Dissertation

Entwicklung von TDMA-basierten QoS-Routing-Protokollen und Simulationskomponenten für Ad-Hoc-Netze

Vom Fachbereich Informatik
der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
genehmigte Dissertation
von

Anuschka Igel

Datum der Einreichung: 07. Juli 2015
Wissenschaftliche Aussprache: 18. Dezember 2015

Dekan: Prof. Dr. rer. nat. Klaus Schneider

Promotionskommission:
Vorsitzender: Prof. Dr.-Ing. Stefan Deßloch
Berichterstatter: Prof. Dr.-Ing. Reinhard Gotzhein
Prof. Dr.-Ing. Jens B. Schmitt

D 386

Zusammenfassung

Ad-Hoc-Netze sind selbstorganisierende Netze ohne zentrale Infrastruktur, die heutzutage in vielen Bereichen Verwendung finden. Sie bestehen aus drahtlosen Knoten, die zur Erfüllung ihrer Aufgaben miteinander kommunizieren. Jedoch befinden sich nicht notwendigerweise alle Knoten in Reichweite zueinander. Damit entfernte Knoten einander erreichen können, werden Routingverfahren benötigt. Die Etablierung einer beliebigen Route ist jedoch oft nicht ausreichend, denn viele Anwendungen stellen spezielle Dienstgüteanforderungen (QoS-Anforderungen) an die Verbindung, beispielsweise die Gewährleistung einer Mindestbandbreite. Um diese QoS-Anforderungen erfüllen zu können, werden sie bereits bei der Ermittlung einer Route berücksichtigt, und die benötigten Ressourcen werden entlang der Route reserviert. Dazu dienen QoS-Routing- und Reservierungsprotokolle.

In dieser Arbeit wird zunächst der Aspekt der deterministischen Reservierung von Bandbreite in Form von konkreten Zeitslots einer TDMA-basierten MAC-Schicht betrachtet. Da sich die Übertragungen verschiedener Knoten in drahtlosen Netzen gegenseitig stören können, wurde ein Interferenzmodell entwickelt. Dieses identifiziert Bedingungen, unter denen Zeitslots innerhalb eines Netzes für mehr als eine Übertragung verwendet werden können. Zudem definiert es durch Aggregation der Informationen anderer Knoten Möglichkeiten zur Ermittlung der benötigten Informationen, um zu entscheiden, welche Zeitslots für eine störungsfreie Übertragung verwendet werden können.

Weiterhin werden existierende QoS-Routing- und Reservierungsprotokolle auf inhärente Probleme untersucht, wobei der Schwerpunkt auf Protokollen liegt, die deterministische Reservierungen von Zeitslots vornehmen. In diese Kategorie fällt auch das im Rahmen der Arbeit entwickelte Protokoll RBBQR, dessen Hauptziel darin besteht, die identifizierten Probleme zu eliminieren. Ferner wird das ebenfalls zu dieser Kategorie gehörende Protokoll QMRP beschrieben, welches zentralisiert Multicast-Routen inklusive der zugehörigen Reservierungen in teilstationären Netzen ermittelt.

Ein weiterer Bestandteil der Arbeit behandelt die Entwicklung von Simulationskomponenten, welche beispielsweise zur Evaluation von QoS-Routing- und Reservierungsprotokollen genutzt werden können. Das existierende Simulationsframework FERAL wurde um eine Komponente erweitert, die die Verwendung von Kommunikationstechnologien des Netzwerksimulators ns-3 ermöglicht. Weiterhin wurde ein Modul zur Simulation eines CC2420-Transceivers entwickelt, welches in eigenständigen ns-3-Simulationen und in Simulationen mit FERAL verwendet werden kann.

Inhaltsverzeichnis

1. Einleitung **9**
- 1.1. Motivation . . . 9
- 1.2. Struktur der Arbeit . . . 10

2. Interferenzmodellierung in drahtlosen TDMA-Netzwerken **13**
- 2.1. Allgemeiner Aufbau einer TDMA-basierten MAC-Schicht . . . 13
- 2.2. Netzwerkmodell . . . 14
  - 2.2.1. Annahmen zum Netzwerk . . . 14
  - 2.2.2. Formale Definition von Netzwerk und Nachbarschaften . . . 15
- 2.3. Globale Reservierungsbedingung . . . 21
- 2.4. Lokale Reservierungsbedingungen . . . 23
  - 2.4.1. Lokale Reservierungsbedingung mit Interferenznachbarschaft . . . 23
  - 2.4.2. Lokale Reservierungsbedingung mit Kommunikationsnachbarschaft . . . 26
- 2.5. Ermittlung einer Route vom Ziel zur Quelle . . . 28
- 2.6. Propagierung getroffener Sendereservierungen . . . 29
- 2.7. Wiederverwendung von Slots entlang einer Route . . . 30
- 2.8. Vergleich mit existierenden Interferenzmodellen . . . 31

3. Grundlagen für QoS-Routing- und Reservierungsprotokolle **37**
- 3.1. Quality of Service in Ad-Hoc-Netzen . . . 37
  - 3.1.1. QoS-Routing . . . 39
  - 3.1.2. Reservierungsprotokoll . . . 40
- 3.2. Modellbasierte Protokollentwicklung mit SDL . . . 44
- 3.3. Black-Burst-basierte Übertragungen . . . 47
  - 3.3.1. Black Bursts . . . 47
  - 3.3.2. Transferprotokolle . . . 49

4. Evaluation existierender Protokolle **57**
- 4.1. Unicast-Protokolle . . . 57
  - 4.1.1. Black-Burst-based QoS-Routing (BBQR) . . . 59
  - 4.1.2. Forward Algorithm (FA) . . . 63
  - 4.1.3. TDMA-based Bandwidth Reservation Protocol . . . 64
  - 4.1.4. Race-Free Bandwidth Reservation Protocol . . . 65
  - 4.1.5. Distributed Slots Reservation Protocol (DSRP) . . . 66
- 4.2. Multicast-Protokolle . . . 67
  - 4.2.1. Hexagonal-Tree QoS Multicast Protocol . . . 67
  - 4.2.2. PSLCB . . . 68

5. Reservation-enhanced Black-Burst-based QoS-Routing (RBBQR) **71**
- 5.1. Voraussetzungen für RBBQR . . . 72

5.2. Routensuche und Reservierung von Datenslots . . . 73
5.2.1. Phase 1 der Routensuche . . . 74
5.2.2. Phase 2 der Routensuche . . . 75
5.2.3. Phase 3 der Routensuche . . . 84
5.3. Datenübertragung in RBBQR . . . 91
5.4. Freigabe von Routen und Reservierungen . . . 92
5.4.1. Explizite und implizite Routenfreigabe . . . 92
5.4.2. Propagierung von Slotreservierungen . . . 93
5.4.3. Routenbrüche . . . 95
5.5. Aufbau und Konfiguration der MAC-Schicht . . . 96
5.5.1. Virtuelle Slots für SPROP und RBRK . . . 98
5.5.2. Virtuelle Slots für die Routensuche . . . 98
5.6. Konkrete Berechnungsbeispiele . . . 102
5.7. Funktionale Simulation . . . 104
5.7.1. Aufbau eines Simulationssystems . . . 104
5.7.2. Beispielhafte Simulation . . . 106

6. QoS Multicast Routing Protocol (QMRP) 111
6.1. Voraussetzungen für QMRP . . . 111
6.2. Stationäre QoS-Routing-Bäume . . . 112
6.2.1. Ermittlung der Routing-Bäume . . . 113
6.2.2. Reservierung von Zeitslots . . . 117
6.3. Erweiterung auf teilweise mobile Netze . . . 119
6.3.1. Ermittlung der Routing-Bäume in teilweise mobilen Netzen . . . 119
6.3.2. Reservierung von Zeitslots für mobile Knoten . . . 122

7. Entwicklung von Simulationskomponenten 125
7.1. Simulationsframework FERAL . . . 125
7.2. Integration von Simulatoren in FERAL . . . 127
7.2.1. Simulatoren für FSCs . . . 127
7.2.2. Simulatoren für CSCs . . . 128
7.3. Konstruktion eines Simulationssystems . . . 129
7.4. Netzwerksimulator ns-3 . . . 130
7.5. Anbindung von ns-3 an FERAL . . . 131
7.5.1. Aufbau des Java-Teils der Simulationskomponente ns-3 . . . 132
7.5.2. Aufbau des C++-Teils der Simulationskomponente ns-3 . . . 138
7.6. Entwicklung eines CC2420-Simulationsmoduls für ns-3 und FERAL . . . 142
7.6.1. Funktionalität des CC2420-Moduls . . . 142
7.6.2. Integration des CC2420-Moduls in ns-3 . . . 144
7.6.3. Erweiterungen . . . 147
7.6.4. Integration des CC2420-Moduls in FERAL . . . 148
7.7. Simulationsszenarien . . . 149
7.7.1. Simulationen mit dem CC2420-Modul . . . 149
7.7.2. Simulation des Adaptive Cruise Control-Szenarios . . . 155
7.8. Vergleich mit existierenden Simulationsansätzen . . . 160
7.8.1. Kopplung und Erweiterung von Simulatoren . . . 160
7.8.2. Integration von Modulen in ns-3 . . . 161
7.8.3. Ansätze zur Simulation des CC2420-Transceivers . . . 161

8. Fazit und Ausblick **163**
8.1. Ergebnisse der Arbeit . . . 163
8.2. Mögliche Weiterentwicklungen . . . 164

A. Paketformate von RBBQR **165**
A.1. Intern verwendete Paketformate . . . 165
A.2. Schnittstelle zur Anwendung . . . 167
A.2.1. Schnittstelle von der Anwendung zu RBBQR . . . 167
A.2.2. Schnittstelle von RBBQR zur Anwendung . . . 168
A.3. Schnittstelle zur MAC-Schicht . . . 168
A.3.1. Schnittstelle von RBBQR zur MAC-Schicht . . . 169
A.3.2. Schnittstelle von der MAC-Schicht zu RBBQR . . . 170

B. Tabellenformate von RBBQR **173**
B.1. Temporäre Zieltabelle . . . 173
B.2. Temporäre Routentabelle . . . 174
B.3. Permanente Routentabelle . . . 177
B.4. Slottabelle . . . 178

C. SDL-Spezifikation von RBBQR **181**
C.1. Service *TimerMgmt* . . . 181
C.2. Service *Data* . . . 182
C.3. Service *SpropRbrk* . . . 184
C.4. Service *RouteSearch* . . . 185
C.4.1. Phase 1 . . . 185
C.4.2. Phase 2 . . . 187
C.4.3. Phase 3 . . . 189

D. Funktionale Simulation von RBBQR **193**
D.1. Erste Simulation . . . 193
D.2. Zweite Simulation . . . 198
D.3. Dritte Simulation . . . 200

E. Verwendung der Simulationskomponente ns-3 **203**

F. Zustandsautomaten des CC2420-Transceivers und -Simulationsmoduls **207**

Publikationsliste **211**

Literaturverzeichnis **213**

Lebenslauf **223**

1 Kapitel 1. Einleitung

1.1. Motivation

Ad-Hoc-Netze bestehen aus drahtlos kommunizierenden Knoten und sind vor allem durch das Fehlen einer zentralen Infrastruktur sowie die selbstständige Vernetzung der Knoten untereinander gekennzeichnet. Häufig sind die Knoten mobil, so dass sie ihre Position mit der Zeit verändern können.

Ad-Hoc-Netze werden heutzutage immer wichtiger, da sie einen vielfältigen Aufgabenbereich abdecken können und oft die einzige Möglichkeit zur Vernetzung von Geräten darstellen. Einsatzgebiete sind beispielsweise die Überwachung von Katastrophengebieten oder modernen Fabriken, die Vernetzung von Sensoren sowie die Kommunikation zwischen Mobiltelefonen oder Fahrzeugen.

Da sich meist nicht alle Knoten eines Netzes in Reichweite zueinander befinden und die Topologie den einzelnen Knoten i. d. R. unbekannt ist, werden Routingverfahren benötigt, damit entfernte Knoten miteinander kommunizieren können. Dabei fungieren einige oder alle Knoten als Router, d. h. sie leiten Pakete für andere Knoten weiter. Wurde eine Route zwischen einer Quelle und einem Ziel ermittelt, so können darüber Daten versendet werden. Dabei ist jedoch zu berücksichtigen, dass eine drahtlose Übertragung zwischen zwei Knoten andere gleichzeitig stattfindende Übertragungen stören oder von diesen gestört werden kann. Ein weiteres Problem besteht darin, dass aufgrund der Mobilität der Knoten Routen brechen können, weil zwei auf der Route benachbarte Knoten sich aus ihrer gegenseitigen Reichweite bewegt haben.

Best-Effort-Routing berücksichtigt keine Restriktionen bei der Routensuche, d. h. Routen werden unabhängig von der Auslastung der beteiligten Knoten ermittelt. Für viele Anwendungen (beispielsweise die Übertragung von Multimediadaten oder die Auswertung von Sensordaten in Echtzeit) ist es jedoch erforderlich, eine bestimmte Dienstgüte (Quality of Service bzw. QoS) zu garantieren. Die gezielte Ermittlung einer Route, welche (eine oder mehrere) QoS-Anforderung(en) erfüllt, wird als QoS-Routing bezeichnet. Eine häufige QoS-Anforderung besteht darin, für eine Route eine bestimmte Bandbreite zu garantieren. Da Übertragungen sich gegenseitig stören können, sind deterministische Garantien jedoch nur unter bestimmten Voraussetzungen möglich, beispielsweise wenn Zeitslots (die von einer TDMA-basierten MAC-Schicht bereitgestellt werden) exklusiv reserviert werden. Da die netzweite Reservierung solcher Zeitslots eine erhebliche Verschwendung von Bandbreite darstellt, sollen diese dort, wo es nicht zu Störungen kommt, wiederverwendet werden. Um dies umzusetzen, müssen zunächst die Bedingungen identifiziert werden, unter denen Übertragungen miteinander interferieren. Dazu wurde im Rahmen dieser Arbeit ein entsprechendes Modell entwickelt.

Die Suche nach einer Route, welche eine Bandbreitenanforderung erfüllt, sowie die Reservierung von Zeitslots entlang dieser Route ist eine komplexe Aufgabe, die geeignete Protokolle erfordert. Viele vorhandene Protokolle reservieren Bandbreite nur in statistischer Form (als

abstrakte Zahlenwerte). Auf diese Weise sind keine Garantien möglich, da interferierende Sendevorgänge nicht berücksichtigt werden. Es existieren jedoch auch Protokolle, welche sowohl QoS-Routen ermitteln als auch Zeitslots reservieren. Solche Protokolle wurden im Rahmen der vorliegenden Arbeit auf Schwächen hin untersucht. So können beispielsweise Kontrollpakete kollidieren bzw. indeterministischen Verzögerungen unterliegen oder die getroffenen Reservierungen können aufgrund von parallelen Routensuchen fehlerhaft sein.

Mit Hilfe der gewonnenen Erkenntnisse wurde in dieser Arbeit ein neues Protokoll entwickelt, das *Reservation-enhanced Black-Burst-based QoS-Routing* (RBBQR). Dieses ermittelt QoS-Routen mit garantierter Bandbreite und etabliert diese im Netz. Dabei werden auch die jeweils benötigten Zeitslots reserviert. Um die Schwächen vorhandener Protokolle zu vermeiden, werden höhere Laufzeiten in Kauf genommen.

Neben der Ermittlung von Routen von einer Quelle zu *einem* Ziel (Unicast-Routing) werden für viele Anwendungen – beispielsweise Audio- oder Videostreaming sowie die Zustellung von Benachrichtigungen – Routen von einer Quelle zu *mehreren* Zielen benötigt (Multicast-Routing). Dazu dient das *QoS Multicast Routing Protocol* (QMRP). Dieses ist für eine spezielle Klasse von Ad-Hoc-Netzen geeignet, bei denen ein Großteil der Knoten keine Mobilität aufweist. Es ermittelt zentralisiert Multicast-Routen mit garantierter Bandbreite.

Da zur Evaluation von Protokollen für Ad-Hoc-Netze häufig Simulationen eingesetzt werden, behandelt ein weiterer Teil dieser Arbeit die Entwicklung geeigneter Simulationskomponenten. Dazu wurde das Framework FERAL [KFBG13, BCG+13, BCG+14] um eine Komponente erweitert, welche es ermöglicht, einige von dem Netzwerksimulator ns-3 [NS-15] bereitgestellte Kommunikationstechnologien aus dem Framework heraus zu nutzen. Auf diese Weise können Protokolle (beispielsweise QoS-Routing- und Reservierungsprotokolle) mit dafür geeigneten Mitteln, z. B. der Spezifikationssprache SDL [Int12a], entwickelt und mit realistischen Übertragungsmedien simuliert werden. Weiterhin wurde ein Modul zur Simulation des im Bereich eingebetteter Systeme gebräuchlichen CC2420-Transceivers [Tex13] erstellt, welches sowohl für eigenständige ns-3-Simulationen als auch für Simulationen mit FERAL nutzbar ist.

1.2. Struktur der Arbeit

Kapitel 2

In Kapitel 2 wird zunächst ein formales Modell eines Netzwerks eingeführt. Anhand dieses Modells werden Bedingungen identifiziert, unter denen verschiedene Übertragungen miteinander interferieren. Diese werden in einer globalen Reservierungsbedingung formalisiert, die angibt, wann die Zeitslots einer TDMA-basierten MAC-Schicht zum Senden von einem Knoten an einen anderen verwendet werden können. Darauf aufbauend werden mehrere lokale Reservierungsbedingungen definiert, welche die für Reservierungen benötigten Informationen aggregieren, um die erforderliche Kommunikation mit Nachbarknoten zu optimieren.

Kapitel 3

In Kapitel 3 werden einige Grundlagen erläutert, die für die folgenden Kapitel benötigt werden. Zunächst wird ein Überblick über QoS im Allgemeinen gegeben und die Rolle von QoS-Routing- und Reservierungsprotokollen aufgezeigt. Häufig auftretende Probleme existierender Protokolle werden allgemein beschrieben. Anschließend wird die *Specification and Description Language* (SDL) [Int12a] vorgestellt, welche zur Spezifikation des Protokolls RBBQR verwendet wurde. Zudem werden einige für RBBQR benötigte Übertragungsverfahren erläutert, die

beispielsweise kollisionsgeschützte Übertragungen ermöglichen.

Kapitel 4

In Kapitel 4 werden existierende QoS-Routing- und Reservierungsprotokolle vorgestellt. Dabei wird untersucht, welche Probleme diese aufweisen bzw. welche Aspekte nicht berücksichtigt wurden. Die daraus resultierenden Erkenntnisse wurden für die Entwicklung der Protokolle RBBQR und QMRP genutzt.

Kapitel 5

Kapitel 5 stellt das Protokoll RBBQR vor. Dabei handelt es sich um ein dezentralisiertes (d. h. verteiltes) QoS-Routing-Protokoll, welches ein Reservierungsprotokoll zur exklusiven Reservierung von Zeitslots integriert. RBBQR ist prinzipiell für beliebige Netze geeignet, sofern diese einigen Voraussetzungen genügen. Um Reservierungen exklusiv und ressourcenschonend treffen zu können, wird das in Kapitel 2 eingeführte Interferenzmodell verwendet.

Kapitel 6

Kapitel 6 stellt das Protokoll QMRP vor. Im Gegensatz zu RBBQR handelt es sich um ein zentralisiertes Protokoll, welches für teilstationäre Netze mit einem großen Anteil stationärer Knoten entwickelt wurde und Multicast-Routen ermittelt sowie Zeitslots reserviert. Mögliche Positionsänderungen mobiler Knoten wurden gezielt berücksichtigt, so dass Routenbrüche vermieden werden. Da Reservierungen von einem zentralen Masterknoten getroffen werden, ist implizit globales Wissen über den Reservierungszustand aller Knoten vorhanden, so dass die globale Reservierungsbedingung direkt berechnet werden kann.

Kapitel 7

Kapitel 7 beschreibt die Erweiterung des Simulationsframeworks FERAL um eine Simulationskomponente ns-3, welche einige Übertragungstechnologien des Simulators ns-3 für das Framework bereitstellt. Weiterhin wird der Aufbau des ns-3-Moduls zur Simulation des CC2420-Transceivers erläutert. Abschließend werden einige Simulationen dargestellt, die mit den entwickelten Komponenten durchgeführt wurden.

Kapitel 8

In Kapitel 8 werden die Ergebnisse der Arbeit zusammengefasst. Zudem wird ein kurzer Ausblick auf mögliche Weiterentwicklungen gegeben.

Anhang A

Anhang A listet die von RBBQR verwendeten Paketformate auf. Die internen Paketformate sind bereits in Kapitel 5 detailliert beschrieben, werden jedoch aus Gründen der Übersichtlichkeit und Vollständigkeit noch einmal zusammengefasst. Darüber hinaus werden die Schnittstellen zwischen der Anwendung und RBBQR sowie zwischen RBBQR und der MAC-Schicht detaillierter betrachtet.

Anhang B

Anhang B beschreibt die von RBBQR verwendeten Tabellen. Diese dienen zur Zwischenspeicherung von Routenanforderungen der Anwendung und von partiellen Routen während der Routensuche sowie zur Speicherung von etablierten Routen und von Slotreservierungen. Die Tabelleneinträge für etablierte Routen und Slotreservierungen werden mit Timern versehen (welche beim Empfang entsprechender Pakete jeweils neu gestartet werden) und bei deren Ablaufen entfernt. Dies wird ebenfalls in Anhang B beschrieben.

Anhang C
Zur funktionalen Simulation von RBBQR wurde eine SDL-Spezifikation erstellt, welche in Anhang C in Auszügen vorgestellt wird. Dabei werden einige Aspekte der Behandlung von Timern, der Übertragung von Daten und Verwaltungsinformationen sowie der Phasen 1 bis 3 einer Routensuche im Detail betrachtet.

Anhang D
Anhang D beschreibt einige funktionale Simulationen des RBBQR-Protokolls, die mit Hilfe der erstellten SDL-Spezifikation durchgeführt wurden. Diese zeigen die grundsätzliche Funktionalität des Protokolls.

Anhang E
Anhang E zeigt die Verwendung der Simulationskomponente ns-3 im Kontext von FERAL-Simulationssystemen. Neben allgemeinen Aspekten wird auch auf die Erstellung eines Mobilitätsmodells für ns-3 aus FERAL-Simulationssystemen heraus eingegangen.

Anhang F
In Anhang F wird sowohl der interne Zustandsautomat des CC2420-Transceivers beschrieben als auch dessen Umsetzung in Form eines simulierten Zustandsautomaten innerhalb des entwickelten CC2420-Simulationsmoduls dargestellt.

Kapitel 2.

