhalb eines VNF-FG; die absoluten Kapazitätsanforderungen werden anhand der relativen benötigten Verarbeitungskapazitäten sowie der Menge an Daten, die durch die VNF-Instanz geroutet wird, berechnet. Einem Substratknoten $n \in N$ kann nur dann eine VNF $v \in V_{\text{inst}}$ zugewiesen werden, wenn gilt: $\text{d}_{\text{total}}(v) \leq \text{res}_{\text{node}}(n)$.
$\mathbf{r}^{i,j}_{\mathbf{total}} : \mathbf{L}^{i,j}_{\text{inst}} \rightarrow \mathbb{Z}$	Absolute benötigte Datenrate eines virtuellen Links. Ähnlich wie die absoluten Kapazitätsanforderungen einer VNF-Instanz berechnet sich die absolute Datenrate auf Grundlage der festgelegten relativen Datenrate des VNF-Links und der Menge an Daten, die durch den virtuellen Link geroutet wird.
$\mathbf{instanceof} : \mathbf{V}^{i,j}_{\text{inst}} \rightarrow \mathbf{V}^i$	Bestimmt, welche VNF die VNF-Instanz $v^{i,j}_{\text{inst}} \in V^{i,j}_{\text{inst}}$ instanziiert.

Tabelle 2.4: NFV-RA Modell: Verkettete Netzwerkfunktionen

bunden; erstreckt sich der Pfad jedoch über Komponenten, die bereits für den Betrieb anderer virtueller Knoten oder Links vorgesehen sind, kann eine solche Zuweisung insgesamt dennoch energieeffizienter ausfallen als die Reservierung bislang inaktiver, kurzer Pfade [54].

Ein weiteres mögliches Optimierungsziel ist die Minimierung von Kommunikationsdelays innerhalb der eingebetteten virtuellen Netze. Die kosteneffizienteste Einbettung ist nicht immer auch gleichzeitig die mit den geringsten Kommunikationsdelays. Dies ist in Abbildung 2.10 veranschaulicht: Hier wurden zwei virtuelle Knoten a und b auf den Substratknoten A und C eingebettet. Der Pfad mit dem geringsten Kommunikationsdelay (20ms) ist A-B-C, während der Delay auf Pfad A-C bei 50ms liegt. Die Einbettungskosten würden in letzterem Fall geringer ausfallen, da insgesamt weniger Ressourcen des Substratnetzes reserviert werden müssen [24]. In der Literatur wurden in diesem Zusammenhang mehrere VNE-Verfahren vorgestellt, die versuchen, die Kommunikationsdelays der Deployments zu minimieren [24, 70, 68].



(a) Ein Deployment mit höheren Einbettungskosten

(b) Ein Deployment mit niedrigeren Einbettungskosten

Abb. 2.9: Verschiedene Möglichkeiten für die Einbettung des ersten virtuellen Netzes aus dem in Abbildung 2.3 gezeigten Szenario – als Knoten- und Kantengewicht sind die jeweils verbleibenden CPU- und Bandbreitenressourcen angegeben

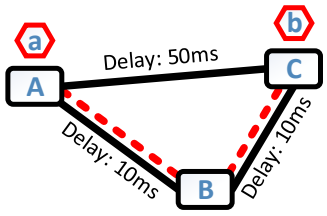


Abb. 2.10: Die Kosten-effizienteste Einbettung ist nicht immer auch diejenige mit den geringsten Kommunikationsdelays

In der VNE-Literatur finden sich zudem einige Verfahren, die zusätzliche Backupressourcen reservieren, um die Ausfallsicherheit eingebetteter virtueller Netze sicherzustellen [93, 94, 104, 100]. All diese Algorithmen versuchen, eine möglichst kosteneffiziente Einbettung zu finden, bei der die virtuellen Netze gegen Ausfälle abgesichert werden.

Da viele dieser Optimierungsziele zueinander in Konkurrenz stehen, bietet sich in vielen Fällen die Definition einer Optimierungsfunktion an, die verschiedene Optimierungskriterien je nach Relevanz gewichtet und zueinander in Relation setzt. Dieses Vorgehen zur Lösung eines multikriteriellen Optimierungsproblems wird von einigen der vorgestellten VNE-Algorithmen gewählt.

2.5 Zusammenfassung

In diesem Kapitel wurde zunächst eine Taxonomie für virtuelle Netzwerkfunktionen, die auf MEC-Servern installiert werden, vorgestellt. Dabei wurde auf verschiedene technische Integrationsmöglichkeiten der MEC-Server ein-

gegangen; allen gemein ist, dass die MEC-Server in unmittelbarer Nähe zur Basisstation installiert werden. Darauf aufbauend wurde eine Klassifikation verschiedener MEC-Anwendungen und Use-Cases vorgestellt.

Anschließend wurde diskutiert, wie sich das Deployment allgemeiner virtueller Netzwerkservices (die nicht zwingend nur auf MEC-Servern installiert werden) auf das VNE- und das NFV-RA-Problem zurückführen lässt.

Für diese beiden Deployment-Probleme wurden unterschiedliche Optimierungsstrategien sowie Evaluationsmetriken beschrieben, die bei der Einbettung verfolgt werden können. Ein in der Literatur sehr häufig gewähltes Optimierungsziel ist dabei die Reduktion der Einbettungskosten.

3 Simulationsbasierte Evaluation von Einbettungsalgorithmen

Um möglichst allgemeingültige Aussagen über die Effizienz von VNE- und NFV-RA-Algorithmen treffen zu können, werden diese im Zusammenhang mit zufällig generierten Einbettungsszenarien evaluiert. Ein solches simulationsbasiertes Vorgehen bietet sich an, da die Evaluation in echten Testbeds sehr aufwändig ist. Übertragbare Ergebnisse lassen sich dort kaum erzielen, da geeignete, große und frei konfigurierbare Testumgebungen nicht immer ohne Weiteres zur Verfügung stehen.

In Vorbereitung zur vorliegenden Arbeit wurde eine umfangreiche Simulationsumgebung (Alevin, *ALgorithms for Embedding of VIrtual Networks*) entwickelt, welche die umfassende Evaluation von Einbettungsalgorithmen ermöglicht. Mit Alevin ist es möglich, derartige Algorithmen automatisiert im Kontext zahlreicher verschiedenartiger Szenarien zu untersuchen und zu bewerten. Die bei der Konzeptionierung, der Umsetzung und dem Einsatz dieses Evaluationsframeworks gesammelten Erfahrungen wurden in [23] publiziert und sollen aufbauend auf dieser Vorveröffentlichung im Folgenden dargelegt werden. Dazu wird zunächst diskutiert, wie Einbettungsalgorithmen systematisch evaluiert werden können, bevor auf die konkrete Umsetzung der Simulationsumgebung eingegangen wird.

3.1 Genereller Simulationsablauf

Abbildung 3.1 veranschaulicht die einzelnen Phasen, in die sich der Simulationsvorgang von Einbettungsalgorithmen typischerweise untergliedern lässt. Im ersten Schritt werden zum einen die Substratnetze sowie zum anderen die VNE- bzw. NFV-RA-Einbettungsanfragen generiert. Die Erzeugung dieser Simulationsszenarien kann beispielsweise unter Zuhilfenahme von zufallsbasierten Topologiegeneratoren erfolgen. Als Zweites werden die Algorithmen ausgeführt, die die Einbettungen berechnen. Die Ergebnisse der Einbettungsberechnungsphase werden daraufhin anhand von Evaluationsmetriken numerisch bewertet. Diese Bewertungen werden schließlich zur Weiterverarbeitung sowie zur graphischen Aufbereitung gespeichert.
Im Folgenden wird auf die einzelnen Schritte dieses Simulationsworkflows eingegangen. Dazu wird zunächst diskutiert, wie innerhalb der ersten Phase des Workflows unterschiedlich komplexe Szenarien generiert werden können, um

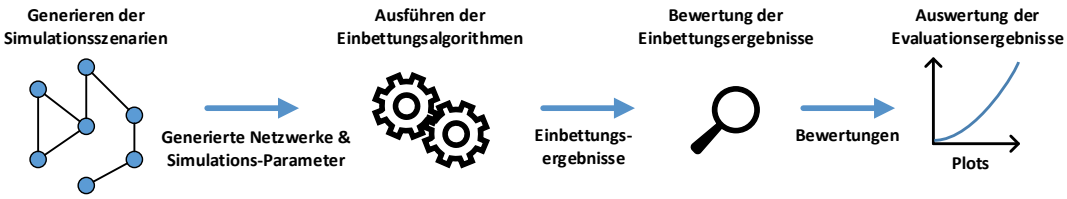


Abb. 3.1: Workflow bei der simulationsgestützten Evaluation von Einbettungsalgorithmen

die Effizienz der Algorithmen zu testen. Anschließend wird die Einbettungsphase beschrieben; in diesem Zusammenhang wird auf den Unterschied zwischen der Simulation in Online- und Offline-Szenarien eingegangen. Darüber hinaus werden verschiedene Evaluationsmetriken vorgestellt, die der numerischen Bewertung der Einbettungsergebnisse dienen.
Abschließend wird die Architektur der Alevin-Simulationsumgebung vorgestellt.

3.2 Komplexität von Simulationsszenarien

Die Komplexität der von einer Simulationsumgebung erzeugten bzw. verwendeten Einbettungsszenarien lässt sich in mehrerlei Hinsicht variieren:

Größe und Topologie der Substratnetze: Die Größe des Substratnetzes kann sich signifikant auf das Laufzeitverhalten eines Zuweisungsalgorithmus auswirken. Aus der Literatur sind viele VNE-Einbettungsalgorithmen bekannt, die zwar im Zusammenhang mit kleinen Netztopologien gute Einbettungsergebnisse liefern, aber nicht mit der Größe der Substratnetze skalieren, vgl. Kapitel 4 und 5. Aus diesem Grund ist es naheliegend, Zuweisungsalgorithmen hinsichtlich ihrer Skalierbarkeit auch in größeren Substratnetzen zu evaluieren. Neben der Anzahl der Knoten und Links eines Substratnetzes hat auch die Topologie des Netzes Auswirkungen auf das Einbettungsverhalten der Algorithmen. So ist es in der Regel einfacher, gültige Einbettungen in vollvermaschten Netzwerk-Graphen zu finden als in *dünnen* Graphen, d.h. in Graphen mit wenigen Kanten.

Größe der Einbettungsanfragen: Die Einbettungskomplexität ist abhängig von der Größe der zu verarbeitenden Anfragen. Mit der Größe der Einbettungsanfragen steigt auch der Komplexitätsgrad der Einbettung, da entsprechend viele Zuweisungsconstraints zu berücksichtigen sind.

Anzahl der Einbettungsanfragen: Neben der *Größe* der Einbettungsanfragen kann auch die *Anzahl* der zu bearbeitenden Anfragen variiert werden. Auch hier steigt die Einbettungskomplexität, da in Abhängigkeit von der Anzahl der Anfragen auch die Anzahl der zuzuweisenden virtuellen Komponenten

Algorithmus 1 Offline-Simulation eines Experiments

```
 1: function SIMULATE-OFFLINE(numberOfRequests)
 2:     allRequests ← {}
 3:     numEvents ← 0
 4:     G ← GENERATESUBSTRATENETWORK
 5:     while numEvents < numberOfRequests do
 6:         request ← GENERATENEWREQUEST
 7:         allRequests ← allRequests ∪ {request}
 8:     end while
 9:     G ← EMBED(allRequests, G)
10:     EVALUATE(G)
11: end function
```

zunimmt. Allerdings unterscheidet sich diese Variante von der ersten Variante, bei der die Größe der Anfragen variiert wird, insofern als dass es grundsätzlich einfacher ist, mehrere Anfragen mit wenigen virtuellen Knoten zuzuweisen als eine einzelne mit vielen Knoten: Im ersteren Fall müssen bei der Platzierung der virtuellen Knoten weniger Zuweisungsconstraints berücksichtigt werden als im zweiten Fall, da sie unabhängig voneinander eingebettet werden können. Im Allgemeinen ist es also viel einfacher, n Anfragen mit nur einem Knoten zuzuweisen als eine einzelne Anfrage mit n Knoten.

Flexibilitätsgrad der NFV-RA-Einbettungsanfragen: Während die Struktur der Einbettungsanfragen beim VNE-Problem durch die Topologie der virtuellen Netze unveränderlich vorgegeben ist, sind die Topologien der Netze, die im Kontext des NFV-RA-Problems eingebettet werden, flexibel. Solche Einbettungsanfragen mit hohem Flexibilitätsgrad sind grundsätzlich einfacher einzubetten als solche, die eine eher starre Struktur vorschreiben. Das liegt daran, dass ein Zuweisungsalgorithmus im ersteren Fall viel mehr Möglichkeiten für die Einbettung virtueller Knoten hat, als in Szenarien, in denen die Reihenfolge der Netzwerkfunktionen fest vorgegeben ist. Denn wenig flexible Einbettungsanfragen schränken die Handlungsmöglichkeiten des Algorithmus stärker ein, wodurch es schwieriger wird, eine gültige Lösung zu finden.

3.3 Offline- und Online-Simulation

Die Evaluation von Einbettungsalgorithmen kann entweder in Offline- oder Online-Szenarien erfolgen. In einer Offline-Simulation wird zu Beginn eine feste Anzahl von Einbettungsanfragen generiert, die von den Algorithmen eingebettet werden sollen. In einem Online-Szenario wird das Eintreffen der Anfragen in diskreten Zeitschritten simuliert. Ferner werden bereits eingebettete Netze nach einer gewissen Zeit wieder entfernt.

3.3.1 Offline-Simulation

Alg. 1 zeigt die SIMULATE-OFFLINE-Funktion, die das Vorgehen bei der Simulation eines Offline-Experiments beschreibt. Die Funktion nimmt die Anzahl der zu erzeugenden Anfragen entgegen, und zwar über den *numberOfRequests*-Parameter. Nach Generieren der Substratnetztopologie und Zuweisung der Substratnetzressourcen (Zeile 4) werden zunächst *numberOfRequests* Einbettungsanfragen erzeugt (Zeile 5). Anschließend wird der Einbettungsalgorithmus aufgerufen (Zeile 9) und abschließend das berechnete Ergebnis anhand der Evaluationsmetriken bewertet (Zeile 10).

3.3.2 Online-Simulation

In Offline-Simulationen wird das Eintreffen und Verarbeiten aller Anfragen in einem einzigen diskreten Zeitschritt simuliert. In der Regel ist diese in der Literatur sehr häufig eingesetzte Simulationsmethodik bereits ausreichend für eine aussagekräftige Evaluation von Einbettungsalgorithmen.
Je nach Anwendungsfall kann es allerdings manchmal Sinn machen, ergänzend zu den Offline-Simulationen zusätzliche Online-Simulationen durchzuführen. In einer Online-Simulation kann beispielsweise die Lebensdauer der eingebetteten virtuellen Netze berücksichtigt werden. Gerade in Cloud-Umgebungen kann davon ausgegangen werden, dass kontinuierlich Reservierungsanfragen zahlreicher Nutzer eintreffen und diese nach einer gewissen Zeit auch wieder aufgehoben werden. „Verfällt“ eine solche Reservierung, können betroffene Ressourcen des Substratnetzes für künftige Anfragen freigegeben werden. Einbettungsalgorithmen können Reorganisationsmaßnahmen veranlassen, um der Fragmentierung von Ressourcenaufteilungen entgegenzuwirken.

Jede Einbettungsanfrage trifft nach einer gewissen diskreten Zeitspanne ein. Eingebettete Netze werden nach Ablauf eines bestimmten Zeitintervalls wieder hinfällig; entsprechend reservierte Ressourcen können dann wieder freigegeben werden. In der Statistik wird ein solcher Prozess „Geburts- und Todesprozess“ genannt.

Alg. 2 beschreibt den Ablauf von Simulationen in Online-Szenarien. Hier wurde eine Exponentialverteilung für die Ankunfts- und Verweilzeiten angenommen mit der Ankunftsrate λ_{arrival} und der Verweilrate $\lambda_{\text{lifespan}}$. In jeder Iteration wird zunächst der diskrete Zeitpunkt bestimmt, zu dem das Eintreffen der nächsten Einbettungsanfrage simuliert werden soll. Dieser Zeitpunkt lässt sich nach dem Inversionsprinzip mit Hilfe der Funktion $F^{-1}(R, \lambda_{\text{arrival}})$ bestimmen. Die Funktion F^{-1} ist definiert als das Inverse der Verteilungsfunktion. Für die Verteilungsfunktion der Exponentialverteilung $F(x) = 1 - e^{-\lambda x}$ ist dies:

$$F^{-1}(R, \lambda) = -\frac{1}{\lambda} \cdot log(1 - R) \tag{3.1}$$

Algorithmus 2 Online-Evaluation eines Experiments

```
 1: function SIMULATE-ONLINE(numberOfRequests, λ_arrival, λ_lifespan)
 2:     activeRequests ← ∅
 3:     numEvents ← 0
 4:     discreteTimeSlot ← 0
 5:     G ← GENERATESUBSTRATENETWORK
 6:     while numEvents < numberOfRequests do
 7:         R ← random()
 8:         //Bestimmung des Zeitpunkts, zu dem die nächste Anfrage eintrifft:
 9:         discreteTimeSlot ← F^{-1}(R, λ_arrival)
10:         numEvents ← numEvents + 1
11:
12:         //Entfernen von Einbettungen, deren Verweilzeiten abgelaufen sind:
13:         for all (disposalTime, request) ∈ activeRequests do
14:             if disposalTime ≤ discreteTimeSlot then
15:                 G ← DISPOSE(request, G)
16:                 activeRequests ← activeRequests \ {(disposalTime, request)}
17:             end if
18:         end for
19:
20:         //Generieren einer neuen Einbettungsanfrage:
21:         R' ← random()
22:         lifespan ← F^{-1}(R', λ_lifespan)
23:         disposalTime ← discreteTimeSlot + lifespan
24:         request ← GENERATENEWREQUEST
25:         activeRequests ← activeRequests ∪ {(disposalTime, request)}
26:
27:         G ← EMBED({request}, G)
28:         EVALUATE(G)
29:     end while
30: end function
```

Als Parameter $R \in [0, 1)$ wird der Funktion F^{-1} eine Zahl übergeben, die mit Hilfe eines Zufallszahlengenerators gezogen wird, welcher gleichverteilte Zufallszahlen erzeugt (Zeile 9). Nach dem Inversionsprinzip sind die über die Funktion F^{-1} erzeugten Zufallszahlen exponentialverteilt [73, S. 87 ff.].

Bevor nun eine neue Einbettungsanfrage erzeugt wird, werden zunächst diejenigen Einbettungen entfernt, deren Verweilzeit inzwischen abgelaufen ist. Der Einbettungsalgorithmus erhält dabei die Gelegenheit, eventuell verbleibende Einbettungen zu reorganisieren, vgl. die DISPOSE-Funktion, Zeile 15.

Im Anschluss wird die Verweilzeit der neuen Einbettungsanfrage bestimmt, und zwar wiederum anhand der inversen Verteilungsfunktion, vgl. Zeile 22. Die neue Einbettungsanfrage wird daraufhin erzeugt und dem Einbettungsalgorithmus zur Bearbeitung übergeben (Zeile 27). Schließlich wird der aktuelle Einbettungszustand durch Aufruf der EVALUATE-Methode evaluiert (Zeile 28).

Die Alevin-Simulationsumgebung wurde im Rahmen dieser Arbeit um ein entsprechendes Online-Evaluationsmodul erweitert, das sowohl zur Evaluation von VNE- als auch von NFV-RA-Algorithmen verwendet werden kann.

3.4 Evaluationsmetriken

Im Folgenden werden gängige Metriken diskutiert, anhand derer sich die Güte der Einbettungsergebnisse bzw. der Einbettungsvorgang selbst numerisch bewerten lassen. Diese Metriken wurden auch für die Alevin-Simulationsumgebung implementiert.

I) **Ressourcenbezogene Metriken**

- **Akzeptanz:** Die Anzahl der erfolgreich in das Substratnetz eingebetteten virtuellen Netzwerkservices im Verhältnis zur Anzahl aller eingetroffenen Anfragen.
- **Einbettungskosten:** Der tatsächliche Ressourcenverbrauch der eingebetteten virtuellen Netzwerkservices; dieser ist üblicherweise definiert als die Summe aller CPU-/Kapazitäts-Ressourcen der Substratknoten addiert mit der Summe der Bandbreiten-Ressourcen der Substratlinks, die für die eingebetteten virtuellen Netzwerkservices reserviert wurden.
- **Einbettungsertrag:** Die Ressourcenanforderungen der virtuellen Netzwerkservices; in der Literatur wird der Einbettungsertrag meist definiert als die Summe aller CPU-/Kapazitäts-Ressourcenanforderungen addiert mit der Summe aller Bandbreiten-Ressourcenanforderungen der eingebetteten virtuellen Netze.
- **Kosten-/Nutzen-Verhältnis:** Für die Einbettung eines Netzes mit hohem Ressourcenbedarf fallen die Einbettungskosten in der Regel höher aus als bei einem Netz mit geringeren Ressourcenanforderungen. Das Kosten-/Nutzen-Verhältnis setzt die Einbettungskosten in Relation zum Einbettungsertrag. Meist wird dazu der Einbettungsertrag durch die Einbettungskosten dividiert, sodass die Werte der Metrik im Allgemeinen in einem Bereich zwischen 0 und 1 liegen. Je mehr sich das so definierte Kosten-/Nutzen-Verhältnis dem Wert 1 nähert, desto effizienter ist die vom Algorithmus bestimmte Einbettung.

II) **Quality of Service-bezogene Metriken**

- **Stress:** Anzahl der virtuellen Knoten bzw. Links, die den Substratknoten bzw. -Links im Schnitt/im Extremfall zugewiesen wurden.

- **Auslastung:** Prozentualer Ressourcenverbrauch der Knoten und/oder der Links des Substratnetzes.
- **Pfadlänge:** Die Länge der Substratpfade, die für die Links der eingebetteten virtuellen Netze reserviert wurden. Da auf jedem Substratlink, der Teil eines Pfades ist, Ressourcen reserviert werden müssen, um die Konnektivität der verbundenen virtuellen Knoten zu gewährleisten, steht diese Metrik in direkter Relation zur Einbettungskosten-Metrik.
- **Delay:** Der durchschnittliche/maximale Delay der zugewiesenen Substratpfade.
- **Jitter:** Die Übertragungsschwankungen, die auf den zugewiesenen Substratpfaden auftreten.

III) **Energieeffizienz-bezogene Metriken**
Diese Metriken sind insbesondere im Zusammenhang mit Einbettungsalgorithmen sinnvoll, deren Optimierungsziel die Maximierung der Energieeffizienz der Deployments.

- **Aktive Knoten/aktive Links:** Anzahl der Knoten bzw. Links, die für den Betrieb der eingebetteten Netzwerkservices notwendig sind.
- **Stromverbrauch:** Der Strombedarf, der für den Betrieb der virtuellen Infrastrukturen notwendig wird. Der Stromverbrauch eines Deployments kann anhand unterschiedlicher Stromverbrauchsmodelle bestimmt werden; i.d.R. ist dieser abhängig von der Auslastung der Komponenten des Substratnetzes.

IV) **Laufzeitbezogene Metriken**

- **Initialisierungs-Overhead:** Einige Einbettungsalgorithmen führen vor der eigentlichen Annahme von Einbettungsanfragen Initialisierungsaufgaben durch (beispielsweise der in Kapitel 4 vorgestellte verteilte DPVNE-Algorithmus). Diese Metrik misst die Zeit, die ein Einbettungsalgorithmus während der Initialisierungsphase benötigt.
- **Laufzeit:** Benötigte Zeit, bis der Algorithmus terminiert und entweder ein Einbettungsergebnis zurückliefert oder die Anfrage verwirft.

V) **Weitere Metriken**

- **Kommunikations-Overhead:** Anzahl der versendeten Nachrichten, die ein verteilter Einbettungsalgorithmus zur Koordination des Einbettungsvorgangs versendet.

Weitere, seltener verwendete Evaluationsmetriken sind in [23, 55] aufgeführt.

3.5 Die Alevin-Simulationsumgebung

In diesem Abschnitt wird die Alevin-Simulationsumgebung vorgestellt. Unter dem Namen Alevin wurde anfangs lediglich eine Sammlung unterschiedlicher VNE-Algorithmen veröffentlicht, die neben den eigentlichen Einbettungsalgorithmen auch eine einfache graphische Benutzeroberfläche enthielt, mit der sich manuell einzelne Einbettungsszenarien zusammenstellen ließen [42]. Dieses Tool bot allerdings keine Möglichkeit, Einbettungsalgorithmen automatisiert miteinander zu vergleichen. Aus diesem Grund wurde mit Alevin, Version 2, die Neuentwicklung eines umfangreichen VNE-Evaluationsframeworks initiiert, die in Zusammenarbeit mit Andreas Fischer und Fabian Kokot vorangetrieben wurde. Das Ergebnis wurde unter einer Open Source-Lizenz veröffentlicht [1]. Vom Autor der vorliegenden Arbeit wurde Alevin um eine erweiterte Simulations-Logik erweitert, mit der sich unter anderem auch Online-Simulationen durchführen lassen. Darüber hinaus wurde Alevin vom Autor so erweitert, dass sich auch NFV-RA-Algorithmen damit evaluieren lassen.

3.5.1 Andere VNE-Simulationsumgebungen

In der Vergangenheit wurden bereits einige andere Simulationstools veröffentlicht. Die meisten von ihnen wurden für die Evaluation eines spezifischen Einbettungsalgorithmus programmiert und implementieren nur die Topologiegeneratoren, Metriken und Einbettungsalgorithmen, die in der entsprechenden Publikation verwendet wurden. Allerdings gibt es mehrere solcher Tools, die im Laufe der Zeit weiter ausgebaut wurden. In diesem Zusammenhang sind die von Chowdhury et al. vorgestellte *ViNE-Yard*-Simulationsumgebung [37] und der von Yu et al. implementierte *VNE Simulator* [105] zu nennen. Beide Umgebungen stellen jedoch keine expliziten, dokumentierten Interfaces für die Integration neuer Szenario-Generatoren, Einbettungsalgorithmen oder Metriken bereit und werden inzwischen auch nicht mehr weiterentwickelt.
Alevin wurde von Beginn an mit dem Ziel der Erweiterbarkeit konzeptioniert. Für jeden der in Abbildung 3.1 gezeigten Simulations-Workflows werden passende Schnittstellen für die Integration neuer Verfahren und Algorithmen bereit gestellt. Im nun folgenden Abschnitt soll ein schematischer Überblick über die Architektur der Simulationsumgebung gegeben werden.

3.5.2 Architektur

Die Module der Alevin-Simulationsumgebung sind in zwei Hauptpakete gegliedert: das Kern-Paket und das Simulations-Paket. Diese sind in Abbildung 3.2 dargestellt. Am linken und rechten Rand der Abbildung sind die Schnittstellen der Module eingezeichnet, über die sich die Simulationsumgebung um neue Funktionen erweitern lässt.
Im Folgenden wird das Zusammenspiel der einzelnen Module näher beschrieben.

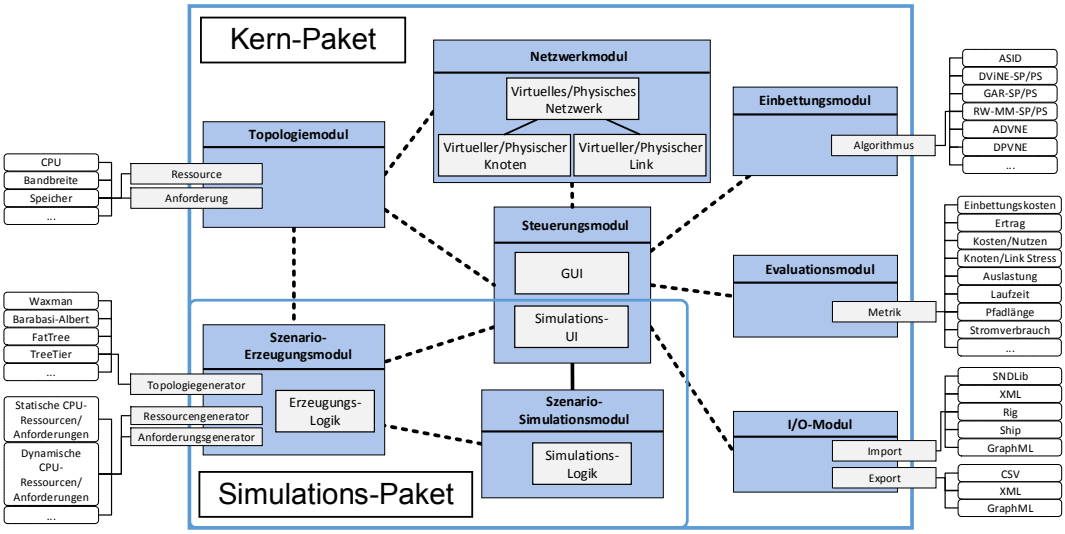


Abb. 3.2: Architektur des Alevin-Frameworks

3.5.2.1 Die Module des Kern-Pakets

Das **Netzwerkmodul** stellt Klassen zur Konstruktion von Substratnetzen und virtuellen Netzen zur Verfügung, die sich aus Substratknoten/-links bzw. aus virtuellen Knoten und Links zusammenstellen lassen.
Mit Hilfe des **Topologiemoduls** lassen sich Netztopologien generieren. Die Alevin-Simulationsumgebung stellt hier mehrere Netzwerktopologiegeneratoren zur Verfügung, die verwendet werden können, um Einbettungsalgorithmen in unterschiedlichen Szenarien zu evaluieren. Es stehen sowohl zufallsbasierte als auch nicht-zufallsbasierte Topologiegeneratoren zur Verfügung:

- Zufallsbasierte Topologiegeneratoren
  Der in der VNE-Literatur am häufigsten verwendete Ansatz zur Erzeugung zufälliger Netztopologien ist der Waxman-Generator [102]. Darüber hinaus wurden komplexere Generatoren integriert, beispielsweise ein Barabási-Albert-Generator zur Erzeugung skalenfreier Netze [10]. Skalenfreie Topologien finden sich in vielen realen Netzstrukturen wieder. Ferner sind ein FatTree- und ein ThreeTier-Generator verfügbar, mit denen sich zufällige Strukturen erzeugen lassen, die den in Rechenzentren üblichen Strukturen ähneln.

- Nicht-zufallsbasierte Topologiegeneratoren
  Das *sndlib*-Projekt hat es sich zum Ziel gesetzt, Daten über die Struktur realer Netzwerke zu sammeln und zur Verfügung zu stellen [7]. Die Alevin-Umgebung stellt eine Schnittstelle zum Einlesen dieser Daten bereit. Darüber ist es möglich, VNE-Algorithmen auch im Kontext von realen Netzwerkstrukturen zu testen, die nicht mit Hilfe eines zufallsbasierten Topologiegenerators erzeugt wurden.

Den Knoten und Links der Substratnetze lassen sich CPU- und Bandbreitenressourcen zuweisen. Ebenso können CPU- und Bandbreitenanforderungen der virtuellen Netze spezifiziert werden. Darüber hinaus stehen noch weitere Klassen für die Spezifikation von Ressourcen sowie von entsprechenden Ressourcenanforderungen zur Verfügung.
Das **Einbettungsmodul** stellt eine Schnittstelle für neue Einbettungsalgorithmen dar: Innerhalb des Einbettungsmoduls sind Interfaces spezifiziert, die von allen Einbettungsalgorithmen eingehalten werden müssen. In Alevin sind bereits verschiedene Einbettungsalgorithmen, die in der Literatur vorgestellt wurden, implementiert. Auf diese kann zurückgegriffen werden, um neuartige Algorithmen in unterschiedlichen Szenarien hinsichtlich verschiedener Evaluationsmetriken zu vergleichen. Da viele VNE-Einbettungsalgorithmen auf dem Konzept der linearen Programmierung basieren, wurde als Bestandteil des Einbettungsmoduls zudem ein *Solver* für lineare Programme, das GNU Linear Programming Kit [4], integriert. Auf diesen Solver können die Einbettungsalgorithmen zurückgreifen.
Das **Evaluationsmodul** enthält Komponenten für die Auswertung berechneter Einbettungsergebnisse. Hier wurden bereits verschiedenste Evaluationsmetriken implementiert, die zur numerischen Bewertung der Einbettungen verwendet werden können (siehe Abschnitt 3.4). Wie beim Einbettungsmodul wurde bei der Konzeptionierung auch hier großer Wert auf eine einfache Erweiterbarkeit des Evaluationsmoduls gelegt.
Über das **I/O-Modul** lassen sich erzeugte Netze exportieren und importieren. Ferner können die anhand der Evaluationsmetriken berechneten Evaluationsergebnisse zur graphischen Aufbereitung abgespeichert werden.
Das **Steuerungsmodul** stellt Schnittstellen zur Verfügung, über die sich Evaluationsszenarien spezifizieren, ausführen und evaluieren lassen. Dieses Modul koordiniert daher alle anderen Module. Die graphische Bedienoberfläche (**GUI**) erlaubt das manuelle Zusammenstellen einzelner Szenarien, während sich über die Simulationsbedienschnittstelle (**Simulations-UI**) die automatisierte Generierung von sehr vielen verschiedenartigen Szenarien einstellen lässt. Die Simulations-UI ist Teil des Simulations-Pakets, dessen Aufbau im Folgenden beschrieben wird.

3.5.2.2 Die Module des Simulations-Pakets

Die Simulationslogik dieses Pakets wurde federführend vom Autor der vorliegenden Arbeit implementiert. Es enthält die folgenden Komponenten:
Mit Hilfe der *Simulations-UI* des **Steuerungsmoduls** können automatisiert unterschiedlichste Szenarien generiert werden. Dazu kann auf verschiedene Topologiegeneratoren des Topologiemoduls zurückgegriffen werden. Mit Hilfe des **Szenario-Erzeugungsmoduls** können Ressourcen bzw. Ressourcenanforderungen der Substratnetze und der virtuellen Netze spezifiziert werden. Die *Erzeugungslogik* erzeugt unter Rückgriff auf Topologiegeneratoren und

Ressourcen- und Ressourcenanforderungs-Generatoren vollständige Substratnetze bzw. virtuelle Netze.
Ein wesentlicher Schwerpunkt bei der Konzeptionierung und der Umsetzung des Simulations-Pakets lag auf der Schaffung einfach zu erweiternder Module. Neue Ressourcen- und Ressourcenanforderungs-Generatoren können daher ebenso einfach integriert werden wie neuartige Netzwerkgeneratoren.
Im Rahmen der vorliegenden Arbeit wurde eine Online-Simulationslogik konzipiert und umgesetzt, die ebenfalls Teil des Simulations-Pakets ist. Dies wurde entsprechend der in Abschnitt 3.3 aufgeführten Überlegungen umgesetzt. Mit der Online-Simulationslogik ist es möglich, Zuweisungsalgorithmen in Szenarien zu evaluieren, in denen Einbettungsanfragen entsprechend einer vorgegebenen Wahrscheinlichkeitsverteilung eintreffen. Zudem kann eine weitere, davon unabhängige Wahrscheinlichkeitsverteilung festgelegt werden, nach der Alevin die Verweilzeiten der bereits eingebetteten virtuellen Netzwerke bestimmt. Das ermöglicht die Simulation von Szenarien, in denen kontinuierlich neu eintreffende Einbettungsanfragen bearbeitet werden müssen. Dies ist insbesondere in NFV- und Cloud-Szenarien der Fall.
Darüber hinaus wurde das Szenario-Erzeugungsmodul um Generatoren erweitert, mit denen die automatisierte, zufallsbasierte Erzeugung von NFV-Einbettungsanfragen möglich ist. Dabei lässt sich der Flexibilitätsgrad der zu erzeugenden NFV-Anfragen einstellen: Es kann also festgelegt werden, wie flexibel sich die Netzwerkfunktionen der zufällig generierten Netzwerkservices verketten lassen. Insgesamt ist es damit möglich, NFV-RA-Ansätze mit Alevin genauso umfassend zu evaluieren wie VNE-Algorithmen.

3.5.3 Laufzeitmessungen

Um zu veranschaulichen, dass sich Alevin zur Evaluation von Einbettungsalgorithmen eignet, wurden Laufzeit- und Speicherverbrauchsmessungen durchgeführt: Abbildung 3.3 zeigt Laufzeitmessungen verschiedener VNE-Einbettungsalgorithmen, die in die Alevin-Simulationsumgebung integriert und damit dann im Rahmen unterschiedlicher Szenarien evaluiert wurden. Tabelle 3.1 zeigt den gemessenen Speicherverbrauch während dieser Simulationen. Konkret wurden hier in jedem Experiment drei zufällige virtuelle Netze mit jeweils zehn virtuellen Knoten erzeugt, die in ein ebenfalls zufällig erzeugtes Substratnetz einzubetten waren. Die Topologien der Netze wurden mit Hilfe des Waxman-Generators erzeugt. Abbildung 3.3 stellt nun dar, wie sich das Laufzeitverhalten der Algorithmen in Abhängigkeit von der Größe der erzeugten Substratnetze verändert. Die Laufzeitmessungen sind vergleichbar mit den von den Autoren der jeweiligen Algorithmen publizierten Messungen. Als Referenz seien hier beispielsweise die von Cheng et al. entworfenen RW-MM-PS/RW-MM-SP-Algorithmen genannt [34]. Cheng et al. bemessen die Laufzeit ihrer Algorithmen im Zusammenhang mit unterschiedlichen Simulationsszenarien für Substratnetze mit 100 Knoten in einem Rahmen von 5 bis 30 Minuten.

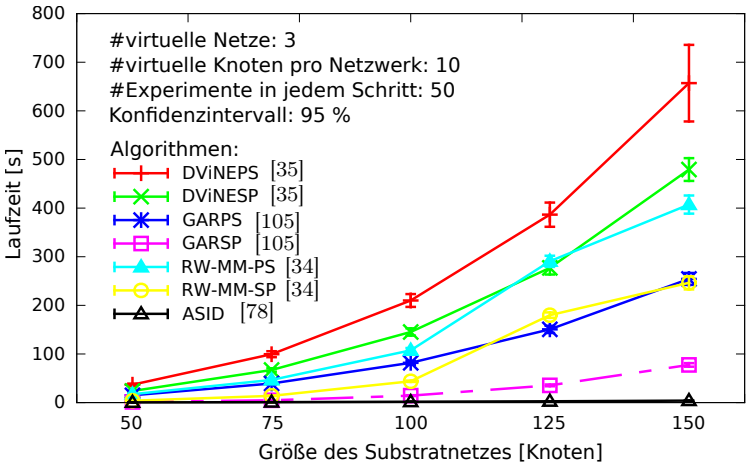


Abb. 3.3: Das mit Hilfe der Alevin-Simulationsumgebung gemessene Laufzeitverhalten unterschiedlicher Einbettungsalgorithmen

#Knoten	Durchschnitt	Maximal
50	116 MB	181 MB
75	175 MB	268 MB
100	232 MB	359 MB
125	267 MB	399 MB
150	388 MB	826 MB

Tabelle 3.1: Hauptspeicherverbrauch von Alevin während der Simulation

Der Speicherverbrauch für die Ausführung der Algorithmen in den einzelnen Szenarien bewegt sich im Durchschnitt im Bereich von einigen hundert Megabytes. Im Extremfall wurde ein Wert von 826 Megabytes gemessen.
Insgesamt zeigt sich, dass Evaluationen von Einbettungsalgorithmen mit Alevin innerhalb vernünftiger Simulationszeiten durchführbar sind und die Laufzeitmessungen durchaus mit den in der Literatur angegebenen Werten vergleichbar sind.

3.6 Zusammenfassung

Das Verhalten von Einbettungsalgorithmen wird üblicherweise durch Simulation vieler unterschiedlicher, meist zufällig generierter Szenarien analysiert. Die berechneten Einbettungsergebnisse werden anhand von Evaluationsmetriken numerisch bewertet. Die Simulationen können entweder offline oder online durchgeführt werden. In Offline-Szenarien wird das Eintreffen und Verarbeiten der Anfragen in einem einzigen diskreten Zeitschritt durchgeführt, während das Eintreffen von Einbettungsanfragen in Online-Szenarien

als ein stochastischer Prozess simuliert wird. In diesem Zusammenhang wurden Funktionsumfang und Architektur der Alevin-Simulationsumgebung vorgestellt und gezeigt, wie sich Einbettungsalgorithmen damit automatisiert evaluieren und miteinander vergleichen lassen.

Die sich nun anschließenden Hauptkapitel der Arbeit befassen sich mit dem effizienten Deployment von virtuellen Netzen in Cloud- und NFV-Szenarien. Anhand der in Abbildung 3.3 und Tabelle 3.1 gezeigten Simulationsergebnisse lässt sich bereits erkennen, dass viele der untersuchten Einbettungsalgorithmen nur schlecht mit der Größe der Substratnetze skalieren. Für die Einbettung von drei virtuellen Netzen in ein Substratnetz mit 150 Knoten benötigten einige Verfahren schon wesentlich länger als 400 Sekunden. Im folgenden Kapitel wird ein neuer Algorithmus vorgestellt, der das Deployment virtueller Netze auch in wesentlich größeren (Cloud-)Infrastrukturen möglich macht. Die aufwändigen Berechnungen, die damit verbunden sind, können mit diesem neuen Verfahren auf gleich mehrere Rechner verteilt werden, die parallel an der Bearbeitung der Einbettungen arbeiten.

4 Verteiltes und skalierbares Einbetten virtueller Netze

In diesem Kapitel wird untersucht, wie die komplexen Berechnungen, die mit der Einbettung virtueller Netze verbundenen sind, durch den Einsatz verteilter Rechner parallelisiert durchgeführt werden können. Dabei wird zunächst diskutiert, in welchen Szenarien ein solches verteiltes Vorgehen überhaupt sinnvoll ist und warum andere, nicht-verteilte Ansätze hier nur wenig geeignet sind. Ferner wird beschrieben, welche Herausforderungen sich durch das verteilte Deployment der virtuellen Netze ergeben und wie sich diese durch den hier vorgestellten neuen Ansatz effizient lösen lassen. Dieser Ansatz heißt DPVNE, wobei DPVNE ein Akronym für *Distributed and Parallel Virtual Network Embedding*) ist.
DPVNE ist ein verteilter VNE-Einbettungsalgorithmus. Im Gegensatz zu vielen bisher vorgestellten Verfahren ist DPVNE also nicht von einem einzelnen Rechner abhängig, der allein für die Durchführung dieser Berechnungen zuständig ist. Stattdessen koordiniert DPVNE mehrere verteilt arbeitende Rechner, die parallel an der Einbettung der virtuellen Netze arbeiten. Diese Rechner werden im Folgenden Einbettungsknoten genannt.
Das ist insbesondere dann sinnvoll, wenn viele Einbettungsanfragen verarbeitet werden müssen – beispielsweise in großen Cloud-Umgebungen. In einem solchen Szenario würde die sequentielle Verarbeitung der Einbettungsanfragen auf einem einzelnen, zentralen Einbettungsknoten aufgrund der Tatsache, dass der Einbettungsvorgang mit aufwändigen Berechnungsoperationen verbunden ist, zu Kapazitätsengpässen auf diesem Knoten führen. Treffen kontinuierlich neue Einbettungsanfragen ein, kommt der zentrale Einbettungsknoten mit der Verarbeitung nicht mehr nach, was die Einbettung weiterer Netze immer mehr verzögert.
Bei verteilten VNE-Verfahren führen dagegen gleich mehrere Einbettungsknoten die Berechnungen parallel durch. Das beschleunigt den Einbettungsvorgang in dem beschriebenen Szenarium deutlich, allerdings ist ein solches verteiltes Vorgehen in mehrerlei Hinsicht komplexer als das Vorgehen eines nicht-verteilten Ansatzes:

I) Der zentrale Einbettungsknoten nicht-verteilter Verfahren weiß zu jeder Zeit, welche Ressourcen innerhalb der Netzwerkinfrastruktur reserviert wurden. Dies ist bei verteilten Verfahren nicht der Fall. Die Einbettungsknoten arbeiten hier parallel am Deployment der virtuellen Netze, was dazu führt, dass jeder Einbettungsknoten nur partiell über die aktuelle Ressourcenverfügbarkeit informiert ist.

II) Der verteilte Einbettungsvorgang muss koordiniert werden. Die Einbettungsknoten müssen sich also über ein Kommunikationsprotokoll austauschen. Diese Koordination ist bei nicht-verteilten Verfahren nicht nötig.