2 Interferenzmodellierung in drahtlosen TDMA-Netzwerken

Um deterministische Quality-of-Service-Garantien geben zu können, ist die exklusive Reservierung von Ressourcen erforderlich. Im Fall von Bandbreite kann dies beispielsweise durch die Verwendung von TDMA (Time Division Multiple Access) erfolgen. Die einfachste Methode besteht darin, die durch TDMA bereitgestellten Zeitslots jeweils netzweit für einen Sendevorgang zu reservieren. Zur besseren Ausnutzung der vorhandenen Bandbreite ist es jedoch möglich, Slots mehrfach zu nutzen, sofern die entsprechenden Sendevorgänge sich nicht gegenseitig stören. Dies wird als SDMA (Space Division Multiple Access) bezeichnet. Zur Realisierung von SDMA ist es zunächst erforderlich, die Bedingungen, unter denen verschiedene Sendevorgänge miteinander interferieren, zu betrachten. Soll beispielsweise eine Nachricht msg von einem Knoten a an einen Knoten b übertragen werden, so darf für einen erfolgreichen Empfang kein Knoten innerhalb der Interferenzreichweite von b im selben Slot senden. Gleichzeitig soll die Übertragung von msg keine anderen Übertragungen stören, d. h. kein Knoten in Interferenzreichweite von a darf in diesem Slot empfangen. Dabei ist auch zu berücksichtigen, dass die Interferenzreichweite größer als die Kommunikationsreichweite sein kann. Im Folgenden wird eine Modellierung vorgestellt, welche dieses Problem (hier als *Interferenzproblem* bezeichnet) formal beschreibt und Möglichkeiten zu dessen Lösung bietet. Ergebnisse dieses Kapitels wurden auch in [IG12] publiziert.

In diesem Kapitel wird zunächst der allgemeine Aufbau einer TDMA-basierten MAC-Schicht beschrieben. Anschließend wird ein Modell zur formalen Beschreibung eines Netzwerks vorgestellt. Danach werden mit Hilfe dieses Modells eine globale Reservierungsbedingung sowie mehrere lokale Reservierungsbedingungen definiert. Diese beschreiben, wann ein Knoten einen Slot zum Senden an einen anderen Knoten verwenden kann. Zudem wird eine Form der Reservierungsbedingung definiert, die für die effiziente Ermittlung einer Route vom Ziel zur Quelle geeignet ist. Anschließend wird betrachtet, wie getroffene Sendereservierungen im Netzwerk propagiert werden können, um die Statusinformationen aller betreffenden Knoten zu aktualisieren. Darüber hinaus wird auf die Wiederverwendung von Slots entlang einer Route eingegangen. Abschließend wird ein Vergleich mit anderen Interferenzmodellen vorgenommen.

2.1. Allgemeiner Aufbau einer TDMA-basierten MAC-Schicht

Um Bandbreite exklusiv reservieren zu können, eignet sich die Verwendung einer TDMA-basierten MAC-Schicht. Zur besseren Ausnutzung der verfügbaren Bandbreite kommt zusätzlich SDMA zum Einsatz. Nähere Informationen zu TDMA und SDMA sind in [Sch03] zu finden.

Im hier verwendeten Modell der MAC-Schicht wird die Zeit in Superslots gleicher Dauer unterteilt, die wiederum aus Makroslots bestehen (diese haben ebenfalls gleiche Dauer). Zur

netzweiten Synchronisation der Zeitslots wird ein Protokoll benötigt, welches in jedem Makroslot ausgeführt wird. Ein entsprechendes Protokoll, welches eine obere Grenze für die Uhrenabweichungen garantiert, ist BBS (Black Burst Synchronization) [GK11]. Die Unterscheidung zwischen Superslots und Makroslots wurde eingeführt, um das Reservierungsintervall vom Resynchronisationsintervall zu entkoppeln.

Die Makroslots werden in Mikroslots gleicher Dauer unterteilt. Diese sind innerhalb eines Superslots konsekutiv nummeriert, wobei die Nummerierung über Makroslotgrenzen hinweg fortgesetzt wird. Mikroslots sind sehr klein und reichen für die Übertragung von Paketen nicht aus. Zum Versenden eines Pakets wird deshalb eine – für dieses ausreichende – Anzahl zusammenhängender Mikroslots zu einer *virtuellen Slotregion* zusammengefasst. Prinzipiell können beliebig viele virtuelle Slotregionen definiert werden, und diese können aus beliebig vielen Mikroslots bestehen, sofern jeweils genügend zusammenhängende Mikroslots vorhanden sind. Dies dient dazu, den Verschnitt gering zu halten, da nicht alle Übertragungen die gleiche Dauer haben. Virtuelle Slotregionen wiederholen sich in jedem Superslot.

Der allgemeine Aufbau eines Superslots ist in Abbildung 2.1 dargestellt. Die schwarzen Bereiche sind dabei für die Zeitsynchronisation reserviert.

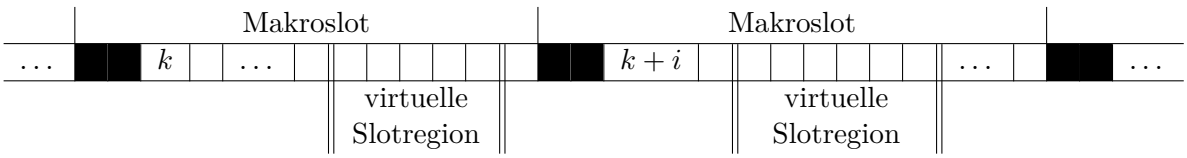

Abbildung 2.1.: Aufbau eines Superslots.

Zur Vereinfachung wird im Folgenden davon ausgegangen, dass jedes Datenpaket die gleiche Länge hat. Neben virtuellen Slotregionen für Kontrollpakete wird eine bestimmte Anzahl virtueller Slotregionen mit gleicher Dauer für solche Datenpakete vorgesehen. Diese werden als Datenslots oder einfach als Slots bezeichnet und sind die von Reservierungsprotokollen verwalteten Einheiten. Zur einfacheren Identifikation werden sie konsekutiv von 0 bis $n_{maxSlot}$ nummeriert. Die Gesamtmenge dieser reservierbaren Slots sei S. Reservierungen beziehen sich stets auf den aktuellen und alle folgenden Superslots.

2.2. Netzwerkmodell

Ein Netz(werk) wird hier als eine Menge drahtlos kommunizierender Knoten aufgefasst, welche räumlich so angeordnet sind, dass jeder Knoten mit jedem anderen kommunizieren kann (ggf. indirekt über weitere Knoten). Das vorgestellte Netzwerkmodell unterscheidet zwischen (drahtlosen) Kommunikations-, Interferenz- und Wahrnehmungslinks.

Im Folgenden werden in Abschnitt 2.2.1 allgemeine Annahmen formuliert, welche für die formalen Definitionen in Abschnitt 2.2.2 benötigt werden.

2.2.1. Annahmen zum Netzwerk

Für die Entwicklung sinnvoller Protokolle werden einige allgemeine Annahmen bezüglich des Netzes getroffen. Existierende Protokolle gehen ebenfalls häufig von solchen Annahmen aus, ohne diese jedoch explizit zu formulieren.

1. Es gilt die *Single Network Property*. Diese besagt, dass es zwischen zwei beliebigen Knoten des Netzes immer einen zuverlässigen Kommunikationspfad (d. h. einen Pfad aus

Kommunikationslinks) gibt und dass keine Knoten mit einem anderen MAC-Protokoll im selben Frequenzband betrieben werden. Um diese Annahme zu erfüllen, ist eine gewisse Kontrolle über die Topologie erforderlich, beispielsweise durch die Verwendung von FDMA (Frequency Division Multiple Access; s. [Sch03]).

2. Das Netz ist hinreichend stationär. Da Reservierungen komplexe Operationen sind, sind sie bei einem Netz mit starker Mobilität nicht sinnvoll. Bewegen sich Knoten auseinander, so können Links zwischen diesen brechen, was bestehende Reservierungen hinfällig macht. Bewegen sich die Knoten aufeinander zu, so können Kollisionen aufgrund von vorher nicht vorhandenen Interferenzen auftreten (bei Verwendung von SDMA). In diesem Fall garantieren die Reservierungen keine Kollisionsfreiheit. Um solche Probleme zu vermeiden, könnte man ein stabiles Teilnetz betrachten (Unterscheidung zwischen stationären und mobilen Knoten) und nach Möglichkeit nur dieses für Routing bzw. Reservierungen verwenden. Auch bei der Slotverteilung könnte man mobile Knoten dann gesondert berücksichtigen. Dies wird bei dem in Kapitel 6 vorgestellten Protokoll QMRP umgesetzt.

3. Alle Links (für Kommunikation, Interferenz und Wahrnehmung) sind bidirektional. Ein unidirektionaler Link könnte entstehen, wenn beispielsweise zwei Knoten unterschiedliche Sendesignalstärken haben oder wenn Reflexionen (z. B. an Wänden) dafür sorgen, dass ein Signal in einer Richtung weiter transportiert wird als in der anderen. Solche Links sollen im Folgenden ausgeschlossen werden, da man dann beispielsweise eine Route nicht vom Zielknoten ausgehend suchen kann. Zur Ermittlung unidirektionaler Links kann beispielsweise das *Automatic Topology Discovery Protocol* (ATDP) [Kra13,KCG15] verwendet werden.

4. Jeder Knoten hat nur eine Antenne. Dies liegt darin begründet, dass existierende Transceiver-Chips i. d. R. nur mit einer Antenne ausgerüstet sind. Die Existenz von nur einer Antenne impliziert, dass ein Knoten nicht an sich selbst senden kann. Außerdem kann ein Knoten während des Sendens das Medium nicht abhören und somit auch nicht empfangen.

2.2.2. Formale Definition von Netzwerk und Nachbarschaften

Ein Knoten a befindet sich in *Kommunikationsreichweite* zu einem Knoten b, wenn b eine Übertragung von a korrekt empfangen kann. Knoten a befindet sich in *Interferenzreichweite* zu b, wenn eine Übertragung von a (an einen beliebigen Knoten) den korrektem Empfang einer gleichzeitigen Übertragung eines weiteren Knotens c zu b verhindert. Knoten a befindet sich in *Wahrnehmungsreichweite* zu b, wenn b eine Übertragung von a mittels Clear Channel Assessment (CCA) wahrnehmen kann.

Definition 2.1

Sei V eine Menge von Knoten. (Drahtlose) Kommunikations-, Interferenz- und Wahrnehmungslinks werden formal durch folgende Relationen beschrieben:

- $CL =_{Df} \{(a, b) \in V \times V : a$ *befindet sich in Kommunikationsreichweite zu* $b\}$ *(Communication Link)*
- $IL =_{Df} \{(a, b) \in V \times V : a$ *befindet sich in Interferenzreichweite zu* $b\}$ *(Interference Link)*

- $SL =_{Df} \{(a,b) \in V \times V : a$ *befindet sich in Wahrnehmungsreichweite zu* $b\}$ *(Sensing Link)*

Die Annahme, dass alle Links bidirektional sind, wird dadurch formalisiert, dass CL, IL und SL symmetrisch sein müssen. Da jeder Knoten nur eine Antenne hat, müssen CL, IL und SL irreflexiv sein, d. h. kein Knoten hat einen Kommunikations-, Interferenz- oder Wahrnehmungslink zu sich selbst. Weiterhin impliziert die Existenz eines Kommunikationslinks die eines entsprechenden Interferenzlinks und die Existenz eines Interferenzlinks impliziert die eines Wahrnehmungslinks. Dieses sogenannte *Konsistenzkriterium* wird als $CL \subseteq IL \subseteq SL$ repräsentiert.

Basierend auf den Relationen CL, IL und SL wird nun ein (drahtloses) Netz definiert. Dabei handelt es sich um einen Graphen mit drei verschiedenen Kantentypen, welche Kommunikations-, Interferenz- und Wahrnehmungslinks repräsentieren, und bei dem je zwei Knoten durch einen Pfad aus Kommunikationslinks verbunden sind (Single Network Property).

Definition 2.2

Sei V eine Menge von Knoten. CL, IL und SL seien Relationen gemäß Definition 2.1. Ein (drahtloses) Netz(werk) *ist formal definiert als gerichteter Graph $G = (V, L, E)$, wobei $L = \{cl, il, sl\}$ eine Menge von Markierungen (Labels) ist und $E \subseteq V \times V \times L$ eine Menge von Kanten. $E = E_{cl} \cup E_{il} \cup E_{sl}$ ist aus den folgenden Mengen zusammengesetzt:*

$E_{cl} =_{Df} \{(a,b,l) \in V \times V \times L : l = cl \wedge CL(a,b)\}$ *(repräsentiert Kommunikationslinks)*

$E_{il} =_{Df} \{(a,b,l) \in V \times V \times L : l = il \wedge IL(a,b)\}$ *(repräsentiert Interferenzlinks)*

$E_{sl} =_{Df} \{(a,b,l) \in V \times V \times L : l = sl \wedge SL(a,b)\}$ *(repräsentiert Wahrnehmungslinks)*

Weiterhin muss der Kommunikationssubgraph $G_{cl} =_{Df} (V, \{cl\}, E_{cl})$ *verbunden sein, d. h.* $\forall\, a, b \in V : \exists\, p =_{Df} (v_1^p, \ldots, v_{|p|+1}^p) \in V^+$, *so dass* $\forall\, i \in \{1, \ldots, |p|\} : (v_i^p, v_{i+1}^p, cl) \in E_{cl}$, $v_1^p = a$ *und* $v_{|p|+1}^p = b$. *p muss ein zyklenfreier Pfad sein, d. h. kein Knoten darf mehr als einmal in p vorkommen. Die Länge $|p|$ von p bezeichnet die Anzahl der Kanten.*

Analog zu G_{cl} sind der Interferenzsubgraph $G_{il} =_{Df} (V, \{il\}, E_{il})$ *und der* Wahrnehmungssubgraph $G_{sl} =_{Df} (V, \{sl\}, E_{sl})$ *definiert.*

Die Kommunikationsdistanz *bzw.* (Hop)Distanz *zweier Knoten $a, b \in V$ ist definiert als*

$$d_{G_{cl}}(a,b) =_{Df} \min_{p \in P_{G_{cl}}(a,b)} |p|\,,$$

wobei $P_{G_{cl}}$ die Menge aller zyklenfreien Kommunikationspfade (Pfade in G_{cl}) beschreibt. $P_{G_{cl}}(a,b) \subseteq P_{G_{cl}}$ ist die Menge aller solchen Pfade, die in a starten und in b enden.

Die Interferenzdistanz $d_{G_{il}}$ *sowie die* Wahrnehmungsdistanz $d_{G_{sl}}$ *sind analog zu $d_{G_{cl}}$ definiert.*

Anmerkung: Einige der in Definition 2.2 verwendeten Notationen (z. B. zyklenfreie Pfade und Pfadmengen) sind an [Nis08, NG09] angelehnt.

Anmerkung: Der Kommunikationssubgraph G_{cl} wird im Folgenden auch als *Kommunikationstopologie* bezeichnet (analog Interferenz- und Wahrnehmungstopologie).

Da CL, IL und SL irreflexiv sind, gilt $\forall\, a \in V$, $\forall\, l \in L : (a,a,l) \notin E$. Da die Relationen symmetrisch sind, gilt außerdem $\forall\, (a,b,l) \in E : (b,a,l) \in E$, was einem ungerichteten

Graphen entspricht.

Anmerkung: Prinzipiell könnte auch direkt eine ungerichtete Graphdefinition verwendet werden. Die gerichtete Graphdefinition wurde aufgrund der größeren Flexibilität gewählt (beispielsweise wäre damit eine Erweiterung auf asymmetrische Links einfach möglich).

Aufgrund des Konsistenzkriteriums gilt ferner:

$$\forall\, a, b \in V : (\,(a, b, cl) \in E_{cl} \implies (a, b, il) \in E_{il}\,) \wedge (\,(a, b, il) \in E_{il} \implies (a, b, sl) \in E_{sl}\,)\,.$$

Im Folgenden wird die Notation $G = (V, E)$ anstelle von $G = (V, L, E)$ verwendet, da die Menge der Markierungen $L = \{cl, il, sl\}$ stets gleich ist.

$d_{G_{cl}}$, $d_{G_{il}}$ und $d_{G_{sl}}$ sind Metriken. Dies ergibt sich direkt aus der folgenden Definition einer Metrik.

Definition 2.3 (*Metrik [For11]*)

Sei X eine beliebige Menge. Eine Abbildung $d : X \times X \to \mathbb{R}$ heißt Metrik *auf X, wenn $\forall\, x, y, z \in X$ folgende Eigenschaften gelten:*

$$d(x, y) = 0 \iff x = y$$

$$d(x, y) = d(y, x) \quad \textit{(Symmetrie)}$$

$$d(x, y) \leq d(x, z) + d(z, y) \quad \textit{(Dreiecksungleichung)}$$

Kommunikations-, Interferenz- und Wahrnehmungslinks werden graphisch durch verschiedene Kantentypen repräsentiert. Für Kommunikationskanten werden durchgezogene Linien verwendet, für Interferenzkanten gestrichelte und für Wahrnehmungskanten gepunktete. Eine Interferenzkante zwischen zwei Knoten wird nur dann eingezeichnet, wenn zwischen diesen Knoten keine Kommunikationskante existiert. Analog wird eine Wahrnehmungskante nur dann eingezeichnet, wenn keine Interferenzkante existiert. Da alle Kanten bidirektional sind, werden für diese keine Pfeile verwendet. Stattdessen werden Sendevorgänge mit Pfeilen dargestellt. Ein Beispiel ist in Abbildung 2.2 zu finden.

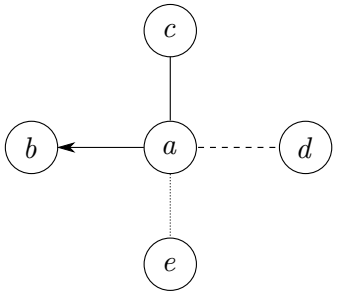

Abbildung 2.2.: Darstellung von Links und Sendevorgängen.

Knoten a hat Kommunikationslinks zu b und c, einen Interferenzlink zu d und einen Wahrnehmungslink zu e. Zudem sendet a an b. c befindet sich in Kommunikationsreichweite von a und empfängt die Übertragung ebenfalls. d empfängt die Interferenz und e kann die Übertragung mittels CCA wahrnehmen. Es ist zu beachten, dass hier nur ein Ausschnitt aus einem Netz dargestellt ist (die Single Network Property ist nicht erfüllt). Der Slot, in dem gesendet werden soll, ist durch den Kontext implizit festgelegt und wird hier nicht separat bezeichnet.

Mit regulären Übertragungen kann nur an Knoten in Kommunikationsreichweite gesendet werden. Sofern Black-Burst-basierte Übertragungen (s. Abschnitt 3.3) verwendet werden, ist es auch möglich, an Knoten in Interferenz- und Wahrnehmungsreichweite zu senden.

Im Folgenden werden Wahrnehmungskanten in der Darstellung meist weggelassen, da die Wahrnehmungsreichweite entweder nicht explizit benötigt oder mit der Interferenzreichweite gleichgesetzt wird (vgl. Abschnitt 3.3.1).

Als Nächstes wird der Begriff von Nachbarschaften zwischen Knoten eingeführt.

Definition 2.4

Sei $G = (V, E)$ ein Netz, sei $a \in V$, und sei $i \geq 0$ ein Integerwert.

- *Die* i-Hop-Kommunikations-*,* Interferenz- *und* Wahrnehmungsnachbarschaft *von a (Communication Neighborhood, Interference Neighborhood, Sensing Neighborhood) ist definiert als*

$$CN_i(a) =_{Df} \{b \in V : d_{G_{cl}}(a, b) = i\},$$

$$IN_i(a) =_{Df} \{b \in V : d_{G_{il}}(a, b) = i\},$$

$$SN_i(a) =_{Df} \{b \in V : d_{G_{sl}}(a, b) = i\}.$$

- *Die* maximale i-Hop Kommunikations-*,* Interferenz- *und* Wahrnehmungsnachbarschaft *von a ist definiert als*

$$CN_{\leq i}(a) =_{Df} \{b \in V : d_{G_{cl}}(a, b) \leq i\},$$

$$IN_{\leq i}(a) =_{Df} \{b \in V : d_{G_{il}}(a, b) \leq i\},$$

$$SN_{\leq i}(a) =_{Df} \{b \in V : d_{G_{sl}}(a, b) \leq i\}.$$

Im Folgenden werden einige Eigenschaften der Nachbarschaftsdefinitionen vorgestellt. Zur Vereinfachung werden diese nur für die Kommunikationsnachbarschaft angegeben; sie gelten jedoch analog auch für die Interferenz- und Wahrnehmungsnachbarschaft. i und j sind im Folgenden stets Integerwerte.

Seien $a, b \in V$. Dann gilt:

- $CN_0(a) = \{a\}$
- $CN_1(a) = \{b \in V : CL(a, b)\}$
- $\forall\, i \geq 0 : CN_{\leq i}(a) = \bigcup_{0 \leq j \leq i} CN_j(a)$
- $\forall i \geq 0 : a \in CN_i(b) \iff b \in CN_i(a)$ (da $d_{G_{cl}}$ eine Metrik ist)
- $\forall i \geq 0 : b \in CN_i(a) \implies \forall\, j > 0 : b \notin CN_{i+j}(a) \wedge b \in CN_{\leq i+j}(a)$
  (da sich die Nachbarschaftsdefinitionen stets auf den kürzesten Pfad beziehen, sind i-Hop-Nachbarn keine $(i + j)$-Hop-Nachbarn für $j > 0$, sie sind jedoch *maximale* $(i + j)$-Hop-Nachbarn)

Weiterhin gilt aufgrund des Konsistenzkriteriums $\forall\, a \in V$:

$$CN_1(a) \quad \subseteq IN_1(a) \quad \subseteq SN_1(a)$$

$$\forall\, i \geq 0 : CN_{\leq i}(a) \subseteq IN_{\leq i}(a) \subseteq SN_{\leq i}(a)$$

Für beliebiges $a \in V$ gilt jedoch i. Allg. *nicht* $CN_i(a) \subseteq IN_i(a) \subseteq SN_i(a)$. Beispielsweise könnten sich zwei Knoten, welche sich in 2-Hop-Kommunikationsreichweite befinden, in 1-Hop-Interferenzreichweite befinden (damit liegen sie nicht in 2-Hop-Interferenzreichweite).

Im Folgenden bezieht sich die Bezeichnung „Nachbarn“ stets auf 1-Hop-Kommunikationsnachbarn, sofern nicht anders angegeben.

Um die Nachbarschaften konkatenieren zu können, wird zunächst die folgende allgemeine Definition benötigt.

Definition 2.5

Sei $G = (V, E)$ ein Netz, sei $a \in V$, und seien $P(a) \subseteq V$ und $R(a) \subseteq V$ einstellige Relationen über V (Relationen abhängig von einem Element aus V), welche für jedes Element aus V existieren. Dann ist die Konkatenation der Relationen folgendermaßen definiert:

$$P(R(a)) =_{Df} \{c \in V : \exists\, b \in R(a) : c \in P(b)\}\,.$$

Anmerkung: Die Notation $P(a)$ bedeutet hier nicht, dass $a \in P$ gilt, sondern bezeichnet eine Relation, die in Abhängigkeit von einem Element $a \in V$ definiert ist.

Die Mengen $CN_i(a)$, $CN_{\leq i}(a)$, $IN_i(a)$, $IN_{\leq i}(a)$, $SN_i(a)$ und $SN_{\leq i}(a)$ sind $\forall\, a \in V$ und $\forall\, i \geq 0$ einstellige Relationen über V. Ein Beispiel zur Verdeutlichung von Definition 2.5 ist die 1-Hop-Interferenznachbarschaft der 1-Hop-Kommunikationsnachbarschaft eines Knotens a, welche mit $IN_1(CN_1(a))$ bezeichnet wird. Falls $CN_1(a) \neq \emptyset$, so gilt beispielsweise $a \in IN_1(CN_1(a))$.