III) Ressourcen des Substratnetzes dürfen nicht mehrfach reserviert werden. Bei einem nicht-verteilten Ansatz lässt sich das sehr einfach vermeiden. Bei einem verteilten Verfahren sollen solche Inkonsistenzen bei der Ressourcenverteilung durch Locking-Regeln ausgeschlossen werden. Dies ist wichtig, wenn mehrere Einbettungsknoten parallel Reservierungen für dieselben Ressourcen des Substratnetzes vornehmen können.

Kernidee hinter dem DPVNE-Algorithmus ist es, das Substratnetz in hierarchisch organisierte Partitionen einzuteilen. Jede dieser Partitionen wird von einem verteilten Einbettungsknoten verwaltet, der Einbettungsanfragen innerhalb der Partition vornimmt und sich mit anderen Einbettungsknoten austauscht.

Ein interessanter Aspekt von DPVNE ist, dass DPVNE in der Lage ist, verschiedene, ursprünglich nicht-verteilte VNE-Verfahren auf mehreren Einbettungsknoten verteilt zu betreiben. DPVNE verwendet diese nicht-verteilten VNE-Algorithmen, um die eigentlichen Einbettungen innerhalb der Partitionen des Substratnetzes zu berechnen. DPVNE versucht, die Einbettungen in möglichst kleinen Partitionen des Gesamtnetzes vorzunehmen, wodurch sich die Problemgröße des VNE-Problems reduziert; dadurch sind die Einbettungsalgorithmen in der Lage, die Einbettung *mehrerer* Netze nicht nur parallel, sondern die *einzelner* Netze oftmals auch schneller zu berechnen.

In der Literatur wurde nach bestem Wissen des Autors bislang nur ein einziger vergleichbarer, verteilter Ansatz vorgestellt, nämlich ADVNE (*Agent-based Distributed Virtual Network Embedding*) [64]. DPVNE unterscheidet sich wie folgt von ADVNE:

I) Im Gegensatz zu ADVNE ist DPVNE in der Lage, die Einbettungen der virtuellen Netze mit signifikant geringerem Kommunikations-Overhead vorzunehmen. Durch die hierarchische Organisation können die Einbettungsknoten bei DPVNE effizienter koordiniert werden, sodass wesentlich weniger Kommunikationsnachrichten ausgetauscht werden müssen.

II) Darüber hinaus gelingt es DPVNE, die virtuellen Netze viel effizienter innerhalb der Infrastruktur des Substratnetzes einzubetten.

III) Anders als ADVNE ist DPVNE zudem ein generisches Einbettungs-Framework, das die Ausführung unterschiedlicher VNE-Algorithmen in einem verteilten Umfeld möglich macht.

DPVNE ist also ein **generisches Framework**, das unterschiedliche, ursprünglich nicht-verteilte VNE-Einbettungsalgorithmen auf mehreren **verteilten** Einbettungsknoten betreiben kann. Dadurch ist es möglich, Einbettungsanfragen **parallel** zu verarbeiten.
Dieses Kapitel beschreibt die Funktionsweise des DPVNE-Algorithmus. Zudem werden die Ergebnisse umfangreicher Experimente diskutiert, die zeigen, dass die Einbettung virtueller Netze mit DPVNE sehr effizient auf mehrere Einbettungsknoten verteilt werden kann. Außerdem wird gezeigt, dass DPVNE dem verteilten ADVNE-Algorithmus in mehrerlei Hinsicht überlegen ist.
Das Kapitel baut auf den in [21, 16, 22] vorgestellten Ideen auf.

4.1 Abgrenzung zu verwandten Arbeiten

Im Folgenden wird DPVNE in Bezug gesetzt zu verschiedenen aus der Literatur bekannten VNE-Algorithmen. Zu *nicht-verteilten* Verfahren, zu *Intra- und Inter-Infrastruktur*-Algorithmen sowie zu anderen *verteilten* Ansätzen.

Nicht-verteilte Einbettungsalgorithmen

Die meisten VNE-Algorithmen arbeiten nicht-verteilt [55]. Das heißt, sie verwenden zur Berechnung des Deployments einen einzelnen, externen Einbettungsknoten, der den Gesamtzustand des Substratnetzes, d.h. insbesondere die vollständige Netztopologie, die zur Verfügung stehenden Netzwerkressourcen, sowie die bereits zugewiesenen virtuellen Netze kennt. Dieser einzelne Rechner ist allein dafür verantwortlich, das Deployment aller virtuellen Netze durchzuführen.

Ein in Hinblick auf DPVNE bemerkenswertes nicht-verteiltes Verfahren ist das von Fürst et al. vorgeschlagene MetaTree-Framework, welches ebenfalls auf einer hierarchischen Partitionierung des Substratnetzes aufbaut [57]. Die Idee dabei ist, die Einbettungen nicht auf Basis des Gesamtnetzes zu berechnen, sondern innerhalb kleinerer Partitionen. Da sich hierdurch die Problemgröße reduziert, kann der Berechnungsvorgang beschleunigt werden. DPVNE baut ebenfalls auf einer hierarchischen Partitionierung des Substratnetzes auf. Der Beschleunigungseffekt, der sich allein schon aufgrund dieser Partitionierung ergibt, kann hier ebenfalls nachgewiesen werden. Anders als bei DPVNE handelt es sich bei MetaTree allerdings nicht um einen verteilten Ansatz; eine parallele Verarbeitung der Einbettungsanfragen ist damit nicht umsetzbar.

In Abgrenzung zu diesen nicht-verteilten Verfahren sieht DPVNE mehrere verteilt arbeitende Einbettungsknoten vor, die bei der Einbettung der virtuellen Netze kooperieren.

Intra-Infrastruktur- und Inter-Infrastruktur-Algorithmen

Bei den meisten VNE-Algorithmen handelt es sich um sogenannte Intra-Infrastruktur-Algorithmen. Intra-Infrastruktur-Algorithmen betten die virtuellen Netze in ein einzelnes, zusammengehörendes Substratnetz ein. Im Gegensatz dazu arbeiten Inter-Infrastruktur-Ansätze mit mehreren, voneinander unabhängigen Substratnetzen, die gegebenenfalls auch von unterschiedlichen Netzwerkprovidern betrieben werden. In der Literatur wurden mehrere, auch verteilte, Inter-Infrastrukturansätze vorgestellt, die sich insbesondere mit der Verteilung der virtuellen Netzwerkanfragen auf die verschiedenen unabhängigen Substratnetze befassen [103, 74, 75, 81, 65, 36]. In Abgrenzung zu diesen Verfahren handelt es sich bei DPVNE um einen Intra-Infrastrukturansatz, der die verteilte Berechnung von Einbettungen innerhalb eines zusammenhängenden, großen Substratnetzes ermöglicht.

Verteilte Intra-Infrastruktur-Algorithmen

In der Literatur wurden bislang zwei weitere, verteilte Intra-Infrastruktur-Einbettungsalgorithmen vorgestellt:

Ghatar et al. stellen einen Ansatz vor, der ähnlich zum DPVNE-Framework das Substratnetz in mehrere, hierarchisch angeordnete Partitionen unterteilt [58]. Im Unterschied zu DPVNE wird die Einbettung jedoch nur zum Teil dezentral berechnet. Für die Einbettung jedes virtuellen Netzes muss die gesamte Substratnetz-Infrastruktur blockiert werden, was eine parallele Verarbeitung der Einbettungsanfragen unmöglich macht. Idee hinter diesem Ansatz ist, jedes virtuelle Netz gleich mehrfach innerhalb der Substratnetzinfrastruktur einzubetten und anschließend lediglich die effizienteste dieser Einbettungslösungen anzuwenden. Im Unterschied zu dem von Ghatar et al. vorgestellten Ansatz handelt es sich bei DPVNE um ein generisches Framework, welches zur verteilten Ausführung beliebiger, ursprünglich nichtverteilter Einbettungsalgorithmen verwendet werden kann.

Houidi et al. stellen einen weiteren verteilten Ansatz vor (ADVNE), bei dem viele Einbettungsknoten gemeinsam das Einbettungsproblem lösen [64]. Mehrere Einbettungsknoten, sogenannte Agenten, betten jeweils unterschiedliche Teile eines virtuellen Netzes ein. Zur Koordination des Einbettungsprozesses tauschen die Agenten Synchronisationsnachrichten untereinander aus.

Nach bestem Wissen des Autors ist ADVNE neben DPVNE der bislang einzige verteilt arbeitende Intra-Infrastruktur-Ansatz, der in der Lage ist, die Einbettung mehrerer virtueller Netze parallel vorzunehmen. In diesem Kapitel wird DPVNE daher mit dem ADVNE-Algorithmus verglichen.

4.2 Der DPVNE-Algorithmus: Ein verteilter Einbettungsalgorithmus

In diesem Abschnitt wird erörtert, wie sich der Deployment-Vorgang mit DPVNE durch Auslagerung der Berechnungen auf mehrere dezentrale Rechner parallelisieren lässt
Bevor die ersten Einbettungsanfragen bearbeitet werden können, müssen die verteilten Einbettungsknoten initialisiert werden. Die Funktionsweise des DPVNE-Algorithmus wird daher in zwei Phasen beschrieben:
In der Initialisierungsphase des Algorithmus wird zunächst die Einteilung des Substratnetzes in Partitionen vorgenommen und jeder Partition ein Einbettungsknoten zugewiesen. In der Einbettungsphase werden dann die eigentlichen Einbettungsanfragen verarbeitet. In diesem Zusammenhang wird die Funktionsweise der Einbettungsknoten beschrieben. Zudem wird das Kommunikationsprotokoll vorgestellt, welches die verteilten Knoten zur Koordination und zur Synchronisation nutzen. Außerdem wird das Locking-Konzept von DPVNE beschrieben, über das sich die Konsistenz der verteilten Zustandsinformationen sicherstellen lässt.
Im Folgenden werden die genannten zwei Phasen beschrieben.

4.2.1 Die Initialisierungsphase

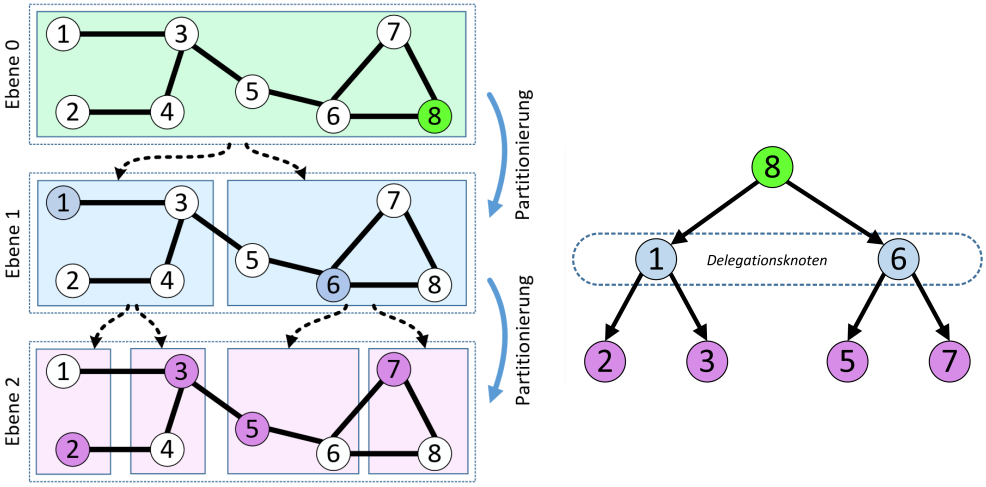
In der Initialisierungsphase wird das Substratnetz in hierarchisch organisierte Partitionen eingeteilt. Daraufhin wird für jede dieser Partitionen ein Einbettungsknoten bestimmt, dessen Aufgabe es ist, innerhalb der jeweils zugeordneten Partition Einbettungen virtueller Netze zu berechnen.

4.2.1.1 Hierarchische Partitionierung des Substratnetzes

In diesem Abschnitt wird zuerst das allgemeine Vorgehen beschrieben, nach dem das Substratnetz in mehrere, hierarchisch gegliederte Partitionen aufgeteilt wird. Im Anschluss werden dann Eigenschaften geeigneter Verfahren diskutiert, die sich für eine solche Partitionierung des Substratnetzes eignen.

Generelles Vorgehen

Das Substratnetz wird zunächst mit Hilfe eines Partitionierungsalgorithmus in kleinere Netzwerkpartitionen unterteilt. Die kleineren Partitionen werden daraufhin wiederum in noch kleinere Einheiten partitioniert. Dieser Vorgang wird nun sukzessive wiederholt, sodass eine Partitions-Hierarchie entsteht. Wichtig dabei ist, dass sich die so gebildeten Partitionen *derselben Ebene* nicht überschneiden, d.h., dass sich jeder Substratknoten *genau einer* und jeder Substratlink *maximal einer* dieser Partitionen zuordnen lässt.
Eine solche Partitionshierarchie ist in Abbildung 4.1a gezeigt: Im ersten Schritt wurde das Gesamtnetz (dargestellt in Ebene 0) in zwei kleinere, überschnei-



(a) Überschneidungsfreie Partitionierung des Substratnetzes

(b) Hierarchie der Einbettungsknoten

Abb. 4.1: Hierarchische, überschneidungsfreie Partitionierung des Substratnetzes

dungsfreie Partitionen unterteilt (vgl. Ebene 1). Im zweiten Schritt wurde jede dieser beiden Partitionen wiederum in zwei überschneidungsfreie Partitionen aufgeteilt (vgl. Ebene 2).
Die Grundidee hinter dem DPVNE-Ansatz ist nun, die Einbettungsalgorithmen nicht innerhalb des großen Gesamt-Substratnetzes auszuführen, sondern in geeigneten kleineren Partitionen. Ein solches Vorgehen ist insbesondere in solchen Szenarien von Vorteil, in denen die einzubettenden virtuellen Netze in der Regel kleiner sind als das eigentliche Substratnetz – beispielsweise in Cloud-Szenarien. Die verteilte Berechnung von Einbettungen innerhalb dieser hierarchischen Partitionierung hat dann zwei Vorteile:

I) Viele Berechnungen können auf Partitionen des Substratnetzes verlagert werden, die kleiner sind als das vollständige Substratnetz. Als Folge dessen lassen sich viele Einbettungen schneller berechnen.

II) Die Einbettung mehrerer virtueller Netze kann in voneinander unabhängigen Teilbereichen des Netzes parallel durchgeführt werden, wodurch sich der Einbettungsvorgang beschleunigt.

Diskussion geeigneter Partitionierungsverfahren

Zur Einteilung des Substratnetzes in Partitionen kann im Prinzip auf unterschiedliche Partitionierungsverfahren zurückgegriffen werden. Das Partitionieren von Graphen ist ein hinlänglich bekanntes Optimierungsproblem, für welches bereits verschiedenste Ansätze vorgestellt wurden. Graph-Partitionierungsalgorithmen sind eng verwandt zu Graph-Clusteringverfahren.

Sie unterscheiden sich dahingehend voneinander, dass ein Partitionierungsalgorithmus Graphen in nicht-überlappende Mengen aufteilt. Dagegen werden bei einem Clustering die Knoten oder Kanten eines Graphen potentiell mehreren Ergebnismengen zugeordnet. Ein detaillierter Überblick zu diversen Clustering- und Partitionierungsmethoden findet sich im Physics Reports Journal [56].

Grundsätzlich eignen sich zur Partitionierung von Computernetzen insbesondere solche Verfahren, die eng zusammenhängende Netzstrukturen erkennen: Topologien realer Netze bestehen gewöhnlicherweise aus Verbünden von Verbünden, die sich wiederum aus Verbünden zusammensetzen, etc.. Diese Hierarchie-Eigenschaft gilt nicht nur für Computernetze, sondern lässt sich ganz allgemein auch in sehr vielen anderen Netzen wie in Forschungsnetzwerken, in Unternehmensstrukturen, bei Flughafen-Verbindungen, sowie in der Topologie von Stromnetzen beobachten [56].

Im Folgenden wird ein quantitatives Kriterium diskutiert, mit dem sich beurteilen lässt, wie gut eine gegebene Aufteilung eines Graphen in Partitionen ist.

Mit Hilfe einer Qualitätsfunktion lässt sich jeder Partition ein numerischer Wert zuordnen, anhand der sich die Güte einer Partitionierung numerisch beurteilen lässt. Eine der populärsten Qualitätsfunktionen für die Partitionierung von Graphen ist das von Newman und Girvan vorgestellte *Modularitätsmaß* [90], welches im Folgenden definiert wird.

Sei $\mathbf{e}$ die symmetrische $k \times k$ Matrix, deren Element e_{ij} die relative Anzahl der partitionsübergreifenden Kanten repräsentiert, die die i-te Partition mit der j-ten Partition im ursprünglichen Graphen verbinden. Auf der Hauptdiagonalen der Matrix sind damit die Kanten aufgezählt, die jeweils Knoten derselben Partition miteinander verbinden. Nun wird die Spur der Matrix, also die Summe der Hauptdiagonalelemente, berechnet:

$$\text{Spur}(\mathbf{e}) = \sum_{i=1}^{k} e_{ii} \tag{4.1}$$

Bei einer „guten“ Partitionierung sollte die Spur der Matrix einen möglichst großen Wert nahe 1 ergeben, da in diesem Fall die Anzahl der partitionsinternen Kanten groß ist.

Die Spur als alleiniges Bewertungskriterium reicht aber nicht aus, da zum Beispiel das von einem Algorithmus berechnete Ergebnis, der einen Graphen in nur eine einzige Partition zerlegt, mit einem maximalen Wert von $\text{Spur}(\mathbf{e}) = 1$ bewertet wird. Daher wird als Teil der Berechnung der Modularität die Anzahl der Kanten berücksichtigt, die die Knoten einer Partition auch mit den Knoten anderer Partitionen verbinden. Für die i-te Partition ist dies $a_i = \sum_{j=1}^{k} e_{ij}$.

Das Modularitätsmaß wird dann wie folgt definiert:

$$Q = \sum_{i=1}^{k} (e_{ii} - a_i^2) = \text{Spur}(\mathbf{e}) - ||\mathbf{e}^2|| \tag{4.2}$$

Dabei ist $||\mathbf{e}^2||$ die Summe aller quadrierten Elemente der Matrix $\mathbf{e}$. Die Formel liefert Werte zwischen 0 und 1. Größere Werte drücken eine höhere Modularität der Partitionierung und damit ein besseres Partitionierungsergebnis aus als kleinere Werte nahe 0 [90]. Eine sinnvolle Erkennung zusammengehörender Netzpartitionen ist für einen Partitionierungsalgorithmus grundsätzlich einfacher zu bewerkstelligen, wenn die Topologien der Substratnetze hierarchisch ausgelegt sind und sich strukturelle Bereiche des Netzes voneinander abheben.

Im Zusammenhang mit DPVNE eignen sich insbesondere solche Partitionierungsverfahren, deren Optimierungskriterium es ist, möglichst gut miteinander vernetzte Knoten in Partitionen zusammenzufassen. Die Wahrscheinlichkeit, dass der Einbettungsversuch innerhalb einer solchen, gut vernetzten Partition gelingt, ist aufgrund der topologischen Eigenschaften des Partitionsgraphen höher als in nur wenig vernetzten Strukturen. In diesem Zusammenhang ist das Modularitätsmaß eine geeignete Qualitätsfunktion, anhand derer sich die Eignung einer Partitionierung auch quantitativ bewerten lässt. Um die Parallelisierbarkeit der Berechnungen zu erhöhen, sollten die Partitionen derselben Hierarchieebene zudem ungefähr gleich groß sein.

Ein solches Partitionierungsverfahren, welches versucht, innerhalb einer Graphstruktur ungefähr gleich große, gut vernetzte Partitionen zu identifizieren, wurde von Karypis et al. vorgestellt [72]. Dieses Verfahren wurde für die Implementation des DPVNE-Algorithmus gewählt.

4.2.1.2 Initialisierung verteilter Einbettungsknoten

Bei DPVNE werden die eigentlichen Einbettungs-Berechnungen von verteilten Einbettungsknoten durchgeführt. Jeder Partition wird dazu ein Einbettungsknoten zugewiesen, der das Deployment virtueller Netze innerhalb des von ihm verwalteten Partitionsbereichs unter Zuhilfenahme eines VNE-Algorithmus berechnet und durchführt. Abbildung 4.1b zeigt die Zuordnung von Einbettungsknoten zu den einzelnen Partitionen. Für das in Abbildung 4.1a dargestellte Szenario wurden die Knoten 2, 3, 5 und 7 als Einbettungsknoten für die Partitionen der Ebene 2 ausgewählt, für Ebene 1 die Knoten 1 und 6 und für Ebene 0 der Knoten 8.
Anschließend werden die Einbettungsknoten bestimmt, die zunächst alle neu eintreffenden Einbettungsanfragen entgegennehmen. Dies erfolgt durch Auswahl einer Hierarchieebene, der sogenannten *Delegationsebene*. Knoten dieser Ebene, die *Delegationsknoten*, delegieren die Bearbeitung eintref-

fender Anfragen an die ihnen untergeordneten Einbettungsknoten weiter und arbeiten unabhängig voneinander. Die untergeordneten Knoten werden *Dienstknoten* genannt. Durch die Festlegung der Delegationsebene kann das Substratnetz daher in mehrere *autonome Regionen* aufgeteilt werden. Dies ist in Abbildung 4.1b veranschaulicht: Hier wurde Ebene 1 als Delegationsebene gewählt, sodass die Einbettungsknoten 1 und 6 die Rolle der Delegationsknoten einnehmen.

Jedem Einbettungsknoten wird eine der folgenden Rollen zugewiesen:

I) Delegationsknoten
Delegationsknoten nehmen innerhalb der autonomen Region des Substratnetzes eine Anführerrolle ein. Sie nehmen neu eintreffende Einbettungsanfragen entgegen und koordinieren die Einbettung mit Hilfe der ihnen untergeordneten Einbettungsknoten. Das Verteilen der Anfragen an die Delegationsknoten geschieht über einen Load Balancer, der die aktuelle Auslastung der Delegationsknoten berücksichtigt und eine faire Verteilung der Arbeitslast sicherstellt.

II) Dienstknoten
Dienstknoten sind die Einbettungsknoten, die den einzelnen Delegationsknoten untergeordnet sind. Die Delegationsknoten können eingehende Einbettungsanfragen an die ihnen zugewiesenen Dienstknoten weiterdelegieren. Dienstknoten versuchen dann, innerhalb ihres Partitionsbereichs eine geeignete Einbettung vorzunehmen.

III) Übergeordnete Knoten
Diese Knoten sind den Delegationsknoten übergeordnet. Sie werden nur dann konsultiert, wenn eine Einbettung innerhalb der einem Delegationsknoten zugeordneten Partitionsbereich nicht möglich ist.

Diskussion: Netz-interne und netz-externe Einbettungsknoten

Jeder Partition muss ein Einbettungsknoten zugewiesen werden, der Einbettungsanfragen stellvertretend für diese Partition entgegennimmt und diese dann mit Hilfe eines Einbettungsalgorithmus verarbeitet. Als Einbettungsknoten können netz-interne oder netz-externe Knoten gewählt werden. Im ersteren Fall werden Substratknoten ausgewählt, die in der Lage sind, mit Hilfe eines VNE-Algorithmus Einbettungen zu berechnen. Im zweiten Fall werden dazu zusätzliche, externe Recheneinheiten verwendet.

Umsetzungsmöglichkeit 1: Netz-interne Einbettungsknoten
Vorteil dieses Ansatzes ist, dass das Substratnetz selbständig in der Lage ist, Einbettungen vorzunehmen. Es müssen keine zusätzlichen Rechner bereitgestellt werden. Netz-interne Einbettungsknoten werden auch für das von Houidi et al. vorgestellte verteilte ADVNE-Verfahren verwendet [64]. Ein Nachteil ist,

dass zur Ausführung der verteilten Berechnungen auf Ressourcen des Substratnetzes zurückgegriffen werden muss, die dann nicht mehr ausschließlich zum Betrieb der eingebetteten virtuellen Netze zur Verfügung stehen. Zum einen werden CPU-Ressourcen für die Durchführung von Einbettungsberechnungen benötigt; zum anderen muss auch berücksichtigt werden, dass die netzinternen Einbettungsknoten Kommunikationsnachrichten untereinander austauschen. Daher müssen für den Einbettungsvorgang selbst sowohl CPU- als auch Bandbreitenressourcen innerhalb des Substratnetzes reserviert werden, die für die Platzierung der virtuellen Netze dann nicht mehr zur Verfügung stehen.
Ein möglicher Ansatz zur Bestimmung netz-interner Einbettungsknoten ist, ausgehend von der obersten Ebene der Partitionshierarchie rekursiv den jeweils rechenstärksten Knoten der aktuellen Partition zu bestimmen und diesen als Einbettungsknoten zu wählen. Aus Lastverteilungsgründen sollte jeder Einbettungsknoten i.d.R. nur für jeweils eine Partition zuständig sein. Dies ist in Abbildung 4.1a veranschaulicht. In jeder Partition ist derjenige Knoten markiert, der als Einbettungsknoten vorgesehen ist. Jeder Partition wurde dabei ein anderer Einbettungsknoten zugewiesen.

Umsetzungsmöglichkeit 2: Netz-externe Einbettungsknoten
Nicht-verteilte Verfahren setzen auf einen einzelnen, abgesondert vom eigentlichen Substratnetz platzierten Rechner, der allein für die Berechnung der Einbettungen zuständig ist. DPVNE kann auf gleich mehreren dieser externen Rechner installiert werden. Beim Einsatz externer Einbettungsknoten vereinfacht sich das Szenario dahingehend, dass die beschriebenen Seiteneffekte, nämlich zusätzlicher Rechen- und Kommunikationsaufwand, nicht mehr innerhalb der Substratnetzinfrastruktur selbst berücksichtigt werden müssen. Dafür müssen allerdings auch zusätzliche Hardwarekomponenten abgestellt werden, auf denen keine virtuellen Instanzen platziert werden können.

4.2.2 Die Einbettungsphase

Im Folgenden wird die Einbettungsphase des DPVNE-Algorithmus beschrieben. Dazu wird zunächst auf die Funktionsweise der Einbettungsknoten eingegangen. Daran anschließend wird beschrieben, wie die Zustände der verteilten Knoten konsistent gehalten werden können. Abschließend wird das Kommunikationsprotokoll vorgestellt, welches zur Koordination der verteilten Knoten eingesetzt wird.

4.2.2.1 Vorgehensweise der Einbettungsknoten

Wie zuvor beschrieben, nimmt jeder Einbettungsknoten entweder die Rolle eines Delegationsknotens, eines Dienstknotens oder eines übergeordneten Knotens ein. Entsprechend unterschiedlich verhalten sich die Knoten bei Empfang einer Einbettungsanfrage.

Algorithmus 3 Die Vorgehensweise der Delegationsknoten bei Empfang einer Einbettungsanfrage. Die Funktion GETEMBEDDINGNODE(p) bestimmt dabei den Einbettungsknoten einer Partition p.

```
1: function RECEIVE(VNR^i v)
2:     result ← None
3:     //Suche nach geeigneten untergeordneten Partitionen
4:     candidates ← {}
5:     for all p in untergeordnete Partitionen} do
6:         if π_CPU(p, v) ≥ 1.0 and π_BW(p, v) ≥ 1.0 then
7:             CANDIDATES.ADD(p)
8:         end if
9:     end for
10:    sortiere candidates abhängig von π (in aufsteigender Reihenfolge)
11:    for p in candidates do
12:        result ← GETEMBEDDINGNODE(p).RECEIVE(v)
13:        if result ≠ None then
14:            return result
15:        end if
16:    end for
17:    p ← eigene Partition
18:    if π_CPU(p, v) ≥ 1.0 and π_BW(p, v) ≥ 1.0 then
19:        result ← EMBED(p, v)
20:    end if
21:    if result = None and self.parent ≠ None then
22:        result ← SELF.PARENT.RECEIVE(v)
23:    end if
24:    return result
25: end function
```

I) Vorgehensweise der Delegationsknoten

Initial werden alle eintreffenden Einbettungsanfragen von einem Load Balancer auf die einzelnen Delegationsknoten aufgeteilt.
Nach Empfang einer solchen Einbettungsanfrage versucht ein Delegationsknoten, die Einbettung des dazugehörigen virtuellen Netzes innerhalb der eigenen, autonomen Region vorzunehmen. Zu diesem Zweck trifft der Delegationsknoten eine Abschätzung, welche der ihm untergeordneten Partitionen über ausreichende Kapazitäten für die Einbettung des virtuellen Netzes verfügen könnte, vgl. dazu die in Alg. 3 gezeigte RECEIVE-Funktion, die das Verhalten des Delegationsknoten nach Empfang einer Anfrage beschreibt.
DPVNE versucht dabei zunächst, eine möglichst kleine Partition ausfindig zu machen, an deren Einbettungsknoten die Anfrage weiterdelegiert werden soll. Zu diesem Zweck schätzt der Delegationsknoten, welche der untergeordneten Partitionen sich potentiell für das Deployment des virtuellen Netzes eignen

könnten. Dies erfolgt durch das In-Bezug-Setzen virtueller Ressourcenanforderungen zu den Ressourcen der einzelnen Substratnetz-Partitionen. Die Abschätzung, ob eine Partition P mit den Knoten N_p und den Links L_p über die nötigen CPU-Ressourcen verfügt, erfolgt dabei über die Funktion:

$$\pi_{\text{cpu}}(P, \text{VNR}^i) = \frac{\sum\limits_{n \in N_p} \text{res}_{\text{cpu}}(n)}{\sum\limits_{n^i \in N^i} \text{dem}_{\text{cpu}}(n^i)} \tag{4.3}$$

Diese Funktion summiert alle verfügbaren CPU-Ressourcen der Partition auf und setzt sie in Relation zu den vom virtuellen Netz geforderten CPU-Ressourcen. Analog lässt sich mit den Bandbreiten-Ressourcen verfahren:

$$\pi_{\text{bw}}(P, \text{VNR}^i) = \frac{\sum\limits_{l \in L_p} \text{res}_{\text{bw}}(l)}{\omega \cdot \sum\limits_{l^i \in L^i} \text{dem}_{\text{bw}}(l^i)} \tag{4.4}$$

In der VNE-Literatur wird üblicherweise davon ausgegangen, dass ein virtueller Knoten auf genau einem Substratknoten eingebettet wird, nicht auf mehreren [55]. Anders verhält es sich bei virtuellen Links: Virtuelle Links werden auf Pfade des Substratnetzes eingebettet, die sich über mehrere Substratlinks erstrecken. Aus diesem Grund enthält die Abschätzungsfunktion π_{bw} aus Formel 4.4 einen Parameter ω, mit dem sich der Optimismus bzw. Pessimismus der Bandbreiten-Abschätzungsheuristik justieren lässt:

1. Optimistische Abschätzung
   Je *kleiner* ω gewählt wird, desto *optimistischer* verhält sich die Heuristik: Es wird dann davon ausgegangen, dass verhältnismäßig wenig Bandbreiten-Ressourcen nötig sind, um die virtuellen Links zu hosten. Für $\omega = 2$ wird beispielsweise angenommen, dass für virtuelle Links im Durchschnitt Ressourcen auf zwei Substratlinks reserviert werden müssen.

2. Pessimistische Abschätzung
   Umgekehrt verhält es sich für große ω-Werte: Je größer ω gewählt wird, desto mehr Bandbreiten-Ressourcen muss eine Partition zur Verfügung stellen, um als möglicher Kandidat in Betracht gezogen zu werden.

Die Einschätzung, ob eine Partition nun für die Einbettung eines virtuellen Netzes in Betracht gezogen wird oder nicht, erfolgt anhand der folgenden Regel (Alg. 3, Zeile 6 ff.): Ergibt eine der beiden Funktionen π_{cpu} oder π_{bw} einen Wert größer oder gleich 1, bedeutet dies, dass sich die Partition möglicherweise für die Einbettung des virtuellen Netzes eignet. Ist sie kleiner 1, erscheint eine Einbettung dagegen wenig vielversprechend – in diesem Fall wird die betroffene Partition für den Einbettungsvorgang nicht weiter berücksichtigt.

Wie im Evaluationsabschnitt noch veranschaulicht wird, ist eine solche ungefähre, vergleichende Abschätzung ausreichend, da diese lediglich der Vorauswahl geeigneter Partitionen dient.

Nach dieser Vorauswahl werden die als geeignet eingestuften Partitionen in Abhängigkeit von ihrem *Einbettungspotential* sortiert. Zur Bestimmung des Potentials wird eine Kombination der zuvor erwähnten Abschätzungsfunktionen verwendet:

$$\pi(P, \mathrm{VNR}^i) = \pi_{\mathrm{cpu}}(P, \mathrm{VNR}^i) \cdot \pi_{\mathrm{bw}}(P, \mathrm{VNR}^i) \tag{4.5}$$

Die Partitionen werden in aufsteigender Reihenfolge nach ihrem π-Wert sortiert (Zeile 10). Die Zuweisungsstrategie versucht nun, das virtuelle Netz mit Hilfe eines VNE-Algorithmus innerhalb der ersten Partition einzubetten. Schlägt der Einbettungsversuch fehl, wird der Vorgang innerhalb der jeweils nächstbesten Partition wiederholt, bis das virtuelle Netz erfolgreich eingebettet werden konnte (Zeile 11). Schlagen alle Einbettungsversuche fehl, versucht der Delegationsknoten, die Einbettung selbst zu berechnen (Zeile 18). Ist auch das nicht möglich, wird die Anfrage schließlich an den übergeordneten Knoten weitergegegeben (Zeile 22).

Die Delegation der Einbettungsanfragen an möglichst kleine Partitionsbereiche hat zwei Vorteile: Zum einen ist es im Allgemeinen einfacher, ein virtuelles Netz in eine kleinere Substratnetzinfrastruktur einzubetten. Dadurch, dass DPVNE versucht, die Einbettung auf eher kleinere Partitionen zu verlagern, wird zudem der Parallelitätsgrad des Verfahrens erhöht, weil somit andere, unabhängige Teile der Partitionen-Hierarchie vom Einbettungsvorgang unberührt bleiben. Weitere Einbettungsanfragen können deshalb parallel von diesen unabhängigen Netzbereichen verarbeitet werden.

II) Vorgehensweise der Dienstknoten

Die Vorgehensweise der Dienstknoten ist recht einfach. Ein Dienstknoten versucht anhand eines VNE-Algorithmus, das virtuelle Netz innerhalb der eigenen Partition einzubetten und sendet das Einbettungsergebnis zurück an den Sender der Nachricht.

III) Vorgehensweise der übergeordneten Knoten

Übergeordnete Knoten haben in der Regel keine aktuellen Informationen über die durchgeführten Ressourcenzuweisungen, da die ihnen untergeordneten Delegationsknoten parallel und autonom arbeiten. Da diese Informationen fehlen, kann nach Empfang einer Einbettungsanfrage auch nicht abgeschätzt werden, welche der untergeordneten Partitionen über entsprechende Kapazitäten verfügen könnte. Daher delegieren übergeordnete Knoten eintreffende Anfragen

Algorithmus 4 Vorgehensweise übergeordneter Knoten bei Empfang einer Einbettungsanfrage. Die Variable *sender* referenziert den Sender der Nachricht.

```
 1: function RECEIVE(VNR^i v, Node sender)
 2:     result ← None
 3:     stoppedNodes ← {}
 4:     for c ∈ direkte Kindknoten ∩ {sender} do
 5:         result ← C.RECEIVE(v) //Senden einer STOP-Nachricht
 6:         stoppedNodes ← stoppedNodes ∪ {c}
 7:         if result ≠ None then
 8:             break
 9:         end if
10:     end for
11:     if result = None then
12:         result ← EMBED(v)
13:     end if
14:     if result = None then
15:         if self.parent ≠ None and self.parent ≠ sender then
16:             result ← SELF.PARENT.RECEIVE(v)
17:         end if
18:     end if
19:     for stoppedNode ∈ stoppedNodes do
20:         sende START-Nachricht an stoppedNode
21:     end for
22:     return result
23: end function
```

einfach sequentiell an die direkten Kindknoten weiter (vgl. Alg. 4, Zeile 4). Alle Kindknoten, die nicht in der Lage sind, die Anfrage innerhalb des eigenen Partitionsbereichs vorzunehmen, beenden alle bereits angestoßenen Berechnungen. Sie pausieren die Weiterverarbeitung aller weiteren Aufgaben, und zwar solange, bis sie vom übergeordneten Knoten Rückmeldung erhalten. Dies geschieht, wenn einer der Knoten die Anfrage einbetten kann. Gelingt dies keinem Knoten, versucht der übergeordnete Knoten selbst, das virtuelle Netz einzubetten (Zeile 12). Misslingt auch das, wird die Anfrage an den Vaterknoten des übergeordneten Knotens weitergereicht (Zeile 16). Der übergeordnete Knoten verfährt dann rekursiv nach demselben Prinzip.

Die Anfrage wird genau dann (final) als nicht-einbettbar zurückgewiesen, wenn auch der Wurzelknoten selbst nicht in der Lage ist, das virtuelle Netz einzubetten.

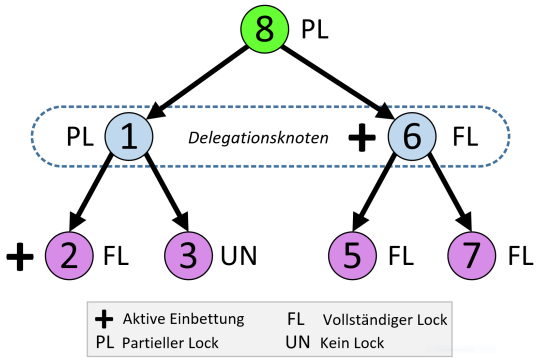


Abb. 4.2: Jeder Einbettungsknoten verwaltet einen Sperr-Baum, in dem über die Zustände der verteilten Knoten Buch geführt wird

4.2.2.2 Parallele Bearbeitung von Einbettungsanfragen und Sicherstellung konsistenter Zustände

Die Einbettungsknoten arbeiten parallel an der Bearbeitung von Einbettungsanfragen. In dem in Abbildung 4.1 gezeigten Szearnio kann beispielsweise der Dienstknoten 2 in der von ihm verwalteten Partition, welche aus den Knoten 1 und 2 besteht, zur gleichen Zeit eine Einbettung vornehmen wie der Dienstknoten 3, der für die Partition mit den Knoten 3 und 4 zuständig ist, da sich diese beiden Partitionen nicht überschneiden.
Bedingt durch die hierarchische Partitionierung des Substratnetzes ist diese für den parallelen Betrieb notwendige Überschneidungsfreiheit für Knoten *unterschiedlicher* Ebenen jedoch nicht in jedem Fall gegeben: Beispielsweise darf Knoten 1, dessen Partition aus den Knoten 1, 2, 3 und 4 besteht, nicht zur gleichen Zeit Einbettungen vornehmen, während Knoten 2 mit dem Deployment eines anderen virtuellen Netzes beschäftigt ist; in diesem Fall würde die parallele Berechnung von Einbettungen aufgrund der Überschneidung der Partitionen zu Mehrfachzuweisungen von Ressourcen und damit zu Inkonsistenzen führen.
Zur Vermeidung solcher Inkonsistenzen verwaltet jeder Einbettungsknoten einen sogenannten Sperr-Baum, in dem über die Belegung der verteilten Knoten Buch geführt wird, vgl. Abbildung 4.2. Jedem Einbettungsknoten kann dabei einer der folgenden Locking-Zustände zugewiesen werden:

- **Vollständiger Lock** (full lock, FL)
  Einbettungsknoten, die die Berechnung einer Einbettung vornehmen, sind während der Berechnung mit einem vollständigen Lock markiert. Ferner sind alle Knoten mit einem vollständigen Lock vermerkt, die Kinder oder Kindeskinder eines Knotens sind, für den ein vollständiger Lock vermerkt ist. In Abbildung 4.2 werden die Knoten 2 und 6 mit einem vollständigen Lock markiert, da sie Einbettungen berechnen. Die Knoten 5 und 7, die dem Knoten 6 untergeordnet sind, sind ebenfalls vollständig gesperrt, da sich ihr Partitionsbereich mit dem von Knoten 6 überschnei-

det.

- **Partieller Lock** (partial lock, PL)
  Berechnet ein untergeordneter Einbettungsknoten eine Einbettung, werden alle ihm übergeordneten Einbettungsknoten mit einem partiellen Lock markiert. Auch in partiell gelockten Partitionen kann keine parallele Einbettung vorgenommen werden, da diese Partitionen Teile der untergeordneten Partitionen enthalten und es anderenfalls zu Inkonsistenzen kommen kann. Beispielsweise kann Knoten 1 in dem in Abbildung 4.2 gezeigten Szenario nicht parallel zu Knoten 2 einbetten, da sich die Partitionsbereiche überschneiden.

- **Kein Lock** (unlocked, UN)
  Berechnet ein Einbettungsknoten weder selbst noch einer seiner ihm direkt oder indirekt untergeordneten Kindknoten eine Einbettung, befindet sich der Einbettungsknoten im Zustand „Kein Lock“.

4.2.2.3 Kommunikationsprotokoll

Die Einbettungsknoten kooperieren bei der verteilten und parallelen Bearbeitung der Einbettungsanfragen. Delegationsknoten leiten Anfragen an ihre Dienstknoten weiter und geben, falls eine Einbettung innerhalb des eigenen Bereichs nicht möglich ist, die Anfrage an den übergeordneten Einbettungsknoten weiter. Da Dienstknoten Einbettungen auf Teilen der Partitionen berechnen, auf denen auch die ihnen übergeordneten Knoten operieren, muss dabei sichergestellt werden, dass Informationen, auf die von mehren Einbettungsknoten zurückgegriffen wird, über den Austausch von Kommunikationsnachrichten synchronisiert werden. Für DPVNE wurden zu diesem Zweck die vier folgenden Nachrichtentypen spezifiziert:

- **Delegations-Nachrichten** `DEREQ(VNR`^{i}`, updates)`
  Delegations-Nachrichten werden zur Delegation von Einbettungsanfragen an Einbettungsknoten verwendet. Diese Art von Nachrichten wird vom Load Balancer verwendet, der eine neu eintreffende Anfrage an die Delegationsknoten weiterreicht. Die Delegationsknoten verwenden Delegations-Nachrichten, um die ihnen zugewiesenen Dienstknoten mit der Bearbeitung von Anfragen zu beauftragen. Übergeordnete Knoten verwenden ebenfalls derartige Nachrichten zur Weitergabe von Anfragen.

  Eine `DEREQ`-Nachricht enthält die übermittelte Einbettungsanfrage `VNR`^{i}. Der Sender kann zudem Update-Informationen über gegebenenfalls vorgenommene Zuweisungsentscheidungen beifügen, die dem Empfängerknoten zur Synchronisierung des Zustandes mitgegeben werden können (`updates`). Jede `DEREQ`-Nachricht muss mit einer `EMRES`-Nachricht beantwortet werden (siehe unten).

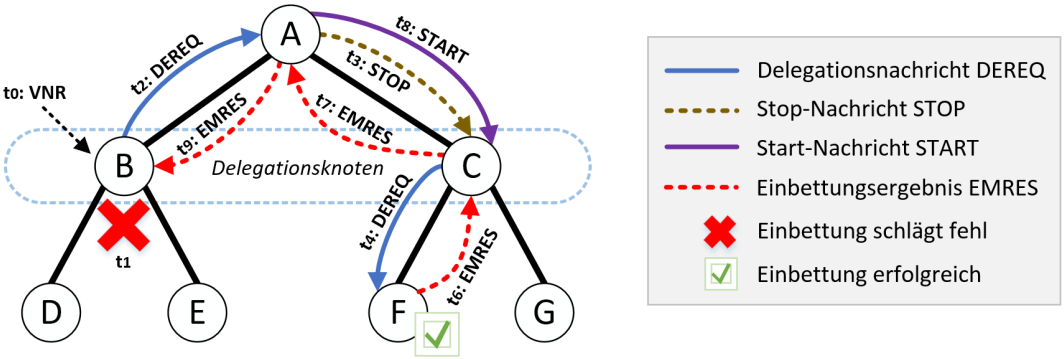


Abb. 4.3: Kommunikationsablauf

- **Einbettungs-Ergebnisse** `EMRES(updates)`
  Nach der Delegation und/oder Bearbeitung einer `DEREQ`-Anfrage teilen die Einbettungsknoten anhand einer `EMRES`-Nachricht mit, ob der Einbettungsversuch erfolgreich war oder nicht. Jeder Einbettungsknoten sendet die `EMRES`-Nachricht dabei jeweils an den Knoten, von dem er die `DEREQ`-Anfrage zuvor empfangen hat. Auch dieser Nachrichtentyp erlaubt das Mitschicken von Synchronisierungs-Informationen (`updates`).