Im Folgenden werden weitere Eigenschaften der Nachbarschaftsdefinitionen gezeigt, die auf der Konkatenation basieren. Dazu werden zunächst die beiden folgenden Sätze benötigt.

Satz 2.1

Sei $G = (V, E)$ ein Netz, sei $a \in V$, und seien $P(a) \subseteq V$, $R_1(a) \subseteq V$ und $R_2(a) \subseteq V$ einstellige Relationen über V. Dann gilt:

$$R_1(a) \subseteq R_2(a) \implies P(R_1(a)) \subseteq P(R_2(a))\,.$$

Beweis: Sei $c \in P(R_1(a))$. Dann existiert ein $b \in R_1(a) \subseteq R_2(a)$ mit $c \in P(b)$. Somit existiert ein $b \in R_2(a)$ mit $c \in P(b)$, d. h. $c \in P(R_2(a))$. ■

Satz 2.2

Sei $G = (V, E)$ ein Netz, sei $a \in V$, und seien $P(a) \subseteq V$, $R_1(a) \subseteq V$ und $R_2(a) \subseteq V$ einstellige Relationen über V. Dann gilt:

$$R_1(a) \subseteq R_2(a) \implies R_1(P(a)) \subseteq R_2(P(a))\,.$$

Beweis: Sei $c \in R_1(P(a))$. Dann existiert ein $b \in P(a)$ mit $c \in R_1(b) \subseteq R_2(b)$. Somit existiert ein $b \in P(a)$ mit $c \in R_2(b)$, d. h. $c \in R_2(P(a))$. ■

Durch Anwendung von Definition 2.5 sowie der Sätze 2.1 und 2.2 lassen sich weitere Eigenschaften der Nachbarschaftsdefinitionen herleiten. Diese sind im folgenden Satz exemplarisch für die Kommunikationsnachbarschaft angegeben; sie gelten jedoch analog auch für die Interferenz- und Wahrnehmungsnachbarschaft.

Satz 2.3

Sei $G = (V, E)$ ein Netz, sei $a \in V$, und seien $i, j \geq 0$ Integerwerte. Dann gilt:

1. $CN_{\leq i}(CN_{\leq j}(a)) = CN_{\leq i+j}(a)$
2. $CN_{\leq i}(CN_{\leq j}(a)) = CN_{\leq j}(CN_{\leq i}(a))$
3. $CN_{i+j}(a) \subseteq CN_i(CN_j(a))$
4. $CN_i(CN_j(a)) \subseteq CN_{\leq i+j}(a)$

Beweis: 1. „$\subseteq$“: Sei $c \in CN_{\leq i}(CN_{\leq j}(a))$. Dann existiert ein $b \in CN_{\leq j}(a)$ mit $c \in CN_{\leq i}(b)$. Somit gilt $d_{G_{cl}}(a, b) \leq j$ und $d_{G_{cl}}(b, c) \leq i$. Da $d_{G_{cl}}$ eine Metrik ist, folgt daraus $d_{G_{cl}}(a, c) \leq i + j$, d. h. $c \in CN_{\leq i+j}(a)$.

„$\supseteq$“: Sei $c \in CN_{\leq i+j}(a)$, d. h. $d_{G_{cl}}(a, c) \leq i + j$. Dann existiert ein $b \in V$ mit $d_{G_{cl}}(a, b) \leq j$ und $d_{G_{cl}}(b, c) \leq i$. Somit gilt $b \in CN_{\leq j}(a)$ und $c \in CN_{\leq i}(b)$, d. h. $c \in CN_{\leq i}(CN_{\leq j}(a))$.

2. Folgt aus Teil 1: $CN_{\leq i}(CN_{\leq j}(a)) = CN_{\leq i+j}(a) = CN_{\leq j+i}(a) = CN_{\leq j}(CN_{\leq i}(a))$.
3. Sei $c \in CN_{i+j}(a)$, d. h. $d_{G_{cl}}(a, c) = i + j$. Dann existiert ein $b \in V$ mit $d_{G_{cl}}(a, b) = j$ und $d_{G_{cl}}(b, c) = i$. Somit gilt $b \in CN_j(a)$ und $c \in CN_i(b)$, d. h. $c \in CN_i(CN_j(a))$.
4. $CN_i(CN_j(a)) \underset{\text{Satz 2.1}}{\subseteq} CN_i(CN_{\leq j}(a)) \underset{\text{Satz 2.2}}{\subseteq} CN_{\leq i}(CN_{\leq j}(a)) \underset{\text{Teil 1}}{=} CN_{\leq i+j}(a)$. ■

Im Allgemeinen gilt jedoch weder $CN_i(CN_j(a)) = CN_j(CN_i(a))$ noch $CN_i(CN_j(a)) \subseteq CN_{i+j}(a)$ noch $CN_{\leq i+j}(a) \subseteq CN_i(CN_j(a))$. Als Gegenbeispiel dient das Netz aus Abbildung 2.3.

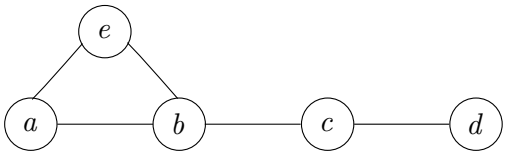

Abbildung 2.3.: Beispielnetzwerk.

Es gilt $c \in CN_1(CN_3(a))$, aber $c \notin CN_3(CN_1(a))$. Weiterhin gilt $b \in CN_1(CN_1(a))$, aber $b \notin CN_2(a)$. Schließlich gilt $b \in CN_{\leq 3}(a)$, aber $b \notin CN_2(CN_1(a))$.

Zusätzlich zu den Nachbarschaften von Knoten (s. Definition 2.4) werden auch die Nachbarschaften von Pfaden definiert (hier exemplarisch für Kommunikationspfade; die Definitionen für andere Typen von Pfaden sind jedoch analog).

Definition 2.6

Sei $G = (V, E)$ ein Netz, und sei $p \in P_{G_{cl}}$ ein zyklenfreier Kommunikationspfad ($P_{G_{cl}}$ bezeichnet die Gesamtmenge aller zyklenfreien Kommunikationspfade in G_{cl}; vgl. Definition 2.2). Die i-Hop-Kommunikationsnachbarschaft *von p ist definiert als*

$$CN_i(p) =_{Df} \bigcup_{a \in p} CN_i(a)\,.$$

Die restlichen in Definition 2.4 aufgeführten Nachbarschaften sind für p analog definiert.

Anmerkung: Für Definition 2.6 wird ein Pfad p als Menge von Knoten aufgefasst. Es ist jedoch auch möglich, p als Menge von Kanten aufzufassen. Insbesondere bezeichnet die Länge eines Pfads die Anzahl seiner Kanten (vgl. Definition 2.2).

2.3. Globale Reservierungsbedingung

Im Folgenden wird die *globale Reservierungsbedingung* $F^s_{TX}(a, b)$ definiert, welche beschreibt, ob ein Slot $s \in S$ für Übertragungen eines Knotens a an einen Knoten b – welcher sich in Kommunikationsreichweite zu a befindet – frei ist. Es muss gewährleistet sein, dass s von keinem Knoten in Interferenzreichweite von a zum Empfangen und von keinem Knoten in Interferenzreichweite von b zum Senden reserviert ist.

Diese Reservierungsbedingung heißt *global*, weil globales Wissen über die Netzwerktopologie und den Reservierungszustand der einzelnen Slots benötigt wird.

Definition 2.7 (*Reservierungszustand*)

Sei $G = (V, E)$ ein Netz und $s \in S$ ein Slot. Der Reservierungszustand $TX^s \subseteq V \times V$ *von Slot s gibt für alle Knotenpaare $a, b \in V$ mit $b \in CN_1(a)$ an, ob s für Übertragungen von a an b reserviert ist (für $b \notin CN_1(a)$ gilt automatisch $\neg TX^s(a, b)$).*

Weiterhin lassen sich folgende Relationen von TX^s ableiten:

- $TX^s(a) =_{Df} \exists\, b \in V : TX^s(a, b)$
  s ist reserviert für Übertragungen von Knoten a.
- $RX^s(a, b) =_{Df} TX^s(b, a)$
  s ist von Knoten a zum Empfangen von Knoten b reserviert.
- $RX^s(a) =_{Df} \exists\, b \in V : RX^s(a, b)$
  s ist von Knoten a zum Empfangen reserviert.

Die von TX^s abgeleiteten Relationen enthalten keine zusätzlichen Informationen, sondern wurden eingeführt, um die im Folgenden definierte globale Reservierungsbedingung übersichtlicher darstellen zu können.

Definition 2.8 (*Globale Reservierungsbedingung*)

Sei $G = (V, E)$ ein Netz, und seien $a, b \in V$ mit $b \in CN_1(a)$. $s \in S$ sei ein Slot, und TX^s sei der Reservierungszustand von Slot s. Die globale Reservierungsbedingung F^s_{TX} *gibt an, ob s für Übertragungen von a an b frei ist:*

$$F^s_{TX}(a, b) =_{Df} \forall\, c \in IN_{\leq 1}(a) : \neg\, RX^s(c) \;\wedge\; \forall\, d \in IN_{\leq 1}(b) : \neg\, TX^s(d)\,.$$

Wegen $a, b \in IN_{\leq 1}(a)$ und $a, b \in IN_{\leq 1}(b)$ beinhaltet die Definition die notwendige Bedingung, dass a und b Slot s weder zum Senden noch zum Empfangen reserviert haben.

Aus der Definition folgt, dass das Interferenzproblem gelöst werden kann, wenn jeder Knoten die aktuelle Kommunikations- und Interferenztopologie des Netzwerks sowie die Sendereservierungen aller Knoten kennt (die Empfangsreservierungen lassen sich aus den Sendereservierungen ableiten). Abbildung 2.4 verdeutlicht die globale Reservierungsbedingung $F^s_{TX}(a, b)$.

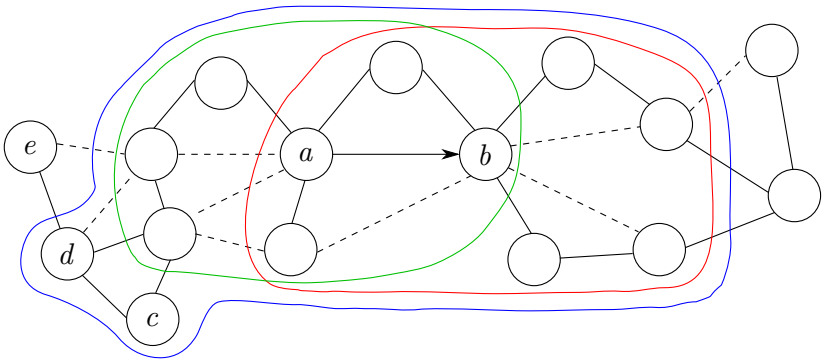

Abbildung 2.4.: Veranschaulichung der globalen Reservierungsbedingung (vgl. [IG12]).

Die inneren Linien stellen die Interferenznachbarschaften von Knoten a (grüne Linie) bzw. Knoten b (rote Linie) dar. Die blaue Linie umfasst alle Knoten, deren TX^s-Reservierungszustand zur Lösung des Interferenzproblems benötigt wird. Dies umfasst alle Knoten in Interferenzreichweite von a und b, sowie die Knoten c und d, jedoch nicht Knoten e. Knoten außerhalb der Interferenzreichweite von a (z. B. c und d) müssen berücksichtigt werden, wenn die Relation RX^s nicht explizit gespeichert wird, sondern aus TX^s abgeleitet werden muss. Wird beispielsweise ein Wert $RX^s(f, g)$ benötigt, so kann dieser aus $TX^s(g, f)$ abgeleitet werden. Zusammengefasst werden also Werte aus $IN_{\leq 1}(a)$ und $IN_{\leq 1}(b)$ benötigt, wenn die Relationen TX^s und RX^s explizit gespeichert werden. Wird RX^s nicht explizit gespeichert, so werden zudem noch Werte aus $CN_1(IN_1(a))$ benötigt.

Ähnliche Kriterien, wie sie in der globalen Reservierungsbedingung Verwendung finden, werden auch von existierenden TDMA-basierten Protokollen für QoS-Routing und Reservierungen formuliert, beispielsweise dem *Forward Algorithm* [ZC02], dem *TDMA-based Bandwidth Reservation Protocol* [LTS02] sowie dem *Distributed Slots Reservation Protocol* [SCCC06]. Diese Protokolle nehmen jedoch $\forall\, a \in V : IN_1(a) = CN_1(a)$ an, d. h. Interferenzen außerhalb der Kommunikationsreichweite werden vernachlässigt. Dies wird im Folgenden als *1-Hop-Interferenzannahme* bezeichnet. Eine genauere Betrachtung der genannten und weiterer Protokolle ist in Kapitel 4 zu finden.

Mit Definition 2.8 lässt sich die Menge der freien Sendeslots für einen Link (a, b) folgendermaßen definieren:

$$F_{TX}(a, b) =_{Df} \{s \in S : F^s_{TX}(a, b)\}\,. \tag{2.1}$$

In Kapitel 6 wird das Protokoll QMRP vorgestellt, bei dem in einem zum Senden von a an b reservierten Slot s auch ein Acknowledgement von b an a gesendet wird. In diesem Fall darf s von keinem Knoten in Interferenzreichweite von a und b zum Senden oder zum Empfangen reserviert sein. Dies wird durch die im Folgenden definierte wechselseitige globale Reservierungsbedingung beschrieben.

Definition 2.9 (*Wechselseitige globale Reservierungsbedingung*)

Sei $G = (V, E)$ ein Netz, und seien $a, b \in V$ mit $b \in CN_1(a)$. $s \in S$ sei ein Slot, und TX^s sei der Reservierungszustand von Slot s. Die wechselseitige globale Reservierungsbedingung *F^s gibt an, ob s für Übertragungen von a an b* und *von b an a frei ist:*

$$F^s(a, b) =_{Df} \forall\, c \in (IN_{\leq 1}(a) \cup IN_{\leq 1}(b)) : (\neg\, RX^s(c) \;\wedge\; \neg\, TX^s(c))\,.$$

F^s lässt sich aus der globalen Reservierungsbedingung F^s_{TX} ableiten:

$$\begin{aligned} F^s(a,b) &=_{Df} \forall\, c \in (IN_{\leq 1}(a) \cup IN_{\leq 1}(b)) : (\neg\, RX^s(c) \ \wedge\ \neg\, TX^s(c)) \\ &\equiv \quad \forall\, c \in IN_{\leq 1}(a) : \neg\, RX^s(c) \ \wedge\ \forall\, d \in IN_{\leq 1}(b) : \neg\, TX^s(d) \ \wedge \\ &\qquad \forall\, c \in IN_{\leq 1}(b) : \neg\, RX^s(c) \ \wedge\ \forall\, d \in IN_{\leq 1}(a) : \neg\, TX^s(d) \\ &\equiv \quad F^s_{TX}(a,b) \ \wedge\ F^s_{TX}(b,a)\,. \end{aligned}$$

Die Menge der freien Slots für einen Link (a, b) gemäß der wechselseitigen globalen Reservierungsbedingung lässt sich definieren als

$$\begin{aligned} F(a,b) &=_{Df} \{s \in S : F^s(a,b)\} \\ &= \quad F_{TX}(a,b) \ \cap\ F_{TX}(b,a)\,. \end{aligned}$$

Anmerkung: Da sich F^s aus F^s_{TX} ermitteln lässt, wird im Folgenden lediglich F^s_{TX} betrachtet.

2.4. Lokale Reservierungsbedingungen

Im Folgenden wird die globale Reservierungsbedingung $F^s_{TX}(a, b)$ in eine lokale Form überführt, so dass jeder Knoten Reservierungsentscheidungen basierend auf seinen lokalen Statusinformationen treffen kann. Dazu werden *lokale Prädikate* eingeführt, welche aus der Sicht eines einzelnen Knotens definiert sind.

Dabei wird in zwei Schritten vorgegangen: In Abschnitt 2.4.1 werden zunächst die Prädikate der Interferenznachbarn in lokale Prädikate des potenziellen Senders umgesetzt, womit sich eine lokale Reservierungsbedingung ergibt, welche äquivalent zur globalen ist. In Abschnitt 2.4.2 wird eine Annahme über die Interferenznachbarschaft eingeführt, die es ermöglicht, das Interferenzproblem anhand von lokalem Wissen über die Kommunikationsnachbarn zu lösen. Dadurch wird die zur Lösung des Interferenzproblems benötigte Statusinformation reduziert. Dies führt zu einer zweiten lokalen Reservierungsbedingung, welche nicht mehr äquivalent zur globalen ist, welche jedoch – unter der getroffenen Annahme – als Basis für Reservierungsprotokolle verwendet werden kann.

2.4.1. Lokale Reservierungsbedingung mit Interferenznachbarschaft

In diesem Abschnitt wird davon ausgegangen, dass jeder Knoten Zugriff auf die Statusinformationen seiner Kommunikations- *und* Interferenznachbarn hat. Aus diesen Statusinformationen können die Knoten lokale Prädikate ableiten, anhand derer die Reservierungsbedingung berechnet werden kann. Abbildung 2.5 zeigt das für die Definition dieser lokalen Reservierungsbedingung verwendete Topologiemuster sowie die lokalen Prädikate.

Um die Reservierungsbedingung zu berechnen, werden Statusinformationen in Form der jeweils unter den entsprechenden Knoten aufgelisteten Prädikate benötigt. Ein Pfeil von einem Prädikat P zu einem Prädikat Q bedeutet, dass P zur Herleitung von Q verwendet wird. Ein gestrichelter Pfeil zeigt an, dass die Werte möglicherweise nicht direkt verfügbar sind. Dies ist der Fall, wenn sich der zugehörige Knoten in Interferenz-, jedoch nicht in Kommunikationsreichweite befindet.

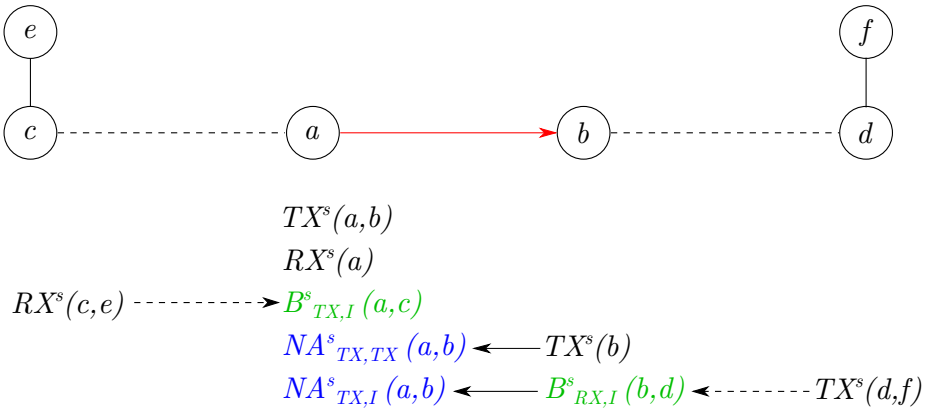

Abbildung 2.5.: Topologiemuster und lokale Prädikate für die lokale Reservierungsbedingung mit Interferenznachbarschaft.

Die globale Reservierungsbedingung verwendet die globalen Prädikate TX^s und RX^s (vgl. Definition 2.8):

$$F^s_{TX}(a,b) =_{Df} \forall\, c \in IN_{\leq 1}(a) : \neg\, RX^s(c) \;\wedge\; \forall\, d \in IN_{\leq 1}(b) : \neg\, TX^s(d)$$

$$\equiv \quad \neg\, RX^s(a) \;\wedge\; \forall\, c \in IN_1(a) : \neg\, RX^s(c) \;\wedge\; \neg\, TX^s(b) \;\wedge\; \forall\, d \in IN_1(b) : \neg\, TX^s(d)\,.$$

Um die globalen Prädikate der Interferenznachbarn in dieser Definition ersetzen zu können, werden die beiden folgenden Definitionen benötigt.

Definition 2.10

$B^s_{TX,I}(a,b) =_{Df} b \in IN_1(a) \;\wedge\; \exists\, c \in CN_1(b),\, c \neq a : RX^s(b,c)$

Slot s ist in Knoten a zum Senden (TX) blockiert (B), da Knoten b, welcher sich in Interferenzreichweite (I) zu a befindet, diesen Slot zum Empfangen reserviert hat. Die Interferenz von a könnte den Empfang in b stören. Dabei ist zu beachten, dass der sendende Knoten nicht a ist, denn wenn a an b senden würde, wäre der Slot in a zum Senden reserviert und nicht blockiert.

Definition 2.11

$B^s_{RX,I}(a,b) =_{Df} b \in IN_1(a) \;\wedge\; \exists\, c \in CN_1(b),\, c \neq a : TX^s(b,c)$

Slot s ist in Knoten a zum Empfangen (RX) blockiert (B), da Knoten b, welcher sich in Interferenzreichweite (I) zu a befindet, diesen Slot zum Senden reserviert hat. Die Interferenz von b könnte den Empfang in a stören. Dabei ist zu beachten, dass der empfangende Knoten nicht a ist, denn wenn b an a senden würde, wäre der Slot in a zum Empfangen reserviert und nicht blockiert.

Mit Hilfe der Prädikate aus den Definitionen 2.10 und 2.11 lässt sich $F^s_{TX}(a,b)$ folgendermaßen umschreiben:

$$F^s_{TX}(a,b) \equiv \neg\, RX^s(a) \;\wedge\; \forall\, c \in IN_1(a) : \neg\, RX^s(c) \;\wedge\; \neg\, TX^s(b) \;\wedge\; \forall\, d \in IN_1(b) : \neg\, TX^s(d)$$

$$\equiv \neg\, RX^s(a) \;\wedge\; \forall\, c \in IN_1(a) : (\underbrace{\neg\, RX^s(c,a)}_{\neg\, TX^s(a,c)} \;\wedge\; \underbrace{\forall\, e \in CN_1(c), e \neq a : \neg\, RX^s(c,e)}_{\neg\, B^s_{TX,I}(a,c)}) \;\wedge$$

$$\neg TX^s(b) \wedge \forall d \in IN_1(b) : (\underbrace{\neg TX^s(d,b)}_{\neg RX^s(b,d)} \wedge \underbrace{\forall f \in CN_1(d), f \neq b : \neg TX^s(d,f))}_{\neg B^s_{RX,I}(b,d)}$$

$$\equiv \neg RX^s(a) \wedge \forall c \in IN_1(a) : \neg B^s_{TX,I}(a,c) \wedge$$

$$\underbrace{\forall c \in CN_1(a), c \neq b : \neg TX^s(a,c)}_{\neg B^s_{RX,I}(b,a)} \wedge \neg TX^s(a,b) \wedge$$

$$\neg TX^s(b) \wedge \forall d \in IN_1(b) : \neg B^s_{RX,I}(b,d) \wedge$$

$$\underbrace{\forall d \in CN_1(b), d \neq a : \neg RX^s(b,d)}_{\neg B^s_{TX,I}(a,b)} \wedge \neg RX^s(b,a)$$

$$\equiv \neg RX^s(a) \wedge \forall c \in IN_1(a) : \neg B^s_{TX,I}(a,c) \wedge$$

$$\neg TX^s(b) \wedge \forall d \in IN_1(b) : \neg B^s_{RX,I}(b,d) \wedge \neg TX^s(a,b)\,. \tag{2.2}$$

Da für Knoten, die sich nicht in Interferenzreichweite zueinander befinden, keine Sendereservierung getroffen werden kann, ist $\forall c \in IN_1(a) : \neg TX^s(a,c)$ äquivalent zu $\forall c \in CN_1(a) : \neg TX^s(a,c)$ (analog für RX^s). Da $a \in IN_1(b)$, ist $\neg B^s_{RX,I}(b,a)$ in $\forall d \in IN_1(b) : \neg B^s_{RX,I}(b,d)$ enthalten. Analog ist $\neg B^s_{TX,I}(a,b)$ in $\forall c \in IN_1(a) : \neg B^s_{TX,I}(a,c)$ enthalten.

Diese Definition von $F^s_{TX}(a,b)$ enthält Prädikate, welche lokal in Bezug auf den Sender a sind, und andere, die lokal in Bezug auf den Empfänger b sind. Im Folgenden werden weitere Prädikate definiert, mit denen $F^s_{TX}(a,b)$ ausschließlich lokal in Bezug auf a dargestellt werden kann.

Definition 2.12

$NA^s_{TX,TX}(a,b) =_{Df} b \in CN_1(a) \wedge TX^s(b)$

Slot s ist zum Senden (TX) von a an b nicht verfügbar (NA; not available), da b diesen Slot bereits zum Senden (TX) reserviert hat.

Definition 2.13

$NA^s_{TX,I}(a,b) =_{Df} b \in CN_1(a) \wedge \exists c \in IN_1(b) : B^s_{RX,I}(b,c)$

Slot s ist zum Senden (TX) von a an b nicht verfügbar (NA), da ein Knoten c in Interferenzreichweite (I) von b diesen Slot zum Senden reserviert hat. Die Interferenz von c könnte den Empfang in b stören.

Anmerkung: Der Begriff „blockiert“ wird verwendet, wenn ein Knoten in einem Slot *insgesamt* nicht senden bzw. nicht empfangen kann, während „nicht verfügbar“ besagt, dass nur an einen bestimmten Knoten nicht gesendet werden kann.

Mit diesen Prädikaten lässt sich $F^s_{TX}(a,b)$ folgendermaßen umschreiben:

$$F^s_{TX}(a,b) \equiv \neg RX^s(a) \wedge \forall c \in IN_1(a) : \neg B^s_{TX,I}(a,c) \wedge$$
$$\neg NA^s_{TX,TX}(a,b) \wedge \neg NA^s_{TX,I}(a,b) \wedge \neg TX^s(a,b)\,.$$

Diese Definition von $F^s_{TX}(a,b)$ ist äquivalent zur globalen Reservierungsbedingung aus Definition 2.8. Ferner basiert sie ausschließlich auf lokalem Wissen über den Reservierungszustand

von Knoten in Interferenzreichweite von a. Deshalb handelt es sich um eine *lokale Reservierungsbedingung*.

Um $F^s_{TX}(a,b)$ berechnen zu können, muss a die Werte für seine lokalen Prädikate bestimmen. Der Wert für $RX^s(a)$ ist aus der Liste der Empfangsreservierungen von a direkt verfügbar; ebenso ist der Wert für $TX^s(a,b)$ aus der Liste der Sendereservierungen verfügbar. Zur Bestimmung der $B^s_{TX,I}$-Werte werden die RX^s-Werte aller *Interferenznachbarn* von a benötigt, welche sich auch außerhalb der Kommunikationsreichweite befinden können. $NA^s_{TX,TX}(a,b)$ kann aus den Sendereservierungen von b abgeleitet werden. Zur Berechnung von $NA^s_{TX,I}(a,b)$ werden die $B^s_{RX,I}$-Werte von b benötigt. Um diese zu berechnen, benötigt b die TX^s-Werte seiner Interferenznachbarn.

Nun liegt zwar eine lokale Definition von $F^s_{TX}(a,b)$ vor; jedoch erfordert diese immer noch die Verfügbarkeit der Statusinformationen von Knoten in Interferenzreichweite.

2.4.2. Lokale Reservierungsbedingung mit Kommunikationsnachbarschaft

Um die Werte der Prädikate $B^s_{TX,I}(a,c)$ und $B^s_{RX,I}(b,d)$ (für $NA^s_{TX,I}(a,b)$ benötigt) der Knoten a bzw. b zu ermitteln, werden die Statusinformationen der Interferenznachbarn c bzw. d benötigt, welche sich außerhalb der Kommunikationsreichweite befinden können. Stehen Black-Burst-basierte Übertragungen (s. Abschnitt 3.3) zur Verfügung, so ist eine Kommunikation mit den Interferenznachbarn möglich. Ist dies jedoch nicht der Fall, so können die Statusinformationen nicht direkt beschafft werden.

Aufgrund der Single Network Property sind zwei beliebige Knoten eines Netzes stets durch einen Pfad von Kommunikationslinks verbunden. Es wird angenommen, dass der Netzwerkdurchmesser durch eine obere Schranke $n_{maxHops}$ beschränkt ist. Somit sind zwei Knoten in Interferenzreichweite immer durch einen Kommunikationspfad höchstens dieser Länge verbunden.

Wird weiterhin angenommen, dass die Topologie zu einem gewissen Grad kontrolliert werden kann, so kann die Kommunikationsdistanz von Knoten in Interferenzreichweite auf einen Wert d beschränkt werden, mit $d \geq 2$. Ist dies der Fall, so gilt $\forall\, a \in V : IN_{\leq 1}(a) \subseteq CN_{\leq d}(a)$. Da Reservierungsentscheidungen konservativ getroffen werden sollen, wird weiterhin angenommen, dass $\forall\, a \in V : IN_{\leq 1}(a) \supseteq CN_{\leq d}(a)$ gilt, d. h. dass sich alle Knoten mit einer Kommunikationsdistanz von höchstens d in Interferenzreichweite befinden. Insgesamt gilt damit $\forall\, a \in V : IN_{\leq 1}(a) = CN_{\leq d}(a)$ bzw. $\forall\, a \in V : IN_1(a) = CN_{\leq d}(a) \setminus \{a\}$.

Dabei ist zu beachten, dass $\forall\, a \in V : IN_{\leq 1}(a) \subseteq CN_{\leq d}(a)$ stets erfüllt sein muss, da sonst keine Kollisionsfreiheit garantiert werden kann. Ist $\forall\, a \in V : IN_{\leq 1}(a) \supseteq CN_{\leq d}(a)$ nicht erfüllt, so führt dies lediglich zu einer schlechteren Ausnutzung der Bandbreite durch überflüssige Sende- bzw. Empfangsblockierungen.

Im Folgenden wird von $d = 2$ ausgegangen, d. h. $\forall\, a \in V : IN_1(a) = CN_1(a) \cup CN_2(a)$ (dies wird als *2-Hop-Interferenzannahme* bezeichnet). Damit haben alle Knoten in Interferenzreichweite eine Kommunikationsdistanz von maximal zwei Hops.

Abbildung 2.6 erweitert das Topologiemuster aus Abbildung 2.5 unter Berücksichtigung der 2-Hop-Interferenzannahme (es sind nicht alle Interferenzlinks explizit dargestellt). Unter Berücksichtigung dieser Annahme werden lokale Prädikate von Knoten in Kommunikationsreichweite hinzugefügt, welche zur Abbildung der Prädikate der Interferenznachbarn verwendet werden.

Die lokalen Prädikate für die Kommunikationsnachbarn werden durch die beiden folgenden Definitionen beschrieben.

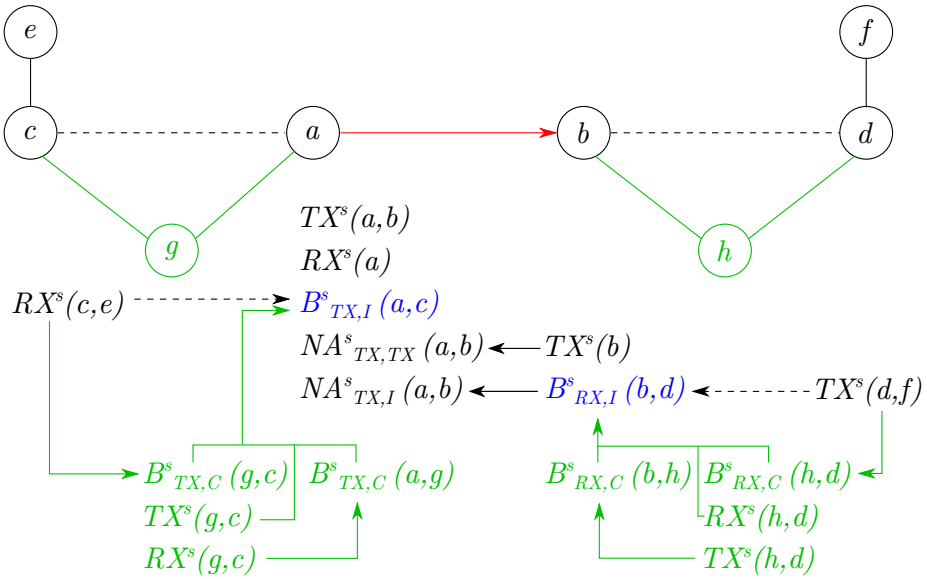

Abbildung 2.6.: Topologiemuster und lokale Prädikate für die lokale Reservierungsbedingung mit Kommunikationsnachbarschaft (unter 2-Hop-Interferenzannahme).

Definition 2.14

$B^s_{TX,C}(a,b) =_{Df} b \in CN_1(a) \ \wedge \ \exists c \in CN_1(b),\, c \neq a : RX^s(b,c)$

Slot s ist in Knoten a zum Senden (TX) blockiert (B), da Knoten b, welcher sich in Kommunikationsreichweite (C) zu a befindet, diesen Slot zum Empfangen (von einem anderen Knoten als a) reserviert hat. Die Interferenz von a könnte den Empfang in b stören.

Definition 2.15

$B^s_{RX,C}(a,b) =_{Df} b \in CN_1(a) \ \wedge \ \exists c \in CN_1(b),\, c \neq a : TX^s(b,c)$

Slot s ist in Knoten a zum Empfangen (RX) blockiert (B), da Knoten b, welcher sich in Kommunikationsreichweite (C) zu a befindet, diesen Slot zum Senden (an einen anderen Knoten als a) reserviert hat. Die Interferenz von b könnte den Empfang in a stören.

Definition 2.14 unterscheidet sich geringfügig von Definition 2.10, da in Definition 2.14 Knoten in Kommunikationsreichweite betrachtet werden, während sich Definition 2.10 auf Knoten in Interferenzreichweite bezieht. Analoges gilt für die Definitionen 2.15 und 2.11.

Die Prädikate $B^s_{TX,I}(a,b)$ und $B^s_{RX,I}(a,b)$ können, basierend auf der 2-Hop-Interferenzannahme, aus dem Reservierungszustand von Knoten in Kommunikationsreichweite von a abgeleitet werden.

$$
\begin{aligned}
B^s_{TX,I}(a,b) \ &=_{Df} b \in IN_1(a) \ \wedge \ \exists c \in CN_1(b),\, c \neq a : RX^s(b,c) \\
&\equiv \quad b \in (CN_1(a) \cup CN_2(a)) \ \wedge \ \exists c \in CN_1(b),\, c \neq a : RX^s(b,c) \\
&\equiv \quad \underbrace{(b \in CN_1(a) \wedge \exists c \in CN_1(b),\, c \neq a : RX^s(b,c))}_{B^s_{TX,C}(a,b)} \ \vee
\end{aligned}
$$

$$(b \in CN_2(a) \wedge \exists\, c \in CN_1(b), \underbrace{c \neq a}_{\text{gilt, da } a \notin CN_1(b)} : RX^s(b,c))$$

$$\equiv \quad B^s_{TX,C}(a,b) \ \vee \ \exists\, d \in CN_1(a) :$$

$$(\underbrace{b \in CN_1(d) \wedge \exists\, c \in CN_1(b),\ c \neq d : RX^s(b,c)}_{B^s_{TX,C}(d,b)} \vee \underbrace{b \in CN_1(d) \wedge RX^s(b,d)}_{TX^s(d,b)})$$

$$\equiv \quad B^s_{TX,C}(a,b) \ \vee \ \exists\, d \in CN_1(a) : (B^s_{TX,C}(d,b) \ \vee \ TX^s(d,b)) \tag{2.3}$$

$$B^s_{RX,I}(a,b) =_{Df} b \in IN_1(a) \ \wedge \ \exists\, c \in CN_1(b),\ c \neq a : TX^s(b,c)$$

$$\equiv \quad B^s_{RX,C}(a,b) \ \vee \ \exists\, d \in CN_1(a) : (B^s_{RX,C}(d,b) \ \vee \ RX^s(d,b)) \tag{2.4}$$

Damit kann a die $B^s_{TX,I}$-Werte aus seinen eigenen $B^s_{TX,C}$-Werten sowie den $B^s_{TX,C}$- und TX^s-Werten seiner Kommunikationsnachbarn herleiten. Für den allgemeinen Fall $IN_1(a) = CN_{\leq d}(a) \setminus \{a\}$ würden dazu die $B^s_{TX,C}$- und TX^s-Werte aller Knoten in $CN_{\leq d-1}(a)$ benötigt. Die $B^s_{TX,C}$-Werte kann jeder Knoten aus den RX^s-Werten seiner Nachbarknoten bestimmen. Analog können die $B^s_{RX,I}$-Werte von b aus den eigenen $B^s_{RX,C}$-Werten sowie den $B^s_{RX,C}$- und RX^s-Werten der Nachbarknoten ermittelt werden (für den allgemeinen Fall würden auch hier die $B^s_{RX,C}$- und RX^s-Werte aller Knoten in $CN_{\leq d-1}(b)$ benötigt). Die $B^s_{RX,C}$-Werte kann jeder Knoten aus den TX^s-Werten seiner Nachbarn berechnen. Auf diese Weise kann a die lokale Form der Reservierungsbedingung $F^s_{TX}(a,b)$ durch Aggregation der Prädikate seiner Nachbarn berechnen.

2.5. Ermittlung einer Route vom Ziel zur Quelle

Die globale und die lokalen Reservierungsbedingung(en) geben an, ob ein Slot s von einem Knoten a zum Senden an einen Knoten b reserviert werden kann. Dies kann für QoS-Routing- und Reservierungsprotokolle verwendet werden, um die freien Slots einer Route zu ermitteln. Das in Kapitel 5 vorgestellte Protokoll RBBQR bestimmt Routen in umgekehrter Richtung, d. h. vom Ziel zur Quelle. Um dies realisieren und dabei für jeden Link die freien Slots möglichst effizient ermitteln zu können, wird eine weitere Form der Reservierungsbedingung definiert.

Definition 2.16

Sei $G = (V, E)$ ein Netz, und sei $b \in V$. $s \in S$ sei ein Slot, und TX^s sei der Reservierungszustand von Slot s. F^s_{RX} gibt an, ob s in b frei zum Empfangen ist:

$$F^s_{RX}(b) =_{Df} \neg\, TX^s(b) \wedge \neg\, RX^s(b) \wedge \forall\, d \in IN_1(b) : \neg\, B^s_{RX,I}(b,d)\,.$$

Damit lässt sich die lokale Reservierungsbedingung (s. Formel (2.2)) folgendermaßen umschreiben:

$$F^s_{TX}(a,b) \ \equiv \neg\, RX^s(a) \ \wedge \ \forall\, c \in IN_1(a) : \neg\, B^s_{TX,I}(a,c) \ \wedge$$

$$\neg\, TX^s(b) \ \wedge \ \forall\, d \in IN_1(b) : \neg\, B^s_{RX,I}(b,d) \ \wedge \ \neg\, TX^s(a,b)$$

$$\equiv \neg\, RX^s(a) \;\wedge\; \forall\, c \in IN_1(a) : \neg\, B^s_{TX,I}(a,c) \;\wedge$$

$$\neg\, TX^s(b) \;\wedge\; \forall\, d \in IN_1(b) : \neg\, B^s_{RX,I}(b,d) \;\wedge$$

$$\underbrace{\neg\, B^s_{TX,I}(a,b) \;\wedge\; \neg\, TX^s(a,b)}_{\neg\, RX^s(b)}$$

$$\equiv \neg\, RX^s(a) \;\wedge\; \forall\, c \in IN_1(a) : \neg\, B^s_{TX,I}(a,c) \;\wedge\; F^s_{RX}(b)\,. \tag{2.5}$$

Anmerkung: Das Prädikat $RX^s(b)$, welches in $F^s_{RX}(b)$ vorkommt, ist für $F^s_{TX}(a,b)$ teilweise redundant, denn der durch $B^s_{TX,I}(a,b)$ beschriebene Teil ist bereits in $B^s_{TX,I}(a,c)$, $c \in IN_1(a)$, enthalten.

Mit Definition 2.16 lässt sich die Menge der freien Empfangsslots eines Knotens b folgendermaßen definieren:

$$\begin{aligned} F_{RX}(b) &=_{Df} \{s \in S : F^s_{RX}(b)\} \\ &= \{s \in S : \neg\, TX^s(b) \;\wedge\; \neg\, RX^s(b) \;\wedge\; \forall\, d \in IN_1(b) : \neg\, B^s_{RX,I}(b,d)\}\,. \end{aligned}$$

Die Menge der freien Sendeslots für den Link (a,b) (bereits in Formel (2.1) definiert) lässt sich mit Hilfe von Formel (2.5) wie folgt berechnen:

$$\begin{aligned} F_{TX}(a,b) &= \{s \in S : F^s_{TX}(a,b)\} \\ &= \{s \in S : \neg\, RX^s(a) \;\wedge\; \forall\, c \in IN_1(a) : \neg\, B^s_{TX,I}(a,c)\} \;\cap\; F_{RX}(b)\,. \end{aligned}$$

Werden in jedem Knoten a die Prädikate $TX^s(a,c)$, $RX^s(a,c)$, $B^s_{TX,I}(a,c)$ und $B^s_{RX,I}(a,c)$ für alle Slots s und alle Kommunikations- bzw. Interferenznachbarn c gespeichert (d. h. proaktiv ermittelt), so ist es möglich, zuerst $F_{RX}(b)$ (d. h. die freien Empfangsslots für den Empfänger b) zu bestimmen, diese zum Sender a zu übertragen und dort $F_{TX}(a,b)$ zu berechnen.

Die *NA*-Prädikate aus den Definitionen 2.12 und 2.13 werden in der obigen Form der Reservierungsbedingung nicht explizit verwendet, sondern sind in $F^s_{RX}(b)$ implizit enthalten. Da diese Prädikate nicht explizit verwendet werden, kann ein Knoten eine Reservierungsentscheidung nicht lokal treffen, sondern er benötigt dazu die F_{RX}-Information des Empfängers.

Die Prädikate $B^s_{TX,I}$ und $B^s_{RX,I}$ können – unter der 2-Hop-Interferenzannahme – gemäß den Formeln (2.3) und (2.4) berechnet werden. Auf diese Weise muss jeder Knoten nur mit seinen Kommunikationsnachbarn kommunizieren. In Kapitel 5 wird jedoch für RBBQR eine Black-Burst-basierte Kommunikation (s. Abschnitt 3.3) verwendet. Damit ist es möglich, direkt mit den Interferenznachbarn zu kommunizieren, so dass die Prädikate $B^s_{TX,I}$ und $B^s_{RX,I}$ unmittelbar berechnet werden können.

2.6. Propagierung getroffener Sendereservierungen

Kann eine Sendereservierung $TX^s(a,b)$ getroffen werden, so muss diese Statusinformation an die betreffenden Knoten propagiert werden. Dies wird durch Abbildung 2.7 verdeutlicht, welche eine Umkehrung des Szenarios von Abbildung 2.6 auf Seite 27 darstellt.

Es wird davon ausgegangen, dass die Reservierung zunächst nur dem Sender a bekannt ist. Kann eine Black-Burst-basierte Übertragung (s. Abschnitt 3.3) verwendet werden, so können die Interferenznachbarn direkt erreicht werden. In diesem Fall wird $TX^s(a,b)$ von a

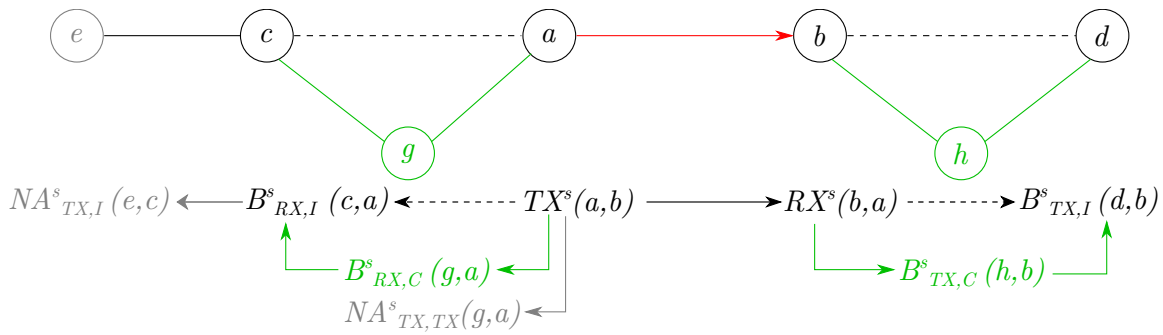

Abbildung 2.7.: Propagierung einer Reservierung $TX^s(a,b)$ (unter 2-Hop-Interferenzannahme).

an alle Interferenznachbarn c übertragen (dazu wird nur eine Übertragung benötigt), welche das Prädikat als $B^s_{RX,I}(c,a)$ speichern. Die Interferenznachbarschaft umfasst auch alle Kommunikationsnachbarn von a; lediglich b speichert es als $RX^s(b,a)$. Anschließend muss b das Prädikat auf gleiche Weise an seine Interferenznachbarn d weiterleiten, welche es als $B^s_{TX,I}(d,b)$ speichern.

Ist eine Black-Burst-basierte Übertragung nicht möglich, so kann die 2-Hop-Interferenzannahme verwendet werden. In diesem Fall wird $TX^s(a,b)$ von a an die Kommunikationsnachbarn g weitergeleitet. Diese speichern das Prädikat als $B^s_{RX,C}(g,a)$ (dies impliziert auch $B^s_{RX,I}(g,a)$) und leiten $B^s_{RX,C}(g,a)$ wiederum an ihre Kommunikationsnachbarn c weiter, welche es als $B^s_{RX,I}(c,a)$ speichern. Nachteilig ist in diesem Fall, dass zwei Übertragungen benötigt werden, um die Interferenznachbarn von a zu erreichen, und dass möglicherweise mehrere Knoten gleichzeitig versuchen, ein Paket weiterzuleiten. Auch hier empfängt b die erste Übertragung, speichert $RX^s(b,a)$ und leitet es an seine Kommunikationsnachbarn h weiter, welche das Prädikat als $B^s_{TX,C}(h,b)$ (impliziert $B^s_{TX,I}(h,b)$) speichern. Diese leiten $B^s_{TX,C}(h,b)$ wiederum an ihre Kommunikationsnachbarn d weiter, welche es als $B^s_{TX,I}(d,b)$ speichern.