- **Stop-Nachrichten** `STOP(VNR`^{i}`)`
  Delegationsknoten haben zudem die Möglichkeit, Anfragen an übergeordnete Knoten weiterzugeben. Wegen der Überschneidung der von den übergeordneten Knoten verwalteten Partitionen mit denen von anderen Delegationsknoten verwalteten Partitionen muss der parallele Betrieb der Delegationsknoten zur Vermeidung von Inkonsistenzen vorübergehend angehalten werden. Dasselbe gilt ebenso für die übergeordneten Knoten. Nach Erhalt einer `STOP`-Nachricht wartet ein Einbettungsknoten, bis alle bereits angestoßenen Berechnungen ausgeführt wurden. Anschließend versucht er, die als Teil der `STOP`-Nachricht mitgesendete Einbettungsanfrage einzubetten. Schlägt dies fehl, pausiert der Einbettungsknoten, bis er von seinem Vaterknoten eine `START`-Nachricht empfängt.

- **Start-Nachrichten** `START(updates)`
  Nachdem das Ergebnis einer Einbettung vorliegt, dessen Berechnung über die übergeordneten Knoten koordiniert wurde, kann der parallele Betrieb der pausierten Einbettungsknoten wieder aufgenommen werden. Dies teilt der entsprechende Vaterknoten mittels einer `START`-Nachricht mit, die an alle pausierten Kindknoten gesendet wird.

Jeder Einbettungsknoten führt Buch über die ihm bekannten letzten Einbettungsvorgänge und darüber, welche dieser Informationen bereits von ihm an welche der anderen Einbettungsknoten mitgeteilt wurden. Der Sender einer Nachricht fügt dann die aus seiner Sicht für den Empfänger jeweils neueren Informationen an die zu versendende Nachricht an.

4.2.2.4 Verteilte Berechnung von Einbettungen mit DPVNE: Ein Beispiel

Abbildung 4.3 veranschaulicht den Ablauf des verteilten Einbettungsvorgangs anhand eines Beispiels. Für dieses Szenario wird angenommen, dass das einzubettende virtuelle Netz nur von den Knoten A, C, F und G eingebettet werden kann. Im Folgenden wird beschrieben, welche Nachrichten zu den jeweiligen Zeitpunkten ausgetauscht werden:

t₀: Ein virtuelles Netz wird als Teil einer Einbettungsanfrage mittels einer `DEREQ`-Nachricht vom Load Balancer an einen der Delegationsknoten gesendet (hier: B). B schätzt nun mit Hilfe der in Abschnitt 4.2.2.1 beschriebenen Heuristik, ob eine Einbettung innerhalb der von D und E verwalteten Partitionen möglich ist, beziehungsweise, ob Knoten B selbst in der Lage ist, die Anfrage innerhalb der eigenen Partition zu bearbeiten. Zur Veranschaulichung wird nun davon ausgegangen, dass die Heuristik für die von D und E verwalteten Partitionen Werte < 1 liefert und für die von B verwaltete Partition einen Wert ≥ 1. In diesem Fall wird angenommen, dass die Partitionen von D und E nicht über genügend Ressourcen verfügen, um das virtuelle Netz aufzunehmen.

t₁: Da aufgrund des Ergebnisses der Heuristik aber davon ausgegangen wird, dass eine Einbettung in die Partition von B möglich sein könnte, führt B selbst den Einbettungsalgorithmus innerhalb der eigenen Partition aus. Knoten B setzt dazu innerhalb seines lokalen Sperr-Baums einen vollständigen Lock für alle Partitionen innerhalb seines Zuständigkeitsbereichs, also für B, D und E. In diesem Szenario wird angenommen, dass nun der Einbettungsversuch des VNE-Algorithmus scheitert.

t₂: Da Delegationsknoten B nicht in der Lage ist, die Anfrage innerhalb des ihm zugewiesenen Bereichs zu bearbeiten, delegiert B die Anfrage an den übergeordneten Knoten A, indem B eine `DEREQ`-Nachricht sendet.

Da die den Delegationsknoten übergeordneten Einbettungsknoten nicht über aktuelle Einbettungsvorgänge untergeordneter Knoten informiert sind, erhält A über die `DEREQ`-Nachricht den aktuellen Zustand der von B verwalteten Partition. Nach dem Versenden der `DEREQ`-Nachricht wartet B nun, bis A über eine `EMRES`-Nachricht ein Einbettungsergebnis mitteilt. Weitere Einbettungsanfragen, die zwischenzeitlich vom Load Balancer an B zugestellt werden, schreibt B in eine Warteschlange, die erst dann abgearbeitet wird, wenn das Ergebnis von A mitgeteilt wurde.

t₃: Trifft bei A eine `DEREQ`-Nachricht ein, weiß A, dass Delegationsknoten B, der diese Nachricht verschickt hat, nicht in der Lage war, die Anfrage selbst zu bearbeiten. Daher wählt A zufällig einen der anderen Delegationsknoten aus und sendet diesem eine `STOP`-Nachricht. Da in diesem

Beispiel nur zwei Delegationsknoten gewählt wurden, kann hier nur C ausgewählt werden.

t4: C empfängt die `STOP`-Nachricht und versucht das virtuelle Netz innerhalb der eigenen Partitionen einzubetten. Dazu wartet C, bis alle von C selbst zuvor delegierten Anfragen von den ihm untergeordneten Einbettungsknoten mit einer `EMRES`-Nachricht beantwortet wurden. C verschiebt daraufhin die Bearbeitung aller später eintreffenden Anfragen. Nun wählt C mit Hilfe der Heuristik den Knoten F aus und delegiert die Anfrage an diesen. C setzt einen lokalen Lock für F; da nun ein Kindknoten von C eine Einbettung vornimmt, setzt C einen partiellen Lock für sich selbst.

t5: F bettet die Anfrage erfolgreich innerhalb der eigenen Partition ein.

t6: F sendet eine `EMRES`-Nachricht, um C mitzuteilen, dass die Einbettung erfolgreich war, und fügt entsprechende Zuweisungsinformationen an.

t7: C entfernt den vollständigen Lock auf F und den partiellen Lock auf C selbst und leitet das Einbettungsergebnis samt der vorgenommenen Resourcenzuweisungen an A weiter. C hängt ggf. weitere Updates an die `EMRES`-Nachricht an, für den Fall, dass von G zwischenzeitlich weitere Einbettungen vorgenommen wurden.

t8: A empfängt die `EMRES`-Nachricht und aktualisiert seine lokalen Zuweisungsinformationen. A sendet daraufhin eine `START`-Nachricht an C, und zwar mit einer leeren Menge an Updates, da in der Zwischenzeit keine für C relevanten Zuweisungen vorgenommen wurden. Daraufhin fährt C fort, weitere Anfragen zu bearbeiten.

t9: A leitet die `EMRES`-Nachricht weiter an B. B fährt schließlich mit der Bearbeitung weiterer Anfragen fort.

4.3 Evaluation des DPVNE-Algorithmus

Die anschließende Evaluation beleuchtet die folgenden Kern-Aspekte des DPVNE-Algorithmus:

I) **Evaluation der Universalität von DPVNE**
DPVNE ist ein generisches Framework, welches in Kombination mit unterschiedlichen Einbettungsalgorithmen eingesetzt werden kann. Zunächst wird DPVNE daher im Zusammenspiel mit verschiedenen Standard-VNE-Algorithmen untersucht; Tabelle 4.1 zeigt eine Auflistung samt kurzer Beschreibung der zum Vergleich herangezogenen Verfahren. Für diese Experimente wurde der ω-Parameter dazu zunächst auf einen

fixen Wert von $\omega = 10$ gesetzt. Die Auswirkungen von ω auf das Verhalten von DPVNE wird im späteren Verlauf des Abschnittes genauer analysiert, wo sich zeigen wird, dass $\omega = 10$ in der Tat eine gute Voreinstellung des Parameters ist.

Um zu evaluieren, wie gut DPVNE in Kombination mit den untersuchten VNE-Algorithmen bezüglich der Akzeptanz und der Einbettungskosten abschneidet, werden diese ursprünglichen nicht-verteilten Algorithmen unabhängig vom DPVNE-Framework in denselben Szenarien ausgeführt und die so erzielten Evaluationsergebnisse denen des verteilten DPVNE-Frameworks gegenübergestellt. Da DPVNE mit dem Ziel konzipiert wurde, gültige Einbettungen bei gleichzeitig geringen Einbettungskosten zu finden, wurden VNE-Algorithmen ausgewählt, deren Optimierungsziel ebenfalls die Reduktion der Einbettungskosten ist. Im Gegensatz zu DPVNE verwenden nicht-verteilte Verfahren einen einzigen, zentralen Einbettungsknoten, der von der vollständigen Kenntnis des Substratnetz-Zustandes profitiert.

Neben diesen nicht-verteilten Algorithmen wird DPVNE zudem mit einem weiteren verteilten VNE-Algorithmus verglichen. Dies ist der ADVNE-Algorithmus, der ebenfalls in Tabelle 4.1 aufgeführt ist. ADVNE ist, nach bestem Wissen des Autors, der bislang einzige vollständig verteilte Intra-Infrastruktur-Ansatz, der bislang in der Literatur vorgestellt wurde.

Wie sich bei der Durchführung dieser Experimente zeigt, skalieren viele der nicht-verteilten VNE-Algorithmen nicht mit der Größe des Substratnetzes. Auch der verteilte ADVNE-Algorithmus kommt in derart großen Szenarien nicht zurecht. Um die verschiedenen Ansätze trotzdem bezüglich der Einbettungsgüte – also der Akzeptanz- und der Einbettungskosten-Metrik – miteinander vergleichen zu können, werden aus diesem Grund zunächst Szenarien gewählt, in denen die Größe der Substratnetze bei 50 Knoten liegt. Dies steht im Einklang mit dem in der Literatur verbreiteten Evaluationsvorgehen: In der Tat werden sehr viele VNE-Algorithmen in Szenarien dieser Größenordnung evaluiert.

II) **Evaluation der Skalierbarkeit von DPVNE**
Um nun zu veranschaulichen, dass DPVNE in der Lage ist, Einbettungen auch in größeren Netzen vorzunehmen, wird das Verhalten von DPVNE im zweiten Schritt in großen Substratnetzen untersucht. Zu diesem Zweck werden wesentlich größere Substratnetze generiert als dies in der Literatur zur Evaluation von VNE-Algorithmen im Allgemeinen üblich ist. Um zu zeigen, dass DPVNE mit derart großen Szenarien zurecht kommt, werden Laufzeitmessungen aufgezeichnet und die Messergebnisse denen der nicht-verteilten Algorithmen gegenübergestellt.

Darüber hinaus wird der Nachrichten-Overhead von DPVNE untersucht und mit dem des verteilten ADVNE-Ansatzes verglichen.

III) **Evaluation der Abschätzungsheuristik von DPVNE**
Schlussendlich wird das Verhalten von DPVNE in Abhängigkeit vom ω-Parameter untersucht. Insbesondere wird dabei gezeigt, wie sich über unterschiedliche Einstellungen dieses Parameters Einfluss auf das Delegationsverhalten des DPVNE-Frameworks nehmen lässt. Zudem wird untersucht, inwieweit der Parameter Auswirkungen auf den erzeugten Nachrichten-Overhead sowie auf die Parallelität des Einbettungsvorgangs hat. Insgesamt kann gezeigt werden, dass $\omega = 10$ eine gute Wahl hinsichtlich des Laufzeitverhaltens, des Nachrichten-Overheads und des Parallelitätsgrades ist.

4.3.1 Evaluationsmethodik

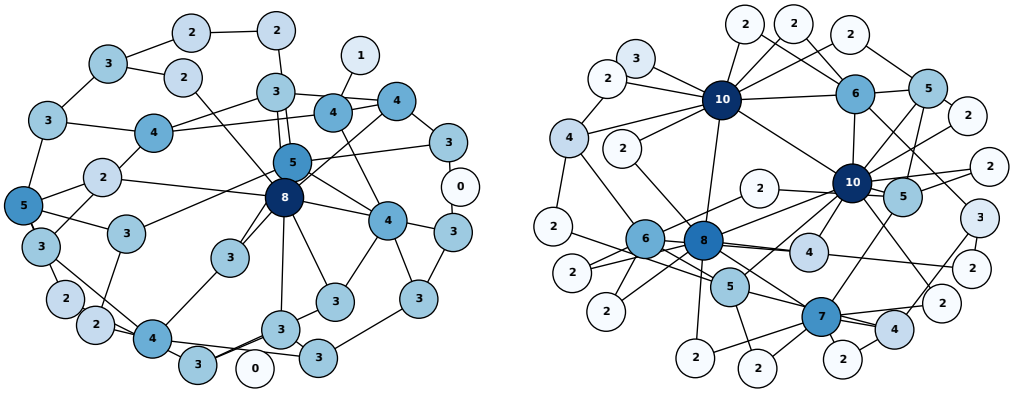
Die im Folgenden gezeigten Simulationen werden mit Hilfe des Alevin-Frameworks durchgeführt, da Alevin bereits Implementationen mehrerer VNE-Algorithmen enthält, mit denen DPVNE verglichen werden kann.

4.3.1.1 Topologie- und Ressourcenerzeugung

Um möglichst umfassende Aussagen über das Verhalten der Algorithmen treffen zu können, werden die Algorithmen in vielen verschiedenen Substratnetz-Topologien evaluiert. Diese werden mit Hilfe eines Topologiegenerators zufällig erzeugt. Eine solche Vorgehensweise ist im Umfeld der VNE-Algorithmen allgemein üblich. In der Literatur wurden in diesem Zusammenhang verschiedene Ansätze vorgestellt, mit denen sich realitätsnahe zufällige Netztopologien generieren lassen. Sehr verbreitet ist die Simulation von VNE-Algorithmen in Netzen, die mit dem Waxman-Algorithmus generiert wurden [102, 55]. Allerdings haben solche Netze nur wenig Gemeinsamkeiten mit realistischen Netzstrukturen – insbesondere dann, wenn größere Topologien mit mehreren hundert Knoten erzeugt werden sollen. Da DPVNE in eben solch großen Netzen simuliert werden soll, wird stattdessen auf den Barabási-Albert-Ansatz zurückgegriffen. Dieser Ansatz erlaubt die Erzeugung zufälliger, skalen-freier Topologien. Solche Netzstrukturen kommen denen realer Netze sehr nahe [10]. Ein skalen-freies Netz ist ein Netz, dessen Verteilung der Knotengrade einem Potenzgesetz folgt. Das heißt, dass ein Knoten mit einer Wahrscheinlichkeit

$$P(k) \sim k^{-\lambda} \tag{4.6}$$

mit k anderen Knoten des Netzes verbunden ist. In der Literatur finden sich zahlreiche statistische Untersuchungen von Strukturen realer Netze, die zeigen, dass viele dieser Strukturen einem Potenzgesetz folgen.
Zur Generierung einer zufälligen Netztopologie startet der Barabási-Albert-Algorithmus mit einer initialen Menge an nicht miteinander verbundenen Knoten und fügt iterativ weitere Knoten hinzu, bis die erforderliche, finale Netzwerkgröße erreicht ist. Neue Knoten werden nach dem *preferential*

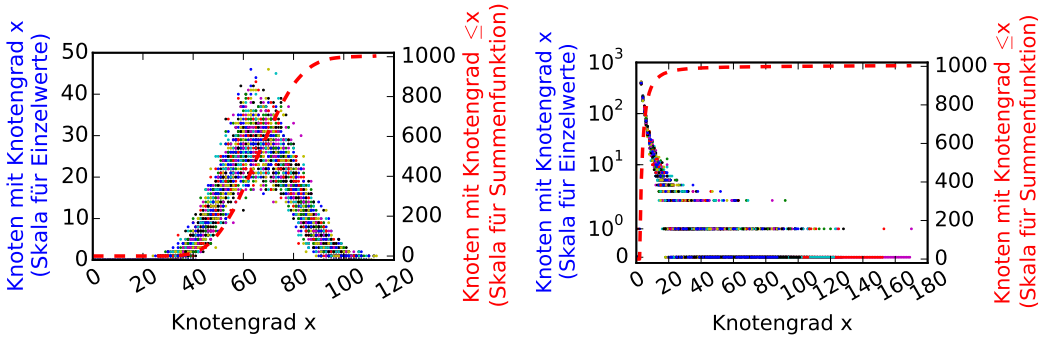


(a) Waxman (b) Barabási-Albert

Abb. 4.4: Vergleich der Knotengrade in zufällig erzeugten Graphtopologien – je dunkler die Farbe eines Knotens, desto höher ist sein Knotengrad

attachment genannten Prinzip mit anderen Knoten des Netzes verbunden: Dabei werden neue Knoten tendenziell eher mit solchen Knoten verbunden, die bereits über viele Verbindungen verfügen. Kanten-„reiche“ Knoten werden dadurch noch „reicher“. Das führt dazu, dass sich nach und nach Cluster von Knoten herausbilden. Dieser Effekt, der durch das *preferential attachment* forciert wird, wird in der Literatur auch der Matthäus-Effekt genannt [89], nach dem biblischen Vers: *„Denn wer da hat, dem wird gegeben werden . . .“ (Matthäus 25.29).*

Im Unterschied dazu ist das Vorgehen des Waxman-Verfahrens wie folgt: Zunächst werden die Knoten des zu erzeugenden Netzes generiert. Diese Knoten werden dann uniform auf einer Ebene verteilt. Die Verbindung der Knoten erfolgt nach einer gewissen Wahrscheinlichkeit, die in Abhängigkeit zu der Entfernung der einzelnen Knoten untereinander steht [102].
Abbildung 4.4 stellt eine nach dem Waxman-Verfahren erzeugte zufällige Topologie einer Barabási-Albert-Topologie gegenüber. Wie hier zu sehen ist, sind die Knoten des Waxman-Netzes relativ gleichmäßig vernetzt. Es lässt sich keine klare Zugehörigkeit von Knoten zu voneinander abgrenzbaren Clustern erkennen. Das ist hingegen bei der Barabási-Albert-Topologie der Fall. Hier lässt sich der sogenannte Matthäus-Effekt gut beobachten: Einige wenige gut vernetzte Knoten verbinden mehrere wenig vernetzte Knoten und tragen somit zu einer Clusterbildung bei.
Abbildung 4.5 veranschaulicht die Verteilung der Knotengrade in zufällig erzeugten Netzen. Der Waxman-Algorithmus wurde mit den in der Literatur üblichen Werten $\alpha = 0.2, \beta = 0.3$ parametrisiert, der Barabási-Albert-Algorithmus mit dem Wert $m = 3$. Die Auswertungen zeigen Ergebnisse für jeweils 50 zufällig erzeugte Netztopologien mit je 1000 Knoten. Wie den Abbildungen zu entnehmen ist, sind die Knoten der beiden Topologien sehr unterschiedlich miteinander vernetzt. Die Summenfunktion der Waxman-Netze



(a) Waxman (b) Barabási-Albert

Abb. 4.5: Vergleich der Knotengrade vieler zufällig erzeugter Graphtopologien

steigt erst im Bereich 40-80 stark an. Beim Barabási-Albert-Modell steigt die Summenfunktion wesentlich früher und nähert sich ihrem Höchstwert bereits im Intervall 0-20. Das bedeutet, dass die einzelnen Knoten der Waxman-Netze im Schnitt deutlich mehr Verbindungen haben als die der Barabási-Albert-Netze. Während der durchschnittliche Knotengrad der Waxman-Netze bei einem Wert von 66 lag, sind die Knoten der Barabási-Albert-Topologien im Schnitt mit nur 6 Knoten verbunden. Wie sich in Abbildung 4.5b beobachten lässt, ist die Verteilung der Knotengrade bei den Barabási-Albert-Netzen skalenfrei und folgt einem Potenzgesetz, nämlich, wie von Barabási et al. gezeigt, $P(k) \sim k^{-3}$ [10].

Um möglichst allgemeingültige Aussagen über das Verhalten der Algorithmen treffen zu können, werden die Simulationen auf Basis vieler unterschiedlicher Netze durchgeführt. Diese werden anhand des Barabási-Albert-Ansatzes zufällig generiert. Nach Erzeugung der Netztopologien wird den Knoten jeweils eine zufällige, normalisierte Menge an CPU-Ressourcen zwischen 1 und 100% zugewiesen; analog werden den Substratlinks zufällige Bandbreiten-Ressourcen aus demselben Wertebereich zugeordnet. CPU-Anforderungen der virtuellen Knoten und Bandbreiten-Anforderungen der virtuellen Links werden zufällig aus dem Bereich 1%-50% gewählt. Diese Parameter wurden in Anlehnung an die in der Literatur üblichen Werte gewählt. Um Aussagen über die Stabilität der Ergebnisse treffen zu können, die auf Basis derart zufällig erzeugter Netzstrukturen erzielt werden, wird jedes Szenario 50 mal mit jeweils anderen Netzen wiederholt. Alle Ergebnisse sind mit einem Konfidenzintervall von 95% angegeben.

4.3.1.2 Partitionierung

Die Partitionierung der Substratnetze wurde anhand des *METIS*-Tools [5] durchgeführt, welches die in Abschnitt 4.2.1.1 diskutierten Eigenschaften erfüllt. *METIS* ist eine Implementation des von Karypis et al. vorgestellten Par-

titionierungsverfahrens, welche von den Autoren zur freien Verfügung gestellt wurde [72]. Das Verfahren eignet sich insbesondere zur Partitionierung großer Graphen, da es in der Lage ist, selbst Graphen mit zehntausenden Knoten innerhalb von Sekunden zu partitionieren.

4.3.1.3 Einbettungsknoten

Wie in Abschnitt 4.2.1.2 näher ausgeführt, können Einbettungsknoten entweder als Teil des Substratnetzes gewählt werden, oder es kann auf externe Knoten zurückgegriffen werden, die exklusiv für die Berechnung der Einbettungen zur Verfügung stehen. Im Folgenden wird von netz-internen Knoten ausgegangen. Dazu wird in der Initialisierungsphase ein Knoten jeder Partition ausgewählt, der über sehr vergleichsweise viele CPU-Ressourcen verfügt und bislang noch keiner anderen Partition als Einbettungsknoten zugewiesen wurde. Es wird davon ausgegangen, dass solche rechenstarken Knoten in realen Szenarien über zusätzliche Ressourcen verfügen, die exklusiv für die Ausführung der VNE-Algorithmen zur Verfügung stehen und nicht für das Deployment von virtuellen Ressourcen verwendet werden.

4.3.1.4 Simulation verteilter Verfahren in einer nicht-verteilten Simulationsumgebung

Die Alevin-Simulationsumgebung sieht eine verteilte Ausführung der Simulationen auf mehreren miteinander vernetzten Rechnern nicht vor. Alle Simulationen werden daher, obwohl DPVNE ein verteilter und paralleler Ansatz ist, sequentiell auf einem einzelnen Rechner und auf einem einzelnen CPU-Kern simuliert. Das bedeutet, dass die Simulationsergebnisse folgende Eigenschaften realer Cloud-Umgebungen nicht berücksichtigen:

I) **Kommunikationsverzögerungen**
Die in realen Netzen auftretenden Kommunikationsverzögerungen, die beim Austausch von Nachrichten zwischen Einbettungsknoten auftreten würden, wurden im Rahmen der Simulation bei den Laufzeitmessungen von DPVNE nicht berücksichtigt. Es kann allerdings davon ausgegangen werden, dass diese Laufzeitunterschiede marginal sind und im Rahmen von wenigen Millisekunden liegen.

II) **Verteilte Ausführung der VNE-Algorithmen**
Der Einbettungsvorgang wird zwar innerhalb der Alevin-Simulationsumgebung als verteilter Prozess simuliert, tatsächlich werden die VNE-Berechnungen aber auf einem einzigen Simulationsrechner durchgeführt – und nicht, wie dies in einer realen Umgebung der Fall wäre, von mehreren verteilten Rechnern. Das bedeutet, dass bei der Ausführung von DPVNE auf mehreren Rechnern von einer zusätzlichen Beschleunigung der Berechnungen ausgegangen werden kann. Dies gilt natürlich ebenso für andere verteilte Verfahren.

Algorithmus	Beschreibung
ASID	Der ASID-Algorithmus versucht unter Berücksichtigung der Ressourcenanforderungen, innerhalb des Substratnetzwerkes Strukturen zu erkennen, die dieselbe oder eine ähnliche Struktur haben wie die virtuellen Netzwerke. Der Algorithmus arbeitet nicht-verteilt und verwendet eine backtracking-basierte Heuristik zum Erkennen von Subgraph-Isomorphismen [78].
GAR	Ein nicht-verteilter Algorithmus, der Substratknoten nach dem Greedy-Verfahren auswählt und für das Einbetten der virtuellen Links die k-kürzesten Substratpfade berücksichtigt [105].
DViNE	Nicht-verteilter Algorithmus, der die Einbettung von virtuellen Knoten und Links zwar ebenfalls in zwei Schritten vornimmt, dies aber koordiniert tut. Für das Einbetten der virtuellen Links werden die k-kürzesten Substratpfade berücksichtigt. Der Algorithmus ist als Mixed Integer Programming (MIP) Problem formuliert [35].
RW-MM	Nicht-verteilter Algorithmus, der bei der Einbettung der Knoten die Netzwerktopologie betrachtet und zum Einbetten der Links die k-kürzesten Substratpfade berücksichtigt [34].
ADVNE	Verteilter Algorithmus, der mit Hilfe dezentral operierender Agenten versucht, ein gültiges Einbettungsergebnis zu finden [64].

Tabelle 4.1: Einbettungsalgorithmen, die zur Evaluation von DPVNE verwendet wurden

Tabelle 4.1 gibt einen Überblick über die Algorithmen, mit denen DPVNE evaluiert wird. DPVNE selbst wird in Kombination mit den hier aufgeführten nicht-verteilten Algorithmen evaluiert. Die eigentlichen Einbettungsberechnungen von DPVNE werden also von den Einbettungsknoten anhand eines dieser VNE-Algorithmen durchgeführt. Die Ergebnisse werden dann mit den Ergebnissen verglichen, die die VNE-Algorithmen ursprünglich liefern, d.h., wenn sie außerhalb des DPVNE-Frameworks ausgeführt werden. Da DPVNE ein verteilter Ansatz ist, wurde zudem der von Houidi et al. vorgestellte ADVNE-Algorithmus nachimplementiert [64], dessen Evaluationsergebnisse denen des DPVNE-Ansatzes gegenübergestellt werden.

4.3.2 Evaluationsmetriken

In Abschnitt 3.4 wurde bereits diskutiert, anhand welcher Metriken sich die Güte von Einbettungsergebnissen bzw. der Einbettungsvorgang selbst numerisch bewerten lassen. Zur Evaluation von DPVNE werden hierbei insbesondere die Akzeptanz in Form der prozentualen Anzahl erfolgreich eingebetteter virtueller Netze, das Laufzeitverhalten, die Anzahl der versendeten Nachrichten und das Kosten-/Nutzen-Verhältnis in unterschiedlichen Szenarien analysiert. In VNE-Szenarien ist das Kosten-/Nutzen-Verhältnis, welches Einbettungskosten und Einbettungsertrag zueinander in Bezug setzt, üblicherweise wie folgt definiert:

Die *Einbettungskosten*-Metrik (*Cost*) bemisst den tatsächlichen Ressourcenverbrauch der eingebetteten virtuellen Netze. In der VNE-Literatur ist folgende Formulierung gebräuchlich, die die reservierten CPU-Ressourcen mit den reservierten Bandbreite-Ressourcen addiert:

$$\text{Cost}(\text{VNR}^i) = \sum_{n^i \in N^i} \text{dem}_{\text{cpu}}(n^i) + \sum_{l^i \in L^i} h(l^i) \cdot \text{dem}_{\text{bw}}(l^i) \tag{4.7}$$

$h(l^i)$ sei hier definiert als eine Funktion, die die Länge des Substratpfades bestimmt, auf dem ein virtueller Link l^i platziert wurde.

Die Einbettungskosten werden in der VNE-Literatur meist in Relation zum Einbettungsertrag gesetzt; das Kosten-/Nutzen-Verhältnis (*Revenue-Cost*) wird dabei wie folgt definiert:

$$\text{Revenue-Cost}(\text{VNR}^i) = \frac{\text{Revenue}(\text{VNR}^i)}{\text{Cost}(\text{VNR}^i)} \tag{4.8}$$

Es wird also der Einbettungsertrag (*Revenue*) durch die Einbettungskosten geteilt. Dabei ist die *Einbettungsertrags*-Metrik als Maß für die Ressourcenanforderungen der virtuellen Netze in der VNE-Literatur üblicherweise als Summe

der angeforderten CPU- und Bandbreiteressourcen definiert:

$$\text{Revenue}(\text{VNR}^i) = \sum_{n^i \in N^i} \text{dem}_{\text{cpu}}(n^i) + \sum_{l^i \in L^i} \text{dem}_{\text{bw}}(l^i) \tag{4.9}$$

Die Werte der Revenue-Cost-Metrik bewegen sich im Bereich zwischen 0 und 1. Je mehr sich die Metrik dem Wert 1 nähert, umso ressourceneffizienter ist die Einbettung. In Fällen, in denen das Co-Hosting von mehreren, miteinander verbundenen Knoten desselben virtuellen Netzes auf demselben Substratknoten erlaubt ist, können die Werte der Metrik durchaus größer als 1 sein.

4.3.3 Evaluationsergebnisse

Als Resultat zahlreicher Experimente und umfangreicher Simulationen lassen sich die folgenden Kernergebnisse zusammenfassen. Tabelle 4.1 listet die in den Grafiken referenzierten Algorithmen:

4.3.3.1 Evaluation der Universalität von DPVNE

In diesem Abschnitt wird untersucht, inwiefern sich DPVNE in Kombination mit unterschiedlichen nicht-verteilten, kosten-minimierenden VNE-Algorithmen verwenden lässt und wie sich die Einbettungsergebnisse jeweils voneinander unterscheiden. Dazu sollen 1 bis 10 virtuelle Netze mit jeweils 10 Knoten in ein Substratnetz mit 50 Knoten eingebettet werden. Im Folgenden werden die Algorithmen bezüglich ihres Kommunikations-Overheads, der Akzeptanz und der Einbettungskosten verglichen.

Kommunikations-Overhead: Abbildung 4.6 vergleicht die Anzahl der von DPVNE und ADVNE versendeten Nachrichten. Da bei nicht-verteilten Algorithmen keine Nachrichten ausgetauscht werden müssen und die Anzahl der versendeten Nachrichten aus diesem Grund immer gleich 0 ist, sind für diese Algorithmen hier keine Werte aufgeführt. Wie sich der Abbildung entnehmen lässt, bleibt der Nachrichten-Overhead von DPVNE in den untersuchten Szenarien sogar in Szenarien mit 10 virtuellen Netzen unter 20 Nachrichten. Der Nachrichten-Overhead des verteilten ADVNE-Algorithmus, der hier ebenfalls abgebildet ist, ist dagegen um ein Vielfaches höher: Hier werden gleich mehrere tausende bis zehntausende Nachrichten verschickt. Diese krassen Unterschiede lassen sich auf die unterschiedliche Organisationsstruktur beider Algorithmen zurückführen. Der ADVNE-Ansatz setzt auf eine große Anzahl verteilter Einbettungsagenten, die als Bestandteil des Substratnetzes selbst gleichberechtigt und autonom Teileinbettungen vornehmen. Jeder Knoten des Substratnetzes kommuniziert mit allen anderen Knoten. DPVNE setzt stattdessen auf eine hierarchische Organisation der Einbettungsknoten, wobei nur relativ wenige

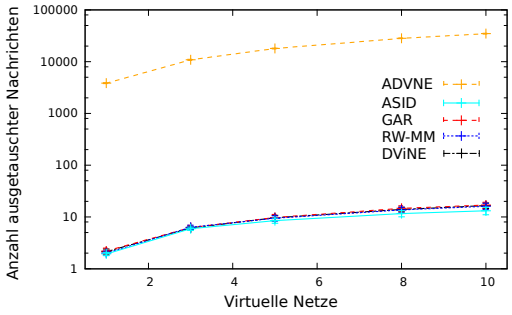
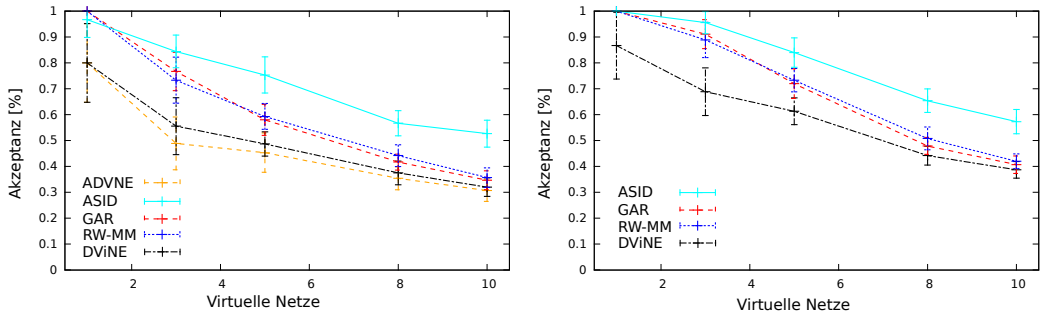


Abb. 4.6: Vergleich von DPVNE und dem verteilten ADVNE-Algorithmus hinsichtlich des Kommunikations-Overheads

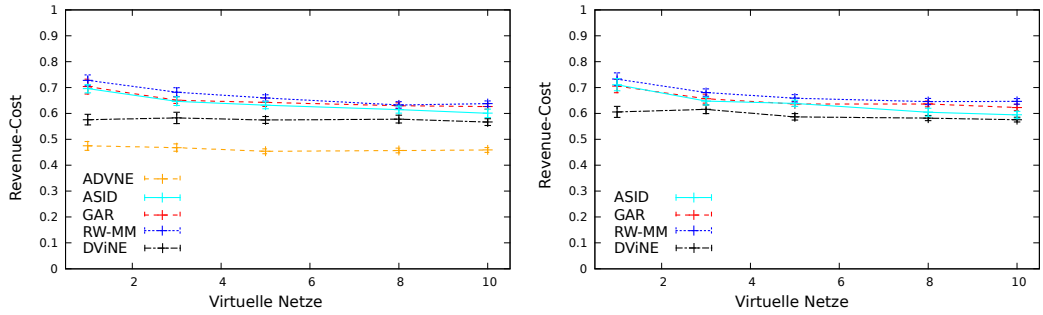


(a) Ursprüngliche Algorithmen (b) DPVNE

Abb. 4.7: Akzeptanz

Knoten des Substratnetzes als Einbettungsknoten vorgesehen sind. Neu einzubettende virtuelle Netze werden anhand der Einbettungsheuristik zu den Partitionen „gelenkt“, in die sie sehr wahrscheinlich eingebettet werden können.

Akzeptanz und Einbettungskosten: Abbildung 4.7 stellt die Akzeptanzwerte der Algorithmen gegenüber. Wie sich hier zeigt, bleiben die Werte des DPVNE-Frameworks vergleichbar mit denen der unveränderten Algorithmen. Dasselbe trifft auch für die in Abbildung 4.8 gezeigte Revenue-Cost-Metrik zu. Für beide Metriken gilt, dass die mit DPVNE erzielten Ergebnisse sogar noch etwas besser werden. Der Grund für diese Verbesserung liegt in der Aufteilung des Substratnetzes in Partitionen während der Initialisierungsphase. Der Partitionierungsalgorithmus fasst Knoten zusammen, die sehr gut untereinander verbunden sind. Die Wahrscheinlichkeit, dass eine VNE-Einbettungsheuristik in der Lage ist, eine mögliche bzw. kosteneffiziente Einbettung in derartigen kleineren, gut miteinander vernetzten Teil-Topologien zu finden, ist im Allgemeinen höher als wenn dieselbe Heuristik mit dem unpartitionierten Gesamtnetz konfrontiert wird. Dieser Effekt macht sich allerdings nur geringfügig bemerkbar. Sowohl die Akzeptanz- als auch die Revenue/Cost-Metrik unterscheidet sich nur geringfügig (zwischen 5 und 10%). Insbesondere verschlechtert sich



(a) Ursprüngliche Algorithmen (b) DPVNE

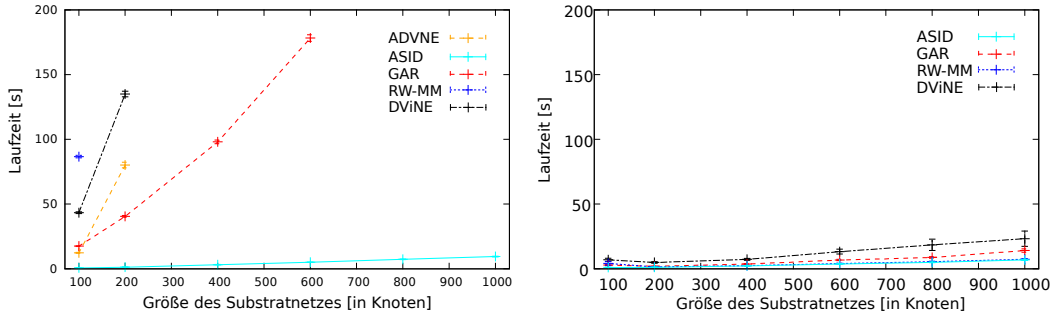
Abb. 4.8: Vergleich der Revenue/Cost

die Akzeptanz aufgrund des von DPVNE angewandten Delegationsvorgehens nicht. Das liegt daran, dass DPVNE in Situationen, in denen ein Einbettungsversuch innerhalb einer Partition scheitert, die Anfrage auch an übergeordnete Partitionen weiterdelegiert, bis schließlich die oberste Hierarchieebene erreicht ist. Der Einbettungsknoten der obersten Ebene versucht schlussendlich, die Einbettung innerhalb des Gesamtnetzes vorzunehmen. Dies entspricht dann dem Vorgehen des unmodifizierten VNE-Algorithmus. Beide Abbildungen zeigen ebenfalls die Ergebnisse für den ADVNE-Algorithmus, der in beiden Fällen jeweils am schlechtesten abschneidet.

4.3.3.2 Evaluation der Skalierbarkeit von DPVNE

Im Folgenden wird DPVNE im Zusammenhang mit verschieden großen Substratnetzen (100 bis 1000 Knoten) evaluiert, in die jeweils 20 virtuelle Netze mit je 5 Knoten eingebettet werden sollen. Ferner wird gezeigt, wie extrem sich das Laufzeitverhalten einiger nicht-verteilter VNE-Algorithmen in Abhängigkeit von der Größe des Substratnetzes verschlechtert.

Laufzeit: Abbildung 4.9a zeigt Laufzeitmessungen für die ursprünglichen Algorithmen, und Abbildung 4.9b stellt diesen Ergebnissen die Laufzeitmessungen für DPVNE in denselben Szenarien gegenüber. Die Ausführung der Algorithmen wurde nach Überschreitung von 200 Sekunden terminiert. Aus diesem Grund sind die in Abbildung 4.9a gezeigten Werte für einige Algorithmen lückenhaft. Beispielsweise ist der DViNE-Algorithmus nicht in der Lage, für Substratnetze mit 300 oder mehr Knoten Einbettungen innerhalb dieser Zeitspanne zu berechnen. Der RW-MM-Algorithmus überschreitet dieses Limit sogar schon ab einer Substratnetzgröße von 200 Knoten. Der einzige dieser nicht-verteilten Algorithmen, der in der Lage ist, alle Einbettungen innerhalb des gesetzten Zeitrahmens effizient zu berechnen, ist der Backtracking-basierte, heuristische ASID-Ansatz.
Abbildung 4.9b zeigt Messergebnisse für das DPVNE-Framework. DPVNE ist



(a) Ursprüngliche Algorithmen (b) DPVNE

Abb. 4.9: Laufzeitmessungen für Szenarien mit großen Substratnetzen

in der Lage, alle Berechnungen innerhalb weniger Sekunden durchzuführen. Die durchschnittliche Berechnungszeit liegt für alle Algorithmen unterhalb von 40 Sekunden – und das trotz der Tatsache, dass alle Berechnungen simulationsbedingt von nur einem Rechner durchgeführt wurden.
Die kürzeren Laufzeiten sind eine Folge der Partitionierung der Substratnetze. Anhand der in Abschnitt 4.2.2.1 beschriebenen Heuristik werden eintreffende Anfragen an im Bezug auf das Gesamtnetz kleineren Partitionen weitergereicht. Der VNE-Algorithmus greift dann zur Berechnung der Einbettung nur noch auf eine Teilmenge aller Substratknoten und -links zurück, was zu einer Reduktion der individuellen Problemgröße führt. Dadurch erfolgt die Berechnung des VNE-Algorithmus in kürzerer Zeit.
Tatsächlich übertrifft DPVNE alle hier evaluierten VNE-Algorithmen bezüglich der Laufzeit. Selbst für den ASID-Algorithmus lässt sich eine signifikante Beschleunigung erzielen, wenn er innerhalb des DPVNE-Frameworks ausgeführt wird. Zwar ist ASID in den hier untersuchten Szenarien deutlich schneller als die anderen in Abbildung 4.9a gezeigten VNE-Algorithmen – in Kombination mit dem DPVNE-Framework lässt sich aber auch in diesem Fall eine Beschleunigung messen: Für Substratnetze mit 1000 Knoten ist DPVNE auch hier im Durchschnitt 73% schneller. Aus diesem Grund kann davon ausgegangen werden, dass sich für ASID weitaus deutlichere Unterschiede bezüglich des Laufzeitverhaltens in Zusammenhang mit größeren Netzen feststellen lassen – insbesondere in Cloud-Szenarien, in denen parallel sehr viele Einbettungsanfragen zu bearbeiten sind. In solchen Szenarien kann DPVNE den Berechnungsaufwand dann, anders als ASID, auf gleich mehrere Rechner verteilen.

An dieser Stelle sei angemerkt, dass die Akzeptanz von DPVNE in diesem Szenario durchgehend bei 100% lag. Die Evaluationsergebnisse spiegeln daher den tatsächlichen Berechnungsaufwand für die erfolgreiche Einbettung aller virtuellen Netze wider. Dadurch, dass die Einbettung in der Regel nicht mehr auf Basis des Gesamtnetzes berechnet wird, sondern innerhalb kleinerer Partitionen, verringert sich die Problemgröße, was zu den hier feststellbaren schnelleren

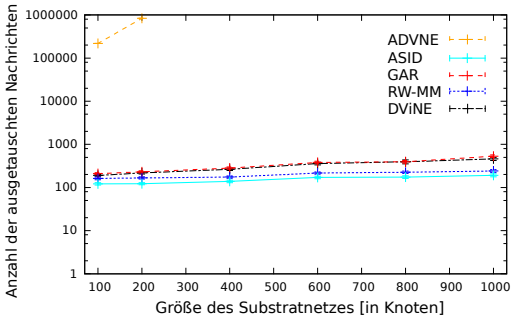


Abb. 4.10: Kommunikations-Overhead in Szenarien mit großen Substratnetzen

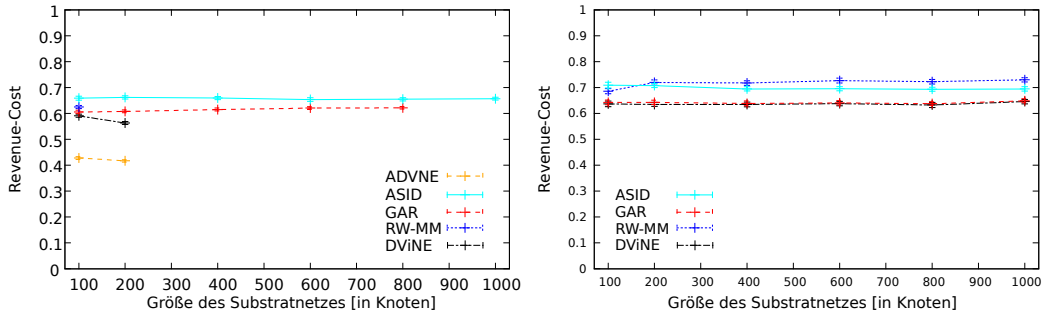
Berechnungsergebnissen führt.