Sollen zusätzlich die NA-Prädikate explizit verwendet werden, so speichern die Kommunikationsnachbarn g von a das Prädikat $NA^s_{TX,TX}(g,a)$ ab, wenn sie $TX^s(a,b)$ empfangen. Zusätzlich muss jeder Interferenznachbar c von a, der $B^s_{RX,I}(c,a)$ gespeichert hat (dies umfasst auch die Kommunikationsnachbarn), das Prädikat an seine Kommunikationsnachbarn e weiterleiten, welche es dann als $NA^s_{TX,I}(e,c)$ vermerken.

Anmerkung: Werden Routen vom Ziel zur Quelle gesucht, so werden die NA-Prädikate i. d. R. nicht explizit verwendet.

2.7. Wiederverwendung von Slots entlang einer Route

Wird die 2-Hop-Interferenzannahme ($\forall\, a \in V : IN_1(a) = CN_1(a) \cup CN_2(a)$) getroffen, so kann ein Slot auf vier aufeinanderfolgenden Links einer Route nur einmal verwendet werden. Dies lässt sich anhand der in Abbildung 2.8 gezeigten Route veranschaulichen (es ist hier nur der Interferenzlink zwischen b und d explizit dargestellt).

Die auf dem Link (d,e) verwendeten Slots können auf (c,d) nicht verwendet werden, da d sendet (es wird hier angenommen, dass die Route in umgekehrter Richtung vom Ziel zur Quelle ermittelt wird). Auf (b,c) und (a,b) können sie nicht verwendet werden, da sowohl c als auch b sich in Interferenzreichweite zu d befinden und die Sendevorgänge von d somit

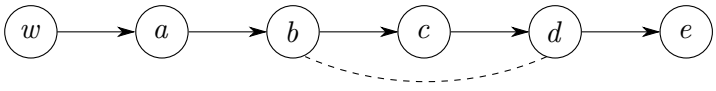

Abbildung 2.8.: Beispielroute zur Illustration der Slotwiederverwendung.

einen erfolgreichen Empfang verhindern. Auf (w, a) können die Slots jedoch wiederverwendet werden.

Anmerkung: Ist $\forall\, a \in V : IN_1(a) \subseteq CN_1(a) \cup CN_2(a)$ nicht erfüllt, so können Kollisionen auftreten. Ist $\forall\, a \in V : IN_1(a) \supseteq CN_1(a) \cup CN_2(a)$ nicht erfüllt, so führt dies nicht zu Fehlern, bewirkt jedoch eine schlechtere Ausnutzung der Bandbreite. Im obigen Beispiel wäre dies der Fall, wenn b sich nicht in Interferenzreichweite zu d befände, da die Slots dann bereits auf (a, b) wiederverwendet werden könnten.

Allgemein betrachtet, kann eine QoS-Anforderung von n Slots also nur erfüllt werden, wenn für jeweils vier aufeinanderfolgende Links (a, b), (b, c), (c, d) und (d, e) folgendes gilt:

1. $|F_{TX}(a, b)| \geq n$, $|F_{TX}(b, c)| \geq n$, $|F_{TX}(c, d)| \geq n$, $|F_{TX}(d, e)| \geq n$
2. $|F_{TX}(a, b) \cup F_{TX}(b, c) \cup F_{TX}(c, d) \cup F_{TX}(d, e)| \geq 4n$

Zunächst müssen auf jedem Link n freie Slots zur Verfügung stehen. Da die Slots pro vier Links nur einmal verwendet werden können, müssen insgesamt mindestens $4n$ Slots zur Verfügung stehen, die nicht identisch sind.

Anmerkung: Es handelt sich um notwendige, jedoch nicht um hinreichende Bedingungen. Beispielsweise könnte der Fall eintreten, dass $F_{TX}(b, c) = F_{TX}(c, d) = F_{TX}(d, e)$, $F_{TX}(a, b) \cap F_{TX}(b, c) = \emptyset$, $|F_{TX}(a, b)| = 3n$ sowie $|F_{TX}(b, c)| = n$ gilt. Dann wären die obigen Bedingungen erfüllt, aber nach Auswahl der Slots für (d, e) gäbe es für (b, c) und (c, d) keine freien Slots mehr.

Eine hinreichende Bedingung wäre $|F_{TX}(a, b)| \geq 4n$, $|F_{TX}(b, c)| \geq 4n$, $|F_{TX}(c, d)| \geq 4n$, $|F_{TX}(d, e)| \geq 4n$. Bei der Routensuche ist die Überprüfung einer hinreichenden Bedingung jedoch nicht sinnvoll, da dann u. U. keine Route gefunden wird, obwohl eine existiert. Statt dessen werden jeweils die notwendigen Bedingungen geprüft. Sind diese erfüllt, so wird versucht, eine Route zu etablieren.

Die Menge der für einen Link (d, e) *ausgewählten* Slots wird im Folgenden mit $LK(d, e)$ bezeichnet (LK steht für Link). Sobald die Slots für einen Link (d, e) ausgewählt wurden, stehen diese den nächsten drei Links nicht mehr zur Verfügung, d. h. es gilt dann beispielsweise $F_{TX}(c, d)' = F_{TX}(c, d) \setminus LK(d, e)$.

2.8. Vergleich mit existierenden Interferenzmodellen

Das Interferenzproblem in drahtlosen Netzen wurde bereits in zahlreichen Arbeiten betrachtet, welche die Bedingungen für Interferenz durch unterschiedliche Modelle beschreiben. Eine umfassende Zusammenfassung derartiger Interferenzmodelle wurde von Cardieri erstellt [Car10]. Im Folgenden wird die dort vorgenommene Einteilung existierender Interferenzmodelle kurz beschrieben und das hier vorgestellte Interferenzmodell eingeordnet.

Im Allgemeinen dienen Interferenzmodelle dazu, zu bestimmen, unter welchen Bedingungen eine Übertragung zwischen zwei Knoten erfolgreich ist. Dies hängt von vielen Faktoren ab, beispielsweise von der räumlichen Anordnung der Knoten, den Umweltbedingungen, den

Eigenschaften des Transceivers und des verwendeten Kanals, den Signaleigenschaften und der Signalausbreitung sowie der zeitlichen Nutzung des Kanals. Insgesamt ist eine exakte Betrachtung von Interferenzen deshalb schwierig.

Existierende Interferenzmodelle werden von Cardieri in drei Kategorien eingeteilt: 1. statistische Interferenzmodelle, 2. Modelle, welche die Effekte von Interferenz beschreiben und 3. graphbasierte Interferenzmodelle.

1. *Statistische Interferenzmodelle* beschreiben nicht die Auswirkungen, sondern direkt die Eigenschaften des Interferenzsignals. Relevante Aspekte, beispielsweise die Übertragungseigenschaften, werden mit Hilfe einer Wahrscheinlichkeits(dichte)funktion dargestellt. Derartige Interferenzmodelle werden beispielsweise benötigt, um eine Performanzanalyse von Empfangstechniken für drahtlose Kommunikation durchzuführen. Auch wenn man die Performanz eines drahtlosen Netzes analysieren und dabei die Effekte des Übertragungskanals, die Positionen der interferierenden Knoten sowie Verkehrsmuster berücksichtigen will, werden diese Modelle benötigt, da dies i. d. R. stochastische Größen sind. Beispielsweise könnte untersucht werden, mit welcher Wahrscheinlichkeit ein Link ausfällt.

2. *Modelle zur Beschreibung von Interferenzeffekten* bilden die Eigenschaften von Transceiver und Kanal mit unterschiedlichem Detailgrad ab. Bekannte Modelle dieser Kategorie sind das Protokollinterferenzmodell und das physikalische Interferenzmodell. Diese Modelle wurden von Gupta und Kumar in [GK00] vorgestellt.

   Das *Protokollinterferenzmodell* legt ein einfaches deterministisches Modell für den Pfadverlust zugrunde und geht davon aus, dass alle Knoten des Netzwerks (Terminals) mit derselben Signalstärke senden. In diesem Modell ist eine Übertragung zwischen einem Terminal X_i und dem zugehörigen Empfänger $X_{R(i)}$ erfolgreich, wenn die Distanz zwischen $X_{R(i)}$ und einem beliebigen, auf demselben Kanal sendenden Terminal X_k um einen Faktor $(1 + \Delta)$ größer ist als die zwischen X_i und $X_{R(i)}$, d. h.

   $$|X_k - X_{R(i)}| \geq (1 + \Delta)|X_i - X_{R(i)}| \,.$$

   Dabei bezeichnet X_i sowohl das Terminal selbst als auch dessen Position, während $|X_i - X_{R(i)}|$ den Abstand zwischen X_i und $X_{R(i)}$ beschreibt. Δ ist die Größe der spezifizierten Sicherheitszone, innerhalb derer keine parallelen Übertragungen stattfinden dürfen. Gleichzeitig können durch Δ auch Schwankungen der Übertragungsreichweite berücksichtigt werden.

   Das *physikalische Interferenzmodell* ist detaillierter als das Protokollinterferenzmodell. Es wird eine Menge $\{ (X_i, X_{R(i)}) : i \in \mathcal{T} \}$ von sendenden und empfangenden Terminals definiert, wobei jedes Terminal X_i eine individuelle Sendesignalstärke P_i hat. Gemäß dieses Modells kann eine Übertragung eines Terminals X_i von $X_{R(i)}$ empfangen werden, sofern die Signal to Interference plus Noise Ratio (SINR) beim Empfänger $X_{R(i)}$ einen Schwellwert β nicht unterschreitet, d. h.

   $$\frac{\frac{P_i}{|X_i - X_{R(i)}|^\alpha}}{N + \sum_{k \in \mathcal{T},\, k \neq i} \frac{P_k}{|X_k - X_{R(i)}|^\alpha}} \geq \beta \,.$$

   Dabei ist N das Grundrauschen (Noise), und die Signalstärke nimmt mit Distanz r im Verhältnis $\frac{1}{r^\alpha}$ ab.

Während das Protokollinterferenzmodell jeweils nur Interferenzen zwischen zwei Terminals betrachtet, berücksichtigt das physikalische Interferenzmodell die aggregierte Interferenz, welche von allen anderen Terminals in Kombination verursacht wird. Das Protokollinterferenzmodell vernachlässigt somit einige Aspekte der Kommunikation, ist dafür aber relativ einfach und wird deshalb oft für Design und Evaluation von Kommunikationsprotokollen verwendet. Das physikalische Interferenzmodell beschreibt die Interferenzeffekte realistischer, weshalb es für die Betrachtung von Aspekten der physikalischen Schicht (z. B. Bestimmung der Kapazität) besser geeignet ist.

3. *Graphbasierte Modelle* beschreiben ebenfalls die Auswirkungen von Interferenz (und sind damit eigentlich Bestandteil von Kategorie 2), nutzen jedoch Konzepte der Graphentheorie. Sie repräsentieren ein Netzwerk als eine Menge von Knoten, welche beispielsweise Terminals darstellen, aber auch Links modellieren können. Diese Knoten sind durch Kanten verbunden (welche Kommunikationslinks darstellen können, oder beispielsweise Verbindungen zwischen Links, die aufgrund von Interferenz nicht gleichzeitig betrieben werden können). Das in diesem Kapitel vorgestellte Interferenzmodell fällt in diese Kategorie.

Es gibt verschiedene Arten graphbasierter Modelle, welche zur Interferenzmodellierung verwendet werden.

- Im *Konnektivitätsgraph* werden die Terminals durch Knoten des Graphen abgebildet, während dessen Kanten Kommunikationslinks repräsentieren. Ein Kommunikationslink zwischen zwei Knoten existiert genau dann, wenn der Empfänger sich innerhalb der Übertragungsreichweite des Senders befindet, d. h. wenn die Signalstärke beim Empfänger über einem festgesetzten Wert liegt. Dies wird auch als *boolesches Modell* bezeichnet. In einfachen Modellen werden Interferenzen nicht berücksichtigt, d. h. nur wenn keine parallelen Übertragungen stattfinden, ist Kollisionsfreiheit garantiert.

  In [DBT05] wird ein komplexeres Modell vorgestellt, welches das physikalische Interferenzmodell verwendet. In diesem existiert ein Kommunikationslink zwischen zwei Knoten genau dann, wenn die SINR – unter Berücksichtigung der kumulierten Interferenz aller anderen Knoten – einen vorgegebenen Schwellwert nicht unterschreitet. Dies muss für beide Knoten gelten, da unidirektionale Links vernachlässigt werden. Damit hängt die Existenz eines Links nicht mehr nur von den Positionen der beiden beteiligten Knoten ab, sondern von den Positionen aller Knoten im Netz.

- Ein *Interferenzgraph* kann entweder die Interferenz zwischen Knoten oder die Interferenz zwischen (Kommunikations)Links modellieren. Wird die Interferenz zwischen Knoten modelliert, so entsprechen die Knoten des Graphen denen des Netzwerks, während die Kanten Interferenzlinks abbilden. Diese Form des Interferenzgraphen ist ähnlich zum Konnektivitätsgraphen; es unterscheiden sich nur die Kanten. Beide Graphen lassen sich auch kombinieren. Einfache Ansätze verwenden einen Interferenzgraphen, der dem Konnektivitätsgraphen entspricht. Solche Modelle werden häufig von existierenden TDMA-basierten QoS-Routing-Protokollen verwendet, z. B. [ZC02, LTS02, SCCC06] (vgl. Kapitel 4). Ein Interferenzgraph, der auch Interferenzen zwischen Knoten außerhalb der Kommunikationsreichweite berücksichtigt, wird beispielsweise in [SK99] verwendet; der dort vorgestellte Ansatz wurde hier adaptiert (s. u.).

  Die in [CF08] und [LRSG12] vorgestellten Arbeiten kombinieren einen graphbasierten Ansatz mit dem physikalischen Interferenzmodell. Zunächst werden potenziell verwend-

bare Slots mit Hilfe eines Graphen identifiziert; anschließend wird geprüft, ob die SINR in diesen Slots ausreichend ist.

Wird die Interferenz zwischen Kommunikationslinks modelliert, so entsprechen die Knoten des Interferenzgraphen Kommunikationslinks und die Kanten der Interferenz zwischen diesen Links. Es kann sowohl das Protokollinterferenzmodell als auch das physikalische Modell zugrunde gelegt werden. Wird das Protokollinterferenzmodell zugrunde gelegt, so existiert ein Interferenzlink zwischen zwei Kommunikationslinks, wenn der Sender des einen Links sich in Interferenzreichweite des Empfängers des anderen Links befindet. Dabei ist zu beachten, dass die Interferenzreichweite hier für jeden Kommunikationslink unterschiedlich ist und vom Abstand zwischen Sender und Empfänger abhängt. Ein derartiger Ansatz wird in [JPPQ05] vorgestellt. Der resultierende Graph wird dort als *Konfliktgraph* bezeichnet und ist ungerichtet. Im Allgemeinen sind solche Interferenzgraphen jedoch gerichtet. Generell gibt es zwei mögliche Formen, solche Modelle zu definieren. Ein empfängerspezifisches Modell legt fest, wie nahe sich ein interferierender Knoten an einem Empfänger befinden kann, ohne den Empfang zu stören. Dagegen definiert ein senderspezifisches Modell, wie nahe ein Empfänger einem Sender sein kann, ohne von diesem gestört zu werden.

Bei Verwendung des physikalischen Interferenzmodells kann beispielsweise ein gerichteter Konfliktgraph erstellt werden, bei dem die Links gewichtet sind und das Gewicht jeweils angibt, wie stark ein Kommunikationslink einen anderen stört. Dies wird ebenfalls in [JPPQ05] vorgestellt.

Graphbasierte Modelle haben gegenüber den anderen beiden Kategorien einige Vorteile. So können beispielsweise Konzepte der Graphentheorie verwendet werden, um Eigenschaften des Netzwerks zu analysieren. Zudem sind sie meist weniger komplex und einfacher zu handhaben, was jedoch mit einem Verlust an Genauigkeit einhergeht. Aufgrund ihrer Natur eignen sie sich besonders dafür, um paarweise Interferenzen zwischen Terminals oder Links zu modellieren. Sie lassen sich beispielsweise verwenden, um Techniken zur Kontrolle der Topologie zu untersuchen.

In dieser Arbeit bestand die Zielsetzung darin, ein Interferenzmodell zu finden, welches kollisionsfreie Reservierungsentscheidungen erlaubt. Aufgrund der genannten Vorteile wurde ein graphbasierter Ansatz gewählt. Da die Reservierungsentscheidungen von einzelnen Knoten (ggf. unter Berücksichtigung von Informationen anderer Knoten) getroffen und zudem Interferenzen betrachtet werden sollen, wurde hier – angelehnt an [SK99] – eine Mischung aus Konnektivitäts- und Interferenzgraph verwendet. In [SK99] wird der Konnektivitätsgraph mit der ersten Version des Interferenzgraphen verbunden und zusätzlich um Wahrnehmungslinks erweitert. Dieser Ansatz wurde von der Idee her adaptiert, jedoch wurden einige Definitionen aus [SK99] angepasst. So besteht dort beispielsweise nur dann ein Kommunikationslink von einem Knoten a zu einem Knoten b, wenn a zu irgendeinem Zeitpunkt Pakete an b sendet. Deshalb sind Kommunikationslinks in diesem Modell gerichtet. Interferenz- und Wahrnehmungslinks sind in [SK99] ungerichtet und entsprechen im Wesentlichen der hier verwendeten Definition (der einzige Unterschied besteht darin, dass in [SK99] jeder Knoten einen Interferenz- und einen Wahrnehmungslink zu sich selbst hat).

Mit Hilfe dieses Graphen lassen sich die globale Reservierungsbedingung (Abschnitt 2.3) und die lokale Reservierungsbedingung mit Interferenznachbarschaft (Abschnitt 2.4.1) formulieren. Ferner können Interferenzlinks unter gewissen Annahmen auf Kommunikationslinks abgebildet werden. Dies ermöglicht eine Variante der Interferenzmodellierung, bei der Interferenz nur dann auftritt, wenn ein Empfänger sich in der 1- oder 2-Hop-Nachbarschaft eines

anderen Senders befindet. Auf diese Weise wurde eine lokale Reservierungsbedingung mit Kommunikationsnachbarschaft definiert (Abschnitt 2.4.2). Ferner lässt sich so eine Variante der Reservierungsbedingung definieren, die besonders für QoS-Routingverfahren geeignet ist, welche Routen vom Ziel zur Quelle ermitteln (Abschnitt 2.5).

3 Kapitel 3.

Grundlagen für Evaluation und Entwicklung von QoS-Routing- und Reservierungsprotokollen

In diesem Kapitel werden einige Grundlagen vorgestellt, die in den folgenden Kapiteln Verwendung finden. Zunächst wird ein kurzer Überblick über Quality of Service (QoS) im Allgemeinen gegeben und eine Einordnung von QoS-Routing- und Reservierungsprotokollen vorgenommen. Dabei werden auch einige Kernprobleme solcher Protokolle beleuchtet. Aspekte dieses Abschnitts wurden auch in [IG11] publiziert. Anschließend wird die Spezifikationssprache *SDL* beschrieben, welche eine modellbasierte Protokollentwicklung ermöglicht und zur Umsetzung des in Kapitel 5 vorgestellten QoS-Routing- und Reservierungsprotokolls RBBQR genutzt wurde. Weiterhin werden die Prinzipien von Black Bursts sowie darauf basierende Transferprotokolle erläutert, die einerseits eine kollisionsgeschützte Kommunikation ermöglichen und andererseits zur Übertragung von Daten an Knoten außerhalb der Kommunikationsreichweite dienen. Diese Transferprotokolle werden von RBBQR genutzt.

3.1. Quality of Service in Ad-Hoc-Netzen

Im Folgenden wird ein kurzer Überblick über QoS in Ad-Hoc-Netzen gegeben (vgl. [IG11]). Da Ad-Hoc-Netze heutzutage immer komplexer werden und die Anforderungen an die Kommunikation immer vielfältiger, ist eine Best-Effort-Übertragung für viele Anwendungen nicht mehr ausreichend, sondern es muss eine bestimmte Dienstgüte gewährt werden. Dabei muss *garantiert* werden, dass der angeforderte Dienst in einer bestimmten Qualität erbracht wird, d. h. es werden Verkehrsverträge zwischen Diensterbringer und Dienstnutzer geschlossen. Dies wird als *Quality of Service* (QoS) bezeichnet. Beispiele für solche Anwendungen sind die regelmäßige Übertragung von Sensorwerten oder von Audio- und Videodaten. In einem Drahtlosmedium sind deterministische Garantien jedoch nur mit zusätzlichen Annahmen möglich, beispielsweise der *Single Network Property* (s. Abschnitt 2.2.1).

Um Verkehrsverträge schließen zu können, müssen die QoS-Anforderungen des Dienstnutzers hinreichend detailliert spezifiziert werden. Eine QoS-Anforderung unterteilt sich in QoS-Metrik und QoS-Wert. QoS-Metriken ermöglichen es, verschiedene Routen zwischen Diensterbringer (Quelle) und Dienstnutzer (Ziel) miteinander zu vergleichen. Solche Metriken können additiv, multiplikativ oder konkav sein. Dies lässt sich folgendermaßen formalisieren:

Definition 3.1 (*QoS-Metriken [Bec07]*)

Sei $d(n_i, n_j)$ der Metrikwert für einen Link (n_i, n_j) und $p = (n_1, n_2, \ldots, n_m)$ ein Pfad zwischen zwei Knoten n_1 und n_m. Dann sind die Metriktypen folgendermaßen definiert:

$$\text{Additiv:} \quad d(p) =_{Df} d(n_1, n_2) + d(n_2, n_3) + \ldots + d(n_{m-1}, n_m)$$

$$\text{Multiplikativ:} \quad d(p) =_{Df} d(n_1, n_2) \cdot d(n_2, n_3) \cdot \ldots \cdot d(n_{m-1}, n_m)$$

$$\text{Konkav:} \quad d(p) =_{Df} \min(d(n_1, n_2), d(n_2, n_3), \ldots, d(n_{m-1}, n_m))$$

Am häufigsten wird die konkave Metrik *Bandbreite* verwendet, weshalb sich viele QoS-Routing-Protokolle auch auf diese beschränken. Weitere Metriken sind *Übertragungsverzögerung* (bzw. *Latenz*), *Hopanzahl* (beides additiv) oder *Verlustwahrscheinlichkeit* (multiplikativ). Auch die *Energie* kann betrachtet werden. Dabei handelt es sich um eine konkave Metrik, wenn jeder Knoten ein Mindestmaß an Energie übrig haben muss, um Bestandteil der Route zu werden, und um eine additive Metrik, wenn die insgesamt benötigte Energie betrachtet wird.

Abbildung 3.1 gibt einen grafischen Überblick über die QoS-Mechanismen, die zur Erfüllung von QoS-Anforderungen benötigt werden. Eine detailliertere Beschreibung ist in [CAH96] zu finden.