Kommunikations-Overhead in großen Netzen: Abbildung 4.10 stellt den Kommunikations-Overhead von DPVNE dem des verteilten ADVNE-Algorithmus gegenüber. Während DPVNE in allen Szenarien mit weniger als 200 Nachrichten auskommt, werden von den verteilten Agenten des ADVNE-Ansatzes signifikant mehr Nachrichten ausgetauscht. Tatsächlich übersteigt der Kommunikations-Overhead von ADVNE schon bei Substratnetzen mit 200 Knoten die Marke von 800 000 Nachrichten.
Aufgrund des massiven Anstiegs der von ADVNE versendeten Nachrichten wurde dieser Algorithmus für Substratnetze mit mehr als 200 Knoten nicht mehr evaluiert: die Alevin-Simulationsumgebung war für diesen Algorithmus nicht in der Lage, mit einem derartigen massiven Nachrichtenaufkommen umzugehen und stürzte aufgrund von Speicherproblemen für größere Szenarien regelmäßig ab.

Einbettungskosten: Abbildung 4.11 veranschaulicht die Revenue-Cost-Werte dieser Szenarien. Ähnlich wie bei den in Abbildung 4.8 gezeigten Ergebnissen sind auch hier die mit DPVNE erzielten Resultate sogar geringfügig besser als diejenigen, die mit den ursprünglichen Algorithmen berechnet wurden. Wie schon in den vorherigen Szenarien bewegen sich die Unterschiede allerdings lediglich im Bereich von 5-10%.

4.3.3.3 Evaluation der Abschätzungsheuristik

Im Allgemeinen hängt die Skalierbarkeit eines verteilten Algorithmus von zwei Faktoren ab: Zum einen der effizienten Lastverteilung auf möglichst viele Knoten und zum anderen der Überschaubarkeit des zur Synchronisation der Knoten notwendigen Kommunikationsaufwands.
Der ω-Parameter der von DPVNE verwendeten Abschätzungsheuristik hat Einfluss auf beide Faktoren. Über ihn lässt sich einstellen, wie optimistisch



(a) Ursprüngliche Algorithmen (b) DPVNE

Abb. 4.11: Vergleich der Einbettungskosten für Szenarien mit großen Substratnetzen

beziehungsweise pessimistisch die von DPVNE verwendete Heuristik die Kapazitäten der Partitionen einschätzt: Je größer der gewählte ω-Wert, desto pessimistischer fallen die Beurteilungen der Heuristik aus, vgl. Abschnitt 4.2.2.1. Im Folgenden werden die Auswirkungen von ω auf das Verhalten des DPVNE-Algorithmus untersucht. In jedem Experiment sollen jeweils 5 virtuelle Netze mit je 5 virtuellen Knoten in ein Substratnetz mit 50 Knoten eingebettet werden.

Kommunikations-Overhead: Abbildung 4.12 zeigt, wie sich der Kommunikations-Overhead von DPVNE für verschiedene Werte von ω verändert. Es lässt sich erkennen, dass die Evaluationswerte für alle vier Algorithmen ähnlich bleiben. Für größer werdende ω-Werte nimmt die Anzahl der ausgetauschten Nachrichten ab. Am stärksten ist dieser Effekt zwischen $\omega = 1$ und $\omega = 5$ zu beobachten. Fällt die Abschätzung zu optimistisch aus, resultiert dies in einem höheren Nachrichtenaufkommen. In diesem Fall wählt die Heuristik viele Einbettungsknoten aus, die nicht in der Lage sind, die Einbettungsanfragen in die eigene Partition einzubetten. Als Konsequenz entsteht zusätzlicher Kommunikations-Overhead, da DPVNE in diesen Fällen zunächst vergeblich versucht, die Einbettung in wenig geeigneten Partitionen vorzunehmen.
Für $\omega \geq 15$ reduziert sich die Anzahl der ausgetauschten Nachrichten nur noch wenig und bleibt identisch für alle verwendeten VNE-Algorithmen. Das liegt daran, dass die Heuristik in diesen Fällen direkt an Partitionen mit ausreichend Hardwareressourcen verweist, für die alle VNE-Algorithmen in der Lage sind, unmittelbar gültige Einbettungen zu finden.

Parallelität: Die Wahl eines möglichst großen ω-Wertes reduziert zwar das Nachrichtenaufkommen auf ein Minimum, allerdings führt dies auch dazu, dass DPVNE eintreffende Anfragen in zunehmend größeren Partitionen verarbeitet. Für $\omega = \infty$ würde DPVNE stets alle Anfragen an den obersten Einbettungsknoten der Hierarchie weiterleiten. Das würde dazu führen, dass DPVNE stets

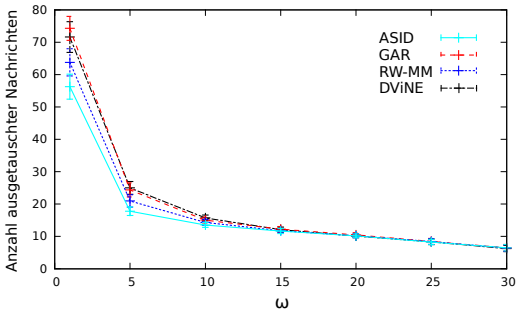


Abb. 4.12: Kommunikations-Overhead

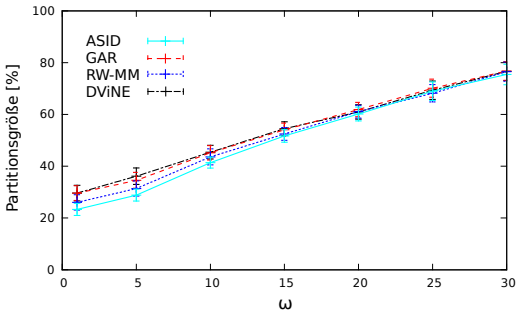


Abb. 4.13: Größe der Partition, in der die Einbettung erfolgte (prozentual in Bezug zur Größe des vollständigen Substratnetzes)

alle Einbettungen von nur einem einzelnen Knoten auf Basis des gesamten Substratnetzes berechnen lässt. Dieses Vorgehen entspricht aber dem Vorgehen der ursprünglichen, nicht-verteilten Algorithmen. Um diesen Zusammenhang zu veranschaulichen, wird in Abbildung 4.13 die durchschnittliche Partitionsgröße dargestellt, in der die virtuellen Netze erfolgreich eingebettet werden konnten. Hier zeigt sich ein linearer Zusammenhang zwischen dem Wert von ω und der Partitionsgröße. In der Abbildung ist die Partitionsgröße prozentual zur Größe des vollständigen Substratnetzes angegeben.
Je höher der hier visualisierte Wert, desto größer ist der Teil des Substratnetzes, der für die Durchführung der Einbettungsberechnungen gesperrt wird. In diesen Substratnetz-Bereichen dürfen also parallel keine weiteren Anfragen verarbeitet werden. Für $\omega = 1$ werden ca. 30% des Substratnetzes gesperrt, sodass im Durchschnitt dreimal so viele VNRs parallel verarbeitet werden können. Für $\omega = 15$ werden bereits 50% in Anspruch genommen, sodass im Schnitt lediglich zwei Einbettungsknoten parallel arbeiten können.

Laufzeit: Abbildung 4.14 zeigt bemerkenswerte Auswirkungen des ω-Parameters auf die Laufzeit. Wie hier zu sehen ist, nimmt die Laufzeit mit steigendem ω-Wert tendenziell zu. Eine Ausnahme ist hier wie auch zuvor schon der ASID-Algorithmus, was daran liegt, dass dieser Algorithmus in Szenarien dieser Größenordnung gut zurechtzukommen scheint. Dass die Laufzeit

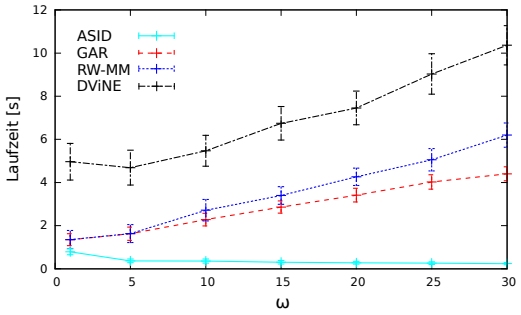


Abb. 4.14: Laufzeit

von DPVNE im Zusammenhang mit allen anderen Algorithmen größer wird, lässt sich mit der steigenden Größe der Partitionen erklären, die von DPVNEs Heuristik zur Berechnung der Einbettungen ausgewählt werden.

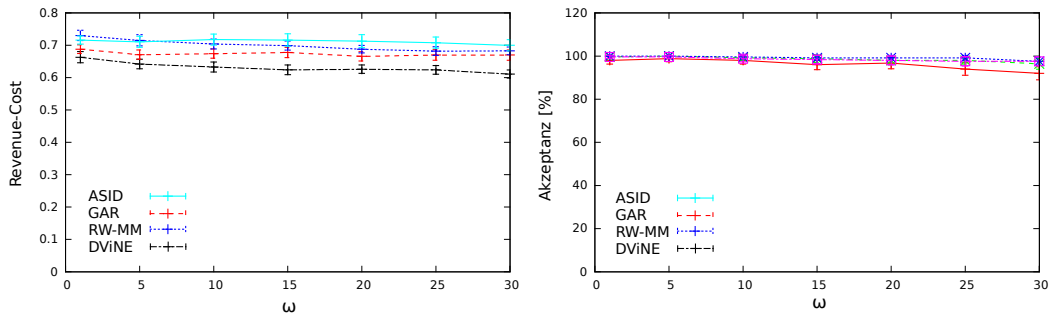
Akzeptanz und Einbettungskosten: Der ω-Parameter hat keine Auswirkungen auf die Akzeptanz, da virtuelle Netze, deren Einbettung in einer Paritition fehlschlägt, sukzessive an andere Partitionen weiterdelegiert werden, vgl. Abbildung 4.15b. Abbildung 4.15a zeigt die Werte der Revenue/Cost-Metrik. Hier wird ersichtlich, dass ω keine bemerkenswerten Auswirkungen auf die Einbettungskosten hat; die Kosten sind für $\omega = 1$ nur zu einem vernachlässigbaren, minimalen Grad geringer.

Zusammenfassend kann $\omega = 10$ als guter Kompromiss bezüglich des Nachrichten-Overheads, des Parallelitätsgrades und des gemessenen Laufzeitverhaltens in diesen Szenarien angesehen werden.

4.3.3.4 Diskussion der Evaluationsergebnisse

Interessanterweise hat der ω-Parameter keinen so starken Einfluss wie dies ursprünglich erwartet wurde. $\omega = 10$ hat sich in den untersuchten Szenarien als ein guter Kompromiss zwischen den unterschiedlichen Evaluationsmetriken ergeben – in der Tat ein auf den ersten Blick recht großer Wert. Schließlich bedeutet dies, dass im Durchschnitt 10-mal so viele Ressourcen des Substratnetzes für die erfolgreiche Einbettung der virtuellen Netze nötig waren. Allerdings muss hierbei berücksichtigt werden, dass eine Partition für die erfolgreiche Einbettung eines Netzes nicht nur über ausreichende CPU- und Bandbreitenressourcen verfügen muss, sondern dass dabei insbesondere auch die Topologie der Partition eine entscheidende Rolle spielt. Dies kann als Grund dafür gesehen werden, dass der ω-Wert entsprechend höher gewählt werden muss.

Die Alevin-Simulationsumgebung führt alle Einbettungsberechnungen auf einem einzigen Rechner aus. Aufgrund dieser Tatsache werden eintreffende Ein-



(a) Einbettungskosten (b) Akzeptanz

Abb. 4.15: Einbettungskosten und Akzeptanz in Abhängigkeit vom gewählten ω-Wert

bettungsanfragen sequentiell verarbeitet. Wie in diesem Abschnitt gezeigt, führt die von DPVNE durchgeführte hierarchische Partitionierung des Substratnetzes sowie die verwendete Delegationsmethodik allein schon zu einer deutlichen Beschleunigung der Einbettungsberechnungen – trotz der Simulation der Szenarien auf nur einem physischen Rechner. Eine Ausnahme war hier der backtracking-basierte ASID-Algorithmus, der insgesamt gute Evaluationsergebnisse in den diskutierten Szenarien zeigte.
Eine Beschleunigung des Einbettungsvorgangs durch den Einsatz von DPVNE lässt sich in Zusammenhang mit ASID nur in einem Testbed mit mehreren verteilten Rechnern nachweisen. Mit dem DPVNE-Framework lassen sich die Einbettungsberechnungen dann auf mehrere Rechner verteilen – das ist gerade in Cloud-Szenarien, in denen viele kontinuierlich eintreffende Einbettungsanfragen verarbeitet werden müssen, sinnvoll. Neben ASID können dabei auch andere Einbettungsstrategien verwendet werden, die ohne DPVNE in derart großen Szenarien gar nicht erst anwendbar wären. Die Ergebnisse eines solchen Experiments, in dem DPVNE in einem Testbed auf mehreren verteilten Knoten installiert wurde, sind in Abbildung 4.16 gezeigt. Hier wird der ursprüngliche ASID-Algorithmus dem DPVNE-Algorithmus gegenübergestellt, welcher ASID auf 7 Rechnern verteilt ausführt. Diese Abbildung visualisiert Messungen, die von Herrn Stephan Holzner im Rahmen seiner Bachelorarbeit durchgeführt wurden [63]. Hier wurden 20 virtuelle Netze mit je 10 Knoten in ein Substratnetz mit 5000 Knoten eingebettet. Als Delegationsknoten wurden die Knoten der Ebene 1 ausgewählt. Es zeigt sich, dass der verteilte DPVNE-Ansatz hier sechs Mal schneller ist als der nicht-verteilte ASID-Algorithmus.

Im Vergleich zum verteilten ADVNE-Algorithmus zeigt sich, dass der Kommunikations-Overhead mit DPVNE wesentlich geringer ausfällt und die erzielten Einbettungsergebnisse deutlich besser sind. Zudem ist DPVNE, anders als ADVNE, ein generisches Framework, welches zur Parallelisierung von Einbettungsalgorithmen mit verschiedensten Optimierungskriterien verwendet

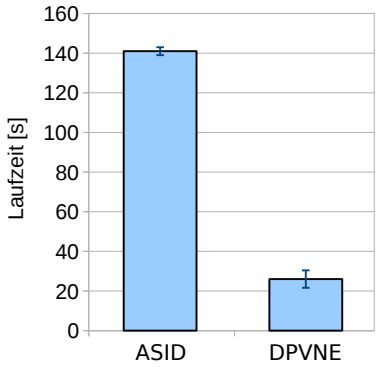


Abb. 4.16: Laufzeit von ASID mit und ohne DPVNE

werden kann.

4.3.4 Ausfallsicherheit

Der Fokus dieses Kapitels liegt auf der Analyse von DPVNE hinsichtlich des Laufzeitverhaltens, der Akzeptanz und der Einbettungskosten. Nichtsdestoweniger ist die Berücksichtigung von möglichen Fehlern Voraussetzung für den Einsatz des verteilten DPVNE-Algorithmus in einem realen Umfeld. Denn das Thema Ausfallsicherheit spielt eine zentrale Rolle in verteilten Systemen. Im Folgenden wird diskutiert, inwiefern sich Ausfälle auf die unterschiedlichen Komponenten des DPVNE-Ansatzes auswirken. Ausgehend von dieser Diskussion lassen sich entsprechende Gegenmaßnahmen ableiten.

Ausfall des Load Balancers: Fällt der Load Balancer aus, können eintreffende Anfragen nicht an die Delegationsknoten verteilt werden. Das bedeutet, dass DPVNE nicht mehr in der Lage ist, weitere Anfragen zu verarbeiten. Zudem können die von den Einbettungsknoten berechneten Ergebnisse nicht nach außen, d.h. an die Serviceprovider, die die Einbettungen in Auftrag geben, kommuniziert werden. Aus diesem Grund muss der Infrastrukturprovider sicherstellen, dass Ausfälle des Load Balancers erkannt und zeitnah behandelt werden können. Hier eignet sich zum Beispiel die Integration eines Hot Spares, dessen Zustand synchron zu dem des Load Balancers gehalten wird.

Ausfall von Delegationsknoten: Nach Ausfall eines Delegationsknotens können zukünftige Anfragen nicht mehr an die dem betroffenen Knoten untergeordneten Einbettungsknoten weitergegeben werden. Ferner steht der Delegationsknoten selbst nicht mehr zur Berechnung von Einbettungen zur Verfügung. Einbettungsknoten, an die der Delegationsknoten bereits vor dem Ausfall Anfragen delegiert hat, sind außerdem nicht mehr in der Lage, berechnete Ergebnisse weiterzureichen. Das bedeutet, dass der gesamte Teilbereich des Substratnetzes, der dem Delegationsknoten zugeordnet ist, nur noch von den

Einbettungsknoten genutzt werden kann, die dem Delegationsknoten übergeordnet sind. Beim Einsatz von DPVNE in einem realen Netzwerk sollte also sichergestellt werden, dass bereits berechnete, aber nicht an den Load Balancer übermittelbare Ergebnisse für ungültig deklariert werden und die betroffenen Anfragen von anderen, intakten Delegationsknoten übernommen werden. Alternativ dazu eignet sich auch hier der Einsatz eines Hot Spares, das den Delegationsknoten zur Seite gestellt und synchron gehalten wird.

Ausfall von Dienstknoten: Fällt einer der Dienstknoten aus, steht er dem ihm übergeordneten Delegationsknoten nicht weiter zur Verfügung. Der Delegationsknoten muss daher zukünftige Anfragen an andere Dienstknoten weiterreichen. Zudem müssen Anfragen, die bereits an den betroffenen Dienstknoten delegiert wurden und deren Einbettungsergebnis dem Delegationsknoten noch nicht vorliegt, erneut bearbeitet werden. Andernfalls würde weder der Delegationsknoten, der die Einbettung in Auftrag gegeben hat, noch der Load Balancer ein Einbettungsergebnis erhalten.

Ausfall übergeordneter Knoten: Nach Ausfall eines übergeordneten Knotens können Anfragen, die von den Delegationsknoten an Vorgängerknoten weitergereicht werden, vom betroffenen übergeordneten Knoten nicht mehr angenommen und verarbeitet werden. Betroffen von diesem Ausfall sind daher all diejenigen Delegationsknoten, die in der Delegationshierarchie Kinder oder Kindeskinder des ausgefallenen Knotens sind. Ferner können Delegationsknoten, die angehalten wurden und auf eine START-Nachricht warten, den Betrieb nicht wieder aufnehmen, wenn ein übergeordneter Knoten ausfällt.

4.4 Zusammenfassung

DPVNE ist ein generisches Framework, mit dem sich die Ausführung verschiedener VNE-Einbettungsalgorithmen parallelisieren lässt. Der Berechnungsaufwand wird auf mehrere Knoten verteilt, die parallel an der Bearbeitung der Einbettungsanfragen arbeiten. Jeder dieser Einbettungsknoten ist dabei für einen bestimmten Teilbereich des Substratnetzes verantwortlich.
In diesem Kapitel wurde zunächst beschrieben, wie das Substratnetz mit Hilfe von Partitionierungsverfahren in kleinere, hierarchisch strukturierte Partitionsbereiche aufgeteilt werden kann. Anschließend wurde eine Heuristik vorgestellt, über die sich bei Eintreffen neuer Einbettungsanfragen abschätzen lässt, welche Partitionen für die Berechnung der Einbettungen geeignet sein könnten. Über einen Parameter ω kann die Heuristik so eingestellt werden, dass die von ihr durchgeführten Abschätzungen optimistischer oder pessimistischer ausfallen. Die Evaluationsergebnisse zeigen, dass $\omega = 10$ eine gute Konfiguration für diesen Parameter darstellt.
Aufbauend auf diesen Partitionierungsüberlegungen wurde beschrieben, wie sich der Berechnungsaufwand, der mit der Einbettung virtueller Net-

ze verbunden ist, auf mehrere Einbettungsknoten verteilen lässt. Dazu wurde in der Initialisierungsphase des Algorithmus jeder Partition ein Einbettungsknoten zugewiesen, der je nach zugeordneter Funktionalität (Delegationsknoten/Dienstknoten/übergeordneter Knoten) unterschiedliche Aufgaben übernimmt. Zur Unterteilung des Substratnetzes in mehrere autonome Bereiche können mehrere Einbettungsknoten als Delegationsknoten deklariert werden. Delegationsknoten nehmen eine „Anführerrolle“ ein. Sie koordinieren die Einbettung eintreffender Anfragen innerhalb ihres autonomen Bereichs und delegieren Einbettungsanfragen an die ihnen untergeordneten Dienstknoten. Ferner wurde das Zusammenspiel der verteilten Einbettungsknoten beschrieben. Insbesondere wurde dazu auf das von den Einbettungsknoten verwendete Kommunikationsprotokoll eingegangen und erörtert, wie sich die Konsistenz der lokalen Zustände der Einbettungsknoten sicherstellen lässt.

Im Zusammenhang mit DPVNE bietet sich die Untersuchung folgender weitergehender Fragestellungen an:

Partitionieren von virtuellen Netzen: DPVNE wurde für den Einsatz in Szenarien konzipiert, bei denen der Infrastrukturprovider viele virtuelle Netze in ein großes Substratnetz einbetten möchte. Dabei wird angenommen, dass die virtuellen Netze in der Regel deutlich kleiner sind als das Substratnetz selbst. Je größer ein virtuelles Netz ist, desto größer muss in der Regel auch die Partition gewählt werden, in die dieses Netz eingebettet wird. Mit der Größe der ausgewählten Partition reduziert sich allerdings auch direkt der Parallelitätsgrad des verteilten Systems: Die Einbettung virtueller Netze, deren Netzstrukturen und Ressourcenanforderungen vergleichbar sind mit der Topologie bzw. dem Ressourcenangebot des Substratnetzes, können auch von DPVNE nur in einem nicht-verteilten Prozess von einem einzelnen Knoten berechnet werden, nämlich dem Wurzelknoten in der Hierarchie der Einbettungsknoten.
Nichtsdestotrotz lässt sich die Anwendbarkeit des DPVNE-Ansatzes recht einfach auch auf Szenarien mit solch großen virtuellen Netzen erweitern. Dazu können die virtuellen Netze selbst mit Hilfe eines Partitionierungsverfahrens in kleinere Partitionen aufgeteilt werden. Diese Partitionen können dann, analog zum üblichen Einbettungsvorgehen, den Delegationsknoten zur weiteren Bearbeitung innerhalb der autonomen Netzbereiche übergeben werden. Bei der Einbettung der virtuellen Teilnetze muss jedoch sichergestellt werden, dass diese anschließend untereinander vernetzt werden können.

Berücksichtigung geographischer Einbettungsconstraints: Sollen bestimmte virtuelle Knoten oder Pfade beispielsweise aus Sicherheitsgründen nur innerhalb eines eingeschränkten regionalen Bereichs des Substratnetzes platziert werden, ist sowohl über eine Erweiterung der Auswahlheuristik als auch über die Verwendung eines alternativen Partitionierungsverfahrens nach-

zudenken. Andernfalls würden diese Anfragen an zahlreiche Einbettungsknoten delegiert, in deren Partitionen jegliche Einbettungsversuche aufgrund dieser zusätzlichen Nebenbedingungen zwangsläufig fehlschlagen müssten. Die Partitionierung sollte die geographische Platzierung der Substratknoten und -Links deshalb im Vorfeld mit einbeziehen und Knoten, die sich innerhalb desselben Ortes befinden, i.d.R. derselben Partition zuordnen.

5 Koordinierte Verkettung und Einbettung virtueller Netzwerkfunktionen

Ein in der Literatur noch recht wenig untersuchtes $\mathcal{NP}$-hartes Optimierungsproblem ist das Deployment virtueller Netzwerkservices mit *flexibler* Netztopologie. Nach bestem Wissen des Autors wurde in diesem Zusammenhang bislang nur ein einziger Deploymentalgorithmus vorgestellt, der in der Lage ist, solche flexiblen Netzwerkservices einzubetten. Allerdings ist dieser andere Ansatz nur in Szenarien mit sehr kleinen Netzwerkinfrastrukturen anwendbar. In diesem Kapitel wird daher ein heuristisches Verfahren vorgestellt, das NFV-Netzwerkservices mit flexibler Netzstruktur effizient auch innerhalb großer Substratnetzinfrastrukturen einbetten kann.

Im NFV-Kontext liegt der Fokus auf der Betrachtung von virtuellen Netzwerkservices, die Datenströme verarbeiten und sich aus einzelnen Netzwerkfunktionen (VNFs) zusammensetzen. Anders als in dem zuvor betrachteten Cloud-Umfeld steht die Topologie der einzubettenden Netzwerkservices in NFV-Szenarien nicht von vornherein fest. Jede VNF eines Netzwerkservices wendet bestimmte Operationen auf die an sie gesendeten Eingangsdaten an und leitet die verarbeiteten Daten an weitere VNF weiter. Konkrete Beispiele dafür sind Kompressions-, Inspektions-, Firewall- und Verschlüsselungsoperationen. Die Reihenfolge, in der die Datenströme durch die VNFs geroutet werden, ist nicht zwingend fest vorgeschrieben. Beispielsweise kann der Betreiber spezifizieren, dass ein Datenstrom zwar zunächst durch eine Kompressionsfunktion geleitet werden soll, bevor er verschlüsselt wird, und nicht in umgekehrter Reihenfolge. Ob der Datenstrom die Firewallfunktion vor oder nach der Verschlüsselungsstufe erreicht, spielt für den Betreiber u.U. aber keine Rolle. Die für jeden Netzwerkservice geltenden Verkettungsregeln spezifiziert der Betreiber innerhalb der jeweiligen VNFR-Einbettungsanfragen; sie müssen beim Deployment des Services beachtet werden.
Anders als bei VNE-Szenarien müssen in NFV-Szenarien also zwei Teilprobleme betrachtet werden:

1. **Die Verkettung von VNFs zu einem Netzwerkservice (*Chaining*-Teilproblem)**
   Die Reihenfolge, in der die Netzwerkfunktionen verkettet werden dürfen, ist durch die in den Einbettungsanfragen enthaltenen Abhängigkeitsspezifikationen eingeschränkt – die VNFs eines Netzwerkservices dürfen also

nicht in beliebiger Reihenfolge angeordnet werden, sondern nur entsprechend der Vorgaben des Serviceproviders.

2. **Das Deployment der verketteten VNFs (VNF-FGE-Teilproblem)**
   Die verketteten Netzwerkservices müssen in die Substratnetzinfrastruktur eingebettet werden. Im Unterschied zum VNE-Problem können mehrere Netzwerkfunktionen innerhalb derselben virtuellen Maschine untergebracht werden.

Dieses Optimierungsproblem wird in der Literatur als das NFV-RA-Problem bezeichnet – eine Abgrenzung gegenüber VNE-Szenarien wurde in Abschnitt 2.3.3 diskutiert.
Dieses Kapitel beschreibt einen heuristischen Ansatz zur effizienten Lösung des NFV-RA-Problems. Der hier vorgestellte Algorithmus nimmt die Verkettung und Einbettung in einem einzigen, koordinierten Schritt vor.

5.1 Abgrenzung zu verwandten Arbeiten

Die meisten aus der Literatur bekannten Arbeiten, die sich mit der Platzierung von VNFs beschäftigten, konzentrieren sich auf die Lösung des VNF-FGE-Teilproblems, also auf die Einbettung von bereits verketteten Netzwerkfunktionen. Hier wird das eigentliche Verketten der Netzwerkfunktionen nicht betrachtet. Stattdessen wird angenommen, dass jeder einzubettende Netzwerkservice durch genau einen vorgegebenen Netzwerkgraphen (VNF-FG) spezifiziert ist und die Reihenfolge, in denen die VNFs innerhalb dieses Graphen angeordnet werden, fest vorgeschrieben ist.
Bari et al. präsentieren ein solches Verfahren, das auf ganzzahliger linearer Programmierung beruht (Integer Linear Programming, ILP) und in der Lage ist, das VNF-FGE-Teilproblem optimal zu lösen [12]. Optimierungsziel dabei ist die Reduktion der Einbettungskosten sowie die Vermeidung von Ressourcen-Fragmentierung. Da dieses optimale, auf linearer Programmierung beruhende Verfahren schlecht mit der Größe der Szenarien skaliert, stellen die Autoren zudem eine schnellere Heuristik vor, die auf dem Prinzip der dynamischen Programmierung basiert und auf den Viterbi-Algorithmus [101] zurückgreift. Die präsentierten Evaluationsergebnisse zeigen, dass dieser heuristische Ansatz je nach Szenario 65 bis 3500 mal schneller terminiert als das ILP-basierte optimale Verfahren, dafür aber im Vergleich 30% schlechtere Einbettungsergebnisse liefert.
Eine weitere Publikation, die von Luizelli et al. vorgestellt wurde, befasst sich ebenfalls mit der Lösung des VNF-FGE-Teilproblems und schlägt eine weitere, auf ganzzahliger linearer Programmierung basierende Formulierung vor [31]. Das Optimierungskriterium ist hier allerdings nicht die Reduktion der Einbettungskosten, sondern die Minimierung der Anzahl der zu instanziierenden virtuellen Maschinen. Darüber hinaus wird auch hier wieder ein heuris-

tischer Ansatz vorgeschlagen, um im Zusammenhang mit größeren Szenarien eine annähernd gute Lösung innerhalb akzeptabler Berechnungszeiten finden zu können.

Mijumbi et al. stellen gleich drei Greedy-Algorithmen sowie eine weitere, auf Tabu-Suche basierende Metaheuristik vor, mit der sich Einbettungen verketteter Netzwerkfunktionen im Kontext des VNF-FGE-Teilproblems heuristisch berechnen lassen [85].

In Abgrenzung zu diesen VNF-FGE-Ansätzen ist nach bestem Wissen des Autors neben dem in diesem Kapitel vorgestellten CoordVNF-Algorithmus bislang nur ein einziger Ansatz (MIVNF) bekannt, der beide Teilprobleme des NFV-RA-Problems gemeinsam betrachtet: Dies ist der von Mehraghdam et al. vorgestellte heuristische Ansatz [82], der das NFV-RA-Problem in zwei voneinander unabhängigen Schritten löst. Im ersten Schritt wird dazu die Verkettung der Netzwerkfunktionen vorgenommen. Im zweiten Schritt wird die optimale Einbettung der verketteten Funktionen berechnet. Die Einbettung wird mit Hilfe eines Mixed Integer Quadratic Constrained Programms (MIQCP) bestimmt. Im Gegensatz zu MILPs (Mixed Integer Linear Programs) sind MIQPs (Mixed Integer *Quadratic* Programs) Mixed Integer Programme, bei denen die Zielfunktion quadratische *Terme* enthält. Enthalten die *Bedingungen* (engl. Constraints) eines Programms wie der in [82] vorgestellte Ansatz quadratische Terme, wird das Programm MIQCP (Mixed Integer *Quadratically Constrained* Programs) genannt.

Das von Mehraghdam et al. vorgestellte Verfahren greift zunächst auf eine Greedy-Strategie zurück, um eine günstige Verkettung der Netzwerkfunktionen zu bestimmen. Nach Abschluss dieser ersten Phase wird auf eine MIQCP-Formulierung zurückgegriffen, mit der sich die optimale Einbettung der Netzwerkfunktionen berechnen lässt. Dabei werden drei verschiedene Optimierungskriterien angegeben, die wahlweise bei der Lösung des VNF-FGE-Teilproblems berücksichtigt werden können: a) die Maximierung der innerhalb des Substratnetzes verbleibenden Bandbreite, b) die Minimierung der Anzahl der Substratknoten, die verwendet werden, mit dem Ziel, einen möglichst energieeffizienten Betrieb der Netzwerkfunktionen zu ermöglichen, sowie c) die Minimierung der Kommunikationslatenzen.

Wie später noch gezeigt wird, ist MIVNF allerdings nur im Zusammenhang mit sehr kleinen Netztopologien anwendbar und skaliert aufgrund der Formulierung des VNF-FGE-Teilproblems als MIQCP nicht in Szenarien realer Größenordnung. Ferner ist durch die Aufteilung des NFV-RA-Problems in zwei voneinander getrennte Schritte nicht garantiert, dass eine optimale Lösung für das Gesamtproblem gefunden werden kann – und das trotz der Tatsache, dass MIVNF das VNF-FGE-Teilproblem optimal lösen kann. Es kann sogar vorkommen, dass es bedingt durch diese Trennung der Teilprobleme zu Situationen kommt, in denen Netzwerkservices aufgrund der im ersten Schritt vorgenommenen Verkettung der Netzwerkfunktionen als

nicht einbettbar zurückgewiesen werden, obwohl die Einbettung desselben Netzwerkservices grundsätzlich möglich wäre, wenn im ersten Schritt auf eine alternative Verkettung zurückgegriffen wird.

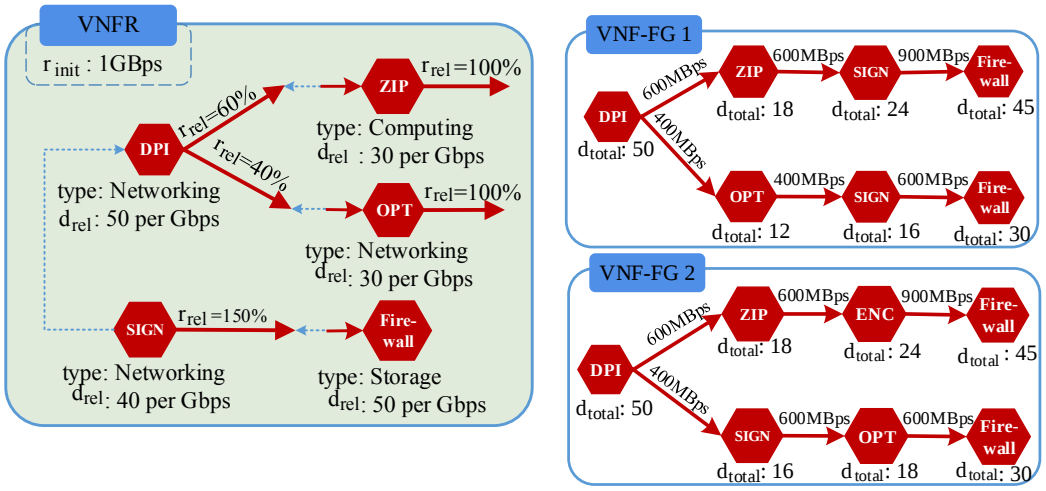
Im Gegensatz zu den in der Literatur vorgestellten Verfahren handelt es sich bei dem im Folgenden vorgestellten CoordVNF-Algorithmus um einen heuristischen Ansatz, der *beide* Teilprobleme *koordiniert* löst (CoordVNF steht für *Coordinated embedding of Virtual Network Functions*). Die Verkettung der Netzwerkfunktionen wird gleichzeitig mit der Zuweisung der Substratressourcen durchgeführt, wodurch noch während der laufenden Einbettungsphasen auf alternative Konfigurationen der einzubettenden Netzwerkservices zurückgegriffen werden kann. Bei CoordVNF handelt es sich um einen Backtracking-basierten Ansatz, der in jedem rekursiven Teilschritt versucht, eine weitere VNF-Instanz zuzuweisen; gelingt dies einmal nicht, wird mittels Backtrackings auf einen vorherigen gültigen Zustand zurückgesprungen, woraufhin alternative Verkettungs- und Zuweisungsoptionen geprüft werden. Bei der Evaluation von DPVNE im vorherigen Kapitel zeigte sich, dass der ebenfalls Backtracking-basierte ASID-Algorithmus [78] in vielen Szenarien sehr performant ist. Inspiriert von ASID wurde CoordVNF daher auch als Backtracking-Algorithmus konzipiert, der einige in [78] diskutierten Ideen aufgreift.

CoordVNF wurde sowohl auf der IEEE GLOBECOM Konferenz vorgestellt [14] als auch im Elsevier Computer Communications Journal publiziert [15].

5.2 Der CoordVNF-Algorithmus

Dieses Kapitel stellt den CoordVNF-Algorithmus vor, der das NFV-RA-Problem in einem koordinierten Schritt löst: Statt erst eine mögliche Verkettung der Netzwerkfunktionen zu bestimmen, deren Einbettung im Anschluss berechnet wird, löst CoordVNF beide Teilprobleme gleichzeitig. Diese koordinierte Vorgehensweise beschleunigt das Finden von validen Lösungen: Kann eine VNF-Instanz innerhalb des Substratnetzes nicht zugewiesen werden, würde ein unkoordinierter Algorithmus den Einbettungsvorgang abbrechen. Die Anfrage würde dann entweder als nicht-einbettbar zurückgewiesen oder es würde versucht, den Einbettungsschritt auf Basis einer anderen Verkettung der VNFs zu wiederholen. Im Gegensatz dazu ist CoordVNF in der Lage, bereits während des laufenden Einbettungsvorgangs alternative Kombinationsmöglichkeiten der VNFs zu berücksichtigen.

Zur Veranschaulichung wird die generelle Funktionsweise von CoordVNF im Folgenden zuerst anhand eines vereinfachten Beispiels vorgestellt. Im Anschluss daran wird der Algorithmus im Detail beschrieben.



(a) NFV-Einbettungsanfrage (VNFR)

(b) Verkettung von Netzwerkfunktionen

Abb. 5.1: Eine Einbettungsanfrage, deren VNFs flexibel verkettet werden können.

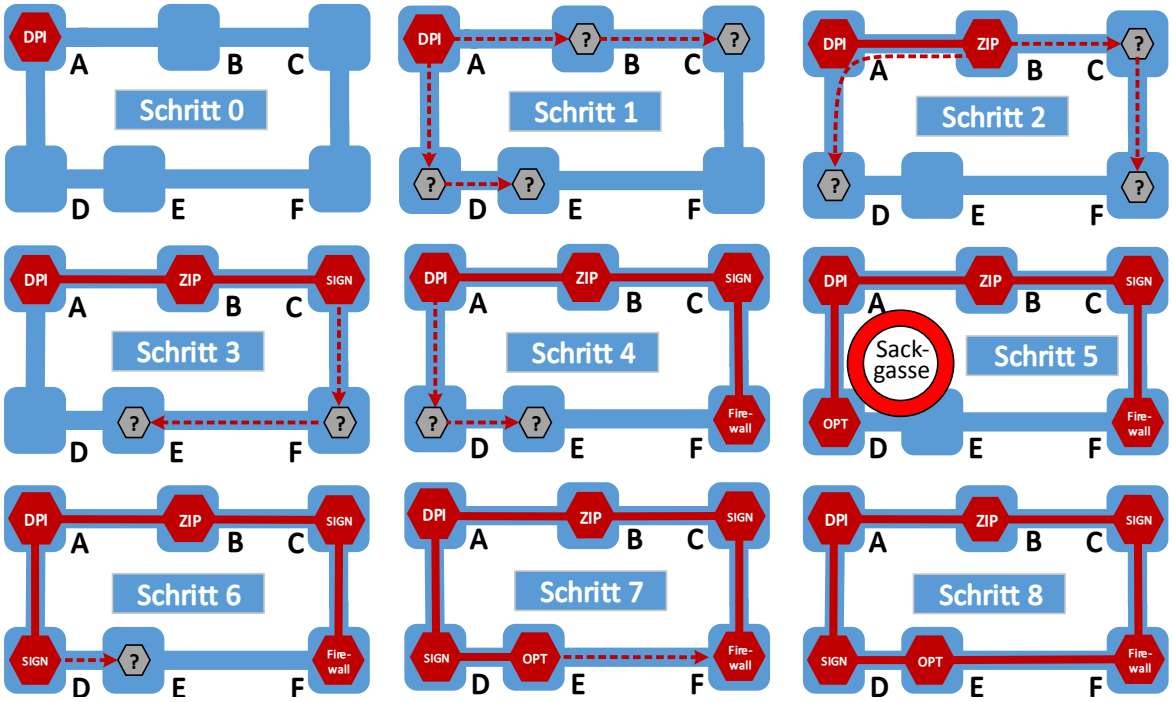


Abb. 5.2: Veranschaulichung der koordinierten Vorgehensweise des CoordVNF-Algorithmus

5.2.1 Generelle Funktionsweise

Im Folgenden wird das generelle Vorgehen des CoordVNF-Ansatzes anhand des in Abbildung 5.1 gezeigten Szenarios beschrieben. Abbildung 5.2 veranschaulicht, wie der Algorithmus die beiden Teilprobleme in neun Schritten löst. Für dieses Beispiel wird vereinfachend angenommen, dass ein Substratknoten maximal eine VNF-Instanz aufnehmen kann, d.h. $\text{host}(n) = 1, \forall n \in N$. Ferner wird die Suchtiefe auf eine maximale Pfadlänge von zwei Substratlinks beschränkt.

In Schritt 0 wählt CoordVNF die `DPI`-VNF als erste einzubettende VNF aus, da diese VNF laut Spezifikation des VNFR von keinen anderen VNFs abhängig ist. Anschließend versucht der Algorithmus, die `DPI`-VNF auf einem passenden, zufällig gewählten Substratknoten zu instanziieren. Da Substratknoten A über genügend Ressourcen zum Betrieb der VNF verfügt, weist CoordVNF die `DPI`-VNF dem Knoten A zu. Die `DPI`-VNF verfügt über zwei ausgehende Links, d.h., sie teilt den Netzwerkfluss in zwei Teilflüsse auf. Es wird nun versucht, den ersten Teilfluss einzubetten. Dazu bestimmt CoordVNF wiederum alle VNFs, deren Abhängigkeiten innerhalb des aktuellen Teilflusses erfüllt sind (`ZIP` und `SIGN`). CoordVNF startet nun, ausgehend von Knoten A, eine Breitensuche, um mögliche Zuweisungskandidaten für diese beiden VNFs zu identifizieren, hier Knoten B, C, D und E, vgl. Schritt 1. Anschließend entscheidet sich CoordVNF für die Zuweisung *einer* dieser VNFs. Dazu muss einerseits sichergestellt werden, dass auf dem Pfad zwischen dem vorherigen Substratknoten und dem Kandidatenknoten genügend Bandbreite zur Verfügung steht, andererseits aber die Kandidatenknoten selbst über ausreichende Rechen-/Netzwerk-/Speicher-Kapazitäten verfügen. Es wird angenommen, dass sich CoordVNF für die Zuweisung der `ZIP`-VNF auf Knoten B entscheidet (vgl. Schritt 2). Der Algorithmus verfährt analog mit den weiteren VNFs des Teilflusses: In Schritt 2 wird anschließend versucht, die `SIGN`-VNF in der Umgebung von B zu platzieren; dazu führt CoordVNF wiederum eine Breitensuche nach passenden Kandidatenknoten durch (C, D und F). Die `SIGN`-Funktion wird auf Knoten C platziert (vgl. Schritt 3), woraufhin nach einer Möglichkeit gesucht wird, die `Firewall`-VNF einzubetten. `SIGN` wird in Schritt 4 auf Knoten F zugewiesen; laut Spezifikation des VNFR müssen zur Vervollständigung des Teilflusses anschließend keine weiteren VNFs angefügt werden. Der von der `DPI`-Funktion abgespaltene erste Teilfluss ist nach diesem Schritt also vollständig zugewiesen.
CoordVNF springt nun zurück zu Knoten A, auf den die `DPI`-VNF eingebettet wurde und beginnt mit der Zuweisung des zweiten Teilflusses. CoordVNF versucht nun, eine `OPT`- oder `SIGN`-Instanz in den zweiten Teilfluss zu integrieren. An dieser Stelle kommen dazu die Knoten D und E in Betracht. CoordVNF entscheidet sich für die Platzierung der `OPT`-Funktion auf Knoten D (vgl. Schritt 5).
Zur Veranschaulichung sei nun angenommen, dass Knoten E keine Instanzen

der `SIGN`-Funktion aufnehmen kann. In diesem Fall befindet sich CoordVNF in einer Sackgasse und ist nicht in der Lage, die Zuweisung des zweiten Teilflusses zu vervollständigen. Aus diesem Grund wird mittels Backtracking auf einen der vorherigen gültigen Zustände zurückgesprungen und eine alternative Zuweisungsentscheidung getroffen. In dem hier gezeigten Szenario wird statt der `OPT`-VNF eine weitere Instanz der `SIGN`-VNF auf Knoten D eingebettet (vgl. Schritt 6). In Schritt 7 kann nun `OPT` auf Knoten E platziert und in Schritt 8 mit der `Firewall`-VNF verbunden werden.
Eingebettet wurde also nicht der in Abbildung 5.1 gezeigte VNF-FG 1, sondern VNF-FG 2.

5.2.2 Detaillierte Beschreibung

Algorithmus 5 zeigt den CoordVNF-Algorithmus in Pseudo-Code. Zur Berechnung einer Einbettung wird initial die CoordVNF-Funktion aufgerufen. Diese Funktion ruft die Rec-Funktion auf, die versucht, rekursiv alle notwendigen VNFs zu instanziieren.
Die Parameter der Rec-Funktion sind die Folgenden:

- VNFR^i: die einzubettende VNFR-Anfrage
- G: Das Substratnetz
- $\text{node}_{\text{previous}}$: Der Substratknoten, dem die im vorherigen rekursiven Schritt eingebettete VNF-Instanz zugewiesen wurde
- dr: Die absolute Datenrate des aktuellen Teilflusses
- *maxPathLength*: Die maximal zulässige Länge der Substratpfade, die zugewiesen werden dürfen
- *numCandidates*: Die maximale Anzahl der Einbettungskandidaten, die in jedem rekursiven Teilschritt berücksichtigt werden dürfen
- M: Die Menge aller Zuweisungen, die bislang vorgenommen wurden

Für die Einbettung der allerersten VNF wird zunächst ein zufälliger, geeigneter Substratknoten gewählt. Anschließend wird in jedem rekursiven Schritt die Liste der VNFs bestimmt, die entsprechend der innerhalb der VNFR-Anfrage spezifizierten Abhängigkeiten als nächstes in den aktuellen Teilfluss eingefügt werden können (Zeile 6). Sind bereits alle VNFs des Teilflusses erfolgreich zugewiesen worden, gibt die Rec-Funktion die Menge M als Teilergebnis zurück (Zeile 7).
Nun wird iterativ versucht, valide Zuweisungsmöglichkeiten für *eine* dieser VNFs zu finden. Dazu wird, ausgehend vom zuletzt zugewiesenen Substratknoten $\text{node}_{\text{previous}}$, eine Breitensuche innerhalb des Substratnetzes durchgeführt (vgl. Zeile 10). Die Breitensuche liefert *alle* geeigneten Einbettungskandidaten, die über genügend Ressourcen zum Betrieb der VNF-Instanz verfügen. Neben den Kapazitätsressourcen der Knoten müssen dabei auch die beschränkten Bandbreitenressourcen der Substratpfade berücksichtigt werden. Algorith-

mus 6 beschreibt das Vorgehen von CoordVNF bei der Suche nach geeigneten Einbettungskandidaten. Bei der Suche wird, ausgehend vom aktuellen Knoten $\text{node}_{\text{current}}$, allen Substratlinks gefolgt, die ausreichend Bandbreite zur Verfügung stellen (Zeile 19). Sobald ein geeigneter Einbettungskandidat gefunden wurde, wird er mitsamt des von $\text{node}_{\text{current}}$ ausgehenden Pfades der Ergebnisliste hinzugefügt, vgl. Zeile 31.
CoordVNF sortiert nun die möglichen Einbettungskandidaten in aufsteigender Reihenfolge nach den folgenden Sortierkriterien (Algorithmus 5, Zeile 11):

> *Sortierkriterium 1*: VNFs, die die Bandbreite des Datenstroms reduzieren, werden bevorzugt. Durch die möglichst frühe Platzierung solcher Netzwerkfunktionen reduziert sich die Bandbreite des gesamten Datenstroms.
>
> *Sortierkriterium 2*: Kandidaten, die sich über kürzere Pfade erreichen lassen, werden bevorzugt. Ziel dabei ist es, die Einbettungskosten in jedem rekursiven Schritt möglichst gering zu halten.

Anschließend wird versucht, eine der möglichen Zuweisungen durchzuführen (Zeile 12ff.); es wird also zum einen die VNF-Instanz auf den gefundenen Substratknoten eingebettet (MAP-NODE-DEMANDS, Zeile 13), zum anderen werden die Ressourcen der Kommunikationsverbindung auf den einzelnen Links des Substratpfades reserviert (MAP-LINK-DEMANDS, Zeile 14).
Sobald alle erforderlichen Teilflüsse einer VNF-Instanz vollständig eingebettet wurden, wird versucht, alle sich anschließenden Teilflüsse einzubetten (Zeile 17 ff.). Dies erfolgt durch rekursives Aufrufen der REC-Funktion für jeden der sich an die VNF-Instanz anschließenden Teilflüsse. Die REC-Funktion wird daher für jeden ausgehenden Link der VNF-Instanz aufgerufen (Zeile 19). Für jeden dieser neuen Teilflüsse wird die absolute Bandbreite des Datenstroms berechnet, welche sich aus der Bandbreite der Datenströme ergibt, die zu der gerade eingebetteten VNF-Instanz geroutet werden, und der relativen Datenrate r^i_{rel} des jeweiligen ausgehenden Links l:

$$\text{dr}' \leftarrow (\sum_{k \in K} r^{i,j}_{\text{total}}(k)) \cdot \text{r}^i_{\text{rel}}(l) \tag{5.1}$$

Dabei ist K die Menge der bereits eingebetteten eingehenden Kanten der VNF-Instanz, vgl. Zeile 20.
Schlägt die rekursive Einbettung der Teilflüsse fehl, wird die letzte Zuweisung rückgängig gemacht und einer der anderen Einbettungskandidaten ausgewählt, dem die VNF zugewiesen wird. Anschließend wird erneut versucht, die Teilflüsse zu vervollständigen. Gelingt dies für keinen der verfügbaren Einbettungskandidaten, wird ein Backtracking-Schritt durchgeführt (Zeile 37). In diesem Fall werden vorherige Zuweisungsentscheidungen aufgehoben und alternative Platzierungsmöglichkeiten in Betracht gezogen (Zeile 34).

Algorithmus 5 Der CoordVNF-Zuweisungsalgorithmus

```
1: function CoordVNF(VNFR^i,G,pathLengthLimit,numCandidates)
2:     return Rec(VNFR^i,G,None,r_init(VNFR^i),pathLengthLimit,numCandidates,{})
3: end function
4:
5: function Rec(VNFR^i,G,node_previous,dr,maxPathLength,numCandidates,M)
6:     VNFs ← VNFOptions(VNFFG^i,M) //VNFs, die angehängt werden können
7:     if VNFs=∅ then
8:         return M
9:     end if
10:    candidates ← BFS(VNFR^i,G,node_previous,VNFs,dr,maxPathLength,numCandidates,M)
11:    sortCandidates(candidates)
12:    for all (VNFInstance,node_current, path_current) ∈ candidates do
13:        M' ← M' ∪ MAP-NODE-DEMANDS(node_current,VNFInstance)
14:        M' ← M' ∪ MAP-LINK-DEMANDS(path_current,dr)
15:        M'' ← M'
16:        success ← True
17:        if alle eingehenden Kanten K der VNF-Instanz sind zugewiesen then
18:            //Einbetten der anschließenden Teilflüsse
19:            for all l ∈ l_i(instanceof(VNFinstance)) do
20:                dr' ← (Σ_{k∈K} r^{i,j}_total(k)) · r^i_rel(l)
21:                M_c ← Rec(VNFR^i,G,node_current,dr',maxPathLength,numCandidates,M'')
22:                if M_c ≠ ∅ then
23:                    M'' ← M'' ∪ M_c
24:                else
25:                    success ← False
26:                    break
27:                end if
28:            end for
29:        end if
30:        if success=True then
31:            return M''
32:        else
33:            //Rückgängigmachen der letzten Zuweisungen:
34:            undoAll(M'')
35:        end if
36:    end for
37:    return ∅ //Backtracking zum vorherigen gültigen Zustand
38: end function
```

Algorithmus 6 Breitensuche nach benachbarten Kandidaten

```
1: function BFS(VNFR^i,G,node_previous,VNFs,dr,maxPathLength,numCandidates,M)
2:     result ← ∅
3:     if node_previous == null then
4:         for all n in G.getNodes() do
5:             for all VNF in VNFs do
6:                 if fulfills(node,VNF) then
7:                     result ← result ∪ {node:n;path:null}
8:                 end if
9:                 if |result| ≥ numCandidates then
10:                    break
11:                end if
12:            end for
13:        end for
14:    else
15:        queue ← {(node:node_previous;path:())}
16:        while |queue| > 0 and |result| ≤ numCandidates do
17:            entry ← queue.poll
18:            for all edge in entry.getOutEdges do
19:                if res_link(edge) ≥ dr then
20:                    o ← destination(edge)
21:                    if o∉result then
22:                        p ← entry.path ∪ {o}
23:                        if |path'| ≤ maxPathLength then
24:                            //Pfaden mit ausreichend Bandbreite folgen:
25:                            queue ← queue ∪ {node:o;path:path'}
26:                            //Verfügt der Substratknoten o über entsprechende
27:                            //Kapazitäten, wird er mitsamt des Pfades p zur
28:                            //Ergebnismenge hinzugefügt
29:                            for all VNF in VNFs do
30:                                if fulfills(o,VNF) then
31:                                    result ← result ∪ {node:o;path:p}
32:                                    if |result| = numCandidates then
33:                                        return result
34:                                    end if
35:                                end if
36:                            end for
37:                        end if
38:                    end if
39:                end if
40:            end for
41:        end while
42:    end if
43:    return result
44: end function
```

5.2.3 Maßnahmen zur Beschränkung der Suchtiefe

CoordVNF ist ein Backtracking-basierter Ansatz, der die Verkettung und Zuweisung der Netzwerkfunktionen in einem koordinierten Schritt vornimmt. Das dabei zugrundeliegende NFV-RA-Problem ist $\mathcal{NP}$-hart. Um in der Lage zu sein, ein derart schwieriges Optimierungsproblem effizient zu lösen, wird der Suchraum des Algorithmus durch folgende Maßnahmen beschränkt:

1. Die Suchtiefe der Breitensuche wird bereits dadurch beschränkt, dass von der BFS-Funktion lediglich die kürzesten Pfade zwischen den Knoten des Substratnetzes in Betracht gezogen werden.

2. Die Anzahl der Zuweisungskandidaten, die in jedem Rekursionsschritt betrachtet werden, kann über den *numCandidates*-Parameter beschränkt werden.

3. Die Länge der zugewiesenen Substratpfade lässt sich über den *pathLengthLimit*-Parameter der CoordVNF-Funktion beschränken. Hierdurch wird insbesondere auch auf die Suchtiefe der Breitensuche Einfluss genommen.

Diese Maßnahmen werden getroffen, um die Suchtiefe des Algorithmus so zu begrenzen, dass bei jedem rekursiven Zuweisungsschritt jeweils nur die am vielversprechendsten erscheinenden Teillösungen in Betracht gezogen werden. Im Folgenden werden die Auswirkungen der beiden Parameter *numCandidates* und *pathLengthLimit* auf das Verhalten des Algorithmus näher beschrieben.

Der *numCandidates*-Parameter

Der *numCandidates*-Parameter limitiert die Anzahl der Zuweisungskandidaten, die in jedem rekursiven Schritt maximal berücksichtigt werden. Hierdurch lässt sich insbesondere das Laufzeitverhalten des Algorithmus beeinflussen.
In jedem rekursiven Teilschritt werden jeweils nur die *numCandidates*-besten Zuweisungsoptionen berücksichtigt.

Der *pathLengthLimit*-Parameter

Die Tiefe der Breitensuche kann über den *pathLengthLimit*-Parameter beschränkt werden. Eine derartige Beschränkung hat folgenden Effekt. Zum einen wird die maximale Pfadlänge zwischen zwei miteinander kommunizierenden VNF-Instanzen reduziert. In jedem rekursiven Schritt werden also nur solche Kandidaten berücksichtigt, die innerhalb eines begrenzten Radius zu erreichen sind. Kann aufgrund der Beschränkung kein passender Kandidat gefunden werden, werden vorherige Zuweisungsentscheidungen zurückgenommen und alternative Einbettungsmöglichkeiten in Betracht gezogen. Die lokale Beschränkung der Pfadlänge führt dazu, dass der Algorithmus Lösungen mit langen Pfaden ignoriert und stattdessen nach kosteneffizienteren Zuweisungslösungen sucht. Zum anderen hat dies auch direkte Auswirkungen auf die Laufzeit des Algorithmus, da in jedem rekursiven Schritt nur eine beschränkte Menge von Einbettungskandidaten berücksichtigt wird.

Eine zu restriktive Einstellung des Parameters hat Auswirkungen auf die Akzeptanz des Algorithmus, da hierdurch die Suchtiefe eingeschränkt wird. Aus diesem Grund sollte der *pathLengthLimit*-Parameter nicht zu niedrig gesetzt werden. Je höher der Parameter allerdings gewählt wird, desto schlechtere Ergebnisse werden von CoordVNF akzeptiert, da der Algorithmus in jedem rekursiven Schritt längere Pfade zuweisen kann, ohne alternative Einbettungskombinationen auszuprobieren.
Zur Lösung dieses Dilemmas bietet sich eine Erweiterung des Algorithmus an, die im Folgenden beschrieben wird.

5.2.4 Der erweiterte CoordVNF-Algorithmus

Statt in jedem rekursiven Schritt eine statische obere Schranke für den *maxPathLength*-Parameter zu verwenden, wird diese Schranke beim erweiterten CoordVNF-Algorithmus sukzessive gelockert. Dies bedeutet, dass CoordVNF zunächst versucht, alle VNF-Instanzen über sehr kurze Pfade zu verbinden; gelingt dies nicht, werden sukzessiv längere Pfade zugelassen, die von der Rec-Funktion berücksichtigt werden dürfen.
Der erweiterte CoordVNF-Algorithmus ist in Algorithmus 7 gezeigt; die hier beschriebene Extended-CoordVNF-Funktion ersetzt die CoordVNF-Funktion aus Algorithmus 5. Beginnend mit einem kleinen *maxPathLength*-Wert wird zunächst versucht, eine gültige Lösung zu finden. Ist dies nicht möglich, wird der *maxPathLength*-Wert inkrementiert, woraufhin ein neuer Einbettungsversuch gestartet wird. Der *pathLengthLimit*-Parameter der CoordVNF-Funktion dient dabei als obere Schranke für den *maxPathLength*-Parameter der Rec-Funktion. Vergleichbar mit dem von Lischka und Karl vorgeschlagen Ansatz wird der Suchraum mit dieser Methodik Schritt für Schritt erweitert [78]. Das inkrementelle Heraufsetzen des *maxPathLength*-Wertes führt dazu, dass CoordVNF nach Einbettungen mit möglichst kurzen Pfaden sucht.

Algorithmus 7 Der erweiterte CoordVNF-Zuweisungsalgorithmus

```
1: function Extended-CoordVNF(VNFR^i,G,pathLengthLimit,numCandidates)
2:     for maxPathLength ∈ {1, ..., pathLengthLimit} do
3:         M ← Rec(VNFR^i,G,None,r_init(VNFR^i),maxPathLength,
                   numCandidates,{})
4:         if M ≠ ∅ then
5:             return M
6:         end if
7:     end for
8:     return ∅
9: end function
```

5.3 Evaluation des CoordVNF-Algorithmus

Im Folgenden wird zunächst die Evaluationsmethodik beschrieben; daran anschließend werden die Evaluationsergebnisse im Einzelnen diskutiert.

5.3.1 Evaluationsmethodik

CoordVNF wird sowohl in Offline- als auch in Onlineszenarien evaluiert. In den Offlineszenarien wird zunächst eine bestimmte Anzahl von VNFR-Anfragen generiert, die von dem Zuweisungsalgorithmus eingebettet werden soll. In Onlineszenarien treffen die Einbettungsanfragen dagegen dynamisch zu zufälligen Zeitpunkten ein.
Mit Hilfe eines Barabási-Albert-Generators werden zufällige Substratnetztopologien erzeugt. Die Kapazitäten der Substratknoten werden zufällig aus einem Bereich zwischen 50 und 100 ausgewählt. Nach demselben Prinzip werden die Bandbreitenressourcen der Substratlinks aus demselben Intervall zugewiesen.
Zunächst wird CoordVNF mit dem MIVNF-Algorithmus verglichen. MIVNF ist der bislang einzige Zuweisungsalgorithmus, der zur Lösung beider Teilprobleme des NFV-RA-Problems vorgeschlagen wurde [82]. Zum Vergleich der beiden Algorithmen wird auf einen ausgewählten, von den Autoren von MIVNF ebenfalls verwendeten NFV-Netzwerkservice zurückgegriffen, der in unterschiedliche, zufällig generierte Substratnetze eingebettet werden soll.
Für alle weiteren Experimente werden auch die VNFR-Anfragen zufällig generiert. Hierzu werden zum einen die Abhängigkeiten der den VNFR-Anfragen zugewiesenen VNFs zufällig festgelegt. Dabei wird angenommen, dass die Flexibilität realer Netzwerkservices beschränkt ist und sich die Netzwerkfunktionen eines solchen Services nur innerhalb eines gewissen Rahmens vertauschen bzw. beliebig verketten lassen. Um diese Annahme bei der Generation zufälliger VNFR-Anfragen zu reflektieren, wird sichergestellt, dass die VNFRs derart erzeugt werden, dass sich ihre VNFs in minimal 2 und maximal 100 möglichen Kombinationsmöglichkeiten anordnen lassen. Zum anderen wird, um möglichst allgemeingültige Evaluationsergebnisse zu erzielen, auch die relative Datenrate der ausgehenden VNF-Links zufällig gewählt:

$$\forall v \in V^i, \forall l \in l^i_{\text{out}}(v),\ r^i_{\text{rel}}(l) = \frac{\mathit{random}(0.75, 1.25)}{|l^i_{\text{out}}(v)|} \tag{5.2}$$

Die Funktion *random* liefert dabei einen zufälligen Wert aus dem Intervall 75-125%. Als relative Datenrate r^i_{rel} wird für jeden ausgehenden Link also ein Zufallswert gewählt, der durch die Anzahl aller ausgehenden Links der VNF dividiert wird. Dadurch, dass der Zufallswert zur Anzahl der ausgehenden Links in Relation gesetzt wird, ist sichergestellt, dass jede VNF-Instanz insgesamt zwischen 75% und 125% des bei ihr ankommenden Datenvolumens an Nachfolgeinstanzen weiterleitet.
CoordVNF wird mit Hilfe des Alevin-Simulationsframeworks anhand der fol-

genden Metriken evaluiert:

- Laufzeit
  Die Zeitdauer, innerhalb der der Algorithmus terminiert und entweder ein Einbettungsergebnis liefert oder die Anfrage ablehnt

- Akzeptanz
  Die prozentuale Anzahl aller VNFR-Anfragen, die erfolgreich eingebettet werden konnten

- Einbettungskosten
  In Anlehnung an verwandte Arbeiten aus dem VNE-Umfeld wurden die Einbettungskosten, die die Zuweisung einer VNFR-Anfrage verursacht, als die Summe der reservierten Verarbeitungskapazität und der reservierten Bandbreite definiert. Dies lässt sich wie folgt formalisieren:

$$\text{Cost}(\text{VNF-FG}^{i,j}) = \sum_{v \in V_{\text{inst}}^{i,j}} d_{\text{total}}^{i,j}(v) + \sum_{l \in L_{\text{inst}}^{i,j}} h(l) \cdot r_{\text{total}}^{i,j}(l) \tag{5.3}$$

  $h(l)$ sei hier definiert als eine Funktion, die die Anzahl der Substratlinks bestimmt, auf die ein virtueller Link l gemappt wurde.

- Backtracking-Schritte
  CoordVNF sucht in jedem rekursiven Schritt nach möglichen Zuweisungskandidaten. Gelingt die Einbettung mit keinem dieser Kandidaten, wird durch Backtracking zu einem der vorherigen Zustände zurückgesprungen. Die Backtracking-Metrik misst die Anzahl dieser Rücksprünge. Sie ist also ein Indiz dafür, wie schwer sich der Algorithmus mit der Durchführung eines Einbettungsvorgangs tut.

- Berücksichtigte Einbettungskandidaten
  Diese Metrik zählt die Anzahl der Einbettungskandidaten, die insgesamt während des Einbettungsvorgangs gefunden wurden. Dazu werden die Kandidaten berücksichtigt, die in jedem rekursiven Schritt gefunden werden. Insofern stehen die Werte nicht nur in Abhängigkeit zu der Anzahl der durchgeführten Backtracking-Schritte, sondern sind auch von der Topologie des Substratnetzes und den dort zur Verfügung stehenden Ressourcen abhängig. Diese Metrik ist insofern von Interesse, als dass sich über den *numCandidates*-Parameter des CoordVNF-Algorithmus einstellen lässt, wie viele Einbettungskandidaten maximal in jedem rekursiven Zuweisungsschritt berücksichtigt werden sollen.

Zur Sicherstellung der Stabilität der Ergebnisse werden bei den Offline-Szenarien für jede Parameterkombination 50 Experimente erzeugt. Durchschnittswerte werden mit einem Konfidenzintervall von 95% angegeben. Die Länge der Whisker bei Box-Whiskerplots ist auf den 1,5-fachen Interquartilsabstand beschränkt.

|N|	CoordVNF	MIVNF
10	16ms	1.8s
15	6ms	41.5s
20	9ms	11m55s
25	4ms	33m45s
30	7ms	12h56m
50	6.6s	–
100	26.6s	–
200	1m50s	–
300	4m48s	–
400	9m42s	–
500	17m43s	–

Tabelle 5.1: Vergleich der Laufzeiten der beiden Algorithmen (für $numCandidates = \infty$)

5.3.2 Evaluationsergebnisse

Im Folgenden wird CoordVNF zunächst mit einem anderen NFV-RA-Algorithmus verglichen. Dazu werden die beiden Algorithmen im Zusammenhang mit unterschiedlich großen Substratnetzen untersucht.
Als nächstes wird untersucht, inwiefern sich die Ergebnisse der in Abschnitt 5.2.4 beschriebenen Erweiterung von CoordVNF von denen des ursprünglichen Algorithmus unterscheiden. Da sich zeigt, dass die erweiterte CoordVNF-Variante bei akzeptablem Laufzeitverhalten in Bezug auf die Einbettungskosten-Metrik sogar noch etwas besser abschneidet, beziehen sich alle darauf folgenden Evaluationsergebnisse auf den erweiterten CoordVNF-Algorithmus.
Anschließend wird untersucht, wie sich die Parameter *pathLengthLimit* und *numCandidates*, mit denen die Suchtiefe von CoordVNF reguliert werden kann, auf die Simulationsergebnisse auswirken. Daraufhin wird untersucht, wie CoordVNF im Zusammenhang mit großen Substratnetzen zurecht kommt. Abschließend wird diskutiert, welche Arten von Szenarien für den Algorithmus besonders „schwierig“ sind.

5.3.2.1 Vergleich von CoordVNF und MIVNF

Dieser Abschnitt vergleicht CoordVNF mit dem MIVNF-Algorithmus. MIVNF ist der derzeit einzige Algorithmus, der zur Lösung des NFV-RA-Problems vorgestellt wurde. Die Evaluationsergebnisse für MIVNF konnten anhand des Original-Quellcodes erhoben werden, da die Autoren des Ansatzes diesen Code

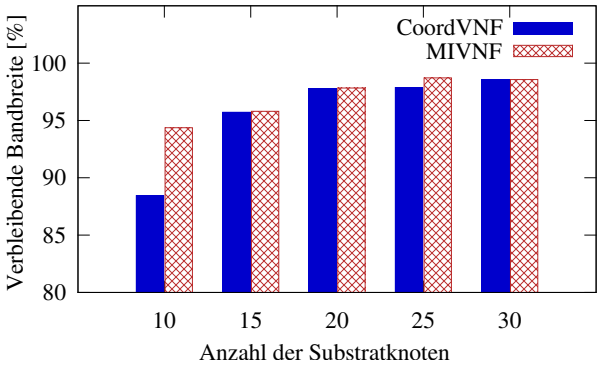


Abb. 5.3: Innerhalb des Substratnetzes zur Verfügung stehende Bandbreitenressourcen nach Einbettung des VNFR

dankenswerterweise zur Verfügung stellten. MIVNF löst das NFV-RA-Problem in zwei unkoordinierten Schritten. Im ersten Schritt wird, unabhängig von der Topologie und der Ressourcenverfügbarkeit des Substratnetzes, die optimale Verkettung der Netzwerkfunktionen bestimmt. Dazu wird der einzubettende VNF-FG erzeugt. Im zweiten Schritt wird dann die Einbettung des zuvor ausgewählten VNF-FG mit Hilfe eines MIQCP (Mixed Integer Quadratically Constrained Programs) berechnet.

Für MIVNF wurden drei Optimierungsstrategien vorgestellt: Die Minimierung der zu reservierenden Bandbreite, das Finden eines energieeffizienten Embeddings und die Minimierung der Kommunikationslatenz. Da CoordVNF mit dem Ziel konzipiert wurde, die Einbettungskosten gering zu halten, um möglichst viele VNFR-Anfragen einbetten zu können, wurde zum Vergleich beider Ansätze die erste Variante von MIVNF verwendet. Im Folgenden wird zuerst die Laufzeit beider Algorithmen in Abhängigkeit von der Problemgröße analysiert. Anschließend wird die Güte der berechneten Lösungen gegenübergestellt. Für den Vergleich werden zufällige Substratnetze generiert. In jedem Szenario wird ein Netzwerkservice eingebettet, der auch von den MIVNF-Autoren verwendet wurde. Er basiert auf den NFV-Use Cases, die von Liu et al. vorgeschlagen wurden [80] und besteht aus 6 VNFs.

Für die hier gezeigten Experimente wurde $pathLengthLimit = 5$ gesetzt, während die durch den *numCandidates*-Parameter einschränkbare maximale Suchtiefe von CoordVNF nicht begrenzt wurde (d.h., $numCandidates = \infty$).

Tabelle 5.1 vergleicht die Laufzeit der beiden Algorithmen in Abhängigkeit von der Größe des Substratnetzes. Während MIVNF in den hier untersuchten Szenarien in der Lage ist, für Topologien mit bis zu 15 Knoten eine Lösung innerhalb weniger Sekunden zu finden, nimmt die Berechnungszeit schon für wenig größere Topologien drastisch zu. Schon für Netze mit 30 Knoten benötigt MIVNF knapp 13 Stunden, während CoordVNF innerhalb von Millisekunden in der Lage ist, ein gültiges Einbettungsergebnis zu finden. MIVNF wurde im

Zusammenhang mit größeren Netzen aufgrund der langen Ausführungszeiten nicht evaluiert. Insgesamt zeigt sich, dass MIVNF in Szenarien realer Größenordnung nicht praktikabel einsetzbar ist. CoordVNF ist in jedem dieser hier untersuchten Szenarien um Größenordnungen schneller und in der Lage, auch in wesentlich größeren Szenarien Einbettungen innerhalb von wenigen Minuten zu berechnen. Für Netze mit 500 Knoten benötigt CoordVNF ohne Beschränkung der Suchtiefe über den *numCandidates*-Parameter ∼17 Minuten, um die relativ kleine VNFR-Anfrage einzubetten.
Da MIVNF darauf abzielt, den Bandbreitenbedarf der Einbettung zu reduzieren, wird im Folgenden untersucht, wie viel Bandbreite den von den Algorithmen durchgeführten Einbettungen zugewiesen wurde. Wie in Abbildung 5.3 zu sehen ist, bezahlt CoordVNF die deutlich schnellere Laufzeit mit geringen Einbußen bei der Einbettungsqualität. In jedem der untersuchten Szenarien findet MIVNF eine etwas kostengünstigere Lösung. Allerdings ist der Unterschied nur gering: im schlechtesten Fall ist CoordVNF um nur 5% weniger kosteneffizient.
Dennoch ist der MIVNF-Algorithmus dem schnelleren CoordVNF-Algorithmus auch bezüglich der Einbettungsqualität nicht in allen Szenarien überlegen: Denn obwohl MIVNF eine optimale Einbettung eines VNF-FG berechnet, ist er nicht immer in der Lage, ein Ergebnis für ein grundsätzlich lösbares NFV-RA-Problem zu finden. Dies ist in Abbildung 5.4 veranschaulicht: In diesem Szenario soll ein VNFR mit 3 VNFs in ein Substratnetz mit 3 Knoten eingebettet werden. Im ersten Schritt berechnet MIVNF dazu die bzgl. des Bandbreitenbedarfs optimale Verkettung der VNFs. In Abbildung 5.4 ist dies VNF-FG 1. Im zweiten Schritt versucht MIVNF, VNF-FG 1 optimal in das Substratnetz einzubetten. Dies ist aufgrund der Netzstruktur unmöglich, weswegen MIVNF die Anfrage zurückweisen muss, während CoordVNF auch hier eine Lösung findet. CoordVNF kann in dem gezeigten Szenario VNF-FG 2 einbetten.
Nachdem CoordVNF in den letzten Experimenten im Zusammenhang mit relativ kleinen VNFR-Anfragen evaluiert wurde, wird CoordVNF im Folgenden in wesentlich komplexeren Szenarien untersucht.

5.3.2.2 Evaluation des erweiterten CoordVNF-Ansatzes

Im Folgenden wird analysiert, inwiefern sich die in Abschnitt 5.2.4 beschriebene Erweiterung des CoordVNF-Algorithmus auf die Evaluationsergebnisse auswirkt. Dazu werden im Folgenden 5 VNFRs mit je 10 VNFs in Substratnetze mit 50 Knoten eingebettet. Die maximale Suchtiefe wurde mit $numCandidates = 4$ beschränkt.
Durch das iterative Erhöhen der maximal zulässigen Pfadlänge erhöht sich zunächst die durchschnittliche Laufzeit des erweiterten CoordVNF-Algorithmus, vgl. Abbildung 5.5a. Die Laufzeit für den einfachen CoordVNF-Ansatz bleibt dagegen relativ unverändert. Eine Ausnahme ist $pathLengthLimit = 1$, was sich durch die mangelnde Verfügbarkeit passender Zuweisungskandidaten erklären

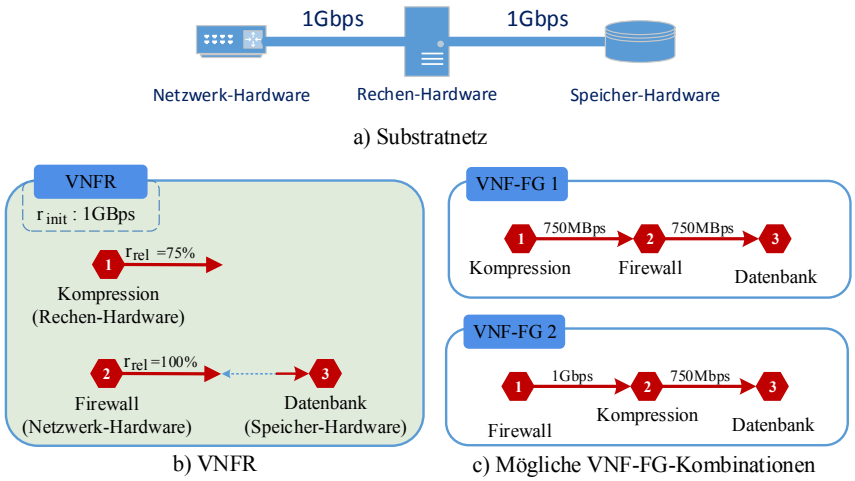
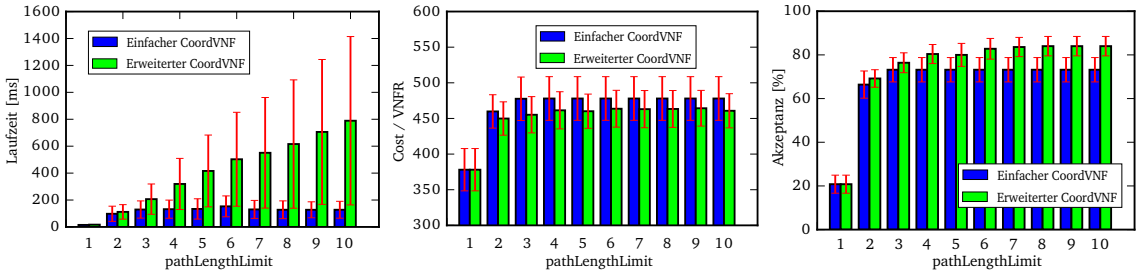


Abb. 5.4: Szenario, in dem der MIVNF-Algorithmus im Gegensatz zu CoordVNF keine Lösung findet

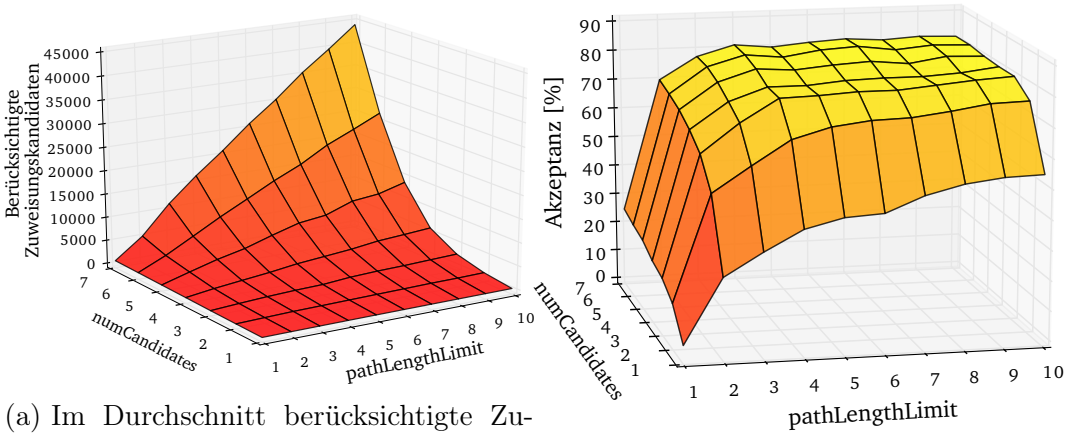


(a) Laufzeit (b) Einbettungskosten (c) Akzeptanz

Abb. 5.5: Durchschnittliche Laufzeit, Kosten und Akzeptanz beim erweiterten CoordVNF-Algorithmus im Vergleich zur ursprünglichen Variante

lässt. Dagegen steigt die durchschnittliche Laufzeit des erweiterten CoordVNF nahezu linear mit dem Wert des *pathLengthLimit*-Parameters. Für größere *pathLengthLimit*-Werte vergrößert sich für den erweiterten Ansatz auch die Größe des Konfidenzintervalls: Das liegt daran, dass der Algorithmus in einigen Fällen gezwungen ist, bis zu *pathLengthLimit*-mal einen Einbettungsversuch zu starten. Dies trifft insbesondere in solchen Fällen zu, in denen keine Einbettung durchführbar ist. Dementsprechend vervielfacht sich die maximale Laufzeit des Verfahrens. Meistens gelingt eine erfolgreiche Einbettung allerdings schon früher, was sich in den Durchschnittswerten widerspiegelt.

Der erweiterte CoordVNF-Algorithmus ist der einfacheren Variante bezüglich der Einbettungskosten überlegen. Da versucht wird, mit der kürzestmöglichen maximalen Pfadlänge Einbettungen zu finden, sinken die Kosten für die Zuweisung der virtuellen Links. Insgesamt führt dies zu einer Reduzierung der Einbettungskosten, vgl. Abbildung 5.5b und Formel 5.3, wodurch zusätzliche Kapazitäten für die Einbettung weiterer VNFRs nutzbar werden. Damit ist auch die Akzeptanz des erweiterten Algorithmus höher als bei der einfacheren



(a) Im Durchschnitt berücksichtigte Zuweisungskandidaten

(b) Durchschnittliche Akzeptanz

Abb. 5.6: Berücksichtigte Zuweisungsoptionen und Akzeptanz in Abhängigkeit von der maximalen Suchtiefe

Variante, vgl. Abbildung 5.5c.
Da die Reduktion der Einbettungskosten und die damit einhergehenden besseren Akzeptanzwerte ein sehr wesentliches Optimierungskriterium für einen Einbettungsalgorithmus sind, wird im Folgenden nunmehr die erweiterte Variante des CoordVNF-Algorithmus genauer analysiert. Im Folgenden bezieht sich der Begriff „CoordVNF“ also immer auf die in Algorithmus 7 gezeigte erweiterte Variante.

5.3.2.3 Suchtiefe

Im Folgenden wird untersucht, in wie weit sich die Beschränkung der Suchtiefe von CoordVNF auf die Güte der berechneten Ergebnisse auswirkt. Die Suchtiefe des Algorithmus ist von zwei Parametern abhängig, nämlich vom *pathLengthLimit*- und vom *numCandidates*-Parameter, vgl. Abschnitt 5.2.3. Im Folgenden werden die Auswirkungen dieser zwei Parameter auf das Verhalten des Algorithmus untersucht.
Für die hier gezeigten Szenarien sollen 5 VNFR-Anfragen mit jeweils 10 VNFs in ein Substratnetz mit 50 Knoten eingebettet werden. Die initiale Datenrate wird auf einen Fixwert von 50 festgesetzt, da, wie im Weiteren noch gezeigt wird, derartige Szenarien für den Algorithmus relativ schwer zu lösen sind.

Berücksichtigte Zuweisungskandidaten: Abbildung 5.6a visualisiert die Anzahl der von CoordVNF berücksichtigten Zuweisungskandidaten. Dies sind die Kandidaten, die CoordVNF mit Hilfe der Breitensuche aufspürt, bis eine Zuweisungsentscheidung getroffen wurde. Wie aus der Abbildung ersichtlich wird, steigen die Werte linear in Abhängigkeit vom *pathLengthLimit*-Parameter sowie exponentiell mit dem *numCandidates*-Wert.

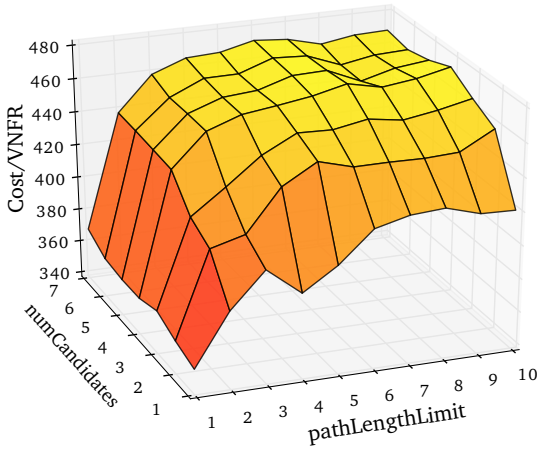
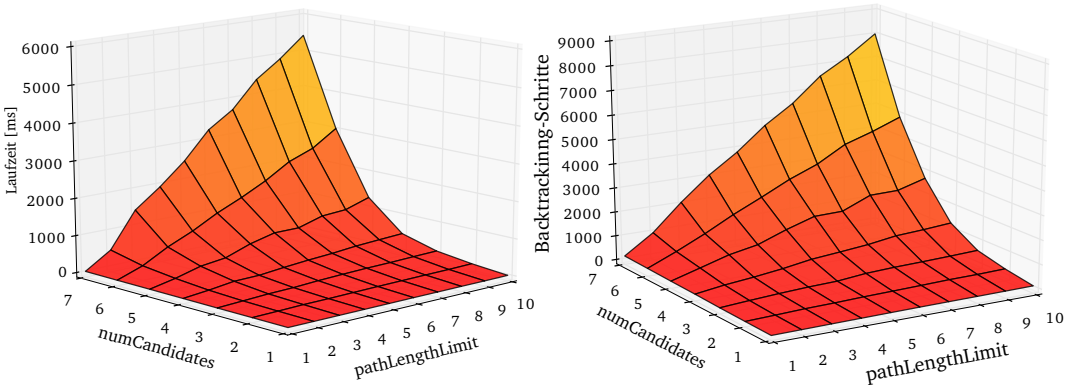


Abb. 5.7: Einbettungskosten in Abhängigkeit von der maximalen Suchtiefe

Während für *numCandidates* = 1 und *pathLengthLimit* = 1 durchschnittlich nur 25 Zuweisungsmöglichkeiten berücksichtigt wurden, waren dies für *numCandidates* = 5 und *pathLengthLimit* = 5 bereits 6950. Für *numCandidates* = 7 und *pathLengthLimit* = 10 wurden durchschnittlich sogar 44085 mögliche Kandidaten mit einbezogen.

Akzeptanz: Je weniger Zuweisungskandidaten berücksichtigt werden, desto höher ist die Wahrscheinlichkeit, dass CoordVNF eine der Anfragen aus Mangel an geeigneten Zuweisungsmöglichkeiten zurückweisen muss. Dieser Effekt lässt sich in Abbildung 5.6b nachvollziehen.
Es lässt sich erkennen, dass die Akzeptanz für größere *pathLengthLimit*-Werte grundsätzlich zunimmt – am deutlichsten an der Stelle *pathLengthLimit* = 2. Diese Auswirkung des Parameters erklärt sich damit, dass die in jedem rekursiven Schritt durchgeführte Breitensuche für größere Werte mehr mögliche Zuweisungskandidaten in Betracht zieht, als sie es für kleinere *pathLengthLimit*-Werte tut – eben auch jene, die nur über entsprechend längere Pfade erreicht werden können. Ein vergleichbarer Effekt lässt sich für den *numCandidates*-Parameter ablesen. Auch hier nimmt die Akzeptanz am deutlichsten für *numCandidates* = 2 zu.
Während die Akzeptanz für kleinere *pathLengthLimit*-Werte mit dem *numCandidates*-Wert steigt und eine Stagnation erst für größere *numCandidates*-Werte erkennbar ist, scheint diese für größere *pathLenghtLimit*-Werte bereits früher einzutreten. So lässt sich beispielsweise erkennen, dass für *pathLengthLimit* = 10 keine erkennbare Verbesserung der Akzeptanz mehr eintritt, wenn *numCandidates* ≥ 4 gewählt wird.

Einbettungskosten: Abbildung 5.7 zeigt, dass die Einbettungskosten für kleinere Parameterwerte geringer ausfallen als bei größeren Werten. Für kleine *pathLengthLimit*-Werte liegt das daran, dass CoordVNF jeweils nur solche



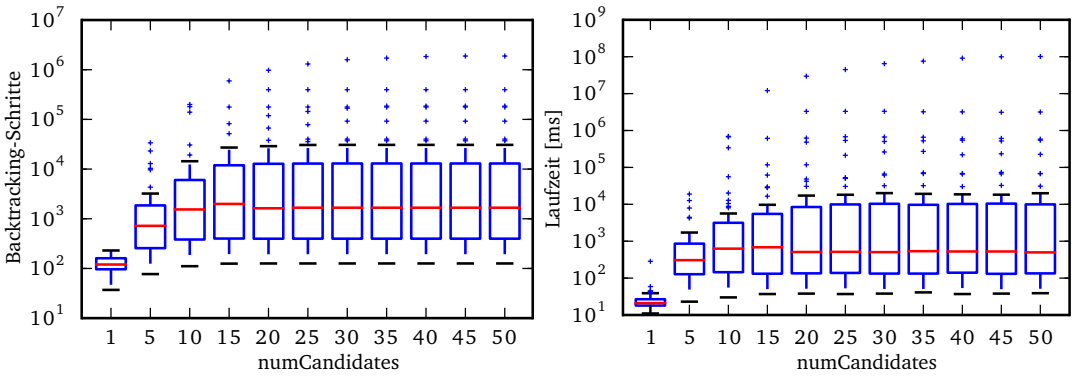
(a) Durchschnittliche Laufzeit

(b) Durchschnittliche Anzahl der Backtracking-Schritte

Abb. 5.8: Durchgeführte Backtracking-Schritte und Laufzeit von CoordVNF in Abhängigkeit von der maximalen Suchtiefe

Zuweisungskandidaten berücksichtigt, die über kurze Distanzen innerhalb des Substratnetzgraphen zu erreichen sind. Werden für einen VNFR nur sehr kurze Pfade reserviert, fallen die Einbettungskosten für diesen Netzwerkservice sehr gering aus. Dies ergibt sich aus der Definition der Einbettungskosten-Metrik nach Formel 5.3.
Weil der Algorithmus längere und damit teurere Pfade nicht berücksichtigt, ist die erfolgreiche Einbettung eines VNFR im Durchschnitt zwar kostengünstiger als für größere *pathLengthLimit*-Werte. Allerdings gelingt eine solche günstige Einbettung nur in seltenen Fällen, was sich in deutlich schlechteren Akzeptanzwerten widerspiegelt, vgl. Abbildung 5.6b – der Algorithmus bettet dann also nur die VNFRs ein, die sich „billig“ zuweisen lassen.
Vergleichbare Effekte zeigen sich für den *numCandidates*-Parameter; es werden nur die in jedem rekursiven Schritt besten Zuweisungskandidaten berücksichtigt und alle anderen Optionen verworfen, was ähnliche Auswirkungen auf die Werte der Einbettungskosten-Metrik hat.
Für $pathLengthLimit \gtrsim 3$ und $numCandidates \gtrsim 3$ stabilisieren sich die Einbettungskosten im Durchschnitt auf einen Wert von ungefähr 470. Allerdings steigen die Einbettungskosten nicht monoton, da in jedem rekursiven Schritt andere Zuweisungskandidaten ausgewählt werden, was Auswirkungen auf zukünftige Einbettungsentscheidungen zur Folge hat. Es gibt also Fälle, in denen eine Einbettung insgesamt günstiger ausfällt, wenn frühzeitig einige weniger günstige Zuweisungen vorgenommen werden, vgl. z.B. $numCandidates = 1$, $pathLengthLimit = 4$.