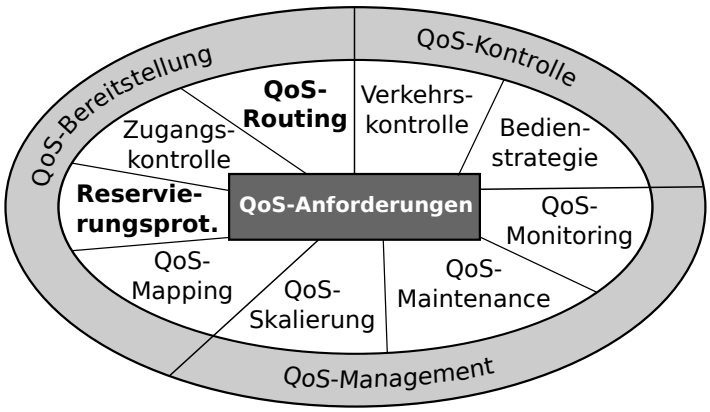

Abbildung 3.1.: QoS-Mechanismen zur Umsetzung von QoS-Anforderungen.

Der äußere Ring stellt die Kategorien dar, der innere die konkreten Mechanismen. Zur *QoS-Bereitstellung* gehören Maßnahmen zur Umsetzung von QoS-Anforderungen in eine geeignete Ressourcensituation (Reservierungsaspekt). Dazu zählt beispielsweise das QoS-Mapping, d. h. die Abbildung von QoS-Anforderungen zwischen den Systemschichten, sowie die Zugangskontrolle, die u. a. überprüft, ob genügend Ressourcen zur Verfügung stehen. Auch das QoS-Routing sowie das Reservierungsprotokoll, welche im Folgenden näher betrachtet werden, gehören zur QoS-Bereitstellung. Die *QoS-Kontrolle* umfasst kurzfristige, das *QoS-Management* hingegen langfristige Maßnahmen, um die vereinbarte Dienstgüte einzuhalten (Benutzungsaspekt). Zur QoS-Kontrolle gehören die Verkehrskontrolle, welche dafür sorgt, dass der Verkehrsvertrag eingehalten wird (z. B. Verwerfen von Paketen, wenn eine Anwendung sich nicht an den Verkehrsvertrag hält), sowie die Bedienstrategie, welche dafür verantwortlich ist, die Aufträge vor Abarbeitung durch eine Bedienstation (z. B. Transceiver) zu priorisieren. Zum QoS-Management gehören QoS-Monitoring, d. h. die Erfassung und Kontrolle des aktuellen Netzzustands, QoS-Maintenance, d. h. das Ausführen von Tuningmaßnahmen auf der Basis des Netzzustands, sowie QoS-Skalierung, d. h. die Manipulation von Paketfolgen zur Anpassung an verschiedene Endgeräte und die Anpassung des Senders an Dienstgüteschwankungen (z. B. Verringerung der Übertragungsrate bei schlechterer Verbindung).

3.1.1. QoS-Routing

Das *QoS-Routing* ist Bestandteil der QoS-Bereitstellung. Da in Multi-Hop-Netzen nicht alle Knoten direkt miteinander kommunizieren können, muss für eine Übertragung mit Hilfe eines Routing-Protokolls ein geeigneter Kommunikationspfad gesucht werden. Best-Effort-Routing berücksichtigt keine Restriktionen bei der Routensuche. Beim QoS-Routing dagegen wird versucht, eine Route zu finden, welche eine oder mehrere QoS-Anforderungen erfüllt (beispielsweise eine Mindestbandbreite oder eine maximale Übertragungsverzögerung).

Viele existierende QoS-Routing-Protokolle haben zwei Kernprobleme (vgl. [BBCG11]). Das erste Problem besteht darin, dass sich mehrere Routenanforderungen gleichzeitig in Bearbeitung befinden können (dies ist z. B. bei [LTS02] der Fall). Oft werden bei der Routensuche die benötigten Netzwerkressourcen der einzelnen Knoten bereits vorreserviert, da diese sonst später möglicherweise nicht mehr zur Verfügung stehen. Dies kann jedoch dazu führen, dass sich Routenanforderungen gegenseitig blockieren, obwohl insgesamt für mindestens eine davon eine Route existiert, welche die QoS-Anforderung erfüllt (vgl. SSNR-Problem in Abschnitt 3.1.2). Werden konkrete Zeitslots reserviert, so können durch parallele Routensuchen Fehlreservierungen (d. h. Reservierungen bereits blockierter Slots) auftreten. Dies kann einerseits dadurch erfolgen, dass die durch Vorreservierungen verursachten Blockierungen nicht an die Knoten in Interferenzreichweite propagiert werden, andererseits aber auch dadurch, dass Ressourcen nach Abschluss einer Routensuche zwar bereits reserviert sind, aber die Interferenznachbarn noch keine Statusaktualisierung erhalten haben (z. B. weil dafür nur bestimmte Kontrollslots verwendet werden können). So könnte z. B. ein Knoten a einen Slot zum Senden reservieren und ein Knoten $b \in IN_1(a)$ denselben Slot bei einer anderen Routensuche zum Empfangen reservieren. Dies kann beim Datenversand zu Kollisionen führen, wenn beide Reservierungen genutzt werden. Solche Fälle sind ähnlich zu dem in Abschnitt 3.1.2 beschriebenen Selbstinterferenzproblem; bei letzterem ist jedoch nur eine Route betroffen. Werden Slots bei einer Routensuche nicht vorreserviert (bzw. werden Vorreservierungen für parallele Routensuchen nicht berücksichtigt), so kann derselbe Slot in einem Knoten mehrfach vergeben werden. Wird dies bei der Etablierung einer Route nicht geprüft, so kommt dadurch ebenfalls eine Fehlreservierung zustande.

Das zweite Problem besteht darin, dass Kontrollpakete häufig mittels wettbewerbsbasiertem Medienzugriff versendet werden. Bei vielen Protokollen wird keine explizite Aussage über das Scheduling der Kontrollpakete getroffen, so dass man ebenfalls von wettbewerbsbasiertem Medienzugriff ausgehen muss (beispielsweise in [SCCC06]). In diesem Fall können Rahmenkollisionen auftreten. Durch unerkannte Kollisionen können Routensuchen scheitern, obwohl passende QoS-Routen existieren. Auch wenn Kollisionen behandelt werden, resultieren daraus indeterministische Verzögerungen. Reaktive Routingprotokolle fluten oft das ganze Netzwerk mit Routenanforderungen, was zu lokalen Broadcasts der einzelnen Knoten führt und die Wahrscheinlichkeit für Kollisionen erhöht. Aufgrund der Broadcasts ist zudem die Verwendung von Acknowledgements (ACKs) schwierig, denn da mehrere Knoten ein Paket empfangen können, müssen geeignete Maßnahmen getroffen werden, um eine Kollision der ACKs beim Sender des Pakets zu verhindern.

Neben diesen beiden Kernproblemen existieren noch weitere Probleme für QoS-Routing. So ist beispielsweise die Kombination mehrerer QoS-Anforderungen schwierig zu realisieren, da die Suche nach einer QoS-Route, welche zwei unabhängige Anforderungen erfüllt, NP-vollständig ist [CN98]. Ein weiteres Problem besteht darin, dass die Erfüllung von QoS-Anforderungen häufig nicht lokal entscheidbar ist, da die Metrikwerte der einzelnen Knoten voneinander abhängen. Wird beispielsweise eine maximale Übertragungsverzögerung gefor-

dert, so entscheidet die Summe der Verzögerungswerte der einzelnen Knoten darüber, ob die gefundene Route die Anforderung erfüllt. Dies führt dazu, dass weitere Nachrichten ausgetauscht oder die benötigten Informationen mit in die Routing-Pakete aufgenommen werden müssen. Allenfalls für konkave Metriken ist eine lokale Entscheidung möglich, beispielsweise wenn man die statistische Reservierung von Bandbreite betrachtet (s. Abschnitt 3.1.2). Doch auch für solche Metriken werden häufig Werte der benachbarten Knoten benötigt, z. B. wenn eine deterministische Reservierung von Bandbreite in Form von Zeitslots vorgenommen werden soll (da die verfügbaren Zeitslots von den Reservierungen anderer Knoten abhängen).

3.1.2. Reservierungsprotokoll

Das *Reservierungsprotokoll* ist wie das QoS-Routing Bestandteil der QoS-Bereitstellung. Seine Aufgabe ist die Koordinierung der verteilten Reservierung von Ressourcen, um die kollisionsfreie Übertragung von Daten sicherzustellen. Es kann eigenständig sein oder in das QoS-Routing-Protokoll integriert werden. Im Fall eines verteilten QoS-Routing-Protokolls ist das Reservierungsprotokoll häufig in das Routingprotokoll integriert, da die Reservierungsentscheidungen eines Knotens von denen anderer Knoten der ermittelten Route abhängen können.

Es gibt zwei Ausprägungen: statistische und deterministische Reservierungen. *Statistische Reservierungen* sind nicht exklusiv, d. h. es gibt keine Garantie, dass die QoS-Anforderungen eingehalten werden. Im Fall von Bandbreite bedeutet dies, dass jeder Knoten diese in Form eines abstrakten Zahlenwerts verwaltet und eine neue QoS-Route durch den entsprechenden Knoten zugelassen wird, solange noch Bandbreite zur Verfügung steht. Die Auswirkungen von Reservierungen auf die Bandbreite der Nachbarknoten werden dabei oft außer Acht gelassen. Aber auch wenn die Bandbreite der Nachbarknoten jeweils angepasst wird, so garantiert dies keine Kollisionsfreiheit (da die Sendezeitpunkte nicht eingeschränkt werden). Die Reservierungen werden im Mittel funktionieren, solange das Netz nicht zu stark ausgelastet ist. Die verwendete MAC-Schicht ist nicht festgelegt, d. h. es kann beispielsweise eine wettbewerbsbasierte MAC-Schicht, z. B. auf Basis von CSMA/CA (Carrier Sense Multiple Access mit Collision Avoidance), CDMA (Code Division Multiple Access) oder auch TDMA (Time Division Multiple Access) verwendet werden. Genaueres zu den einzelnen Verfahren findet sich in [Sch03].

Im Gegensatz dazu sind *deterministische Reservierungen* exklusiv, d. h. unter den Annahmen, dass die Single Network Property erfüllt ist und keine Linkbrüche auftreten, ist garantiert, dass die QoS-Anforderungen eingehalten werden. Aufgrund der Charakteristiken von drahtlosen Netzen (z. B. Störanfälligkeit, Knotenmobilität) ist es ohne solche Annahmen nicht möglich, harte QoS-Garantien zu geben. Im Fall von Bandbreite können beispielsweise konkrete Zeitslots der MAC-Schicht exklusiv reserviert werden, so dass kollisionsfreie Übertragungen stattfinden können.

Als Voraussetzung für deterministische Reservierungen von Bandbreite werden einige Basisfunktionalitäten benötigt. So ist es i. d. R. nicht möglich, eine wettbewerbsbasierte Mediumarbitrierung zu verwenden. Da die Verwendung von FDMA (Frequency Division Multiple Access) aufgrund der hohen Hardwareanforderungen oft nicht sinnvoll ist, kommt vor allem TDMA als Arbitrierungsverfahren auf MAC-Ebene in Frage. TDMA setzt jedoch eine globale Zeitsynchronisation mit maximalen Uhrenabweichungen voraus, da Kollisionen auftreten können, wenn das Zeitverständnis der einzelnen Knoten zu stark divergiert. In diesem Fall könnten Übertragungen in benachbarten Slots kollidieren, obwohl jeder Knoten in seinem reservierten Slot sendet. Ein geeignetes Verfahren mit beschränkter Konvergenzzeit ist BBS [GK11].

Zusätzlich zu TDMA wird i. d. R. SDMA (Spatial Division Multiple Access) verwendet, d. h. Zeitslots werden mehrfach genutzt, sofern die entsprechenden Knoten sich nicht in Interferenzreichweite befinden.

Um TDMA und SDMA zu realisieren, wird ein Modell des Netzwerks benötigt, wobei nicht jeder Knoten die gesamte Topologie kennen muss. Ein Knoten muss aber zumindest Informationen über seine Nachbarn und ggf. über weitere Knoten erhalten, um entscheiden zu können, ob ein Slot reserviert werden kann. Dabei ist auch zu beachten, dass zwischen Kommunikations- und Interferenzreichweite unterschieden werden muss, wobei die Interferenzreichweite i. d. R. größer ist als die Kommunikationsreichweite. Ein Modell für kollisionsfreie Reservierungen unter Verwendung von TDMA und SDMA wurde in Kapitel 2 betrachtet.

Viele TDMA-basierte QoS-Routing- und Reservierungsprotokolle berücksichtigen zwar, dass die Sendevorgänge verschiedener Knoten sich gegenseitig beeinflussen; jedoch wird oft vernachlässigt, dass die Interferenzreichweite größer sein kann als die Kommunikationsreichweite (d. h. es wird die 1-Hop-Interferenzannahme $\forall\, a \in V : IN_1(a) = CN_1(a)$ getroffen). Dies findet sich beispielsweise in den Protokollen [ZC02, LTS02, SCCC06].

Bei der Reservierung von Zeitslots unter Verwendung von SDMA sind das *Hidden-* und das *Exposed-Station-Problem* zu beachten. Diese Probleme sind vor allem im Kontext wettbewerbsbasierter Datenübertragungen bekannt; aufgrund der Verwendung von SDMA müssen sie jedoch auch bei wettbewerbsfreien Übertragungen berücksichtigt werden. Das Hidden-Station-Problem ist in Abbildung 3.2 dargestellt.

Abbildung 3.2.: Veranschaulichung des Hidden-Station-Problems.

Ein Sendevorgang eines Knotens a an einen Knoten b wird durch einen gleichzeitigen Sendevorgang eines Knotens c, welcher sich in Interferenzreichweite zu b befindet, gestört. Da a und c sich nicht in Wahrnehmungsreichweite befinden, kann c die Übertragung von a nicht per Clear Channel Assessment (CCA) detektieren.

Das Exposed-Station-Problem ist in Abbildung 3.3 dargestellt.

Abbildung 3.3.: Veranschaulichung des Exposed-Station-Problems.

Wenn Knoten a an b sendet, so kann c gleichzeitig an d senden, da sich a und d nicht in Interferenzreichweite befinden. Wird dieser Sendevorgang jedoch verhindert, da c sich in Wahrnehmungsreichweite zu a befindet und somit dessen Übertragung per CCA detektiert, so führt dies zu Ineffizienz und schlechter Netzauslastung.

Zur Vermeidung des Hidden-Station-Problems verwenden einige Protokolle CDMA zusätzlich zu TDMA (beispielsweise [LL99, Lin01, CTSK04]). Dabei werden unterschiedliche Codes für Datenübertragungen verwendet, die sich sonst gegenseitig stören würden. Hierfür werden jedoch entsprechende Codes sowie ein Algorithmus zu deren Verteilung benötigt, was die Realisierung aufwendig macht.

Anmerkung: Die in den Abschnitten 2.3, 2.4 und 2.5 beschriebenen Reservierungsbedingungen berücksichtigen das Hidden- sowie das Exposed-Station-Problem implizit. Das Hidden-Station-Problem wird eliminiert, da ein Sendevorgang nur dann stattfinden kann, wenn kein

Interferenznachbar im selben Zeitslot empfängt. Das Exposed-Station-Problem wird beachtet, da ein Knoten unabhängig von den Sendereservierungen seiner Wahrnehmungsnachbarn senden kann. Entscheidend ist nur, dass keine vorhandenen Übertragungen gestört werden, und dass kein Interferenznachbar des Empfängers im gleichen Slot sendet.

Ein weiteres Problem, welches ein Reservierungsprotokoll lösen muss, ist die Verteilung der zu reservierenden Zeitslots in den einzelnen Knoten. Es muss nicht nur eine Route gefunden werden, die die QoS-Anforderung(en) erfüllt, sondern auch ermittelt werden, welche der freien Slots jeweils reserviert werden sollen, um an den nächsten Knoten der Route zu senden. Ziel ist es, die vorhandene Bandbreite möglichst gut auszunutzen. In [Zhu02] wurde gezeigt, dass die Bestimmung der maximalen Bandbreite (gemessen in Slots) einer Route NP-vollständig ist. Somit kommen in der Praxis Heuristiken zum Einsatz.

Dies führt zu zwei weiteren Problemen, welche in [SCCC06] beschrieben werden, nämlich das *Slot Shortage for Self Route* (SSSR)- und das *Slot Shortage for Neighboring Routes* (SSNR)-Problem. Das SSSR-Problem besteht darin, dass für eine Route, welche eigentlich die QoS-Anforderung erfüllt, keine passende Slotverteilung gefunden werden kann, weil am Anfang der Route die „falschen“ Slots gewählt wurden. Abbildung 3.4 zeigt ein Beispiel. Über den Links sind die jeweils freien Slots angegeben, welche prinzipiell zum Senden auf dem jeweiligen Link verwendet werden können.

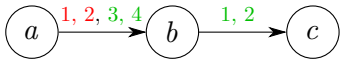

Abbildung 3.4.: Veranschaulichung des SSSR-Problems.

Angenommen, es soll eine Route von a zu c mit einer QoS-Anforderung von zwei Slots etabliert werden. Die Wahl der Slots 1 und 2 für den Link (a, b) führt dazu, dass auf dem Link (b, c) keine Slots mehr zur Verfügung stehen. Würden jedoch die Slots 3 und 4 für (a, b) gewählt, so könnten 1 und 2 für (b, c) verwendet werden.

Das SSNR-Problem besteht darin, dass für eine Route keine passende Slotverteilung gefunden werden kann, obwohl dies möglich wäre, wenn für eine andere Route die Slotwahl modifiziert würde. Ein Beispiel ist in Abbildung 3.5 zu finden.

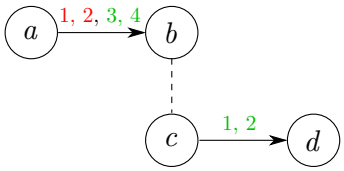

Abbildung 3.5.: Veranschaulichung des SSNR-Problems.

Wenn zuerst eine QoS-Route zwischen a und b etabliert wird (mit einer QoS-Anforderung von zwei Slots), für die die Slots 1 und 2 verwendet werden, so stehen für die QoS-Route zwischen c und d keine Slots mehr zur Verfügung, da c sich in Interferenzreichweite zu b befindet und somit in den Slots 1 und 2 nicht senden kann. Würden jedoch die Slots 3 und 4 für (a, b) verwendet, so könnten 1 und 2 für (c, d) verwendet werden.

Bei verteilten Routensuchen und Reservierungen ergibt sich ein weiteres Problem. Normalerweise wird zunächst versucht, eine Route zwischen Quelle und Ziel mit einer ausreichenden Anzahl von Slots für jeden Link zu finden, bevor die Slots reserviert und die Reservierun-

gen an die Nachbarknoten propagiert werden. Hierbei kann das *Selbstinterferenzproblem* auftreten, welches zu Fehlreservierungen führen kann. Dabei handelt es sich um eine Variante des Hidden-Station-Problems, die dadurch zustande kommt, dass benötigte Reservierungsinformationen noch nicht propagiert wurden, wenn weitere Reservierungen getroffen werden. Geht man allgemein von einer Interferenzreichweite von d Kommunikationshops aus (d. h. $\forall\, a \in V\colon IN_1(a) = CN_{\leq d}(a) \setminus \{a\}$), so besteht das Selbstinterferenzproblem darin, dass Knoten, die sich im Netzwerk in Interferenznachbarschaft befinden, auf der Route durch mehr als d Hops getrennt sind. Dieses Problem kann natürlich dadurch auftreten, dass die Interferenzannahme nicht erfüllt ist. Ist sie erfüllt, so kann das Problem trotzdem vorkommen. In diesem Fall existiert stets eine kürzere Route, welche jedoch nicht gewählt wurde (z. B. weil sie die QoS-Anforderung nicht erfüllen kann). Beide Fälle werden bei der Entwicklung des Protokolls RBBQR näher betrachtet (vgl. Abschnitt 5.2.3.2).

Existierende Protokolle gehen üblicherweise von der 1-Hop-Interferenzannahme aus, und es wird vorausgesetzt, dass diese Annahme immer erfüllt ist. Damit kann das Selbstinterferenzproblem bei solchen Protokollen nur dadurch entstehen, dass Knoten, die innerhalb des Netzes Kommunikationsnachbarn sind, auf der Route einen größeren Abstand haben. Abbildung 3.6 zeigt ein Beispiel.

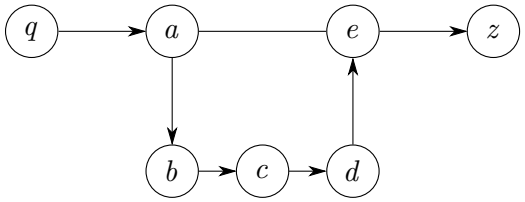

Abbildung 3.6.: Route mit Selbstinterferenzproblem.

Um das Hidden-Station-Problem zu vermeiden, kann ein Slot unter der getroffenen Interferenzannahme auf jeweils drei aufeinanderfolgenden Links einer Route nur einmal verwendet werden. Es wird hier angenommen, dass jeder Knoten sein Route Request per Broadcast versendet. Im Folgenden werden die Probleme betrachtet, die bei Knoten e auftreten können. Zunächst erhält e ein Route-Request-Paket von a. Kann für die (partielle) Route $q \to a \to e$ die QoS-Anforderung erfüllt werden, so wird diese Route gewählt. Ist dies jedoch nicht möglich (z. B. aufgrund der vorherigen Slotauswahl), so sendet d zu einem späteren Zeitpunkt ein Route-Request-Paket, sofern die Route bis zu d fortgesetzt werden kann. Ist eine Fortsetzung der Route zu e möglich, so könnten für (e, z) Slots gewählt werden, die bereits für (q, a) verwendet wurden, da (q, a) und (e, z) sich nicht in einem Umkreis von drei Links zueinander befinden. In diesem Fall liegt eine *Fehlreservierung bei den Sendeslots von e* vor (d. h. für e werden Sendeslots gewählt, die bereits zum Senden blockiert sind), die beim Datenversand zu kollidierenden Paketen führen kann (wenn beide Reservierungen genutzt werden). Außerdem ist es möglich, dass für (d, e) Slots gewählt werden, die bereits auf (a, b) verwendet wurden (*Fehlreservierung bei den Empfangsslots von e*, d. h. für e werden Empfangsslots gewählt, die bereits zum Empfangen blockiert sind). Auch dies kann beim Datenversand zu Problemen führen.

Durch das Selbstinterferenzproblem auftretende Fehlreservierungen werden von existierenden Protokollen oft nicht berücksichtigt (z. B. [ZC02]).

Anmerkung: Eine Fehlreservierung bei den Sendeslots von e entspricht einer Fehlreservierung bei den Empfangsslots von a. Dennoch ist es sinnvoll, zwischen Fehlreservierungen bei Sende- und Empfangsslots zu unterscheiden. Eine mögliche Lösung des Selbstinterferenz-

problems besteht darin, die Slots beim Reservieren jeweils sofort an die Interferenznachbarn zu propagieren. In diesem Fall werden Fehlreservierungen immer beim zweiten Knoten, der die Slots reservieren will, festgestellt. Bei diesem müssen dann die Sende- und / oder Empfangsslots geändert werden (werden die Empfangsslots geändert, so müssen dem Vorgänger die geänderten Sendeslots mitgeteilt werden).