Laufzeitverhalten: Die Begrenzung der Suchtiefe von CoordVNF wirkt sich auf das Laufzeitverhalten des Algorithmus aus, vgl. Abbildung 5.8a. Je größer beide Werte gewählt werden, desto höher ist im Allgemeinen die durchschnittliche Laufzeit. Das liegt daran, dass CoordVNF in jedem rekursiven Schritt

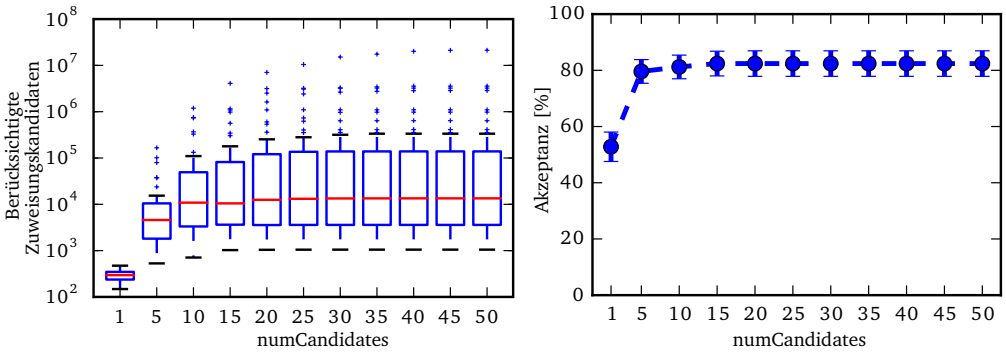


(a) Durchgeführte Backtracking-Schritte für *pathLengthLimit*=10 (b) Laufzeit für pathLengthLimit=10

Abb. 5.9: Durchgeführte Backtracking-Schritte und Laufzeit von CoordVNF in Abhängigkeit von der maximalen Suchtiefe (für *pathLengthLimit*=10)

nach allen möglichen Zuweisungskandidaten sucht, die sich ausgehend vom zuletzt zugewiesenen Substratknoten erreichen lassen. Diese Kandidaten werden als „Reserve“ bereit gehalten für den Fall, dass bereits getroffene Zuweisungsentscheidungen wieder rückgängig gemacht werden müssen. Das Rückgängigmachen ist allerdings nur dann nötig, wenn sich der Algorithmus „verläuft“. Gelingt die Einbettung mit nur wenigen Rückgriffen auf bereit gehaltene alternative Zuweisungsmöglichkeiten, terminiert der Algorithmus entsprechend früher. Ein vollständiges Ausprobieren *aller* Zuweisungsoptionen wird also immer dann durchgeführt, wenn der Algorithmus keine mögliche Einbettung finden kann.

Backtracking-Schritte: Die Backtracking-Metrik eignet sich zur Veranschaulichung der Häufigkeit, mit der CoordVNF bei der Zuweisung der VNF-Instanzen scheitert. Misslingt ein solcher Zuweisungsversuch, werden ein oder mehrere Zuweisungsentscheidungen widerrufen und Alternativmöglichkeiten in Betracht gezogen. Neben der in Abbildung 5.6a gezeigten Anzahl der berücksichtigten Zuweisungskandidaten eignet sich daher die Zahl der von CoordVNF durchgeführten Backtracking-Schritte für die Nachvollziehbarkeit der gemessenen Laufzeitwerte, vgl. Abbildung 5.8b. Da die Backtracking-Metrik widerspiegelt, wie oft sich der Algorithmus bei der Zuweisung „verläuft“, kann die Anzahl der Backtracking-Schritte neben den reinen Laufzeitwerten als Maß dafür angesehen werden, wie schwer sich der Ansatz mit der Einbettung der VNFR-Anfragen tut.
In der Tat ähneln sich die Abbildungen 5.8a und 5.8b sehr. Je häufiger der Algorithmus also gezwungen ist, frühere Zuweisungsentscheidungen zu widerrufen, desto länger braucht er in der Regel auch für den Einbettungsvorgang.
Abbildungen 5.9 und 5.10 zeigen weitere Untersuchungen des Algorithmus be-



(a) Berücksichtigte Zuweisungskandidaten für *pathLengthLimit*=10 (b) Akzeptanz für *pathLengthLimit*=10

Abb. 5.10: Berücksichtigte Zuweisungskandidaten und Akzeptanz in Abhängigkeit von der maximalen Suchtiefe (für *pathLengthLimit*=10)

züglich des *numCandidates*-Parameters. Hier wurden wesentlich größere Werte für *numCandidates* untersucht, wobei $pathLengthLimit = 10$ gesetzt wurde.
Wie aus den Grafiken hervorgeht, kommt es auch für größere *numCandidates*-Werte im Mittel zu keinen statistisch signifikanten Unterschieden hinsichtlich der untersuchten Metriken. Insbesondere verbessert sich bei Wahl einer noch größeren Suchtiefe der gemessene Akzeptanzwert nicht.
Bemerkenswert ist allerdings, dass das Laufzeitverhalten in einigen Ausnahmefällen für größere *numCandidates*-Werte wesentlich vom Mittelwert abweicht, vgl. dazu die in Abbildung 5.9b dargestellten Ausreißer: Die maximale Laufzeit für $numCandidates = 10$ liegt hier bei 12 Minuten, während der Algorithmus für $numCandidates = 50$ im Extremfall sogar erst nach über 28 Stunden terminierte. Wie aus Abbildung 5.9a und Abbildung 5.10a hervorgeht, ist CoordVNF in diesen Szenarien gezwungen, sehr viele Backtracking-Schritte durchzuführen und entsprechend viele alternative Zuweisungskandidaten aufzuspüren und Zuweisungsmöglichkeiten auszuprobieren. Interessant ist, dass die *mittlere* Laufzeit für $numCandidates \geq 10$ in etwa gleich bleibt, während die *maximale* Laufzeit für diese Werte weiterhin steigt.
Auch die Werte der Backtracking-Metrik und die Anzahl der berücksichtigten Zuweisungskandidaten zeigen weder im Mittel noch hinsichtlich ihres Maximalwertes signifikante Unterschiede bei größeren Parametereinstellungen.

Insgesamt lässt sich den Ergebnissen entnehmen, dass ein entsprechend größer gewählter *numCandidates*-Wert in dem gezeigten Szenario nicht zu besseren Akzeptanzwerten führt, vgl. Abbildung 5.10b. Aus diesem Grund kann hier $numCandidates = 4$ und $pathLengthLimit = 10$ als eine geeignete Parameterkombination betrachtet werden.

Zwischenfazit: Grundsätzlich ist für beide Parameter ein Tradeoff zwischen dem Laufzeitverhalten von CoordVNF und der Akzeptanz erkennbar. Je klei-

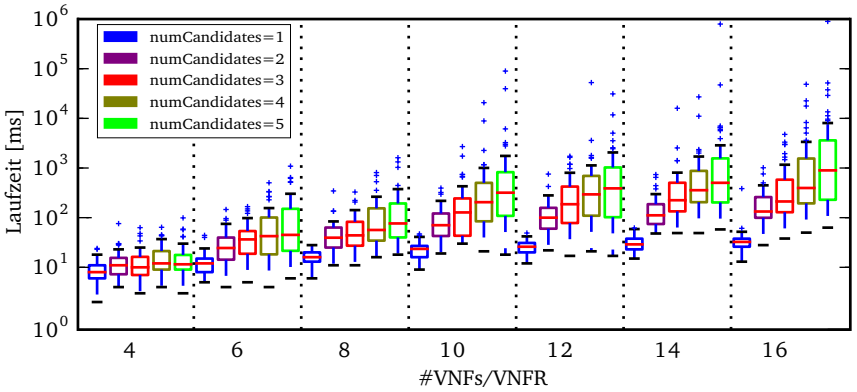


Abb. 5.11: Laufzeit von CoordVNF in Abhängigkeit von der Größe der VNFRs und der maximalen Suchtiefe

ner die Werte gesetzt werden, desto schneller terminiert CoordVNF. Werden die Parameter allerdings auf einen zu kleinen Wert gesetzt, beeinflusst dies die durchschnittliche Akzeptanz des Algorithmus, die in diesem Fall drastisch abnimmt.
Für die im Folgenden gezeigten Experimente wurde $pathLengthLimit = 10$ und $numCandidates = 4$ gesetzt. Diese Einstellung stellt in den untersuchten Szenarien eine gute Wahl hinsichtlich der beiden Metriken Akzeptanz und Laufzeit dar. Da die Laufzeit für $pathLengthLimit \geq 4$ nur geringfügig steigt und sich gleichzeitig die Akzeptanz einem stabilen Durchschnittswert annähert, wird für die später folgenden Experimente $pathLengthLimit = 10$ und $numCandidates = 4$ gewählt. $pathLengthLimit = 10$ erscheint als eine großzügige und angemessene obere Schranke für die Länge der zugewiesenen Pfade.

Die nun folgenden Experimente zeigen, dass diese Parameterkombination in der Tat eine gute Wahl ist und der Algorithmus auch in anderen, komplexeren Szenarien gute Ergebnisse hinsichtlich der untersuchten Metriken liefert.

5.3.2.4 Einbettung größerer VNFR-Anfragen:

Im letzten Abschnitt wurde das Verhalten von CoordVNF in einem Szenario analysiert, in dem die Einbettung von VNFR-Anfragen mit jeweils 10 VNFs simuliert wurde. Nun soll untersucht werden, inwiefern die Größe der VNFR-Anfragen Auswirkungen auf die Performanz des Algorithmus hat. Ferner soll evaluiert werden, inwieweit der *numCandidates*-Parameter Einfluss auf die Laufzeit und die Akzeptanz des Algorithmus hat, wenn größere Einbettungsanfragen in Betracht gezogen werden. Zu diesem Zweck werden im Folgenden jeweils 5 VNFR-Anfragen unterschiedlicher Größe in ein Substratnetz aus 50 Knoten eingebettet. Der *pathLengthLimit*-Parameter wurde auf einen Fixwert von 10 gesetzt, während *numCandidates* aus dem Bereich $[1, 5]$ gewählt wurde. Je nach Szenario werden pro Anfrage 4 bis 16 VNFs generiert. Grundsätzlich ist die Einbettung größerer Anfragen schwieriger, vgl. Abschnitt 3.2.

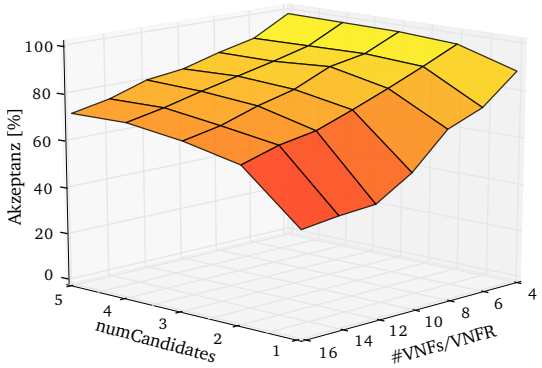


Abb. 5.12: Durchschnittliche Akzeptanz von CoordVNF in Abhängigkeit von der Größe der VNFRs und der maximalen Suchtiefe

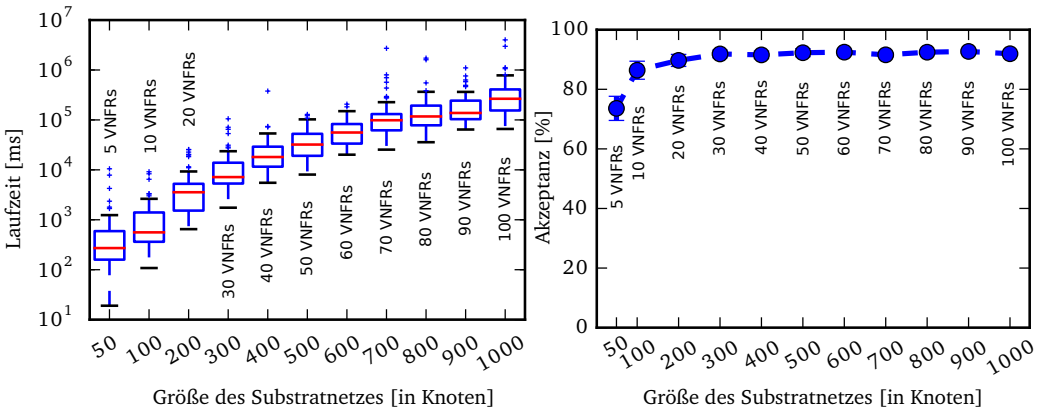
Laufzeitverhalten: Abbildung 5.11 zeigt die durchschnittliche Laufzeit des Algorithmus, die stark von der Größe der VNFR-Anfragen und dem gewählten *numCandidates*-Wert abhängt. Als Maximalwert wurde für $numCandidates = 5$ eine Laufzeit von 15 Minuten für die Einbettung von 5 VNFRs mit jeweils 16 VNFs gemessen, während der Maximalwert für $numCandidates = 4$ lediglich bei 49 Sekunden lag. In jedem der Szenarien liegt der Medianwert im Bereich weniger Millisekunden, während die Ausreißer sowohl mit der Größe der VNFR-Anfragen als auch mit der durch den *numCandidates*-Parameter vorgegebenen maximalen Suchtiefe stark zunehmen.

In den meisten Fällen ist CoordVNF also in der Lage, die Einbettungsergebnisse innerhalb von Millisekunden zu berechnen. Wird der Algorithmus allerdings mit einem „schwierigen“ Einbettungsproblem konfrontiert, kann sich dies im Extremfall sehr negativ auf sein Laufzeitverhalten auswirken. Aus diesem Grund sollte die Suchtiefe von CoordVNF adäquat eingeschränkt werden, solange die Auswirkungen auf die Akzeptanz des Algorithmus in einem vertretbaren Rahmen bleiben.

Akzeptanz: Wie sich aus Abbildung 5.12 ablesen lässt, führt die durch Setzen von $numCandidates = 5$ bedingte größere Suchtiefe im Durchschnitt nicht zu besseren Ergebnissen bezüglich der Akzeptanzmetrik.

In der Tat scheint, unabhängig von der Größe der VNFR-Anfragen, ein Wert zwischen 3 und 4 als geeignete Einstellung für den *numCandidates*-Parameter. Dies passt zu den im vorherigen Abschnitt diskutierten Ergebnissen.

Da CoordVNF auch für $numCandidates = 4$ in der Lage ist, innerhalb einer akzeptablen Laufzeit von wenigen Millisekunden Einbettungslösungen zu finden, wird für alle nun folgenden Experimente $numCandidates = 4$ und $pathLengthLimit = 10$ gesetzt. Im Extremfall wurden für diese Einstellung eine Laufzeit von 49 Sekunden gemessen.



(a) Laufzeit (b) Akzeptanz

Abb. 5.13: Laufzeit und Akzeptanz von CoordVNF in Abhängigkeit von der Größe des Substratnetzes

5.3.2.5 Evaluation in großen Substratnetzen

Im Folgenden wird nun analysiert, inwiefern die Größe der Substratnetze Auswirkungen auf die Performanz des Algorithmus hat. Dazu wird CoordVNF sowohl in einem Offlineszenario als auch in einem Onlineszenario evaluiert. Beim Offlineszenario wird für jedes Experiment ein zufälliges Substratnetz erzeugt, in welches eine bestimmte Anzahl ebenfalls zufällig generierter VNFR-Anfragen einzubetten ist. Jedes Experiment wird 50 Mal wiederholt. Dies entspricht grundsätzlich der Vorgehensweise, die schon für die zuvor beschriebenen Szenarien gewählt wurde. Bei dem Onlineszenario wird ein einzelnes großes Substratnetz erzeugt, in das viele kontinuierlich eintreffende VNFR-Anfragen eingebettet werden sollen; das Eintreffen der VNFR-Anfragen wird dabei als Poisson-Prozess simuliert, siehe Abschnitt 3.3.

Offlineszenario: Für das Offlineszenario werden mit Hilfe des Barabási-Albert-Generators zufällige Substratnetztopologien mit 50 bis 1000 Knoten generiert, in welche VNFR-Anfragen mit je 10 VNFs eingebettet werden sollen. Da die Einbettungskomplexität im Allgemeinen mit der Anzahl der einzubettenden VNF-Instanzen zusätzlich steigt, wird die Menge der einzubettenden VNFR-Anfragen in jedem Experiment in Abhängigkeit von der Größe des Substratnetzes bestimmt. Für Experimente mit 50 Substratknoten werden also 5 VNFR-Anfragen mit jeweils 10 VNFs generiert, für Substratnetze mit 100 Knoten entsprechend 10 Anfragen (mit ebenfalls 10 VNFs) und für Netze mit 1000 Knoten 100 Anfragen.
Wie in Abbildung 5.13a zu erkennen ist, nimmt die Laufzeit von CoordVNF mit der Größe des Substratnetzes zu. Allerdings muss hierbei berücksichtigt werden, dass neben der Größe des Substratnetzes auch die Anzahl der einzubettenden VNFR-Anfragen linear gesteigert wird. Der dadurch entsprechend höhere Komplexitätsgrad spiegelt sich in den Laufzeitmessungen wider. Die

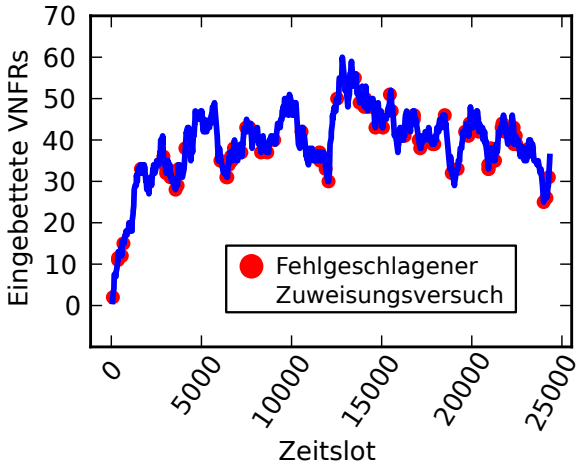


Abb. 5.14: Online-Szenario: Einbettung von 1000 VNFRs in ein Substratnetz mit 1000 Knoten

mittlere Laufzeit bleibt selbst bei großen Substratnetzen mit 1000 Knoten in diesem Szenario bei etwa 4 Minuten. Für die Bearbeitung einer einzelnen VNFR-Anfrage benötigt CoordVNF im Mittel also nur wenige Sekunden. Als Extremwert wurde eine maximale Laufzeit von $\sim$ 67 Minuten gemessen, und zwar für eines der Experimente mit 1000 Knoten. In diesem Extremfall benötigte CoordVNF für die Einbettung von 100 VNFRs insgesamt $\sim$ 67 Minuten. Bezüglich der in Abbildung 5.13b gezeigten Akzeptanz-Metrik ist auffallend, dass die Akzeptanzwerte mit steigender Substratnetzgröße sogar zunehmen. Während für Netze mit 50–100 Knoten eine mittlere Akzeptanz von $\sim$ 80% erzielt wird, sind es bei den Experimenten mit 200–1000 Knoten $\sim$ 90%. Dies erscheint auf den ersten Blick gerade deswegen verwunderlich, weil die Anzahl der einzubettenden Anfragen in diesem Szenario in Relation zur Netzwerkgröße gewählt wurde. Allerdings muss berücksichtigt werden, dass die Wahrscheinlichkeit höher ist, innerhalb eines großen Netzes eine passende Teil-Topologie ausfindig zu machen, die den Anforderungen der VNFRs entspricht.
Insgesamt lässt sich festhalten, dass CoordVNF auch in diesem komplexen Szenario in der Regel nur wenige Sekunden für die Bearbeitung der einzelnen Einbettungsanfrage benötigt. CoordVNF zeigt auch hier sehr gute Akzeptanzwerte, gerade in den Szenarien mit größeren Substratnetz-Topologien.

Onlineszenario – Einbettung vieler kontinuierlich eintreffender Anfragen: Abbildung 5.14 zeigt Simulationsergebnisse aus einem Onlineszenario. Hier wird ein einzelnes, mit Hilfe des Barabási-Albert-Ansatzes generiertes Substratnetz mit 1000 Knoten erzeugt, in welches kontinuierlich neu eintreffende VNFR-Anfragen eingebettet werden sollen.
Das Eintreffen der VNFR-Anfragen wird als Poisson-Prozess simuliert mit einer Ankunftsrate $\lambda_{\text{arrival}} = 4/100$. Die Lebensdauer der Anfragen wurden durch Ziehen exponentialverteilter Zufallszahlen bestimmt, wobei eine Verweilrate

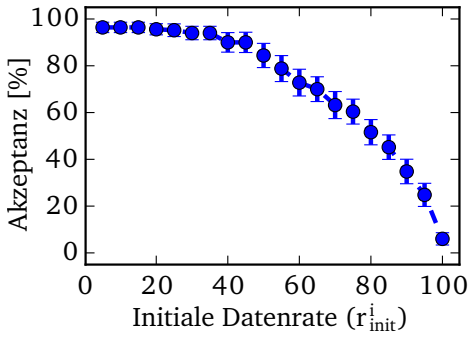


Abb. 5.15: Durchschnittliche Akzeptanz in Abhängigkeit von der Komplexität der Szenarien

von $\lambda_{\text{lifespan}} = 1/1000$ angenommen wird. Die Werte für λ_{arrival} und $\lambda_{\text{lifespan}}$ wurden dabei wie in [53] gewählt.
In Abbildung 5.14 ist die Anzahl der zu jedem diskreten Simulationszeitpunkt aktiven Netzwerkservices visualisiert. Für jeden Zeitpunkt sind das also diejenigen Netzwerkservices, die erfolgreich eingebettet werden konnten und deren Verweildauer noch nicht überschritten ist. Situationen, in denen CoordVNF nicht in der Lage war, eine VNFR erfolgreich einzubetten, sind in der Grafik rot markiert. Wie sich ablesen lässt, kann CoordVNF den größten Teil der 1000 einzubettenden Anfragen erfolgreich zuweisen. In der Tat werden 918 Anfragen eingebettet; lediglich 82 der VNFRs, also 8,2%, müssen zurückgewiesen werden.
Ergänzend zu den hier geplotteten Ergebnissen wurden Laufzeitmessungen durchgeführt. Ähnlich wie in den vorherigen Experimenten ist CoordVNF auch hier in der Lage, die Anfragen in der Regel innerhalb weniger Sekunden zu bearbeiten. Dies gilt sowohl für diejenigen VNFRs, die erfolgreich eingebettet werden können, als auch für diejenigen, die zurückgewiesen werden müssen. Als Extremwert wurde eine Laufzeit von 8 Minuten gemessen.

5.3.2.6 Analyse des Laufzeitverhaltens in schwierigen Einbettungsszenarien

Im Folgenden wird gezeigt, wann es CoordVNF besonders schwer fällt, eine Einbettungsentscheidung zu treffen. Zu diesem Zweck werden die Ressourcenanforderungen der VNFR-Anfragen variiert, indem die initialen Datenraten der VNFRs, r^i_{init}, unterschiedlich hoch gewählt werden. Dies wirkt sich auf das Volumen des Datenstroms aus, der von den Netzwerkfunktionen der VNFRs verarbeitet werden muss. Entsprechend höher fallen die Ressourcenanforderungen der VNF-Instanzen sowie die der virtuellen Links aus. In jedem Szenario wurde die Einbettung von 5 VNFRs mit jeweils 10 VNFs in ein Substratnetz mit 50 Knoten berechnet.
Abbildung 5.15 zeigt, dass mit zunehmender Datenrate immer weniger VNFRs eingebettet werden können. Je höher die initiale Datenrate gesetzt wird, desto

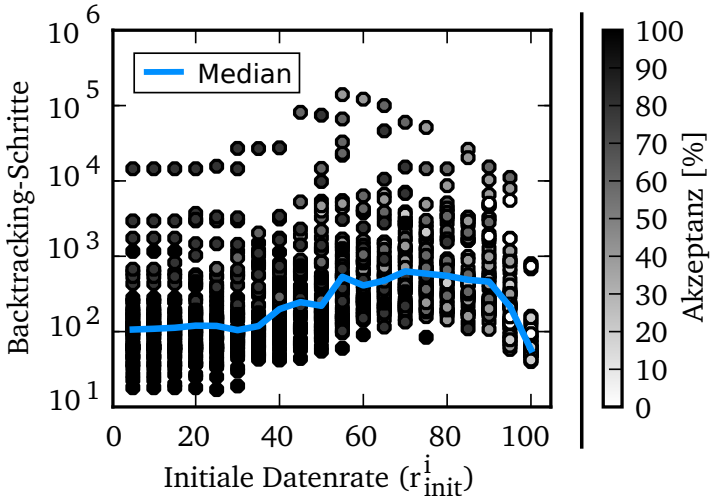


Abb. 5.16: Zusammenhang von Akzeptanz und Backtracking-Verhalten in Abhängigkeit von der Komplexität der Szenarien

höher fällt auch die Anzahl der Experimente aus, in denen nur ein Teil der jeweils 5 simulierten VNFR-Anfragen erfolgreich eingebettet werden kann. Für r^i_{init}-Werte zwischen 90 und 100 werden fast alle Anfragen zurückgewiesen.

Abbildung 5.16 visualisiert, wie schwer sich CoordVNF mit der Bearbeitung der Einbettungsanfragen tut. In dieser Graphik sind gleich zwei Metriken dargestellt. Über die Position der Einzelwerte lässt sich die Anzahl der Backtracking-Schritte ablesen, die im jeweiligen Experiment durchgeführt wurden. Ebenfalls wird der Median dieser Werte angegeben. Als zweite Metrik ist die Akzeptanz durch Einfärbung der Einzelmessungen markiert. Je heller ein Punkt ist, desto weniger VNFR-Anfragen konnten in dem entsprechenden Experiment erfolgreich eingebettet werden.

Wie sich aus der Abbildung ablesen lässt, nimmt die Anzahl der durchgeführten Backtracking-Schritte mit steigender Einbettungslast zunächst zu, vgl. Werte für $r^i_{\text{init}} \in [10, 80]$. Eine bemerkenswerte Veränderung des Backtracking-Verhaltens lässt sich im Intervall $r^i_{\text{init}} \in [90, 100]$ erkennen. Anders als zuvor steigt die Anzahl der Backtracking-Schritte plötzlich nicht mehr, sondern sie fällt sogar auf noch niedrigere Werte als bei Experimenten mit sehr geringer Einbettungslast.

Wie nun aus Abbildung 5.17 ersichtlich wird, ist CoordVNF in der Lage, die meisten der Anfragen mit *geringem* Ressourcenbedarf unmittelbar einzubetten (hier der Fall für $r^i_{\text{init}} \leq 30$). Für solche Szenarien werden im Durchschnitt nur verhältnismäßig wenige Backtracking-Schritte benötigt. In diesen Fällen fällt es dem CoordVNF-Algorithmus zunächst leicht, passende Einbettungen ausfindig zu machen. Hier ist die Wahrscheinlichkeit, dass CoordVNF bereits aus der Menge der ersten ausgewählten Zuweisungskandidaten passende Platzierungsmöglichkeiten findet, verhältnismäßig hoch. Entsprechend schneller terminiert

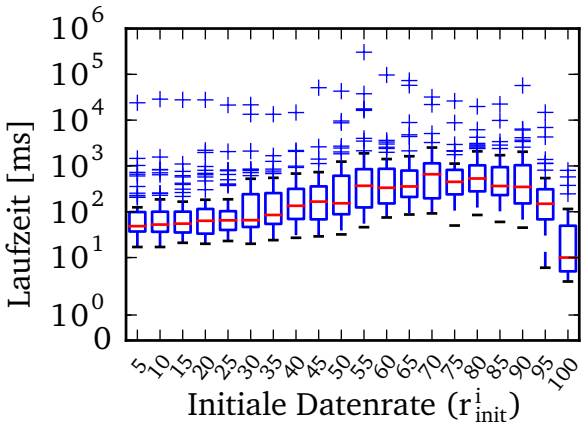


Abb. 5.17: Laufzeit von CoordVNF in Abhängigkeit von der Komplexität der Szenarien

der Algorithmus in diesen Szenarien.

In Szenarien, in denen eine besonders hohe Datenrate gewählt wurde, terminiert der Algorithmus wiederum sehr schnell (vgl. $r^i_{\text{init}} > 90$). Tatsächlich führt CoordVNF in solchen Fällen sogar noch weniger Backtracking-Schritte aus und terminiert in Folge dessen noch schneller als in Szenarien mit niedriger Datenrate. Das liegt daran, dass in diesen Fällen nur wenige geeignete Zuweisungsoptionen innerhalb des Substratnetzes zur Verfügung stehen, die CoordVNF iterativ ausprobieren kann. In diesen Fällen kann CoordVNF alle verbleibenden Zuweisungsoptionen mit nur wenig Aufwand durchprobieren und sehr schnell solche Anfragen verwerfen, für die sich keine dieser Optionen eignet. In diesen Fällen reduziert sich die Laufzeit des Algorithmus, da nur eine sehr überschaubare Menge von Einbettungsmöglichkeiten geprüft werden muss.

„Schwierig“ sind für CoordVNF folglich solche Experimente, in denen die Anforderungen der VNFRs gerade so hoch sind, dass es einerseits sehr viele Zuweisungsmöglichkeiten innerhalb des Substratnetzes für die Platzierung der VNF-Instanzen gibt, es dem Algorithmus aber andererseits nicht gelingt, die VNFRs zu platzieren. In den gezeigten Experimenten ist das dann der Fall, wenn die Einbettungslast weder besonders niedrig noch besonders hoch ist. Dann ist CoordVNF weder unmittelbar in der Lage, auf Anhieb Zuweisungen für die VNFs ausfindig zu machen, noch zu entscheiden, dass die Einbettung der VNFR-Anfrage aufgrund mangelnder Ressourcen nicht machbar ist. In diesen Fällen müssen zunächst sehr viele Zuweisungsoptionen geprüft werden, bevor die Anfrage entweder akzeptiert oder zurückgewiesen werden kann. Dieser Zusammenhang lässt sich direkt am in Abbildung 5.16 gezeigten Backtracking-Verhalten ablesen. Für $r^i_{\text{init}} = 80$ ist die Anzahl der Backtracking-Schritte besonders hoch.

5.4 Zusammenfassung

CoordVNF ist ein heuristischer Ansatz zur effizienten Lösung des NFV-RA-Problems. Beim NFV-RA-Problem müssen virtuelle Netzwerkfunktionen zunächst zu einem Netzwerkservice verkettet und dann in die Substratnetzinfrastruktur eingebettet werden. CoordVNF löst die beiden Teilprobleme in einem gemeinsamen, koordinierten Schritt. Wie die Evaluationsergebnisse zeigen, ist CoordVNF in der Lage, Einbettungen auch innerhalb großer Netzwerktopologien vorzunehmen. Das Laufzeitverhalten unterscheidet sich in solchen Szenarien signifikant vom aus der Literatur bekannten MIVNF-Algorithmus. Die Ergebnisse der umfangreichen Simulationen zeigen, dass es CoordVNF in sehr vielen Szenarien gelingt, innerhalb kürzester Zeit valide Einbettungsmöglichkeiten ausfindig zu machen.

Wie in diesem Kapitel gezeigt, kann die Backtracking-Metrik als ein Maß dafür angesehen werden, wie schwer sich CoordVNF mit der Einbettung von VNFR-Anfragen tut. Weiterführende Optimierungsüberlegungen sollten daher insbesondere darauf aufbauen, diese Kennzahl niedrig zu halten. Ein möglicher Ansatz dazu könnte die Integration einer ausgefeilteren Bewertungsfunktion sein, die bei der Sortierung und Auswahl der in jedem rekursiven Schritt in Betracht gezogenen Zuweisungskandidaten angewendet wird, vgl. die in Abschnitt 5.2.2 beschriebenen Sortierkriterien. Insbesondere könnten bei der Bewertung einer Zuweisungsoption auch die zur Verfügung stehenden Ressourcen der Nachbarknoten und -links mit einbezogen werden. Darüber hinaus könnte CoordVNF um eine Rekonfigurationslogik erweitert werden. Schlägt die Einbettung eines Netzwerkservices fehl, weil in der Vergangenheit andere Services ungünstig platziert wurden, weist CoordVNF die neue Anfrage zurück. Alternativ würde es sich anbieten, bereits durchgeführte Einbettungen so abzuändern, dass die Einbettung neuer Anfragen wieder möglich wird. Da die Migration bereits aktiver Netzwerkfunktionen i.d.R. mit Unterbrechungen der betroffenen Netzwerkservices verbunden ist, sollte ein Rekonfigurationsalgorithmus allerdings darauf ausgelegt sein, die Auswirkungen der Rekonfigurationen zu berücksichtigen.

In diesem Kapitel stand die Reduktion der Einbettungskosten im Vordergrund. Darüber hinaus ist die Betrachtung anderer Optimierungsziele naheliegend. Beispielsweise versuchen Betreiber von Mobilfunknetzen, die Übertragungsverzögerungen innerhalb der Infrastruktur gering zu halten. Diesbezüglich können Algorithmen entworfen werden, in deren Fokus das Deployment von NFV-Netzwerkservices mit niedrigem Kommunikations- und Verarbeitungsdelay steht. Ferner können Backupmaßnahmen in Betracht gezogen werden, mit denen sich eingebettete Netzwerkservices gegenüber Ausfällen absichern lassen [40]. Ein erstes Verfahren, über das sich die Robustheit eingebetteter NFV-Netzwerkservices steigern lässt, wird im nächsten Kapitel vorgestellt.

6 Ausfallsichere Einbettung virtueller Netzwerkservices

Für Infrastrukturprovider spielt das Thema Ausfallsicherheit eine zentrale Rolle. Der Ausfall von Netzwerkkomponenten muss zeitnah kompensiert werden, da ansonsten als Folge anhaltender Serviceausfälle mit drohenden Vertragsstrafen zu rechnen ist.
Dieses Kapitel diskutiert das Thema Ausfallsicherheit im Kontext des Deployments NFV-fähiger Netze. Das Deployment ausfallsicherer Netze wird in der Literatur vor allem im Zusammenhang mit dem VNE-Optimierungsproblem diskutiert. In Bezug auf das NFV-RA-Problem finden sich hier bislang noch keine Quellen.
Aufbauend auf den Inhalten des vorherigen Kapitels wird eine konzeptionelle Erweiterung des Platzierungsproblems vorgestellt, mit dem sich NFV-Netzwerkservices durch die Integration von Backup-Ressourcen gegen Ausfälle absichern lassen.
Aufbauend auf dem CoordVNF-Algorithmus wird gezeigt, wie ein effizientes Deployment solcher ausfallsicheren Services algorithmisch umgesetzt werden kann. Dazu wird der SVNF-Algorithmus vorgestellt (SVNF für *Survivable embedding of Virtual Network Functions*). SVNF stellt sicher, dass eingebettete Netzwerkservices im Falle eines Fehlers auf Backup-Ressourcen ausweichen können und innerhalb des physischen Netzes ausreichend Kapazitäten zum Betrieb dieser Backup-Konfigurationen zur Verfügung stehen.

Im Kontext ausfallsicherer Netzwerkinfrastrukturen ist zwischen den sogenannten *Primärinstanzen* und den *Backup-Instanzen* zu unterscheiden. Primärinstanzen sind für den Normalbetrieb der Netzwerkservices vorgesehen, während auf Backup-Instanzen nach Ausfällen von Primärressourcen zurückgegriffen wird.
SVNF ist in der Lage, die Robustheit von NFV-Deployments anhand der folgenden Strategien zu steigern:

- Robustes Deployment virtueller Links
  Diese Strategie erhöht die Robustheit virtueller Netzwerkservices gegenüber Ausfällen physischer Links: Fällt eine Primär-Verbindung innerhalb des physischen Netzes aus, ist sichergestellt, dass Knoten, die über den betroffenen physischen Link miteinander verbunden sind, zu jedem Zeitpunkt über Backup-Pfade kommunizieren können.

- Robustes Deployment von VNF-Instanzen

Diese Strategie erhöht die Robustheit virtueller Netzwerkservices gegenüber Ausfällen physischer Knoten. Durch Reservierung von *Cold Backups* werden Primär-VNF-Instanzen gegenüber Hard- und Softwareausfällen abgesichert. Nach dem Ausfall einer Primärinstanz ist dann sichergestellt, dass die Funktionalität dieser Instanz auf andere Backup-Knoten umgezogen werden kann. Dazu werden auf anderen physischen Knoten Reservekapazitäten bereit gehalten. Darüber hinaus sorgt SVNF dafür, dass die Backup-Instanzen an die übrigen Funktionen des Netzwerkservices angebunden sind.

Bei der Reservierung von Backup-Pfaden muss sichergestellt sein, dass ein Backup-Pfad durch den Ausfall des Primärpfades nicht in Mitleidenschaft gezogen wird. Für eine Backup-Verbindung sollten also Pfade innerhalb des physischen Netzes reserviert werden, die disjunkt zum Primärpfad sind. Ebenso sollte eine Backup-VNF-Instanz auf einem anderen physischen Knoten eingebettet werden als seine Primärinstanz, damit die Backup-Instanz unabhängig von der Funktionsfähigkeit der Primärinstanz bleibt.

In diesem Kapitel beziehen sich die Begriffe *Substratnetz*, *Substratknoten* und *Substratlink* immer auf das physische Netz, die physischen Knoten bzw. die physischen Links des Netzes. Die Einbettung ausfallsicherer Netzwerkservices in Szenarien mit verschachtelter Virtualisierung (*Nested virtualization*, vgl. Kapitel 1, Abschnitt 1.2) wird hier nicht betrachtet.

Der im Folgenden diskutierte SVNF-Algorithmus wurde auf der IEEE NFV-SDN Konferenz vorgestellt [17].

6.1 Abgrenzung zu verwandten Arbeiten

In Kapitel 5, Abschnitt 5.1, wurde bereits ein Überblick über die aus der Literatur bekannten Ansätze zur Lösung des allgemeinen NFV-RA-Deployment-Problems gegeben. Darüber hinaus finden sich bislang noch keine vergleichbaren Ansätze, die das Thema Ausfallsicherheit im Zusammenhang mit diesem Optimierungsproblem behandeln.
Allerdings wurden für das verwandte VNE-Problem bereits mehrere Ansätze vorgeschlagen, die sich mit der Frage befassen, inwiefern sich virtuelle Netze gegen Ausfälle absichern lassen. Obwohl sich das VNE-Problem vom NFV-RA-Problem in mehrerlei Hinsicht unterscheidet (vgl. Abschnitt 2.3.3), lassen sich einige der in diesem Zusammenhang diskutierten Ideen als Gedankenanstoß für die Konzeptionierung ausfallsicherer NFV-RA-Deployment-Algorithmen nutzen. Auf diese VNE-Verfahren soll im Folgenden eingegangen werden:

Beispielsweise wurde von Rahman et al. ein VNE-Verfahren beschrieben, welches Backup-Kapazitäten auf den Links des physischen Netzes reserviert. Nach

Ausfall einzelner Kommunikationsverbindungen kann die Konnektivität der betroffenen virtuellen Links wiederhergestellt werden [93, 94]. In [94] werden zwei unterschiedliche Möglichkeiten diskutiert, mit denen sich die schnelle Wiederherstellbarkeit der Kommunikationsverbindungen grundsätzlich umsetzen lässt.

- Lokale Backup-Maßnahmen
  Bei diesem Ansatz werden Backup-Verbindungen für die Links des Substratnetzes reserviert. Für jeden einzelnen Substratlink eines zugewiesenen Pfades werden Backup-Verbindungen aufgespannt: Für einen Substratlink, der zwei Substratknoten A und B miteinander verbindet, wird also ein Alternativpfad innerhalb des Netzes reserviert, über den die beiden Knoten miteinander kommunizieren können.

- Globale Backup-Maßnahmen
  Bei diesem Ansatz werden Ende-zu-Ende-Verbindungen durch Reservierung von Backup-Pfaden vor Ausfällen geschützt. Im Unterschied zum zuvor beschriebenen lokalen Ansatz sind hier keine Backup-Ressourcen für die Absicherung einzelner Substratlinks eines Pfades vorgesehen. Stattdessen wird für jeden Primärpfad ein eigenständiger, disjunkter Backuppfad reserviert.

In beiden Fällen handelt es sich um sogenannte *fast restoration*-Mechanismen, mit denen ein umgehendes Umschalten auf die reservierten Backup-Ressourcen nach Ausfällen von Primärverbindungen möglich ist. Der Vorteil lokaler Backup-Maßnahmen liegt in der grundsätzlich schnelleren Wiederherstellbarkeit einzelner ausfallender Links. Solche Ausfälle lassen sich sehr schnell vom Netzwerk abfangen, da der Datenverkehr nur lokal zwischen den Teilelementen des betroffenen Substratpfades umgeleitet werden muss. Werden dagegen globale Backup-Maßnahmen getroffen, muss die komplette Ende-zu-Ende-Verbindung auf den disjunkten Backup-Pfad migriert werden, was innerhalb des Netzwerks mit einem geringfügig höheren Migrationsaufwand verbunden ist. Da in diesem Fall jedoch lediglich ein einzelner Backup-Pfad reserviert werden muss und nicht wie bei lokalen Backups für jeden Link des Primärpfades ein dedizierter Pfad, fällt der Ressourcenbedarf in der Regel geringer aus. Die von Rahman et al. vorgestellte Arbeit konzentriert sich, wie auch alle weiteren hierzu relevanten Publikationen, auf die Diskussion eines Ende-zu-Ende-Ansatzes, bei dem globale Backup-Maßnahmen getroffen werden. Aus diesem Grund wird im Folgenden ebenfalls davon ausgegangen, dass die Kommunikationspfade durch die Reservierung solcher globalen Backup-Verbindungen geschützt werden sollen.

Xu et al. präsentieren einen weiteren VNE-Algorithmus, der nicht nur für die Kommunikationsverbindungen Backup-Ressourcen reserviert, sondern auch virtuelle Knoten gegen Ausfälle absichert [104]. Das Verfahren greift auf Kon-

zepte der linearen Programmierung zurück und löst das Optimierungsproblem heuristisch.

Guo et al. schlagen vor, nicht für jede Primärressource dedizierte Backupressourcen zu reservieren, da der zeitgleiche Ausfall gleich mehrerer Primärressourcen unwahrscheinlich ist [62]. Stattdessen teilt ein Pool mehrerer Primärressourcen ein gemeinsames Backup-Kontingent. Das VNE-Verfahren stellt die Ausfallsicherheit der eingebetteten virtuellen Infrastrukturen sicher, solange pro Pool höchstens eine Primärressource ausfällt.

In Abgrenzung zu diesen Ansätzen aus der Literatur wird im Folgenden das Thema Ausfallsicherheit im Zusammenhang mit dem NFV-RA-Problem motiviert. Dazu wird zunächst diskutiert, wie die verschiedenen Überlegungen aus dem VNE-Umfeld in diesem neuen Kontext Anwendung finden können. Anschließend wird ein auf CoordVNF aufbauender Einbettungsalgorithmus vorgestellt, der in der Lage ist, sowohl virtuelle Kommunikationsverbindungen als auch VNF-Instanzen gegen Ausfälle abzusichern.

6.2 Absicherung virtueller Netzwerkservices gegen Ausfälle

Im Kontext des VNE-Problems wie auch des NFV-RA-Problems lässt sich die Widerstandsfähigkeit virtueller Netzwerkservices durch die Reservierung von Fallback-Ressourcen erhöhen. Da die Bereithaltung von Backup-Kapazitäten zusätzliche Kosten verursacht, kann es je nach Anwendungsfall sinnvoll sein, solche Schutzmaßnahmen nur für bestimmte Netzwerkfunktionen zu integrieren.

Bei Services, denen weniger hohe Priorität zugeordnet wird, kann gegebenenfalls von einer umgehenden Reparatur abgesehen und stattdessen abgewartet werden, bis der Infrastrukturprovider die Hardwareschäden behoben hat. Beispiele für solche gegebenenfalls nicht-eiligen Dienste, die sich in begrenztem Rahmen zeitlich verschieben lassen, sind die regelmäßige Durchführung von Virenscans sowie die Synchronisation von Backups [96].

Bei Netzwerkservices, die von Dritten und nicht vom Infrastrukturprovider selbst betrieben werden, sind Bestimmungen, die die Ausfallsicherheit betreffen, in den Service Level Agreements geregelt, die zwischen den Parteien vertraglich vereinbart wurden. Der Infrastrukturprovider muss sicherstellen, dass die Zusicherungen für diese Services eingehalten werden.