Anmerkung: Im Gegensatz zum eigentlichen Hidden-Station-Problem, welches durch das in Kapitel 2 beschriebene Interferenzmodell berücksichtigt wird, muss das Selbstinterferenzproblem gesondert betrachtet werden. Da dieses durch das Fehlen von Reservierungsinformation verursacht wird, kann es nicht allgemein berücksichtigt werden.

3.2. Modellbasierte Protokollentwicklung mit SDL

Die von der ITU-T (International Telecommunication Union – Telecommunication Standardization Sector) standardisierte *Specification and Description Language* (SDL) [Int12a] ist eine formale Spezifikationssprache, welche sich insbesondere für verteilte und reaktive Systeme sowie für Kommunikationsprotokolle eignet. Umfassende Informationen zu SDL sind in [EHS97] zu finden. SDL existiert sowohl in einer graphischen Version (SDL-GR), welche für Menschen gut lesbar ist, als auch in einer textuellen (SDL-PR), welche besser maschinenverarbeitbar ist.

Das Verhalten eines Systems wird in SDL mittels erweiterter endlicher Automaten beschrieben, welche durch gerichtete Kanäle verbunden sind und durch den Austausch von Signalen asynchron kommunizieren. Da neben geschlossenen auch offene Systeme möglich sind, werden Kanäle ebenso verwendet, um ein System mit seiner Umgebung zu verbinden.

Im Folgenden werden diese Konzepte anhand des *DemonGame*-Systems dargestellt. Dabei handelt es sich um ein in der SDL-Community weit verbreitetes Beispiel, welches auch im SDL-Standard Verwendung findet (vgl. [Int12b]). Die oberste Strukturierungsebene von SDL ist das *System*. Im gewählten Beispiel ist dies das in Abbildung 3.7 dargestellte System *DemonGame*.

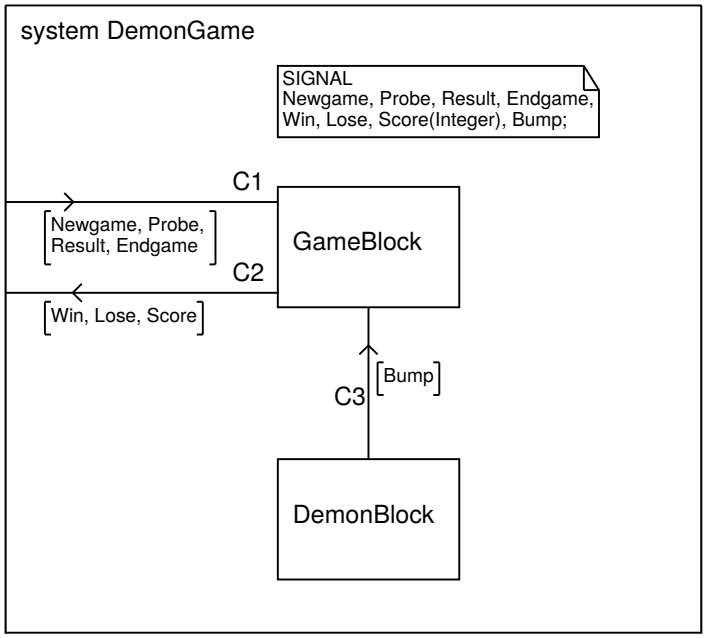

Abbildung 3.7.: Das *DemonGame*-System [Int12b].

Zur Strukturierung eines Systems dienen *Blöcke* (im Beispiel *GameBlock* und *DemonBlock*). Die Deklaration eines Blocks beschreibt eine Menge von Instanzen. Die Instanzmengen von Blöcken enthalten i. d. R. jeweils nur eine Instanz; es ist jedoch auch möglich, eine größere Anzahl zu verwenden. Dies ist analog zu den Instanzmengen von Prozessen (s. u.), wo wesentlich häufiger eine größere Anzahl als 1 verwendet wird.

Blöcke können durch *Kanäle* (*C1*, *C2*, *C3*) miteinander und mit der Umgebung des Systems verbunden werden. Es existieren sowohl verzögernde (Pfeilspitzen zur Mitte hin verschoben) als auch nicht-verzögernde Kanäle (Pfeilspitzen an den Enden). Kanäle sind stets zuverlässig und reihenfolgeerhaltend. Darüber hinaus sind sie gerichtet und typisiert; sie können uni- oder bidirektional sein. Die möglichen Signale für die spezifizierten Übertragungsrichtungen werden an den Kanälen angegeben (z. B. *Newgame*, *Probe*, ...). Weiterhin können Parameter für Signale definiert werden (z. B. hat *Score* einen Parameter vom Typ *Integer*). Anstatt die Signale direkt an den Kanälen anzugeben, können auch Signallisten deklariert werden. In diesem Fall wird der Name einer Signalliste in runden Klammern am jeweiligen Kanal angegeben.

Blöcke können entweder in weitere Blöcke oder in *Prozesse* unterteilt sein. Abbildung 3.8 zeigt den Aufbau des Blocks *DemonBlock*.

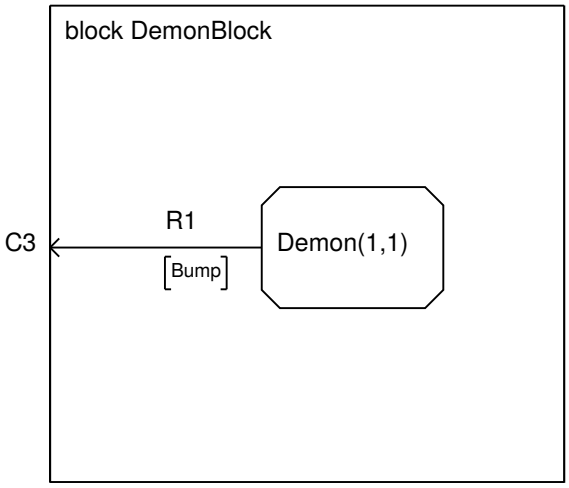

Abbildung 3.8.: Der *DemonBlock* des *DemonGame*-Systems [Int12b].

Die Deklaration eines Prozesses beschreibt eine Menge von Instanzen; die Zahlen in Klammern geben die initiale und maximale Anzahl von Prozessinstanzen in der entsprechenden Menge an. Im Beispiel gibt es sowohl initial als auch maximal genau einen *Demon*-Prozess. Beide Zahlen können (unabhängig voneinander) weggelassen werden; die Standardwerte sind 1 für die initiale und ∞ für die maximale Anzahl. Während es sich bei Systemen und Blöcken um passive (zur Strukturierung verwendete) Komponenten handelt, beschreiben Prozesse das Verhalten der entsprechenden Systemkomponenten und sind somit aktiv. Alle Prozesse sind nebenläufig und asynchron.

Die Verbindung zwischen Prozessen oder von Prozessen zu Blockgrenzen erfolgt durch nicht-verzögernde Kanäle, welche *Signalrouten* genannt werden (im Beispiel *R1*). Signalrouten, die zur Blockgrenze führen, müssen an einen Kanal angeschlossen werden, der dieselben Signale in der entsprechenden Richtung überträgt (im Beispiel wird *R1* an *C3* angeschlossen).

Die Kommunikation zwischen Prozessen erfolgt asynchron durch das Senden von Signalen. Zur Verarbeitung von empfangenen Signalen hat jeder Prozess eine Eingangswarteschlange,

die in FIFO-Reihenfolge abgearbeitet wird. Ein von einem Prozess P1 gesendetes Signal kann nur dann an einen Prozess P2 zugestellt werden, wenn es einen passend typisierten Pfad aus Kanälen und Signalrouten von P1 zu P2 gibt. Gibt es mehrere mögliche Empfänger, so wird einer davon zufällig ausgewählt. Zur expliziten Adressierung eines Prozesses kann dessen eindeutiger Process Identifier (PId) verwendet werden, welcher vom Laufzeitsystem vergeben wird. Daneben gibt es noch die Möglichkeit, eine implizite Adressierung zu verwenden, indem man ein Gate (s. u.), einen Kanal oder eine Signalroute bzw. den Namen einer Prozessinstanzmenge angibt. Damit ist es möglich, die Menge potenzieller Empfänger einzuschränken. Einen bestimmten Prozess kann man auf diese Weise jedoch nicht adressieren.

Das durch einen Prozess spezifizierte Verhalten wird mittels eines erweiterten endlichen Automaten beschrieben. Abbildung 3.9 zeigt den Aufbau des Prozesses *Demon*.

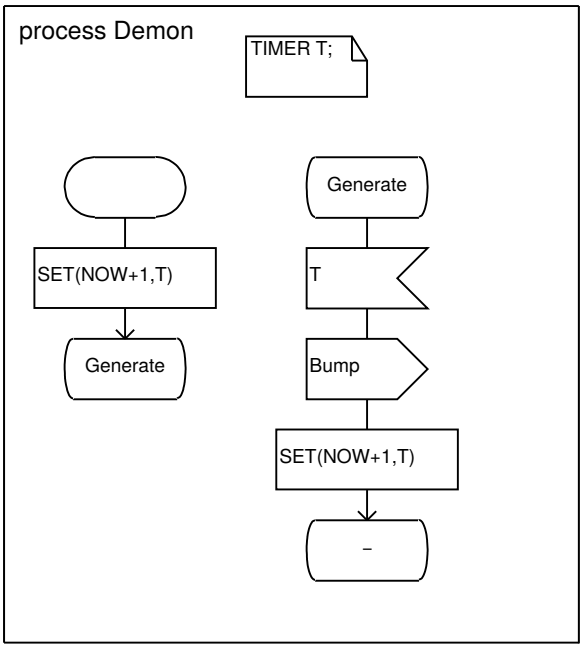

Abbildung 3.9.: Der *Demon*-Prozess des *DemonGame*-Systems [Int12b].

Transitionen werden durch Signale ausgelöst, welche ein Prozess in einem bestimmten Zustand empfängt. Ablaufende Timer erzeugen ebenfalls Signale, deren Typ dem Namen des Timers entspricht. Sie werden analog zu regulären Signalen in die Eingangswarteschlange des zugehörigen Prozesses eingereiht und anschließend verarbeitet. Im Beispiel erzeugt der Timer T beim Ablaufen ein Signal T, welches vom Prozess *Demon* konsumiert wird. Befindet sich an der ersten Position der Eingangswarteschlange ein Signal, welches in einem Zustand nicht explizit empfangen wird, so wird dieses implizit konsumiert und das nächste Eingabesignal verarbeitet. Innerhalb einer Transition können beispielsweise Ausgabesignale erzeugt, Timer gestellt oder Variablen gesetzt werden. Auch Verzweigungen anhand von Entscheidungen sind möglich. Nach Ausführung der Transition geht der Prozess in einen Folgezustand über. Das Symbol "–" bedeutet, dass dieser identisch mit dem Ausgangszustand ist.

Zur besseren Strukturierung von Prozessen ist es möglich, diese in *Services* zu zerlegen. Services implementieren jeweils einen eigenen Zustandsautomaten, teilen sich jedoch die Eingangswarteschlange des umgebenden Prozesses und können auch auf dessen Variablen zugreifen. Um Mehrdeutigkeiten zu vermeiden, müssen die von den einzelnen Services verarbeiteten

Eingabesignale disjunkt sein.

Weiterhin ist es möglich, in sich abgeschlossene Teile von Prozessen in *Prozeduren* auszulagern. Dies ist nützlich, wenn derselbe Teil einer Prozessspezifikation mehrfach benötigt wird oder wenn bestimmte Details aus Gründen der Übersichtlichkeit verborgen werden sollen. Eine Prozedur kann auf Eingangswarteschlange und Variablen des umgebenden Prozesses zugreifen und stellt einen Zustandsautomaten innerhalb eines Prozesses dar. Wenn eine Prozedur beginnt, wird der Aufrufer unterbrochen und erst nach deren Abschluss weiter ausgeführt.

Anstatt Blöcke, Prozesse und Services direkt zu instanziieren, ist es auch möglich, Block-, Prozess- und Servicetypen zu deklarieren, welche dann an anderer Stelle instanziiert werden können. Dies erleichtert die Strukturierung und verbessert die Wiederverwendbarkeit von Systemen. Um solche Typen verbinden zu können, werden bei der Deklaration sogenannte *Gates* verwendet. Dabei handelt es sich um typisierte Verbindungspunkte, welche die Schnittstellen des entsprechenden Block-, Prozess- oder Servicetyps zur Umgebung beschreiben. Bei der Instanziierung werden die Gates dann mit Kanälen bzw. Signalrouten des umgebenden Kontextes verbunden.

Für Systeme, die Echtzeiteigenschaften garantieren müssen, ist SDL nur bedingt geeignet. Beispielsweise ist die prozessübergreifende Priorisierung bestimmter Abläufe im Standard nicht vorgesehen. Auch ermöglichen Timer es nur, eine minimale Verzögerung zu spezifizieren, jedoch keine maximale (da nicht festgelegt werden kann, wann das beim Ablaufen des Timers erzeugte Signal spätestens verarbeitet sein muss). Zudem muss der durch die Laufzeitumgebung verursachte Overhead berücksichtigt werden. Es wurden jedoch einige Forschungen dazu durchgeführt, wie die Echtzeitfähigkeit von SDL verbessert werden kann. Die Ergebnisse sind u. a. in [Chr15] und [BCGI12] zu finden.

3.3. Black-Burst-basierte Übertragungen

Für das in Kapitel 5 vorgestellte Protokoll RBBQR werden Übertragungsverfahren benötigt, welche einerseits der Übermittlung von Informationen an die Interferenznachbarn eines Knotens dienen und andererseits zur kollisionsgeschützten Arbitrierung zwischen mehreren Sendern bzw. zur kollisionsgeschützten Übertragung eines Werts im Falle mehrerer Sender verwendet werden können. Unter der in Kapitel 2 getroffenen 2-Hop-Interferenzannahme (vgl. Abschnitt 2.4.2) ist ersteres zwar mit zwei regulären Übertragungen möglich. Allerdings ergibt sich dabei das Problem, dass für einen sendenden Knoten a der Fall $|CN_1(a)| > 1$ eintreten kann, wodurch bei den Knoten aus $CN_2(a)$ Kollisionen auftreten könnten, wenn mehrere Knoten aus $CN_1(a)$ gleichzeitig weiterleiten.

Zur Lösung dieser Probleme existieren spezielle Übertragungsverfahren, welche bereits in zahlreichen Arbeiten beschrieben wurden (s. z. B. [CGK09, Chr10, BBCG11, CGR12, Chr15]). Diese Verfahren basieren auf dem Konzept von *Black Bursts*; einer Kommunikationsprimitive zur kollisionsgeschützten Übertragung einzelner Bits. In den folgenden Abschnitten werden Black Bursts sowie die darauf aufbauenden Übertragungsverfahren beschrieben. Dabei handelt es sich im Wesentlichen um die Essenz der Beschreibungen in den genannten Arbeiten. Neu ist hierbei jedoch das Konzept des *Bit-based Transfers*, welches im Rahmen dieser Arbeit für RBBQR entwickelt wurde (s. Abschnitt 3.3.2.4).

3.3.1. Black Bursts

Bei Black Bursts handelt es sich um Perioden von Energie auf dem Medium, welche eine definierte Dauer haben und jeweils zu festgelegten Zeitpunkten gesendet werden müssen. Ein

Black Burst enthält keine Information im Sinne einer Payload, jedoch werden Informationen durch seinen Anfangszeitpunkt sowie die Dauer der Medienbelegung codiert. Zur Codierung einer binären 0 wird ein Black Burst mit Dauer 0 s verwendet, d. h. es findet keine Übertragung statt. Eine 1 wird dagegen durch einen Black Burst mit Dauer > 0 s codiert, wodurch die 1 dominant ist (d. h. wenn gleichzeitig 1 und 0 gesendet wird, empfangen alle Knoten in Reichweite beider Sender eine 1). Ferner impliziert dies, dass ein Knoten, der eine 0 „überträgt“, nicht senden muss. Er kann somit das Medium abhören und feststellen, ob ein anderer Knoten eine 1 sendet.

Die 1 wird als MAC-Rahmen mit minimaler Byteanzahl codiert, was mit handelsüblichen Transceivern realisiert werden kann (um einen Black Burst von einem regulären Rahmen unterscheiden zu können, wird vorausgesetzt, dass er kürzer ist als der kürzeste reguläre Rahmen). Somit ist keine spezielle Hardware erforderlich; allerdings ist die Performanz von der minimalen Byteanzahl eines MAC-Rahmens sowie von Datenrate und Umschaltzeiten des gewählten Transceivers abhängig. Die Erkennung von Black Bursts erfolgt mittels CCA. Diesen Mechanismus stellen fast alle drahtlosen Transceiver mit CSMA/CA-Unterstützung bereit.

Wenn mehr als ein Knoten eine 1 sendet, ändern sich Startzeit und Dauer auf dem Medium nur geringfügig, da die Sendevorgänge zu definierten Zeitpunkten stattfinden (es wird davon ausgegangen, dass die Uhrenabweichungen durch eine geeignete Synchronisation – beispielsweise BBS [GK11] – begrenzt werden). Da die Inhalte der gesendeten Rahmen nicht ausgewertet werden, sondern für den korrekten Empfang einer 1 nur Startzeit und Dauer auf dem Medium relevant sind, können Empfänger in Reichweite mehrerer Knoten die überlappenden Black-Burst-Übertragungen immer noch korrekt erkennen. Aus diesem Grund sind Kollisionen von Black Bursts zerstörungsfrei.

Zur Übertragung einer Bitsequenz beliebiger Länge (auch Black-Burst-Rahmen genannt) wird diese bitweise mit Black Bursts codiert. Übertragen mehrere Knoten gleichzeitig unterschiedliche Black-Burst-Rahmen, so ergibt sich daraus auf dem Medium eine bitweise Veroderung, da die 1 dominant ist. Da sich Stationen, welche eine 0 senden, im Empfangsmodus befinden, ist anschließend allen Knoten in Reichweite aller Sender – inklusive der Sender selbst – die Veroderung der gesendeten Sequenzen bekannt.

Jeder Black-Burst-Rahmen beginnt mit einem dominanten 1-Bit, dem sogenannten Start-Of-Frame (SOF)-Bit. Dies ist erforderlich, da man ansonsten den Fall, dass gerade kein Rahmen gesendet wird, nicht von einem mit Nullen beginnenden Rahmen unterscheiden kann. Für das SOF-Bit wird ein Black Burst mit größerer Dauer verwendet, so dass dieses von anderen 1-Bits unterschieden werden kann. Es ist jedoch trotzdem kürzer als der kürzeste reguläre Rahmen. Damit kann ein Knoten stets feststellen, ob gerade ein Black Burst oder ein regulärer Rahmen empfangen wird.

Da Black Bursts mittels CCA empfangen werden, haben sie prinzipiell Wahrnehmungsreichweite. Da diese die Interferenzreichweite umfasst, können Informationen, die an die Interferenznachbarn eines Knotens übermittelt werden sollen (z. B. die von einem Knoten zum Senden bzw. Empfangen reservierten Slots) als Black-Burst-Rahmen gesendet werden. Werden diese von Knoten außerhalb der Interferenzreichweite empfangen, so kann dies dazu führen, dass die vorhandene Bandbreite schlechter genutzt wird, da Slots unnötig blockiert werden. Es können dadurch jedoch keine Kollisionen auftreten. Deshalb wird im Folgenden davon ausgegangen, dass die Interferenzreichweite der Wahrnehmungsreichweite entspricht, d. h. dass $\forall\, a \in V : IN_1(a) = SN_1(a)$ gilt. Es wird daher nur noch der Begriff „Interferenzreichweite“ verwendet.

Die Eigenschaft, dass mit einer Übertragung alle Interferenznachbarn eines Knotens erreicht

werden, kann auch unerwünscht sein. Deshalb wird in dieser Arbeit davon ausgegangen, dass die Reichweite von Black Bursts eingeschränkt werden kann, so dass genau die Kommunikationsnachbarn erreicht werden (z. B. durch Anpassung des CCA-Schwellwerts). Unter den getroffenen Annahmen haben Black Bursts somit entweder Kommunikations- oder Interferenzreichweite.

3.3.2. Transferprotokolle

Die im Folgenden beschriebenen Transferprotokolle basieren auf Black Bursts und wurden primär zur kollisionsgeschützten Arbitrierung bzw. zur kollisionsgeschützten Übertragung von Black-Burst-Rahmen über mehrere Hops entwickelt. Sie können jedoch auch verwendet werden, um Rahmen an Knoten außerhalb der Kommunikationsreichweite zu übermitteln (bei dem Protokoll RBBQR kommen beide Verwendungsmöglichkeiten zum Einsatz).

Die Verzögerung der Transferprotokolle hängt nicht von der Anzahl an Knoten ab, sondern nur von der Anzahl an Hops, über die ein Rahmen propagiert wird. Diese wird im Folgenden als n_{hops} bezeichnet. Soll der Rahmen durch das ganze Netz propagiert werden, so wird für n_{hops} eine obere Schranke für den Netzwerkdurchmesser verwendet, welche als $n_{maxHops}$ bezeichnet wird. Zur Ermittlung von $n_{maxHops}$ werden Kommunikationshops verwendet, da dies auch für die Interferenzhops eine obere Schranke darstellt. Wird der Rahmen nicht durch das gesamte Netz propagiert, so gelten einige Einschränkungen (vgl. Abschnitt 3.3.2.3).

Im Folgenden werden zunächst Arbitrating und Cooperative Transfer im Detail vorgestellt, wobei von $n_{hops} = n_{maxHops}$ ausgegangen wird. Der Arbitrating Transfer wird verwendet, um eine kollisionsgeschützte Arbitrierung zwischen mehreren potenziellen Sendern zu realisieren. Der Cooperative Transfer dient zur kollisionsgeschützten Übertragung gleicher Rahmen (und zur Übertragung von Rahmen an Knoten außerhalb der Kommunikationsreichweite).

Zur Illustration dienen ähnliche Beispiele wie in [BBCG11], wobei hier jedoch zwischen Kommunikations- und Interferenzlinks unterschieden wird. Die verwendete Topologie ist in Abbildung 3.10 dargestellt. Sie besteht aus fünf Knoten und hat einen maximalen Netzwerkdurchmesser von $n_{maxHops} = 3$ Kommunikationshops.

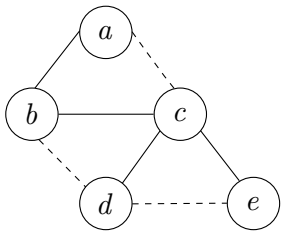

Abbildung 3.10.: Topologie zur Veranschaulichung der Transferprotokolle.

Neben Arbitrating und Cooperative Transfer wird ein weiteres, auf dem Cooperative Transfer basierendes Übertragungsverfahren definiert, der *Bit-based Transfer*. Dieser wird für das Protokoll RBBQR benötigt. Er kann zum Senden über einen Interferenzhop verwendet werden, wenn keine konkurrierenden Sender auftreten, und dient zur Übertragung von Rahmen mit vorher unbekannter Länge. Bei Arbitrating und Cooperative Transfer wird hingegen vorausgesetzt, dass die Längen der zu übertragenden Rahmen allen Knoten vorab bekannt sind.