Im Folgenden wird das robuste Deployment von virtuellen Links sowie von VNF-Instanzen im Kontext des NFV-RA-Problems näher beleuchtet.

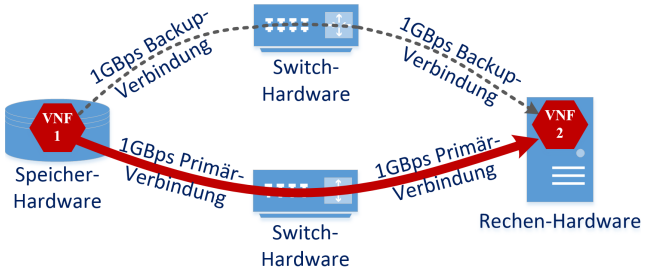


Abb. 6.1: Integration von Backup-Pfaden

6.2.1 Robustes Deployment virtueller Links

Zur Steigerung der Ausfallsicherheit virtueller Links können Backup-Pfade zugewiesen werden, die zusätzlich zu den Primärpfaden reserviert werden. Fällt ein Primärpfad aus, kann das System den Netzwerkfluss umgehend über den Backup-Pfad umleiten. Der Backup-Pfad, der innerhalb des Substratnetzes reserviert wird, kann sich dabei ebenso wie der Primärpfad über mehrere Zwischenknoten erstrecken (diese Zwischenknoten werden in der Literatur auch als *hidden hops* bezeichnet [28]. Das sind die Knoten, durch die ein Substratpfad führt, die aber weder Start- noch Zielknoten des Pfades sind). Dies ist in Abbildung 6.1 visualisiert. Hier wird zusätzlich zu dem Primärpfad eine Backup-Verbindung installiert. Wie beim Primärpfad auch, werden hier auf jedem Substratlink, der Teil des Backup-Pfads ist, jeweils zusätzlich 1 Gbps an Bandbreite reserviert. Aus Gründen der Lesbarkeit werden in diesem Kapitel lediglich die logischen Verbindungen der Netzwerkfunktionen visualisiert und die Links des Substratnetzes ausgeblendet.
Eine wesentliche Eigenschaft, die für jeden Backup-Pfad aus Gründen der Ausfallsicherheit erfüllt sein muss, ist die Link- und Knoten-Disjunktheit zum Primärpfad. Andernfalls wäre der Backup-Pfad nicht unabhängig von der Funktionalität des Primärpfades, was dazu führen würde, dass auch der Backup-Pfad von einem Ausfall des Primärpfades in Mitleidenschaft gezogen werden kann. Formal lässt sich die Bedingung der *Link-Disjunktheit* wie folgt spezifizieren:

$$\forall l \in L_{\text{inst}}^{i,j} : \exists p, b \in P \wedge p \cap b \equiv \emptyset \tag{6.1}$$

Für jeden virtuellen Link l einer verketteten Netzwerkfunktion muss ein Backup-Pfad b reserviert werden, der link-disjunkt zu dem Primärpfad p ist, der für l zugewiesen wird. $P \subseteq 2^L$ repräsentiert dabei die Menge aller möglichen Pfade innerhalb des Substratnetzes. Fällt ein Kommunikationslink innerhalb des Substratnetzes aus, ist durch diese Bedingung sichergestellt, dass von einem Linkausfall des Primärpfades nicht auch gleichzeitig der Backup-Pfad betroffen ist.
Zusätzlich zu der Link-Disjunktheit muss außerdem gelten, dass der Backup-Pfad nicht über Knoten des Primärpfades führt, da ein ausgefallener Substratknoten im Allgemeinen nicht mehr in der Lage ist, eintreffende Netzwerkflüsse

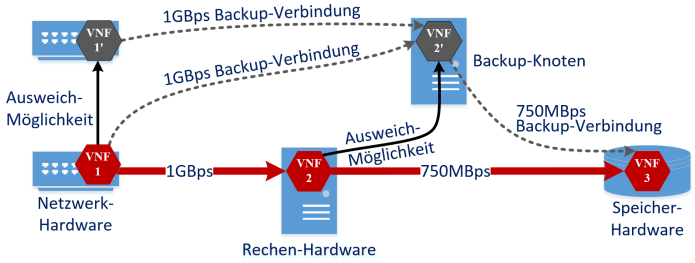


Abb. 6.2: Integration einer Backup-VNF-Instanz für eine Primärinstanz *VNF 2*

weiterzuleiten. Formal lässt sich diese *Knoten-Disjunktheitseinschränkung* wie folgt spezifizieren:

$$\forall l \in L_{\text{inst}}^{i,j} : \exists p, b \in P \wedge v(p) \cap v(b) \equiv \emptyset \tag{6.2}$$

Dabei bestimmt die Funktion $v : P \rightarrow 2^N$ die *hidden hops* eines Substratpfades. Ein NFV-RA-Deployment-Algorithmus, der virtuelle Kommunikationsverbindungen ausfallsicher zuweisen soll, muss also sowohl die durch Formel 6.1 spezifizierte Link-Disjunktheitsregel als auch die in Formel 6.2 angegebene Knoten-Disjunktheitsregel berücksichtigen. Für die ausfallsichere Einbettung eines virtuellen Links l muss insgesamt also gelten:

$$\forall l \in L_{\text{inst}}^{i,j} : \exists p, b \in P \wedge p \cap b \equiv \emptyset \wedge v(p) \cap v(b) \equiv \emptyset \tag{6.3}$$

6.2.2 Robustes Deployment von VNF-Instanzen

Damit auch VNF-Instanzen selbst vor Ausfällen geschützt werden können, müssen zusätzliche Absicherungsmaßnahmen getroffen werden. Für VNF-Instanzen, die einem Substratknoten zugewiesen werden sollen, müssen entsprechende Backupressourcen auf einem oder mehreren Backup-Knoten bereitgestellt werden. Fällt ein Substratknoten aus, muss für jede der dort installierten VNF-Instanzen klar sein, auf welchem Ersatzknoten die Backup-Instanz betrieben werden kann.
Aus Gründen der Ausfallsicherheit muss sichergestellt sein, dass eine Backup-VNF-Instanz nicht auf demselben Substratknoten eingebettet wird wie die Primärinstanz. Ist $\text{node}^{\text{primary}}$ der Substratknoten, auf dem die Primär-VNF-Instanz gehostet wird und $\text{node}^{\text{backup}}$ der Knoten, auf dem die Backup-Instanz liegt, muss also gelten:

$$\text{node}^{\text{primary}} \neq \text{node}^{\text{backup}} \tag{6.4}$$

Zusätzlich zu den Backup-Ressourcen, die für die Backup-VNF-Instanzen selbst reserviert werden, müssen weitere Ressourcen bereitgehalten werden, damit die Konnektivität der VNF-Instanzen nach Ausfall des Primärknotens auch

in der Backup-Konfiguration gewährleistet bleibt. Dies ist in Abbildung 6.2 gezeigt. Zu sehen ist hier ein Szenario, in dem die Integration einer Backup-VNF-Instanz exemplarisch dargestellt wird: Für die Primär-VNF-Instanz *VNF 2* (rot) wird zunächst eine Backup-Instanz auf einem anderen Substratknoten bereit gestellt (grau: *VNF 2'*). Da *VNF 2* Daten von der Primär-*VNF 1* entgegennimmt und nach der Verarbeitung an *VNF 3* weitergibt, muss die Backup-Instanz *VNF 2'* ebenfalls an *VNF 1* und *VNF 3* angebunden sein. Nach einem Ausfall von *VNF 2* ist so sichergestellt, dass der Datenfluss über *VNF 2'* weitergeleitet werden kann, vgl. Regel I und II. Damit der Netzwerkservice auch in Situationen funktionsfähig bleibt, in denen *beide* VNFs, also sowohl *VNF 1* als auch *VNF 2* ausfallen, muss *VNF 1'* darüber hinaus auch mit *VNF 2'* verbunden werden.
Für jede abzusichernde VNF-Instanz müssen also die folgenden Backup-Pfade reserviert werden:

I) Backup-Pfade zwischen allen Primärinstanzen, von denen Daten entgegengenommen und verarbeitet werden, und der Backup-Instanz

II) Backup-Pfade zwischen der Backup-Instanz und allen benachbarten Primärinstanzen, an die die Primärinstanz Daten weiterleitet

III) Backup-Pfade zwischen Backup-Instanzen benachbarter Primärinstanzen

Da zum robusten Deployment von VNF-Instanzen lediglich die Ausfallsicherheit der VNF-Instanzen sichergestellt werden muss, nicht aber die der virtuellen Links, können Backup-Pfade installiert werden, die nicht-disjunkt zu den jeweiligen Primärpfaden sind. Auf solchen sich überschneidenden Teilpfaden müssen Ressourcen zudem nicht „doppelt“ reserviert werden. Fällt nämlich eine VNF-Instanz aus, werden die Bandbreitenressourcen des Primärpfades nicht mehr benötigt, da als Folge des Ausfalls keine Daten mehr von bzw. zu der Primär-VNF-Instanz gesendet werden.

6.3 Der SVNF-Algorithmus

Die grundsätzliche Vorgehensweise von SVNF beim Deployment der VNF-Instanzen gleicht der Herangehensweise des in Kapitel 5 beschriebenen erweiterten CoordVNF-Algorithmus. Zuweisungen von VNF-Primärinstanzen erfolgen in rekursiven Schritten. Für den Fall, dass der Algorithmus nicht in der Lage ist, das Deployment aufgrund früher durchgeführter Zuweisungsmaßnahmen zu komplettieren, wird mittels Backtracking zu vorherigen Deployment-Zuständen zurückgekehrt. Ist der Algorithmus also einmal nicht in der Lage, passende Primär- oder Backup-Ressourcen zu finden, wird in jedem rekursiven Schritt zunächst die letzte Zuweisungsentscheidung rückgängig gemacht und daraufhin geprüft, ob eine andere Zuweisungsoption zum Erfolg führt. Eignet sich keine dieser Alternativen, wird wie beim CoordVNF-Algorithmus ein

Backtracking-Schritt durchgeführt, um vorherige Fehlentscheidungen zu revidieren und auf einen Zustand zurückzuspringen, in dem weitere alternative Zuweisungsmöglichkeiten geprüft werden können.

Die eigentliche Auswahl und Zuweisung von Backup-Ressourcen erfolgt innerhalb einer nun neu eingeführten AllocateBackups-Funktion, die in jedem rekursiven Schritt aufgerufen wird. Abgesehen vom Aufruf dieser Funktion sowie diverser Anpassungen verschiedener Funktionensignaturen entspricht die generelle Vorgehensweise von SVNF der des in Kapitel 5 vorgestellten CoordVNF-Algorithmus. Wird der Algorithmus instruiert, gar keine Backups zuzuweisen, sind die von ihm berechneten Ergebnisse folglich identisch mit denen des CoordVNF-Ansatzes.

Die Integration dieser Backup-Funktion in die rekursive Vorgehensweise des Algorithmus ist in Algorithmus 8 gezeigt. Die AllocateBackups-Funktion wird in jedem rekursiven Schritt aufgerufen. Der Aufruf der Funktion ist in Algorithmus 8 rot hervorgehoben, vgl. Zeile 21ff.. Tabelle 6.1 listet die Parameter der AllocateBackups-Funktion auf.

Parameter	**Beschreibung**
$G = (N, L)$	Das Substratnetz G mit den Knoten N und den Links L
$\text{node}^{\text{primary}}_{\text{previous}}$	Der Substratknoten, dem im vorherigen rekursiven Schritt die dort ausgewählte Primärinstanz einer VNF zugewiesen wurde
$\text{node}^{\text{backup}}_{\text{previous}}$	Der Substratknoten, dem im vorherigen rekursiven Schritt die dort ausgewählte Backup-Instanz einer VNF zugewiesen wurde
$\text{node}^{\text{primary}}_{\text{current}}$	Der Substratknoten, dem im aktuellen rekursiven Schritt die ausgewählte Primärinstanz einer VNF zugewiesen wurde
$\text{path}^{\text{primary}}_{\text{current}}$	Der Pfad innerhalb des Substratnetzes, der im aktuellen rekursiven Schritt ausgewählt wurde, um $\text{node}^{\text{primary}}_{\text{previous}}$ mit $\text{node}^{\text{primary}}_{\text{current}}$ zu verbinden
dr	Die Datenrate des aktuellen Teilflusses

Tabelle 6.1: Parameter der *AllocateBackup*-Funktion

Je nach Backup-Strategie bestimmt die hier neu eingeführte AllocateBackups-Funktion einen Knoten und/oder Pfad innerhalb des Substratnetzes, auf dem sich Backup-Ressourcen für die zuletzt zugewiesene Primärinstanz und/oder den Primärpfad bereitstellen lassen. Die Vorgehensweise bei der Auswahl von Backup-Ressourcen unterscheidet sich je nach angewandter Strategie, wie im Folgenden näher beschrieben wird.

```
1: function SVNF(VNFR^i,G,pathLengthLimit,numCandidates)
2:     for maxPathLength ∈ {1, ..., pathLengthLimit} do
3:         M ← REC(VNFR^i,G,None,None,
               r_init(VNFR^i),maxPathLength,numCandidates,{})
4:         if M ≠ ∅ then
5:             return M
6:         end if
7:     end for
8:     return ∅
9: end function
10:
11: function REC(VNFR^i,G,node^primary_previous,node^backup_previous,dr,maxPathLength,numCandidates,M)
12:     VNFInstances ← VNFOPTIONS(VNFFG^i,M)
13:     if VNFInstances=∅ then
14:         return M
15:     end if
16:     candidates ←
            BFS(VNFR^i,G,node^primary_previous,VNFInstances,dr,maxPathLength,numCandidates,M)
17:     sortCandidates(candidates)
18:     for all (VNFInstance,node^primary_current,path^primary_current) ∈ candidates do
19:         M' ← MAP-NODE-DEMANDS(node^primary_current,VNFInstance)
20:         M' ← M' ∪ MAP-LINK-DEMANDS(path^primary_current,dr)
21:         (M_backup, node^backup_current) ← AllocateBackups(
22:             G,node^primary_previous,node^backup_previous,node^primary_current,path^primary_current,VNFInstance,dr)
23:         if M_backup = ∅ then
24:             undoAll(M')
25:             continue
26:         end if
27:         M'' ← M'
28:         success ← True
29:         if alle eingehenden Kanten K der VNF-Instanz sind zugewiesen then
30:             //Einbetten der anschließenden Teilflüsse:
31:             for all l ∈ l_i(instanceof(VNFInstance)) do
32:                 dr' ← Σ_{k∈K} r^{i,j}_total(k) · d(l)
33:                 M_c ← REC(VNFR^i,G,node^primary_current,node^backup_current,dr',maxPathLength,
                        numCandidates,M'')
34:                 if M_c ≠ ∅ then
35:                     M'' ← M'' ∪ M_c
36:                 else
37:                     success ← False
38:                     break
39:                 end if
40:             end for
41:         end if
42:         if success=True then
43:             return M''
44:         else
45:             undoAll(M'')
46:         end if
47:     end for
48:     return ∅ //Backtracking zum vorherigen gültigen Zustand
49: end function
```

Algorithmus 8: Generelle Funktionsweise des SVNF-Algorithmus

6.3.1 Die SVNF-Strategie zum robusten Deployment virtueller Links

Zur Implementierung der Backup-Strategie für virtuelle Links berechnet die AllocateBackups-Funktion einen Backup-Pfad zwischen dem Substratknoten $\text{node}^{\text{primary}}_{\text{previous}}$, auf dem die im vorherigen rekursiven Schritt zugewiesene VNF-Primärinstanz eingebettet wurde, und dem Substratknoten $\text{node}^{\text{primary}}_{\text{current}}$, auf dem die aktuelle VNF-Instanz platziert ist. Diese Suche wird anhand des Algorithmus von Dijkstra durchgeführt, vgl. dazu Algorithmus 9, Zeile 3ff..
Die folgenden Bedingungen müssen dabei eingehalten bleiben:

1. Bei der Suche muss sowohl die Knoten- als auch die Link-Disjunktheitsregel eingehalten werden, vgl. Bedingung 6.3.
2. Auf jedem Link des Pfades muss genügend Bandbreite zur Verfügung stehen.

Daher wird der Dijkstra-Algorithmus auf einem Graph G' ausgeführt, der neben den Links des Primärpfades diejenigen Links des Substratnetzes G nicht enthält, die zu *hidden hops* des Primärpfades führen. Die Menge der Links L' des Graphen G' ist also:

$$\begin{aligned} L' \;=\; \{l \in L \mid\; & l \notin \text{path}^{\text{primary}}_{\text{current}} \\ & \wedge\; \text{destination}(l) \notin v(\text{path}^{\text{primary}}_{\text{current}}) \\ & \wedge\; \text{res}_{\text{link}}(l) \geq \text{dr}\ \} \end{aligned} \tag{6.5}$$

Dabei bestimmt die Funktion $v : P \to 2^V$ die *hidden hops* eines Substratpfades. Diese Strategie wird im Folgenden als die *Link-Backup-Strategie* bezeichnet.

6.3.2 Die SVNF-Strategie zum robusten Deployment von VNF-Instanzen

Um die Ausfallsicherheit von VNF-Instanzen sicherzustellen, muss für jede abzusichernde Primärinstanz ein Knoten bestimmt werden, auf dem ein entsprechendes Backup instanziiert wird. Wie in Abschnitt 6.2 erläutert, muss gleichzeitig auch die Konnektivität der Backup-Instanz gewährleistet werden. Aus diesem Grund sind zusätzlich entsprechende Backup-Verbindungen zu reservieren.

Die in Algorithmus 9 beschriebene AllocateBackups-Funktion geht in jedem rekursiven Schritt dazu wie folgt vor:

1. Suche nach einem Substratknoten für die Zuweisung der aktuellen VNF-Backup-Instanz, vgl. Zeile 15

Nachdem in einem rekursiven Schritt die Primär-VNF-Instanz auf einem Substratknoten platziert wurde, wird versucht, die entsprechende Backup-Instanz einzubetten. Dazu wird ausgehend vom Substratknoten $\text{node}_{\text{current}}^{\text{primary}}$, dem die aktuelle Primärinstanz zugewiesen wurde, eine Breitensuche nach einem möglichen Backup-Kandidaten $\text{node}_{\text{current}}^{\text{backup}}$ gestartet, vgl. Aufruf der $\text{BFS}^{\text{backup}}$-Funktion in Algorithmus 9, Zeile 15. Bei der Breitensuche werden (wie auch schon bei der Suche nach dem Primärknoten) nur solche Pfade weiterverfolgt, die über ausreichend Backup-Kapazitäten verfügen. Somit lässt sich die Konnektivität der Backup-Kandidaten zu vorherigen Primärknoten sicherstellen, vgl. Regel I, Abschnitt 6.2.2.

2. Verbinden der vorherigen VNF-Backup-Instanz mit der aktuell eingebetteten VNF-Primärinstanz, vgl. Zeile 18
   Wurde im vorherigen rekursiven Schritt eine Backup-VNF-Instanz platziert, wird sie an die aktuelle Backup-Instanz angebunden. Dazu wird mit Hilfe des Dijkstra-Algorithmus ein kürzester Pfad zwischen dem im vorherigen Rekursionsschritt reservierten Backup-Knoten $\text{node}_{\text{previous}}^{\text{backup}}$ und dem aktuellen Primärknoten $\text{node}_{\text{current}}^{\text{primary}}$ berechnet.

   Dieses Vorgehen führt dazu, dass Backup-Instanzen mit den benachbarten Primärinstanzen verbunden werden, an die Daten weitergegeben werden müssen, vgl. Regel II, Abschnitt 6.2.2.

3. Verbinden der vorherigen VNF-Backup-Instanz mit der aktuellen Backup-Instanz, vgl. Zeile 20
   Der Dijkstra-Algorithmus wird verwendet, um den kürzesten Pfad zwischen dem im vorherigen rekursiven Schritt zugewiesenen Backup-Knoten $\text{node}_{\text{previous}}^{\text{backup}}$ und dem aktuellen Backup-Knoten $\text{node}_{\text{current}}^{\text{backup}}$ zu finden.

   Anhand dieses Vorgehens werden benachbarte Backup-Instanzen miteinander verbunden, sodass die Konnektivität des Netzwerkservices auch bei gleichzeitigem Ausfall mehrerer benachbarter Primärinstanzen gewährleistet ist, vgl. Regel III, Abschnitt 6.2.2.

Die Suchtiefe der Breitensuche sowie des Dijkstra-Algorithmus richtet sich auch hier wieder nach dem *maxPathLength*-Parameter.
Diese Backup-Strategie wird im Folgenden als die *VNF-Backupstrategie* bezeichnet.

Gelingt es dem Algorithmus nicht, geeignete Backup-Ressourcen ausfindig zu machen, widerruft SVNF zuvor getroffene Einbettungsentscheidungen und prüft alternative Zuweisungsoptionen, vgl. Algorithmus 8, Zeile 23ff..

1: **function** AllocateBackups(
$G = (N, L)$,$\text{node}^{\text{primary}}_{\text{previous}}$,$\text{node}^{\text{backup}}_{\text{previous}}$,$\text{node}^{\text{primary}}_{\text{current}}$,$\text{path}^{\text{primary}}_{\text{current}}$,VNFInstance,dr)
2: $M^{\text{backup}} \leftarrow \{\}$; $\text{node}^{\text{backup}}_{\text{current}} \leftarrow$ None
3: **if Link-Backups** OR **VNF- und Link-Backups then**
4: //Alle geeigneten Links, die sowohl die Knoten- als auch
5: //die Link-Disjunktheitsregel erfüllen:
6: $L' = \{l \in L \mid l \notin \text{path}^{\text{primary}}_{\text{current}} \wedge \text{destination}(l) \notin v(\text{path}^{\text{primary}}_{\text{current}}) \wedge \text{res}_{\text{link}}(l) \geq \text{dr}\}$
7: $G' \leftarrow (N, L')$
8: $\text{path}^{\text{backup}} \leftarrow$ findShortestPath(G', $\text{node}^{\text{primary}}_{\text{previous}}$, node)
9: **if** $\text{path}^{\text{backup}}_{\text{current}} \neq \emptyset$ **then**
10: $M^{\text{backup}} \leftarrow M^{\text{backup}} \cup$ MapDemands($\text{path}^{\text{backup}}_{\text{current}}$,dr)
11: **end if**
12: **end if**
13: **if VNF-Backups** OR **VNF- und Link-Backups then**
14: //Finde einen Backup-Knoten $\text{node}^{\text{backup}}_{\text{current}} \neq \text{node}^{\text{primary}}_{\text{current}}$:
15: $(\text{node}^{\text{backup}}_{\text{current}}, \text{path}^{\text{backup}}_{\text{current}}) \leftarrow \text{BFS}^{\text{BACKUP}}(G,\text{node}^{\text{primary}}_{\text{previous}},\text{dr},M)$
16: **if** $\text{node}^{\text{backup}}_{\text{current}} \neq$ None **then**
17: //Verbindung vorheriger Backup-Knoten → aktueller Primärknoten:
18: $\text{path}^{\text{backup}}_{\text{to primary}} \leftarrow$ findShortestPath(G, $\text{node}^{\text{backup}}_{\text{previous}}$,$\text{node}^{\text{primary}}_{\text{current}}$)
19: //Verbindung vorheriger/aktueller Backup-Knoten:
20: $\text{path}^{\text{backup}}_{\text{to backup}} \leftarrow$ findShortestPath(G, $\text{node}^{\text{backup}}_{\text{previous}}$,$\text{node}^{\text{backup}}_{\text{current}}$)
21: **if** $\text{path}^{\text{backup}}_{\text{to primary}} \neq None$ and $\text{path}^{\text{backup}}_{\text{to backup}} \neq None$ **then**
22: //Reserviere nur auf solchen Substratlinks zusätzliche Ressourcen,
23: //die nicht Teil des Primärpfades sind:
24: $\text{path}^{\text{backup}}_{\text{current}} \leftarrow \text{path}^{\text{backup}}_{\text{current}} \setminus \text{path}^{\text{primary}}_{\text{current}}$
25: $\text{path}^{\text{backup}}_{\text{to primary}} \leftarrow \text{path}^{\text{backup}}_{\text{to primary}} \setminus \text{path}^{\text{primary}}_{\text{current}}$
26: $\text{path}^{\text{backup}}_{\text{to backup}} \leftarrow \text{path}^{\text{backup}}_{\text{to backup}} \setminus \text{path}^{\text{primary}}_{\text{current}}$
27: $M^{\text{backup}} \leftarrow M^{\text{backup}} \cup$ map-node-demands($\text{node}^{\text{backup}}_{\text{current}}$,VNFInstance)
28: $M^{\text{backup}} \leftarrow M^{\text{backup}} \cup$ map-link-demands($\text{path}^{\text{backup}}_{\text{current}}$,dr)
29: $M^{\text{backup}} \leftarrow M^{\text{backup}} \cup$ map-link-demands($\text{path}^{\text{backup}}_{\text{to primary}}$,dr)
30: $M^{\text{backup}} \leftarrow M^{\text{backup}} \cup$ map-link-demands($\text{path}^{\text{backup}}_{\text{to backup}}$,dr)
31: **end if**
32: **end if**
33: **end if**
34: **return** (M^{backup}, $\text{node}^{\text{backup}}_{\text{current}}$)
35: **end function**

Algorithmus 9: Ausfindigmachen von Backup-Ressourcen

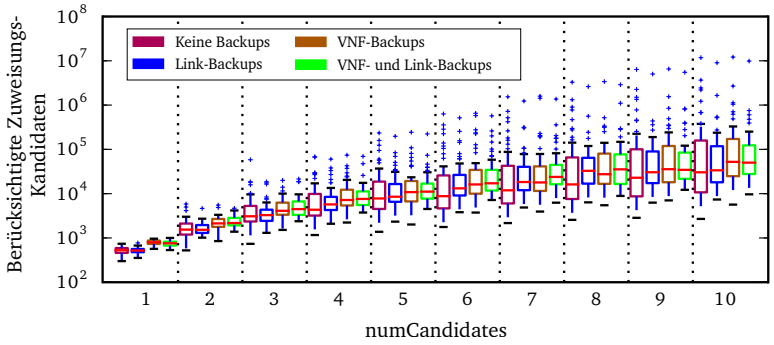


Abb. 6.3: Berücksichtigte Zuweisungskandidaten

6.4 Evaluation des SVNF-Algorithmus

Die Evaluation von SVNF erfolgt entsprechend der auch für den CoordVNF-Algorithmus gewählten Vorgehensweise, vgl. Abschnitt 5.3.1. Darüber hinaus wird angenommen, dass 50% der Primär-VNF-Instanzen bzw. 50% der Primärpfade gegen Ausfälle abzusichern sind. Die abzusichernden Elemente werden dabei für jede Einbettungsanfrage von der Simulationsumgebung zufällig bestimmt.

Zunächst wird SVNF in Offline-Szenarien evaluiert, in denen 10 VNFRs mit je 10 VNFs in ein Substratnetz mit 100 Knoten einzubetten sind. In diesen Szenarien wird die Auswirkung der Suchtiefe des Algorithmus auf die erzielten Deployment-Ergebnisse untersucht. Ferner wird analysiert, wie hoch die Einbettungskosten für das Deployment von Primärinstanzen im Verhältnis zu den Backup-Kosten ausfallen.
Anschließend wird untersucht, ob der Algorithmus auch in Szenarien mit großen Substratnetztopologien anwendbar ist. Dazu wird die Einbettung von 1000 VNFRs mit je 10 VNFs simuliert, die von SVNF in ein Substratnetz mit 1000 Knoten eingebettet werden sollen.

6.4.1 Suchtiefe

Zunächst soll untersucht werden, inwieweit die Suchtiefe des Algorithmus Auswirkungen auf das Laufzeitverhalten und die Akzeptanzwerte hat. Abbildung 6.3 zeigt die Anzahl der berücksichtigten Zuweisungskandidaten in Abhängigkeit von der durch den *numCandidates*-Parameter eingestellten Suchtiefe.
Für $numCandidates = 1$ werden bei jeder Strategie insgesamt nur wenige hundert bis eintausend Kandidaten berücksichtigt. Für $numCandidates = 5$ sind dies bereits um die zehntausend und für $numCandidates = 10$ werden in gleich mehreren Experimenten bereits hunderttausende, in Einzelfällen sogar Millionen von Zuweisungskombinationen geprüft. Auffallend ist hier, dass sich die Ausreißer für größere *numCandidates*-Werte häufen. Dies

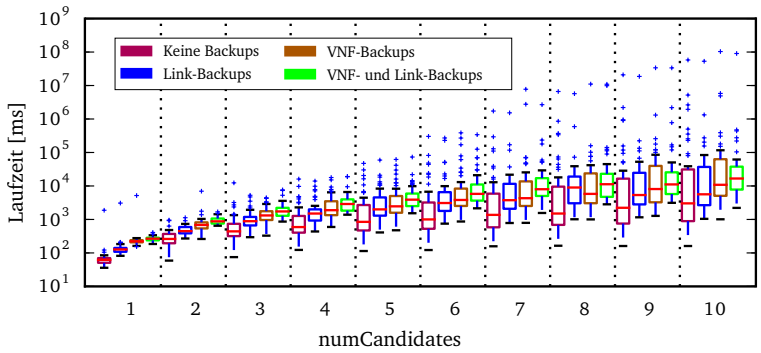


Abb. 6.4: Laufzeitverhalten in Abhängigkeit von der Suchtiefe

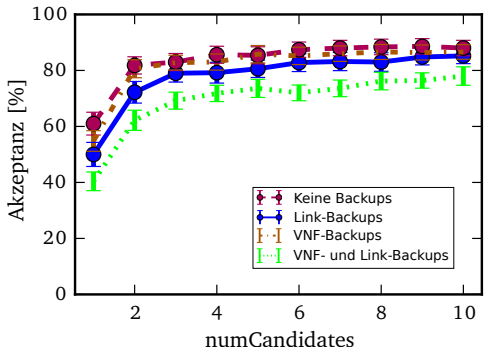
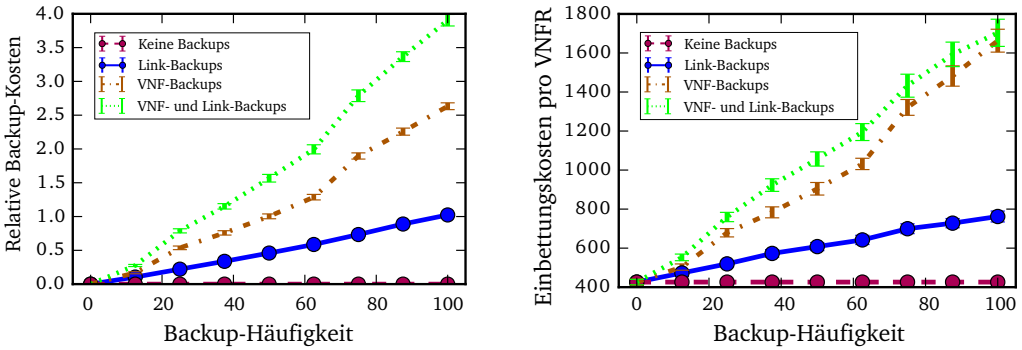


Abb. 6.5: Akzeptanz in Abhängigkeit von der Suchtiefe

stimmt mit den im vorherigen Kapitel diskutierten Messungen, die für den CoordVNF-Algorithmus durchgeführt wurden, überein.

Entsprechend vergleichbar verhalten sich die Backup-Strategien bezüglich des in Abbildung 6.4 gezeigten Laufzeitverhaltens. Für die einzelnen *numCandidates*-Werte lassen sich klare Unterschiede bezüglich des Laufzeitverhaltens der verschiedenen Strategien ablesen. Hier zeigt sich, dass die Link-Strategie im Mittel schneller ist als die VNF-Strategie und dass die kombinierte Backup-Strategie am meisten Berechnungszeit benötigt. Mit größeren *numCandidates*-Werten nimmt analog mit den in Abbildung 6.3 gezeigten Werten die Zahl der Ausreißer zu. Im Mittel liegt die Laufzeit in allen Fällen zwar im Sekundenbereich. in einigen Fällen benötigt der Algorithmus aber mehrere Minuten. Im Extremfall wurde für die VNF-Strategie für $numCandidates = 10$ eine Laufzeit von über 28 Stunden gemessen.
Bei allen Strategien ist das Verhältnis, in dem die Akzeptanz mit steigendem *numCandidates*-Wert zunimmt, vergleichbar, vgl. Abbildung 6.5. Auffallend ist, dass die kombinierte Backup-Strategie hier am schlechtesten abschneidet. Dagegen erzielt sowohl die Link- als auch die VNF-Strategie sogar annähernd ähnlich gute Akzeptanzwerte wie die Strategie, bei der keine Reservierungen von Backup-Ressourcen vorgenommen werden. Insgesamt lässt sich erkennen, dass $numCandidates = 4$ für alle Backup-Strategien in den hier untersuchten



(a) Relative Backup-Kosten (b) Einbettungskosten pro VNFR

Abb. 6.6: Relative Backup-Kosten und Einbettungskosten in Abhängigkeit von der Anzahl der abzusichernden Elemente

Szenarien zu guten Akzeptanz- und Laufzeitwerten führt. Aus diesem Grund wurde dies als fixer Wert für die weiteren Experimente verwendet.

6.4.2 Kosten für die Bereitstellung von Backups

Im Folgenden wird die Performanz des Algorithmus in Abhängigkeit von der Anzahl der zu reservierenden Backup-Ressourcen analysiert. Dazu wird die Backup-Häufigkeit der VNFR-Anfragen variiert, indem unterschiedlich viele VNFs bzw. unterschiedlich viele Links der VNFs ausgewählt werden, die durch Integration von Backup-Ressourcen gegen Ausfälle abzusichern sind. Zu diesem Zweck wird die Alevin-Simulationsumgebung angewiesen, bei der Generierung der VNFRs unterschiedlich viele VNFs bzw. Links zu spezifizieren, die SVNF beim Deployment durch Backups absichern soll.
Abbildung 6.6a veranschaulicht das Verhältnis zwischen den für die Primärinstanzen bzw. Primärpfade zugewiesenen Ressourcen und den reservierten Backup-Ressourcen. Interessant sind hier zwei Beobachtungen. Zum einen sind die Backup-Kosten der Link-Strategie, die linear steigen, für eine Backup-Häufigkeit von 100% gerade genauso groß wie die Kosten für die Einbettung der Primärressourcen. Zum anderen sind die Backup-Kosten für die beiden anderen Backup-Strategien deutlich höher. Der Backup-Aufwand der kombinierten Backupstrategie ist in etwa 4-mal so hoch wie der der Link-Strategie.
Abbildung 6.6b zeigt, dass die absoluten Einbettungskosten pro VNFR für alle Backupstrategien nahezu linear steigen. Durch die Backups der VNF-Instanzen steigen die absoluten Backup-Kosten sowohl bei der VNF-Strategie als auch bei der kombinierten Backupstrategie deutlich stärker an als bei der Link-Strategie. Für eine Backup-Häufigkeit von 100% sind die Einbettungskosten der kombinierten Strategie sogar mehr als doppelt so hoch wie die der Link-Strategie.
Entsprechend bemerkbar macht sich dies hinsichtlich der Akzeptanz, vgl. Abbildung 6.7: Hier zeigt sich, dass die Werte der kombinierten Strategie auf

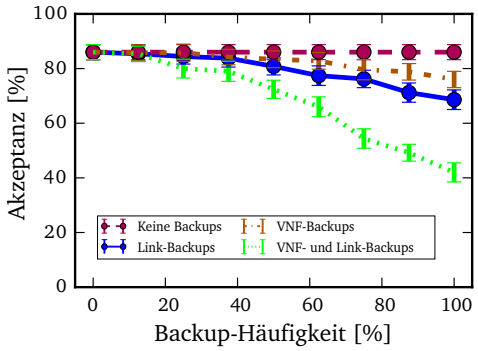


Abb. 6.7: Akzeptanz

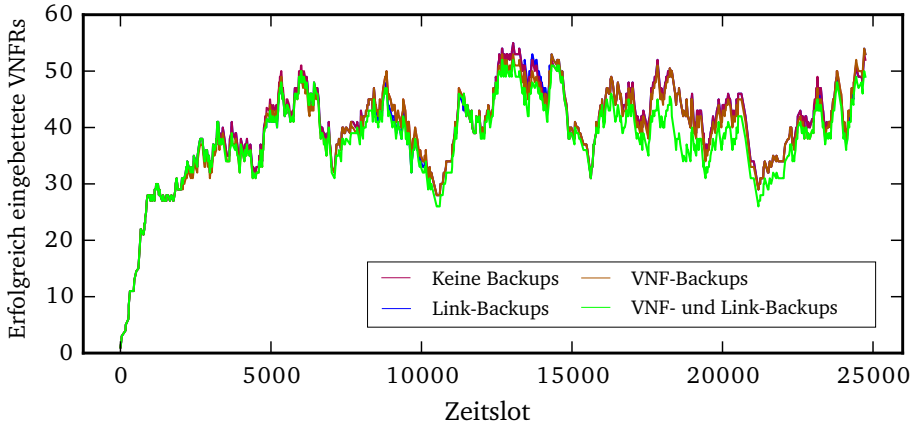


Abb. 6.8: Online-Szenario: Einbettung von 1000 VNFRs in ein Substratnetz mit 1000 Knoten

$\sim$ 40% abfallen, während die anderen beiden Backup-Strategien sich auch bei einer Backup-Häufigkeit von 100% in einem Bereich von 65-80% bewegen.

6.4.3 Einbettung vieler kontinuierlich eintreffender Anfragen in ein großes Substratnetz

Im Folgenden wird SVNF im Zusammenhang mit vielen kontinuierlich eintreffenden Anfragen evaluiert, die innerhalb eines großen Substratnetzes eingebettet werden sollen.
Hier wurde ein mit Hilfe des Barabási-Albert-Ansatzes generiertes Substratnetz mit 1000 Knoten erzeugt, in welches sukzessive 1000 VNFRs mit je 10 VNFs eingebettet werden sollen. Das Eintreffen der VNFR-Anfragen wurde als Poisson-Prozess mit der Ankunftsrate $\lambda_{\text{arrival}} = 4/100$ simuliert. Für die Lebensdauer der Anfragen wurde eine Exponentialverteilung angenommen mit einer durchschnittlichen Verweildauer von $\lambda_{\text{lifespan}} = 1000$ Zeitslots. Dies entspricht den für die Evaluation von CoordVNF gewählten Werten, die in Anlehnung an eine Publikation aus dem VNE-Umfeld [53] gewählt wurden.

Backup-Strategie	Einbettungen	Akzeptanz	∅ Laufzeit/VNFR
Keine Backups	3	99,7%	2s
Link-Backups	8	99,2%	3s
VNF-Backups	13	98,7%	5s
VNF- und Link-Backups	44	95,6%	6s

Tabelle 6.2: Online-Szenario: Vergleich der verschiedenen Backup-Strategien in Bezug auf die Anzahl der fehlgeschlagenen Einbettungsversuche und die durchschnittliche Einbettungszeit

Abbildung 6.8 zeigt die Anzahl der VNFRs, die zum jeweiligen Simulationszeitpunkt aktiv sind. Dies sind diejenigen VNFRs, die erfolgreich eingebettet werden konnten und deren Verweildauer noch nicht abgelaufen ist. Ergänzend dazu listet Tabelle 6.2 die Anzahl der fehlgeschlagenen Einbettungsversuche, die Akzeptanz und die Laufzeit des Algorithmus. Insgesamt lässt sich hier Folgendes beobachten. Die meisten VNFRs können zugewiesen werden, wenn überhaupt keine Backup-Ressourcen reserviert werden müssen. Denn in diesem Fall verbleiben nach einer Einbettung entsprechend mehr Kapazitäten ungenutzt, auf die bei Eintreffen künftiger Anfragen zurückgegriffen werden kann. Die Link-Strategie zeigt die zweitbesten Ergebnisse bezüglich der Akzeptanz. Etwas schlechter dagegen schneidet die VNF-Strategie ab. Wie auch in den Experimenten zuvor zeigt sich hier, dass die kombinierte Backup-Strategie am häufigsten Anfragen zurückweisen muss. Wie Tabelle 6.2 zeigt, ist die Anzahl der zurückgewiesenen Anfragen bei der kombinierten Strategie insgesamt sogar mehr als dreimal so hoch wie bei der VNF-Strategie.

In Tabelle 6.2 sind neben der Anzahl der fehlgeschlagenen Einbettungsversuche die durchschnittlichen Laufzeiten der verschiedenen Strategien aufgeführt. Es zeigt sich hier, dass die Link-Strategie der VNF-Strategie im Bezug auf die durchschnittliche Laufzeit überlegen ist. Das liegt daran, dass für jede VNF-Instanz, für die ein Backup bereitgestellt werden soll, zunächst eine Breitensuche durchgeführt werden muss. Für den damit gefundenen Backup-Knoten muss mit Hilfe des Dijkstra-Algorithmus eine Anbindungsmöglichkeit an den im vorherigen Rekursionsschritt platzierten Primärknoten sowie den ggf. zuvor zugewiesenen Backup-Knoten gefunden werden. Im Gegensatz dazu erfolgt bei der Link-Strategie lediglich eine einzelne Pfadsuche mit dem Dijkstra-Algorithmus, die Breitensuche entfällt. Die durchschnittliche Berechnungszeit pro VNFR lag in allen Fällen aber im Bereich von wenigen Sekunden.

6.5 Zusammenfassung

In diesem Kapitel wurde zunächst erörtert, wie sich Netzwerkservices durch Integration von Backup-Ressourcen gegen Ausfälle absichern lassen. Aufbauend auf dem CoordVNF-Algorithmus wurde dann SVNF als ein Ansatz zum Deployment robuster Netzwerkservices vorgestellt. Mit SVNF können sowohl Backup-Verbindungen zwischen VNF-Instanzen als auch Backups von VNF-Instanzen selbst reserviert werden.

Die Evaluationsergebnisse zeigen, wie sich die unterschiedlichen Backup-Strategien auf die Performanz des Algorithmus auswirken. Insgesamt zeigt sich, dass sich jede der Strategien auch im Zusammenhang mit großen Substratnetzen einsetzen lässt. Die durchschnittliche Laufzeit zur Berechnung eines Deployments bewegte sich im Bereich von wenigen Sekunden. Aufgrund der Tatsache, dass SVNF neben den Primärressourcen noch zusätzliche Backup-Ressourcen reservieren muss, ist die Akzeptanz dieses Algorithmus im Vergleich zu CoordVNF zwar geringer – insbesondere dann, wenn neben den virtuellen Kommunikationsverbindungen gleichzeitig auch die VNF-Instanzen gegen Ausfälle abgesichert werden sollen. Die Einbettungsergebnisse sind dafür aber robust gegenüber Knoten- und/oder Link-Ausfällen.

In den hier beschriebenen Szenarien wurde angenommen, dass der Serviceprovider innerhalb der Einbettungsanfrage spezifiziert, für welche Komponenten des Netzwerkservices Backupressourcen zuzuweisen sind und für welche nicht. Zudem wurde angenommen, dass die Komponenten des physischen Netzes mit der jeweils gleichen Wahrscheinlichkeit von Ausfällen betroffen sind. Für zukünftige Arbeiten würde sich hier eine differenziertere Betrachtung des Problems anbieten. Der Serviceprovider möchte vom Infrastrukturprovider in der Regel harte Zusicherungen über die Verfügbarkeit seiner Netzwerkdienste erhalten. Der Infrastrukturprovider soll die Dienste innerhalb des physischen Netzes so deployen, dass die vereinbarte Zuverlässigkeit der Dienste eingehalten werden kann.

Ein Ansatzpunkt für weitere Überlegungen in dieser Richtung ist die Miteinbeziehung von Ausfallwahrscheinlichkeiten physischer Hardwarekomponenten. Das Deployment kann dann in Abhängigkeit von diesen Ausfallwahrscheinlichkeiten erfolgen. Backup-Maßnahmen könnten in diesem Fall in Abhängigkeit vom Deployment der Primärressource erfolgen. Kann die Primärressource nur so platziert werden, dass die Wahrscheinlichkeit eines Ausfalls oberhalb des vereinbarten Levels liegt, kann die Robustheit der Einbettung durch Hinzuziehung geeigneter Backup-Ressourcen wieder entsprechend ausgeglichen werden.

Eine weitere Idee wäre, dieselben Backup-Ressourcen gleich für mehrere Primärressourcen zu reservieren. Im vorliegenden Kapitel wurde von einer *de-*

dizierten Reservierung von Backup-Ressourcen ausgegangen. Die exklusive, dedizierte Reservierung von Backup-Ressourcen führt jedoch in der Regel zu einer wesentlichen Erhöhung der Einbettungskosten. Für jeden virtuellen Link, der abgesichert wird, müssen zwei zueinander disjunkte Pfade innerhalb des Substratnetzes vorgehalten werden und für jeden dieser Pfade müssen entsprechende Ressourcen reserviert werden.
Allerdings ist es unwahrscheinlich, dass alle Backup-Ressourcen zur gleichen Zeit zum Einsatz kommen. Das wäre nur dann der Fall, wenn alle Primärknoten beziehungsweise -links zur gleichen Zeit ausfallen, ohne dass der Infrastrukturprovider in der Lage ist, zwischenzeitlich einige der defekten Komponenten auszutauschen. Aus diesem Grund ist eine kostengünstigere Reservierungsstrategie naheliegend, bei der die Backup-Ressourcen für mehrere virtuelle Links bzw. VNF-Instanzen gleichzeitig reserviert werden. Beispielsweise kann auf einem Substratknoten ein gewisses Kontingent an Rechenkapazität reserviert werden, auf welches zurückgegriffen werden kann, wenn einer der darüber abgesicherten Primärknoten ausfällt. Allerdings kann das System dann keine Ausfälle auffangen, wenn gleich mehrere der VNF-Instanzen betroffen sind, die sich dasselbe Backup-Kontingent teilen. Aus diesem Grund sollten die Ausfallwahrscheinlichkeiten der physischen Hardwarekomponenten bei der Zuweisung derartiger mehrfach zugewiesener Backup-Ressourcen in Betracht gezogen werden.

7 Fazit und Ausblick

Network Function Virtualization ist ein ganz aktueller Hype im Telekommunikationsbereich. In der Praxis hat sich gezeigt, dass der Aufbau klassischer Netze auf Basis von Einheiten, deren Funktionalität eng mit den Hardwarekomponenten verwoben ist, unflexibel und mittel- bis langfristig zu teuer ist. Durch die Virtualisierung von Netzwerkfunktionen wird diese enge Verbindung zwischen Hard- und Software aufgehoben.
Herzstück der NFV-Infrastruktur ist der sogenannte Orchestrator, der für die Platzierung der virtuellen Netzwerkservices innerhalb der Netzwerkinfrastruktur zuständig ist. Im Rahmen dieser Arbeit wurden verschiedene Algorithmen vorgestellt, mit denen der Orchestrator das Deployment dieser Services effizient berechnen kann. Die Bestimmung optimaler Deployment-Lösungen ist $\mathcal{NP}$-hart, weswegen die Anwendung optimaler Verfahren nur in sehr kleinen Szenarien möglich ist. Im Fokus dieser Arbeit stand daher die Konzeptionierung von heuristischen Verfahren, die auch im Zusammenhang mit Netzwerkstrukturen realer Größenordnung anwendbar sind.

Mit dem in dieser Arbeit vorgestellten DPVNE-Algorithmus ist es möglich, den Berechnungsaufwand, der mit der Lösung dieses komplexen Optimierungsproblems verbunden ist, auf mehrere Rechner zu verteilen. Dies ermöglicht die effiziente, parallele Verarbeitung von Einbettungsanfragen, was insbesondere dann von Vorteil ist, wenn viele virtuelle Netze in sehr große physische Infrastrukturen eingebettet werden sollen. DPVNE versucht, die Einbettungen parallel in mehreren Partitionen des Substratnetzes durchzuführen. Die Simulationsergebnisse zeigen, dass es mit DPVNE sehr effizient möglich ist, die Ausführung ursprünglich nicht-verteilter VNE-Deploymentalgorithmen auf mehrere Einbettungsknoten zu verteilen. Dabei bleiben die von DPVNE berechneten Einbettungsergebnisse vergleichbar mit denen der nicht-verteilten Verfahren.
DPVNE wurde darüber hinaus mit einem aus der Literatur bekannten anderen verteilten Einbettungsansatz verglichen. Wie die Ergebnisse zeigen, ist DPVNE diesem Ansatz überlegen – insbesondere kommt DPVNE als Folge der hierarchischen Strukturierung des Netzes mit einem um Größenordnungen geringeren Kommunikations-Overhead aus.
DPVNE wurde im Rahmen dieser Arbeit als Ansatz zur Lösung des VNE-Problems präsentiert. In VNE-Szenarien wird davon ausgegangen, dass die Struktur der virtuellen Netze fest vorgegeben ist. Die Anordnung der virtuellen Knoten ist, anders als bei NFV-Netzwerkservices, unveränderlich. Generell lässt sich der bei DPVNE verfolgte Gedanke, das physische Netz in autonom arbeitende Teilbereiche zu gliedern, auch auf NFV-Szenarien

übertragen. DPVNE kann das rechenintensive Deployment also grundsätzlich nicht nur in VNE- sondern auch in NFV-RA-Szenarien auf mehrere Rechner verteilen.

Anschließend wurde der Backtracking-basierte CoordVNF-Algorithmus vorgestellt. CoordVNF ist ein schneller Algorithmus zur kosteneffizienten Platzierung flexibler NFV-Netzwerkservices. Umfangreiche Simulationen zeigen, dass CoordVNF auch im Zusammenhang mit großen physischen Infrastrukturen anwendbar ist – anders als bisher bekannte Verfahren. Im Mittel benötigte CoordVNF für das Deployment eines Netzwerkservices lediglich wenige Sekunden. Nach bestem Wissen des Autors gibt es in der Literatur bislang nur den MIVNF-Algorithmus zur vollständigen Lösung des NFV-RA-Problems. Die im Rahmen dieser Arbeit durchgeführten Evaluationen zeigen, dass CoordVNF in der Lage ist, das Deployment von Netzwerkservices um Größenordnungen schneller vorzunehmen als MIVNF. In Szenarien mit großen Netzinfrastrukturen benötigt MIVNF sogar Stunden oder gar Tage, während CoordVNF das Problem innerhalb von Sekunden oder Minuten löst. Die weiteren Evaluationsergebnisse zeigen, dass CoordVNF das Optimierungsproblem nicht nur *schnell*, sondern auch *gut* löst. CoordVNF gelingt es, die Netzwerkservices sehr kosten- bzw. ressourceneffizient einzubetten. CoordVNF ist also der erste skalierbare, koordinierte Ansatz zur effizienten Lösung des NFV-RA-Deployment-Problems.

Durch die Integration von Backup-Ressourcen lassen sich virtuelle Netzwerkservices vor Ausfällen schützen. Beim Deployment können dabei grundsätzlich zwei Backup-Strategien verfolgt werden: Zunächst lässt sich die Robustheit virtueller Kommunikationsverbindungen durch Reservierung von Backup-Pfaden steigern. Darüber hinaus können Backups reserviert werden, um auch die Funktionalität von VNF-Instanzen zu gewährleisten. Der vorgestellte SVNF-Algorithmus ist in der Lage, beide Backup-Strategien beim Deployment der virtuellen Netzwerkservices umzusetzen. Die Evaluationsergebnisse zeigen, dass auch dieser Ansatz sinnvoll in großen Szenarien einsetzbar ist – trotz des Aufwands, der mit der zusätzlichen Reservierung von Backup-Ressourcen verbunden ist. SVNF ist der erste Deployment-Algorithmus, der in diesem Zusammenhang vorgestellt wurde.

Im Folgenden soll auf weitere Arbeiten eingegangen werden, die im Kontext dieser Thematik verfolgt werden sollten.

Der Fokus der vorliegenden Arbeit lag auf der Betrachtung von Deployment-Algorithmen, die in der Lage sind, virtuelle Netzwerkservices effizient einzubetten. Optimierungsziel der in diesem Kontext vorgestellten Verfahren war dabei die Reduktion des Ressourcenbedarfs der einzubettenden Dienste. Insbesondere in Telekommunikationsnetzen spielen weitere Faktoren, wie die Reduktion von Kommunikations- und Verarbeitungsverzögerungen, eine zentrale Rolle.

Bei der Platzierung von Netzwerkfunktionen sollten daher derartige Verzögerungen mit einbezogen werden, die durch das Co-Hosting mehrerer virtueller Kommunikationsverbindungen auf denselben Pfadsegmenten entstehen. Ferner sollte analysiert werden, welche Arten von VNFs effizient auf demselben physischen Rechner betrieben werden können. In Cloud-Umgebungen ist es beispielsweise nicht sinnvoll, mehrere virtuelle Maschinen, die I/O-intensive Aufgaben durchführen, auf einem gemeinsamen Rechner zu platzieren. Durch die gegenseitige Beeinflussung der virtuellen Maschinen kann es zu I/O-Bottlenecks kommen, die die Performanz der einzelnen Dienste beeinträchtigen. Solche Performanz-Abhängigkeiten der VNFs können und sollten bei der Platzierung von Netzwerkdiensten berücksichtigt werden. Darüber hinaus können die Algorithmen um Skalierbarkeitsaspekte erweitert werden. Soll beispielsweise der Kapazitätsbedarf einzelner Komponenten der eingebetteten virtuellen Netze zur Laufzeit erhöht werden, müssen zusätzliche Ressourcen innerhalb der Substratnetzinfrastruktur vorgenommen und entsprechende Anpassungen des Deployments vorgenommen werden. Dabei müssen die Auswirkungen, die mit der Migration bereits zugewiesener Komponenten verbunden sind, mit einkalkuliert werden. Dies gilt insbesondere auch dann, wenn neben der Kapazitätsanpassung bereits zugewiesener Netzwerkfunktionen auch die Struktur der eingebetteten Netzwerkservices zur Laufzeit vom Serviceprovider angepasst werden kann. Möchte der Serviceprovider also zur Laufzeit neue Netzwerkfunktionen in einen bestehenden Netzwerkservice integrieren oder alte Netzwerkfunktionen durch neuere ersetzen, sollten Anpassungen der Netzkonfiguration so durchgeführt werden, dass der Betrieb des rekonfigurierten Services möglichst unterbrechungsfrei fortgeführt werden kann.
Eine wesentliche Voraussetzung für die Integration der NFV-Technologie in produktiv eingesetzte Telekommunikationsnetze ist – unabhängig von der algorithmischen Umsetzung von Deploymentstrategien – die Einführung offener, herstellerübergreifender Standards zur Spezifikation sowie zur Integration virtueller Netzwerkservices. Eine wichtige Rolle spielen dabei die Fragen, wie virtualisierte Dienste sicher, zuverlässig und ressourceneffizient in die bestehende Infrastruktur des Netzes integriert werden können und wie die in Telekommunikationsnetzen gegebenen harten Performanzvorgaben erfüllt bleiben können. Tools und Technologien, die in heutigen Cloud-Rechenzentren eingesetzt werden, müssen zu diesem Zweck um NFV-spezifische Konzepte ergänzt und angepasst werden.

Die in der vorliegenden Arbeit diskutierten Algorithmen ermöglichen das effiziente Deployment von virtuellen Netzwerkservices – sogar in großen Netzinfrastrukturen. Sie können als Ausgangspunkt für die Umsetzung dieser weiteren Arbeiten dienen.

Glossar

Content Distribution Network (CDN)
Ein Netzwerk aus verteilten Servern, das zur Auslieferung von Internetinhalten verwendet wird. Die Server sind auf viele unterschiedliche geographische Regionen verteilt, sodass die Inhalte innerhalb dieser Regionen sehr schnell ausgeliefert werden können.

Deep Packet Inspection (DPI)
Eine Netzwerkfunktion, die in der Lage ist, den Inhalt von Datenflüssen anhand des Datenteils der Netzwerkpakete (und nicht nur anhand der Header-Informationen) zu analysieren.

Einbettungsknoten
Einbettungsknoten berechnen das Deployment virtueller Netze / virtueller Netzwerkservices. Bei nicht-verteilten Deploymentalgorithmen ist dafür lediglich ein einzelner Einbettungsknoten vorgesehen, während bei verteilten Deploymentalgorithmen mehrere Einbettungsknoten zum Einsatz kommen.

Infrastrukturprovider (ISP)
Der Infrastrukturprovider ist für den reibungslosen Betrieb der physischen Infrastruktur verantwortlich.

Load Balancer
Ein Load Balancer kann eingesetzt werden, um viele zu bearbeitende Anfragen auf mehrere parallel arbeitende Rechner aufzuteilen.

Long Term Evolution (LTE)
LTE ist eine Bezeichnung für einen Mobilfunkstandard.

Mobile Edge Computing (MEC)
MEC ist eine Mobilfunktechnologie, bei der Cloud-Hardware am Rande des Mobilfunknetzes installiert werden. Durch die unmittelbare Nähe zum Mobilfunk-Endnutzer können solche MEC-Server für den Betrieb äußerst latenzsensitiver Dienste verwendet werden.

Network Function Virtualization (NFV)
NFV ist ein Konzept, das die logische Funktionalität von der physischen Hardware trennt. Statt hochspezialisierter Hardwareboxen, deren Funktionalität eng mit der physischen Hardware verbunden ist, kommen in NFV-Netzen

Standardkomponenten zum Einsatz, auf denen verschiedene Netzwerkfunktionen installiert werden.

Network Function Virtualization-Resource Allocation (NFV-RA)
NFV-RA bezeichnet das theoretische Optimierungsproblem, das mit dem Deployment von NFV-Netzwerkservices verbunden ist. Zur Lösung des Problems müssen zwei Teilprobleme betrachtet werden: 1) Die Verkettung von Netzwerkfunktionen zu einem Netzwerkservice und 2) Das Deployment der verketteten Netzwerkfunktionen innerhalb des Substratnetzes (das VNF-FGE-Teilproblem).

Service Provider (SP)
Der SP definiert Netzwerkservices, die innerhalb der Infrastruktur des ISP installiert werden sollen.

Software Defined Networking (SDN)
SDN ist ein Konzept zur Flexibilisierung von Netzwerkinfrastrukturen. Grundidee dabei ist die Trennung der Kontroll- von der Datenebene.

Substratnetz, Substratknoten, Substratlink
In vielen Fällen beziehen sich die Begriffe Substratnetz/Substratknoten/Substratlink auf das physische Netz/einen physischen Knoten/einen physischen Link. In ein Substratnetz können parallel mehrere virtuelle Netze bzw. virtuelle Netzwerkservices eingebettet werden. Über verschachtelte Virtualisierung (*nested Virtualization*) ist es möglich, virtuelle Netze auch auf andere virtuelle Netze einzubetten. In einem solchen Szenario gibt es also mehrere Ebenen von Substratnetzen, von denen nur die unterste Ebene aus einem physischen Netz besteht. Daher wird insbesondere in der VNE-Literatur verallgemeinernd von Substratnetzen/-knoten/-links gesprochen.

Virtual Network Embedding (VNE)
Das VNE-Problem bezeichnet das theoretische Optimierungsproblem, das mit dem Deployment virtueller Netze verbunden ist.

Virtual Network Function embedding Request (VNFR)
Eine NFV-Einbettungsanfrage, in der insbesondere angegeben ist, in welcher Reihenfolge die VNFs des spezifizierten Netzwerkservices angeordnet werden müssen.

Virtual Network Function Forwarding Graph (VNF-FG)
Eine Verkettung von VNFs, die entsprechend der Vorgaben des VNFR gebildet wurde. Die Verkettung der VNFs ist das erste Teilproblem des VNF-RA-Problems. Für jeden NFV-Netzwerkservice wird ein VNF-FG bestimmt, der

in das Substratnetz eingebettet wird.

Virtuelle Maschine (VM)
Innerhalb einer VM kann ein Gastbetriebssystem installiert werden, was isoliert von anderen Gastsystemen auf demselben physischen Host betrieben werden kann.

Virtuelle Netzwerkfunktion (VNF)
Eine VNF ist eine logische Funktion, die zur Verarbeitung von Datenströmen eingesetzt werden kann. Beispiele für solche VNFs sind Decodierfunktionen, Kompressionsfunktionen oder Deep Packet Inspection-Funktionen.

Virtuelles Netz
Ein virtuelles Netz besteht aus virtuellen Knoten, die über virtuelle Links miteinander verbunden sind. Beim VNE-Problem sollen mehrere virtuelle Netze in ein Substratnetz eingebettet werden.

VNF Forwarding Graph Embedding (VNF-FGE)
Das VNF-FGE-Problem ist ein Teilproblem des VNF-RA-Problems, bei dem die eigentliche Platzierung der virtuellen Netzwerkfunktionen innerhalb des Substratnetzes berechnet werden soll.

Literaturverzeichnis

[1] *Alevin Simulations-Framework.* – `http://alevin.sf.net/`, abgerufen am 3. Juli 2016

[2] *Amazon Elastic Compute Cloud (EC2).* – `https://aws.amazon.com/de/ec2/`, abgerufen am 10. Juli 2016

[3] *Fujitsu Cloud-Lösungen.* – `http://www.fujitsu.com/de/cloud/`, abgerufen am 10. Juli 2016

[4] *GLPK (GNU Linear Programming Kit).* – `https://www.gnu.org/software/glpk/`, abgerufen am 11. Juli 2016

[5] *METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering.* – `http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`, abgerufen am 4. Juli 2016

[6] *Microsoft Azure: Cloud-Computing-Platform und -Dienste.* – `https://azure.microsoft.com/de-de/`, abgerufen am 10. Juli 2016

[7] *Die SNDlib Netzwerkbibliothek.* – `http://sndlib.zib.de/home.action`, abgerufen am 3. Juli 2016

[8] Ahmed, Arif ; Ahmed, Ejaz: A survey on mobile edge computing. In: *the Proceedings of the 10 t h IEEE International Conference on Intelligent Systems and Control (ISCO 2016), Coimbatore, India*, 2016

[9] Amaldi, Edoardo ; Coniglio, Stefano ; Koster, Arie M. ; Tieves, Martin: On the computational complexity of the virtual network embedding problem. In: *Electronic Notes in Discrete Mathematics* 52 (2016), S. 213–220

[10] Barabási, Albert-László ; Albert, Réka: Emergence of scaling in random networks. In: *science* 286 (1999), Nr. 5439, S. 509–512

[11] Bari, Md F. ; Boutaba, Raouf ; Esteves, Rafael ; Granville, Lisandro Z. ; Podlesny, Maxim ; Rabbani, Md G. ; Zhang, Qi ; Zhani, Mohamed F.: Data center network virtualization: A survey. In: *IEEE Communications Surveys & Tutorials* 15 (2013), Nr. 2, S. 909–928

[12] Bari, Md. F. ; Chowdhury, Shihabur R. ; Ahmed, Reaz ; Boutaba, Raouf: On Orchestrating Virtual Network Functions in NFV. In: *CoRR* abs/1503.06377 (2015). – `http://arxiv.org/abs/1503.06377`

[13] Bari, Md F. ; Chowdhury, Shihabur R. ; Ahmed, Reaz ; Boutaba, Raouf ; Duarte, Otto Carlos Muniz B.: Orchestrating Virtualized Network Functions. (2015)

[14] Beck, Michael T. ; Botero, Juan-Felipe: Coordinated Allocation of Service Function Chains. In: *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, S. 1–6

[15] Beck, Michael T. ; Botero, Juan-Felipe: Scalable and Coordinated Allocation of Service Function Chains. In: *Computer Communications Journal* (2016)

[16] Beck, Michael T. ; Botero, Juan-Felipe ; Fischer, Andreas ; de Meer, Hermann ; Hesselbach, Xavier: A Distributed, Parallel, and Generic Virtual Network Embedding Framework. In: *23th IEEE International Conference on Communications (ICC)*, IEEE, 2013, 3471–3475

[17] Beck, Michael T. ; Botero, Juan-Felipe ; Samelin, Kai: Resilient Allocation of Service Function Chains. In: *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016

[18] Beck, Michael T. ; Feld, Sebastian ; Fichtner, Andreas ; Linnhoff-Popien, Claudia ; Schimper, Thomas: ME-VoLTE: Network Functions for Energy-Efficient Video Transcoding at the Mobile Edge. In: *18th International Conference on Intelligence in Next Generation Networks (ICIN)* IEEE, 2015, S. 38–44

[19] Beck, Michael T. ; Feld, Sebastian ; Linnhoff-Popien, Claudia ; Puetzschler, Uwe: Mobile Edge Computing. In: *Informatik-Spektrum* 39 (2016), Nr. 2, 108–114. `http://dx.doi.org/10.1007/s00287-016-0957-6`. – DOI 10.1007/s00287–016–0957–6. – ISSN 1432–122X

[20] Beck, Michael T. ; Feld, Sebastian ; Schimper, Thomas: Vom Zugangs- zum Dienstanbieter: Wie Mobilfunkprovider das Internet der Zukunft mitgestalten können. In: Claudia Linnhoff-Popien (Hrsg.) ; Michael Zaddach (Hrsg.) ; Andreas Grahl (Hrsg.): *Marktplätze im Umbruch – Digitale Strategien für Services im Mobilen Internet.* Springer, 2015, S. 123–131

[21] Beck, Michael T. ; Fischer, Andreas ; Botero, Juan-Felipe ; Linnhoff-Popien, Claudia ; de Meer, Hermann: Distributed and scalable embedding of virtual networks. In: *Journal of Network and Computer Applications* 56 (2015), 124–136. `http://dx.doi.org/http://dx.doi.org/10.1016/j.jnca.2015.06.012`. – DOI http://dx.doi.org/10.1016/j.jnca.2015.06.012. – ISSN 1084–8045

[22] Beck, Michael T. ; Fischer, Andreas ; de Meer, Hermann: Distributed Virtual Network Embedding. In: *7th GI/ITG KuVS Workshop on Future Internet*, University of Kaiserslautern, 2012. – http://www.future-internet.org/2011/7_Fachgespraech/archiv_2011_KuVS7.shtml.en

[23] Beck, Michael T. ; Fischer, Andreas ; Kokot, Fabian ; Linnhoff-Popien, Claudia ; de Meer, Hermann: A Simulation Framework for Virtual Network Embedding Algorithms. In: *Proceedings of the 16th IEEE Telecommunications Network Strategy and Planning Symposium (Networks), Madeira Island, Portugal*, Networks, 2014, S. 1–6

[24] Beck, Michael T. ; Linnhoff-Popien, Claudia: On Delay-Aware Embedding of Virtual Networks. In: *Proc. of the 6th International Conference on Advances in Future Internet*, 2014, S. 55–59

[25] Beck, Michael T. ; Werner, Martin ; Feld, Sebastian ; Schimper, Thomas: Mobile Edge Computing: A taxonomy. In: *Proc. of the 6th International Conference on Advances in Future Internet*, 2014, S. 48–54

[26] Botero, Juan F. ; Hesselbach, Xavier ; Duelli, Michael ; Schlosser, Daniel ; Fischer, Andreas ; de Meer, Hermann: Energy Efficient Virtual Network Embedding. In: *IEEE Communications Letters* 16 (2012), may, Nr. 5, S. 756–759. http://dx.doi.org/10.1109/LCOMM.2012.030912.120082. – DOI 10.1109/LCOMM.2012.030912.120082. – ISSN 1089–7798

[27] Botero, Juan F. ; Hesselbach, Xavier ; Fischer, Andreas ; de Meer, Hermann: Optimal mapping of virtual networks with hidden hops. In: *Telecommunication Systems* 51 (2012), Nr. 4, S. 273–282

[28] Botero, Juan F. ; Hesselbach, Xavier ; Fischer, Andreas ; de Meer, Hermann: Optimal mapping of virtual networks with hidden hops. In: *Telecommunications Systems: Special Issue "Future Internet Services and Architectures: Trends and Visions"* 51 (2012), Nr. 4, S. 273–282. http://dx.doi.org/10.1007/s11235-011-9437-0. – DOI 10.1007/s11235–011–9437–0. – ISSN 1018–4864

[29] Botero, Juan F. ; Molina, Miguel ; Hesselbach-Serra, Xavier ; Amazonas, José R.: A novel paths algebra-based strategy to flexibly solve the link mapping stage of VNE problems. In: *Network and Computer Applications* 36 (2013), Nr. 6, S. 1735–1752

[30] Butt, Nabeel F. ; Chowdhury, Mosharaf ; Boutaba, Raouf: *Topology-awareness and reoptimization mechanism for virtual network embedding*. Springer, 2010

[31] Caggiani Luizelli, M. ; Richter Bays, L. ; Salete Buriol, L. ; Pilla Barcellos, M. ; Gaspary, L.P.: Piecing Together the NFV Provisioning Puzzle: Efficient Placement and Chaining of Virtual Network Functions. In: *Integrated Network Management (IM 2015), 2015 IFIP/IEEE International Symposium on*, 2015

[32] Chakrabarti, Amit ; Chekuri, Chandra ; Gupta, Anupam ; Kumar, Amit: Approximation algorithms for the unsplittable flow problem. In: *Algorithmica* 47 (2007), Nr. 1, S. 53–78

[33] Cheng, Xiang ; Su, Sen ; Zhang, Zhongbao ; Shuang, Kai ; Yang, Fangchun ; Luo, Yan ; Wang, Jie: Virtual network embedding through topology awareness and optimization. In: *Computer Networks* 56 (2012), Nr. 6, S. 1797–1813

[34] Cheng, Xiang ; Su, Sen ; Zhang, Zhongbao ; Wang, Hanchi ; Yang, Fangchun ; Luo, Yan ; Wang, Jie: Virtual network embedding through topology-aware node ranking. In: *SIGCOMM Comput. Commun. Rev.* 41 (2011), April, S. 38–47. – ISSN 0146–4833

[35] Chowdhury, M. ; Rahman, M.R. ; Boutaba, R.: ViNEYard: Virtual Network Embedding Algorithms With Coordinated Node and Link Mapping. In: *IEEE/ACM Transactions on Networking* 20 (2012), feb., Nr. 1, S. 206–219. http://dx.doi.org/10.1109/TNET.2011.2159308. – DOI 10.1109/TNET.2011.2159308. – ISSN 1063–6692

[36] Chowdhury, Mosharaf ; Samuel, Fady ; Boutaba, Raouf: Polyvine: policy-based virtual network embedding across multiple domains. In: *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures* ACM, 2010, S. 49–56

[37] Chowdhury, N. M. M. K. ; Rahman, Muntasir R. ; Boutaba, Raouf: Virtual Network Embedding with Coordinated Node and Link Mapping. In: *Proc. IEEE INFOCOM* IEEE Infocom, 2009

[38] Chowdhury, N.M. Mosharaf K. ; Boutaba, Raouf: A survey of network virtualization. In: *Computer Networks* 54 (2010), Nr. 5, 862–876. http://dx.doi.org/DOI:10.1016/j.comnet.2009.10.017. – DOI DOI: 10.1016/j.comnet.2009.10.017. – ISSN 1389–1286

[39] Cui, Chunfeng ; Deng, Hui ; Telekom, Deutsche ; Michel, Uwe ; Damker, Herbert ; Italia, Telecom ; Guardini, Ivano ; Demaria, Elena ; Minerva, Roberto ; Manzalini, Antonio: Network Functions Virtualisation. In: *SDN & OpenFlow World Congress, Düsseldorf* (2014). – http://course.ipv6.club.tw/SDN/nfv_white_paper.pdf (accessed on 2016-02-26)

[40] Cui, Chunfeng ; Xie, Yunpeng ; Gao, Gongying ; Telekom, Deutsche ; Martiny, Klaus ; Carapinha, Jorge ; Telecom, S. K. ; Lee, D. K. ; Argela, Turk T. ; Ergen, Mustafa: Network Functions Virtualisation (NFV). In: *SDN & OpenFlow World Congress, Düsseldorf* (2014). https://portal.etsi.org/portals/0/tbpages/nfv/docs/nfv_white_paper3.pdf

[41] Deng, M. ; Tian, H. ; Lyu, X.: Adaptive sequential offloading game for multi-cell Mobile Edge Computing. In: *2016 23rd International Conference on Telecommunications (ICT)*, 2016, S. 1–5

[42] Duelli, Michael ; Schlosser, Daniel ; Botero, Juan F. ; Hesselbach, Xavier ; Fischer, Andreas ; de Meer, Hermann: VNREAL: Virtual Network Resource Embedding ALgorithms in the Framework ALEVIN. In: *Proc. of the 7th Euro-NF Conf. on Next-Generation Internet (NGI 2011)*, IEEE, 2011, S. 1–2

[43] Eramo, V. ; Miucci, E. ; Compagnone, L. ; Ammar, M.: Evaluation of reconfiguration cost and Energy Aware Virtual Network Embedding policies. In: *2015 IEEE International Conference on Communication Workshop (ICCW)*, 2015. – ISSN 2164–7038, S. 2824–2829

[44] Eramo, V ; Miucci, E ; Compagnone, L ; Ammar, M: Evaluation of reconfiguration cost and Energy Aware Virtual Network Embedding policies. In: *Communication Workshop (ICCW), 2015 IEEE International Conference on* IEEE, 2015, S. 2824–2829

[45] Ericcson: *Ericcson Mobility Report – June 2013.* http://www.ericsson.com/res/docs/2013/ericsson-mobility-report-june-2013.pdf. – abgerufen am 3. Juli 2016

[46] European Telecommunications Standards Institute (ETSI): *ETSI - Mobile Edge Computing.* – http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing, abgerufen am 4. Juli 2016

[47] European Telecommunications Standards Institute (ETSI): *ETSI NFV.* – http://www.etsi.org/technologies-clusters/technologies/nfv, abgerufen am 3. Juli 2016

[48] European Telecommunications Standards Institute (ETSI): Network Functions Virtualisation (NFV): Architectural Framework. In: *ETSI GS NFV* 2 (2013). http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf

[49] European Telecommunications Standards Institute (ETSI): *Mobile-Edge Computing - Introductory Technical White Paper*. Sep 2014. – https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf, abgerufen am 3. Juli 2016

[50] European Telecommunications Standards Institute (ETSI): Network Functions Virtualisation (NFV): Terminology for Main Concepts in NFV. In: *ETSI GS NFV* (2014), 12. http://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.02.01_60/gs_nfv003v010201p.pdf

[51] European Telecommunications Standards Institute (ETSI): *New ETSI group develops Open Source for NFV*. Feb 2016. – http://www.etsi.org/index.php/news-events/news/1064-2016-02-new-etsi-group-develops-open-source-for-nfv, abgerufen am 3. Juli 2016

[52] Fajjari, Ilhem ; Aitsaadi, Nadjib ; Pujolle, Guy ; Zimmermann, Hubert: VNE-AC: Virtual network embedding algorithm based on ant colony metaheuristic. In: *Communications (ICC), 2011 IEEE International Conference on* IEEE, 2011, S. 1–6

[53] Fajjari, Ilhem ; Aitsaadi, Nadjib ; Pujolle, Guy ; Zimmermann, Hubert: Vnr algorithm: A greedy approach for virtual networks reconfigurations. In: *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE* IEEE, 2011, S. 1–6

[54] Fischer, Andreas ; Beck, Michael T. ; de Meer, Hermann: An Approach to Energy-efficient Virtual Network Embeddings. In: *IFIP/IEEE International Symposium on Integrated Network Management (IM)* IEEE, 2013, 1142–1147

[55] Fischer, Andreas ; Botero, Juan-Felipe ; Beck, Michael T. ; de Meer, Hermann ; Hesselbach, Xavier: Virtual Network Embedding: A Survey. In: *IEEE Communications Surveys and Tutorials Journal* 15 (2013), Nr. 4, 1888–1906. http://www.net.fim.uni-passau.de/pdf/Fischer2013a.pdf. – ISSN 1553–877X

[56] Fortunato, Santo: Community detection in graphs. In: *Physics Reports* 486 (2010), Nr. 3, S. 75–174

[57] Fuerst, Carlo ; Schmid, Stefan ; Feldmann, Anja: Virtual network embedding with collocation: Benefits and limitations of pre-clustering. In: *Cloud Networking (CloudNet), 2013 IEEE 2nd International Conference on* IEEE, 2013, S. 91–98

[58] Ghazar, Tay ; Samaan, Nancy: Hierarchical Approach for Efficient Virtual Network Embedding Based on Exact Subgraph Matching. In: *GLOBECOM*, 2011, S. 1–6

[59] Ghaznavi, Milad ; Shahriar, Nashid ; Ahmed, Reaz ; Boutaba, Raouf: Service Function Chaining Simplified. In: *arXiv preprint arXiv:1601.00751* (2016)

[60] Gropp, Martin: *Im Wagen vor mir fährt – ein vernetzter Computer*. http://www.faz.net/aktuell/wirtschaft/neue-mobilitaet/auf-der-a9-wird-der-vernetzte-verkehr-erprobt-13902816.html, 11 2015. – abgerufen am 3. Juli 2016

[61] Guan, Xinjie ; Choi, Baek-Young ; Song, Sejun: Energy efficient virtual network embedding for green data centers using data center topology and future migration. In: *Computer Communications* 69 (2015), S. 50–59

[62] Guo, Tao ; Wang, Ning ; Moessner, Klaus ; Tafazolli, Rahim: Shared backup network provision for virtual network embedding. In: *Communications (ICC), 2011 IEEE International Conference on* IEEE, 2011, S. 1–5

[63] Holzner, Stephan: *Verteilte Einbettung virtueller Netze*. Germany, 2016. – Masterarbeit, in Vorbereitung

[64] Houidi, I. ; Louati, W. ; Zeghlache, D.: A Distributed Virtual Network Mapping Algorithm. In: *Communications, 2008. ICC '08. IEEE International Conference on Communications*, 2008, S. 5634–5640

[65] Houidi, Ines ; Louati, Wajdi ; Ameur, Walid B. ; Zeghlache, Djamal: Virtual network provisioning across multiple substrate networks. In: *Computer Networks* 55 (2011), Nr. 4, S. 1011–1023

[66] Hu, Yun C. ; Patel, Milan ; Sabella, Dario ; Sprecher, Nurit ; Young, Valerie: Mobile Edge Computing—A Key Technology Towards 5G. In: *ETSI White Paper* 11 (2015)

[67] IBM Corporation: *Smarter wireless networks; Add intelligence to the mobile network edge*. `http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=WSW14201USEN&appname=WWWSEARCH`, 2013. – abgerufen am 4. Juli 2016

[68] Inführ, Johannes ; Raidl, Günther R.: Introducing the virtual network mapping problem with delay, routing and location constraints. In: *Proceedings of the 5th international conference on Network optimization*. Berlin, Heidelberg : Springer-Verlag, 2011 (INOC'11). – ISBN 978–3–642–21526–1, S. 105–117

[69] Intel and Nokia Siemens Networks: *Increasing Mobile Operators' Value Proposition With Edge Computing*. `http://nsn.com/system/files/document/edgecomputingtechbrief_328909_002_0.pdf`, 2013. – abgerufen am 4. Juli 2016

[70] Ivaturi, Karthikeswar ; Wolf, Tilman: Mapping of delay-sensitive virtual networks. In: *Computing, Networking and Communications (ICNC), 2014 International Conference on* IEEE, 2014, S. 341–347

[71] Jarray, Abdallah ; Karmouch, Ahmed: Decomposition approaches for virtual network embedding with one-shot node and link mapping. In: *IEEE/ACM Transactions on Networking (TON)* 23 (2015), Nr. 3, S. 1012–1025

[72] Karypis, G. ; Kumar, V. ; Kumar, Vipin: Multilevel k-way Partitioning Scheme for Irregular Graphs. In: *Journal of Parallel and Distributed Computing* 48 (1998), S. 96–129

[73] Kolonko, M.: *Stochastische Simulation: Grundlagen, Algorithmen und Anwendungen*. Vieweg+Teubner Verlag, 2009 (Studienbücher Wirtschaftsmathematik). `https://books.google.de/books?id=wHhVcQ1WNfAC`. – ISBN 9783834892904

[74] Lee, Tae-Ho ; Tursunova, Shahnaza ; Choi, Tae-Sang: Graph clustering based provisioning algorithm for virtual network embedding. In: *2012 IEEE Network Operations and Management Symposium* IEEE, 2012, S. 1175–1178

[75] Leivadeas, Aris ; Papagianni, Chrysa ; Papavassiliou, Symeon: Efficient resource mapping framework over networked clouds via iterated local search-based request partitioning. In: *IEEE Transactions on Parallel and Distributed Systems* 24 (2013), Nr. 6, S. 1077–1086

[76] Leivadeas, Aris ; Papagianni, Chrysa ; Paraskevas, Evripidis ; Androulidakis, Georgios ; Papavassiliou, Symeon: An architecture for virtual network embedding in wireless systems. In: *Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on* IEEE, 2011, S. 62–68

[77] Li, H. ; Shou, G. ; Hu, Y. ; Guo, Z.: Mobile Edge Computing: Progress and Challenges. In: *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, 2016, S. 83–84

[78] Lischka, Jens ; Karl, Holger: A virtual network mapping algorithm based on subgraph isomorphism detection. In: *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, New York, USA, Aug. 2009, S. 81–88

[79] Liu, Hui ; Cui, Hongyan ; Wang, Jia ; Lu, Bo ; Chen, Jianya: An adaptive energy-aware virtual network embedding scheme. In: *2014 IEEE 79th Vehicular Technology Conference (VTC Spring)* IEEE, 2014, S. 1–5

[80] Liu, W. ; al. et: *Service Function Chaining (SFC) Use Cases, Active Internet-Draft, IETF Secretariat, Internet-Draft.* 2014

[81] Lv, Bo ; Wang, Zhenkai ; Huang, Tao ; Chen, Jianya ; Liu, Yunjie: Virtual resource organization and virtual network embedding across multiple domains. In: *2010 International Conference on Multimedia Information Networking and Security* IEEE, 2010, S. 725–728

[82] Mehraghdam, S. ; Keller, M. ; Karl, H.: Specifying and placing chains of virtual network functions. In: *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, 2014, S. 7–13

[83] Mell, Peter ; Grance, Tim: The NIST definition of cloud computing. (2011). `http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf`

[84] Melo, Márcio ; Sargento, Susana ; Killat, Ulrich ; Timm-Giel, Andreas ; Carapinha, Jorge: Optimal virtual network embedding: Energy aware formulation. In: *Computer Networks* 91 (2015), 184 - 195. `http://dx.doi.org/http://dx.doi.org/10.1016/j.comnet.2015.08.011`. – DOI http://dx.doi.org/10.1016/j.comnet.2015.08.011. – ISSN 1389–1286

[85] Mijumbi, R. ; Serrat, J. ; Gorricho, JL ; Bouten, N. ; De Turk, F. ; Davy, S: Design and Evaluation of Algorithms for Mapping and Scheduling of Virtual Network Functions. In: *Network Softwarization, IEEE 1st International Conference on*, 2015

[86] Mijumbi, Rashid ; Serrat, Joan ; Gorricho, Juan-Luis ; Boutaba, Raouf: A path generation approach to embedding of virtual networks. In: *Network and Service Management, IEEE Transactions on* 12 (2015), Nr. 3, S. 334–348

[87] Mijumbi, Rashid ; Serrat, Joan ; Gorricho, Juan-Luis ; Bouten, Niels ; De Turck, Filip ; Boutaba, Raouf: Network function virtualization: State-of-the-art and research challenges. In: *IEEE Communications Surveys & Tutorials* 18 (2015), Nr. 1, S. 236–262

[88] Moens, Hendrik ; De Turck, Filip: *VNF-P: A Model for Efficient Placement of Virtualized Network Functions*

[89] Newman, Mark E.: The structure and function of complex networks. In: *SIAM review* 45 (2003), Nr. 2, S. 167–256

[90] Newman, Mark E. ; Girvan, Michelle: Finding and evaluating community structure in networks. In: *Physical review E* 69 (2004), Nr. 2, S. 026113

[91] Nogueira, Joao ; Melo, Márcio ; Carapinha, Jorge ; Sargento, Susana: Virtual network mapping into heterogeneous substrate networks. In: *Computers and Communications (ISCC), 2011 IEEE Symposium on* IEEE, 2011, S. 438–444

[92] Nonde, Leonard ; El-Gorashi, Taisir E. ; Elmirghani, Jaafar M.: Energy efficient virtual network embedding for cloud networks. In: *Journal of Lightwave Technology* 33 (2015), Nr. 9, S. 1828–1849

[93] Rahman, Muntasir R. ; Aib, Issam ; Boutaba, Raouf: Survivable virtual network embedding. In: *NETWORKING 2010.* Springer, 2010, S. 40–52

[94] Rahman, Muntasir R. ; Boutaba, Raouf: SVNE: Survivable Virtual Network Embedding Algorithms for Network Virtualization. In: *IEEE Transactions on Network and Service Management* 10 (2013), Nr. 2, S. 105–118

[95] Razzaq, Adil ; Rathore, Muhammad S.: An approach towards resource efficient virtual network embedding. In: *Evolving Internet (INTERNET), 2010 Second International Conference on* IEEE, 2010, S. 68–73

[96] Rincon, David ; Agusti-Torra, Anna ; Botero, Juan-Felipe ; Raspall, Federico ; Remondo, David ; Hesselbach, Xavier ; Beck, Michael T. ; de Meer, Hermann ; Niedermeier, Florian ; Giuliani, Giovanni: A Novel Collaboration Paradigm for Reducing Energy Consumption and Carbon Dioxide Emissions in Data Centres. In: *The Computer Journal* 56 (2013), Nr. 12, S. 1518–1536

[97] Shanbhag, Shashank ; Kandoor, Arun R. ; Wang, Cong ; Mettu, Ramgopal ; Wolf, Tilman: VHub: Single-stage virtual network mapping through hub location. In: *Computer Networks* 77 (2015), S. 169–180

[98] Soualah, Oussama ; Fajjari, Ilhem ; Aitsaadi, Nadjib ; Mellouk, Abdelhamid: A reliable virtual network embedding algorithm based on game theory within cloud's backbone. In: *Communications (ICC), 2014 IEEE International Conference on* IEEE, 2014, S. 2975–2981

[99] Su, Sen ; Zhang, Zhongbao ; Liu, Alex X. ; Cheng, Xiang ; Wang, Yiwen ; Zhao, Xinchao: Energy-aware virtual network embedding. In: *Networking, IEEE/ACM Transactions on* 22 (2014), Nr. 5, S. 1607–1620

[100] Sun, Gang ; Yu, Hongfang ; Li, Lemin ; Anand, Vishal ; Di, Hao ; Gao, Xiujiao: Efficient algorithms for survivable virtual network embedding.

In: *Asia Communications and Photonics Conference and Exhibition* International Society for Optics and Photonics, 2010

[101] Viterbi, Andrew: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In: *IEEE transactions on Information Theory* 13 (1967), Nr. 2, S. 260–269

[102] Waxman, Bernard M.: Routing of Multipoint Connections. In: *IEEE Journal on selected areas in communications* 6 (1988), Nr. 9

[103] Xin, Yufeng ; Baldine, Ilia ; Mandal, Anirban ; Heermann, Chris ; Chase, Jeff ; Yumerefendi, Aydan: Embedding virtual topologies in networked clouds. In: *Proceedings of the 6th International Conference on Future Internet Technologies* ACM, 2011, S. 26–29

[104] Xu, Jielong ; Tang, Jian ; Kwiat, K. ; Zhang, Weiyi ; Xue, Guoliang: Survivable Virtual Infrastructure Mapping in Virtualized Data Centers. In: *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012. – ISSN 2159–6182, S. 196–203

[105] Yu, Minlan ; Yi, Yung ; Rexford, Jennifer ; Chiang, Mung: Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration. In: *ACM SIGCOMM CCR* 38 (2008), April, Nr. 2, S. 17–29

[106] Zhang, Sheng ; Qian, Zhuzhong ; Guo, Song ; Lu, Sanglu: FELL: A flexible virtual network embedding algorithm with guaranteed load balancing. In: *Communications (ICC), 2011 IEEE International Conference on* IEEE, 2011, S. 1–5

[107] Zhang, Zhongbao ; Cheng, Xiang ; Su, Sen ; Wang, Yiwen ; Shuang, Kai ; Luo, Yan: A unified enhanced particle swarm optimization-based virtual network embedding algorithm. In: *International Journal of Communication Systems* 26 (2013), Nr. 8, S. 1054–1073

[108] Zhang, Zhongbao ; Su, Sen ; Zhang, Junchi ; Shuang, Kai ; Xu, Peng: Energy aware virtual network embedding with dynamic demands: Online and offline. In: *Computer Networks* 93, Part 3 (2015), 448 - 459. `http://dx.doi.org/http://dx.doi.org/10.1016/j.comnet.2015.09.036`. – DOI http://dx.doi.org/10.1016/j.comnet.2015.09.036. – ISSN 1389–1286. – Cloud Networking and Communications {II}

[109] Zhu, Yong ; Ammar, Mostafa H.: Algorithms for Assigning Substrate Network Resources to Virtual Network Components. In: *INFOCOM*, IEEE, 2006