3.3.2.1. Arbitrating Transfer

Der Arbitrating Transfer ermittelt aus verschiedenen gleichzeitig übertragenen Bitsequenzen einen Gewinner, dessen Sequenz anschließend im gesamten Netz bekannt ist (in diesem Abschnitt wird von $n_{hops} = n_{maxHops}$ ausgegangen). Dabei setzt sich immer die numerisch höchste Bitsequenz durch. Die Rahmen werden bitweise übertragen, wobei jedes Bit einzeln durch das gesamte Netz propagiert wird. Dabei werden rezessive Nullen durch dominante Einsen überschrieben, so dass Knoten, welche eine 0 senden, erkennen können, dass sie die Arbitrierung verloren haben. Sie leiten dann nur noch empfangene Bits weiter. Alternativ wäre auch eine Variante des Arbitrating Transfers denkbar, bei der Knoten weiter senden, auch wenn ihr gesendetes Bit überschrieben wurde. Auf diese Weise würden alle Knoten die logische Veroderung der gesendeten Bitsequenzen erhalten. In dieser Arbeit wird der Arbitrating Transfer jedoch nur zur Arbitrierung verwendet.

Im Folgenden bezeichnet eine *Burstrunde* des Arbitrating Transfers die Übertragung eines einzelnen Bits über einen Hop. Die zugehörige Dauer wird mit $d_{arbBurst}$ bezeichnet und hängt von der verwendeten Hardware sowie der Synchronisationsungenauigkeit ab (Herleitungen für $d_{arbBurst}$ sind beispielsweise in [Chr10, CGR12, Chr15] zu finden). *Bitrunde* bezeichnet die Übertragung eines einzelnen Bits über n_{hops} Hops. Für die Übertragung eines Black-Burst-Rahmens mit n_{bits} Bits über n_{hops} Hops ergibt sich damit eine Ausführungszeit d_{arb} von

$$d_{arb} = n_{bits} \cdot n_{hops} \cdot d_{arbBurst} \,. \tag{3.1}$$

Das Beispiel in Abbildung 3.11 zeigt den Arbitrating Transfer unter Verwendung der Topologie aus Abbildung 3.10, d.h. $n_{hops} = n_{maxHops} = 3$ (die SOF-Bits der übertragenen Rahmen werden in diesem Beispiel nicht dargestellt). Es wird hier davon ausgegangen, dass die Black-Burst-Reichweite der Interferenzreichweite entspricht, so dass das Netz eigentlich nur einen Durchmesser von zwei (Interferenz)Hops hat. Die Arbitrierungssequenzen haben jeweils eine Länge von vier Bit.

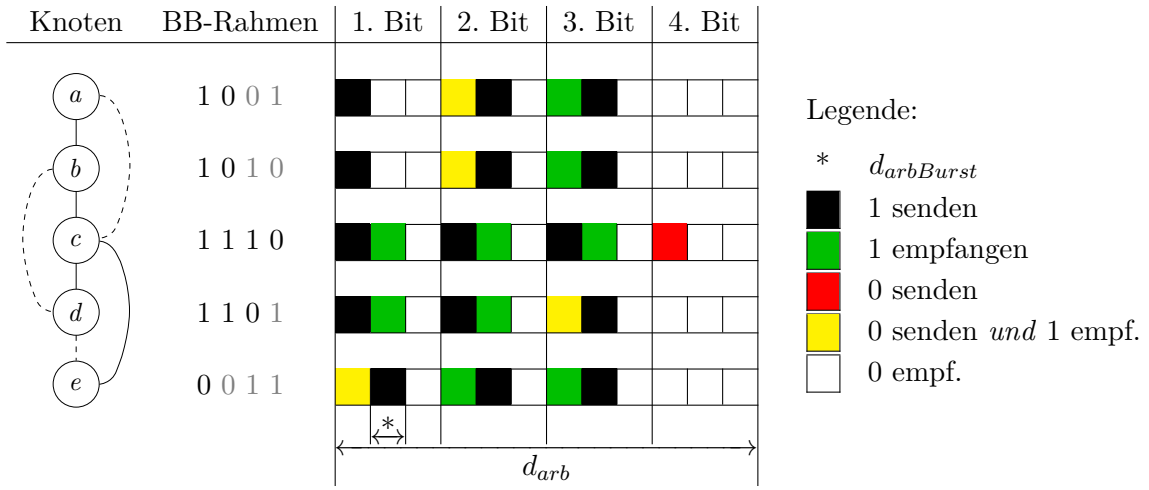

Abbildung 3.11.: Beispiel für Arbitrating Transfer mit Interferenzreichweite.

In der ersten Burstrunde der ersten Bitrunde überträgt jeder Knoten sein erstes Bit (d. h. e „sendet“ ein rezessives 0-Bit und alle anderen Knoten senden ein dominantes 1-Bit). e empfängt die 1 und leitet sie in der zweiten Burstrunde weiter. Da die von e gesendete 0 überschrieben wurde, nimmt e nicht mehr an der Arbitrierung teil. In der zweiten Bitrunde

übertragen die Knoten a und b in der ersten Burstrunde ein 0-Bit, welches von den 1-Bits der Knoten c und d überschrieben wird. In der zweiten Burstrunde wird das 1-Bit von a, b und e weitergeleitet. a und b nehmen nun ebenfalls nicht mehr an der Arbitrierung teil. In der dritten Bitrunde überträgt Knoten d in der ersten Burstrunde ein 0-Bit und c ein 1-Bit, welches anschließend von a, b, d und e weitergeleitet wird. d hat nun ebenfalls die Arbitrierung verloren. In der vierten Bitrunde ist nur noch Knoten c übrig. Dieser sendet in der ersten Burstrunde ein 0-Bit, welches in der restlichen Bitrunde nicht überschrieben wird, und hat damit die Arbitrierung gewonnen.

Im Folgenden wird das obige Beispiel etwas modifiziert, indem die Black-Burst-Reichweite auf die Kommunikationsreichweite eingeschränkt wird. Das Resultat ist in Abbildung 3.12 zu finden (auch hier werden die SOF-Bits nicht dargestellt).

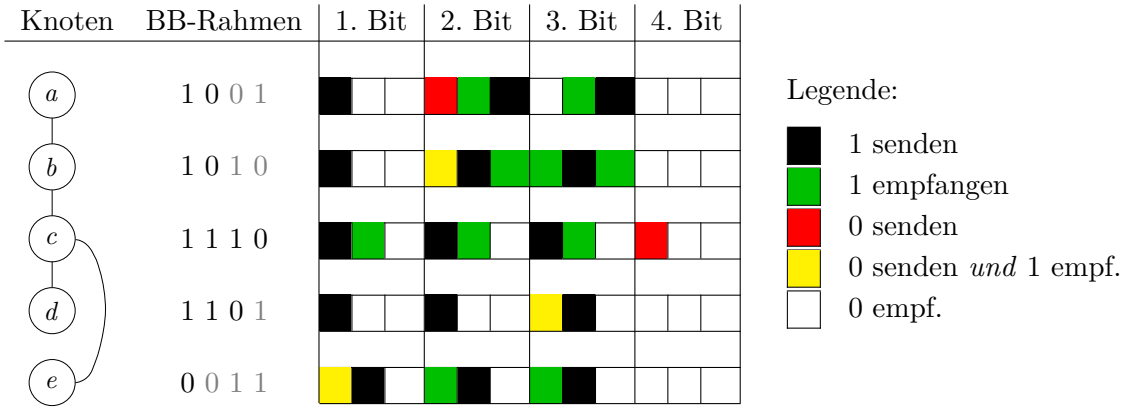

Abbildung 3.12.: Beispiel für Arbitrating Transfer mit Kommunikationsreichweite.

Das Verhalten ist ähnlich zum vorherigen Beispiel. In der zweiten Burstrunde der ersten Bitrunde empfängt Knoten d jedoch nicht das von e weitergeleitete 1-Bit. Ein weiterer Unterschied findet sich in der zweiten Bitrunde. Dort empfängt Knoten a in der ersten Burstrunde keine 1, da c sich nicht in Kommunikationsreichweite befindet. Die 1 wird erst in der zweiten Burstrunde von b empfangen und anschließend weitergeleitet. Dasselbe Verhalten für Knoten a findet sich auch in der dritten Bitrunde. Zudem empfängt Knoten d in der zweiten Burstrunde der zweiten Bitrunde nicht das von b und e weitergeleitete 1-Bit.

3.3.2.2. Cooperative Transfer

Der Cooperative Transfer dient dazu, dass mehrere Knoten gleichzeitig denselben Rahmen kollisionsgeschützt weiterleiten können. Diese Situation kann eintreten, wenn ein Knoten einen Rahmen durch das gesamte Netz propagieren will (in diesem Abschnitt wird von $n_{hops} = n_{maxHops}$ ausgegangen). Wird dieser Rahmen während des Weiterleitens von mehreren Knoten empfangen, welche ihn dann zeitgleich weiterleiten, würde es ohne eine kollisionsgeschützte Codierung bei gemeinsamen Empfängern zu einer Kollision kommen. Im Gegensatz zum Arbitrating Transfer erfolgt die Weiterleitung beim Cooperative Transfer nicht bitweise, sondern rahmenweise. Jeder sendende Knoten überträgt also den kompletten Rahmen, bevor dieser von den Empfängern weitergeleitet wird.

Im Folgenden bezeichnet eine *Burstrunde* des Cooperative Transfers die Übertragung eines einzelnen Bits über einen Hop (analog zum Arbitrating Transfer). Die zugehörige Dauer (abhängig von verwendeter Hardware und Synchronisationsungenauigkeit) wird mit $d_{coopBurst}$

bezeichnet und ist durch $d_{arbBurst}$ nach oben begrenzt (eine Herleitung für $d_{coopBurst}$ ist beispielsweise in [Chr10] zu finden). *Rahmenrunde* bezeichnet die Übertragung des Black-Burst-Rahmens über einen Hop. In jeder Rahmenrunde wird eine Verarbeitungszeit $d_{procCoop}$ hinzugerechnet, die ein Knoten nach Empfang eines Black-Burst-Rahmens benötigt. Diese hängt ebenfalls von der verwendeten Hardware ab. Für die Übertragung eines Black-Burst-Rahmens mit n_{bits} Bits über n_{hops} Hops ergibt sich damit eine Ausführungszeit d_{coop} von

$$d_{coop} = n_{hops} \cdot (n_{bits} \cdot d_{coopBurst} + d_{procCoop}) \,. \tag{3.2}$$

Im Allgemeinen ist der Cooperative Transfer etwas effizienter als der Arbitrating Transfer, da die zusätzliche Verarbeitungszeit $d_{procCoop}$ durch eine geringere Dauer der Burstrunde ausgeglichen wird. Ist dies nicht der Fall (z. B. weil nur wenige Bits zu übertragen sind), oder soll aus anderen Gründen nur ein Übertragungsverfahren für alle Übertragungen verwendet werden, so kann der Cooperative Transfer durch den Arbitrating Transfer ersetzt werden. Anstatt unterschiedliche Arbitrierungssequenzen zu verwenden, sendet dabei jeder Knoten die gleiche Sequenz. Die Verwendung des Arbitrating Transfers zum kooperativen Senden von Bitsequenzen ist beispielsweise in [Chr15] beschrieben.

Das Beispiel in Abbildung 3.13 zeigt den Cooperative Transfer unter Verwendung der Topologie aus Abbildung 3.10 ($n_{hops} = n_{maxHops} = 3$). Die SOF-Bits der übertragenen Rahmen werden nicht dargestellt. Die Black-Burst-Reichweite entspricht der Interferenzreichweite.

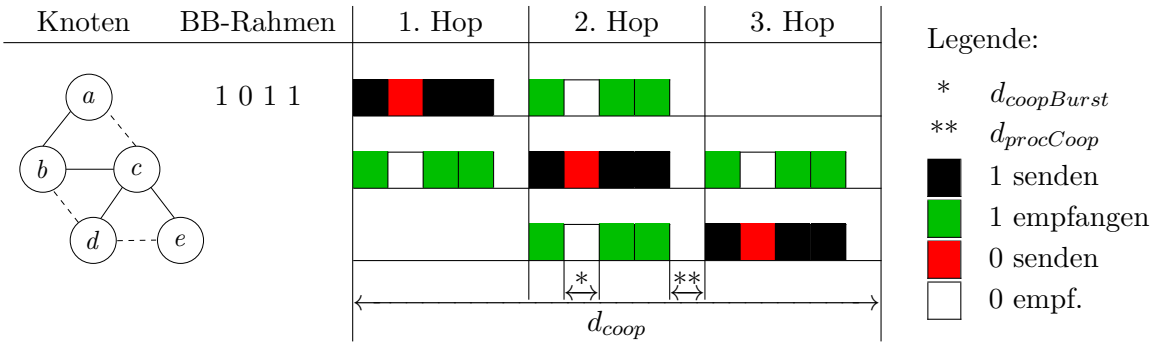

Abbildung 3.13.: Beispiel für Cooperative Transfer mit Interferenzreichweite.

In der ersten Rahmenrunde sendet Knoten a den Rahmen 1011 (in vier Burstrunden), welcher von b und c empfangen wird. In der zweiten Rahmenrunde wird dieser von beiden Knoten weitergeleitet. Durch die kollisionsgeschützte Codierung wird er korrekt von a, d und e empfangen und in der dritten Rahmenrunde von d und e weitergeleitet (da diese den Rahmen zuvor noch nicht weitergeleitet haben).

Im Folgenden wird das obige Beispiel modifiziert, indem die Black-Burst-Reichweite auf die Kommunikationsreichweite eingeschränkt wird. Das Resultat ist in Abbildung 3.14 dargestellt (auch hier ohne SOF-Bits).

Das Verhalten ist ähnlich zum vorherigen Beispiel. Der von Knoten a in der ersten Rahmenrunde gesendete Rahmen wird nun jedoch nur von b empfangen, da c sich nicht in Kommunikationsreichweite zu a befindet. b leitet den Rahmen in der zweiten Rahmenrunde weiter, woraufhin er von a und c empfangen und von c in der dritten Rahmenrunde weitergeleitet wird. Daraufhin wird er von b, d und e empfangen.

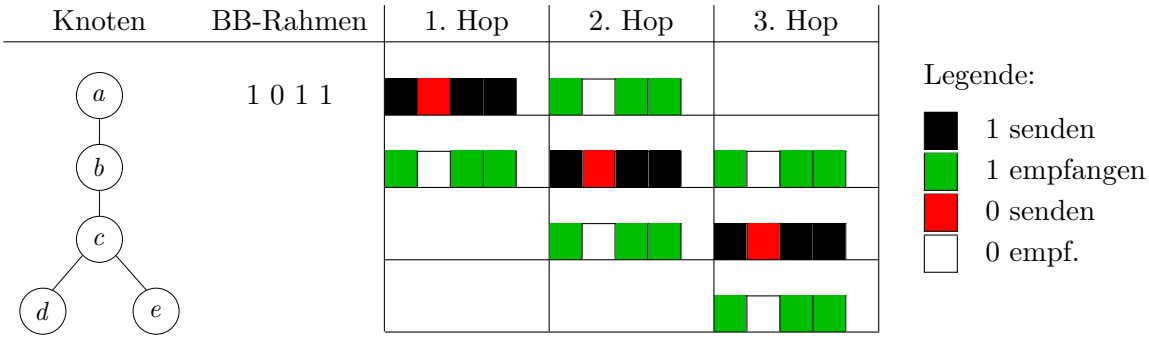

Abbildung 3.14.: Beispiel für Cooperative Transfer mit Kommunikationsreichweite.

3.3.2.3. Einschränkung von Arbitrating und Cooperative Transfer

Sowohl beim Arbitrating als auch beim Cooperative Transfer ist es möglich, diesen nicht netzweit, sondern nur in einem begrenzten Bereich um einen Knoten auszuführen (d. h. mit einem Radius $n_{hops} < n_{maxHops}$).

Beim Arbitrating Transfer kann dies zur Eliminierung des Hidden-Station-Problems verwendet werden (dabei entspricht die Black-Burst-Reichweite der Interferenzreichweite). Wird der Arbitrating Transfer mit $n_{hops} = 2$ ausgeführt, so befindet sich anschließend jeder Knoten, der selbst kein Gewinner ist, in Interferenzreichweite zu höchstens einem Gewinner. Abbildung 3.15 verdeutlicht die Eliminierung des Hidden-Station-Problems.

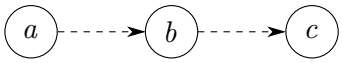

Abbildung 3.15.: Eliminierung des Hidden-Station-Problems.

Angenommen, a will einen Rahmen an b übertragen (per Cooperative Transfer, da b sich nicht in Kommunikationsreichweite zu a befindet) und c will gleichzeitig senden. Dazu wird eine Arbitrierung durchgeführt, an der a und c teilnehmen. Knoten a habe die größere Arbitrierungssequenz. Somit gewinnt a die Arbitrierung und c weiß dies, da c sich innerhalb des Arbitrierungsradius von a befindet. Auf diese Weise ist sichergestellt, dass c anschließend nicht sendet, so dass b den Rahmen von a fehlerfrei empfängt.

Nach der Arbitrierung gibt es zwei Möglichkeiten: Soll anschließend regulär gesendet werden, so müssen sich die beteiligten Knoten natürlich in Kommunikationsreichweite befinden. Es ist aber auch denkbar, dass – wie im obigen Beispiel – eine Übertragung mittels Cooperative Transfer (über einen Hop) stattfinden soll, um die Interferenznachbarn eines Knotens zu erreichen.

Wird der Arbitrating Transfer nicht netzweit ausgeführt, so kann dies dazu führen, dass Knoten Rahmen empfangen, die überhaupt nicht gesendet wurden (sogenannte Phantomrahmen). Dieses Phänomen wurde in [BBCG11, Chr15] beschrieben und kann beispielsweise dadurch zustande kommen, dass ein Knoten von keinem Gewinner erreicht wird und nur den Präfix eines Verlierers empfängt (dieser hört auf zu senden, sobald er die Arbitrierung verloren hat). Um feststellen zu können, ob ein Knoten sich im Bereich von n_{hops} Hops um einen Gewinner befindet, kann ein End-Of-Frame (EOF)-Bit verwendet werden. Dabei handelt es sich um ein dominantes Bit, welches an die übertragenen Rahmen angehängt wird. Damit kann jedoch immer noch nicht sichergestellt werden, dass ein tatsächlich gesendeter Rahmen

empfangen wurde. Eine weitere Möglichkeit, wie ein Phantomrahmen zustande kommen kann, besteht nämlich darin, dass ein Knoten sich im n_{hops}-Umkreis von mehreren Gewinnern befindet und somit die logische Veroderung der gesendeten Sequenzen empfängt. Man kann sich also nicht mehr darauf verlassen, dass ein per Arbitrating Transfer empfangener Wert einem tatsächlich gesendeten entspricht, wenn $n_{hops} < n_{maxHops}$ gilt. Somit muss der Wert anschließend mit einem anderen Übertragungsverfahren erneut übertragen werden, wenn dieser nach der Arbitrierung noch benötigt wird.

Beim Cooperative Transfer kann die Einschränkung des Bereichs dazu verwendet werden, um Informationen in einem bestimmten Teil des Netzes kollisionsgeschützt bekanntzugeben, beispielsweise wenn ein Knoten Informationen über zwei Hops propagieren will. Ein weiterer Anwendungsfall ist die Propagierung eines sich in Abhängigkeit von der Hopanzahl ändernden Pakets durch das gesamte Netz. Zwar gibt es im ersten Hop nur einen Sender, aber ab dem zweiten Hop kann es mehrere Sender geben, die jedoch alle identische Pakete versenden. Mittels wiederholtem Cooperative Transfer über jeweils einen Hop kann diese Übertragung kollisionsgeschützt durchgeführt werden. Ein Cooperative Transfer mit $n_{hops} < n_{maxHops}$ ist nur sinnvoll, wenn jeder Empfänger sich nur im Bereich solcher Sender befindet, die gleiche Rahmen übertragen. Um dies sicherzustellen, kann z. B. ein vorgeschalteter Arbitrating Transfer mit $n_{hops} = 2$ verwendet werden (s. o.).

3.3.2.4. Bit-based Transfer

Der Bit-based Transfer, welcher im Rahmen dieser Arbeit als Konzept definiert wird, ist eigentlich ein Spezialfall des Cooperative Transfers. Er erlaubt jedoch den Empfang von Rahmen, deren Länge nicht im Voraus bekannt ist. Black-Burst-Rahmen, die mittels Bit-based Transfer übertragen werden, erfordern daher ein Längenfeld, aus dem die Länge während des Empfangs ermittelt wird (auf diese Weise können empfangende Knoten feststellen, wie viele Bits noch folgen).

Anmerkung: Dies ist analog zu regulären Rahmen, da auch bei diesen die Länge i. d. R. nicht vorab bekannt ist. In dieser Arbeit wird deshalb davon ausgegangen, dass reguläre Rahmen ebenfalls ein Längenfeld enthalten.

Weiterhin darf es beim Bit-based Transfer keine konkurrierenden Sender geben. Dies bedeutet nicht, dass es im gesamten Netz nur einen Sender geben darf, sondern lediglich, dass jeder Empfänger sich nur in Reichweite eines Senders befinden darf (dies kann beispielsweise durch eine vorherige Arbitrierung mittels Arbitrating Transfer mit $n_{hops} = 2$ sichergestellt werden). Diese Einschränkung wird benötigt, da die MAC-Schicht beim Bit-based Transfer den höheren Schichten in einem separaten Bit mitteilt, ob sich der Sender eines Black-Burst-Rahmens in Kommunikationsreichweite befindet (diese Information wird für das Protokoll RBBQR benötigt, bei dem der Bit-based Transfer zum Einsatz kommt). Ob sich der Sender in Kommunikationsreichweite befindet, wird anhand der empfangenen Signalstärken ermittelt (es wird nur dann angenommen, dass der Sender sich in Kommunikationsreichweite befindet, wenn die Signalstärke für *alle* empfangenen Bits groß genug ist). Befände sich ein Empfänger in Reichweite mehrerer Sender, so könnte dieser Wert durch die Überlagerung der Signale verfälscht werden.

Aus der Einschränkung, dass es keine konkurrierenden Sender geben darf, folgt automatisch, dass Rahmen immer nur über einen (Interferenz-)Hop weitergeleitet werden dürfen, da beim Weiterleiten nicht mehr gewährleistet werden kann, dass sich jeder Empfänger nur in Reichweite eines Senders befindet. Somit gilt für den Bit-based Transfer stets $n_{hops} = 1$. Die

Ausführungszeit entspricht der des Cooperative Transfers mit $n_{hops} = 1$, d. h.

$$d_{bitb} = n_{bits} \cdot d_{coopBurst} + d_{procCoop}\,. \tag{3.3}$$

Eine Einschränkung der Black-Burst-Reichweite auf die Kommunikationsreichweite, wie sie bei Arbitrating und Cooperative Transfer vorgenommen werden kann, ist beim Bit-based Transfer nicht sinnvoll, da in einem solchen Fall auch ein reguläres Paket für die Übertragung verwendet werden könnte.

Kapitel 4.

4 Evaluation existierender Protokolle

In diesem Kapitel werden existierende QoS-Routing- und Reservierungsprotokolle für Ad-Hoc-Netze betrachtet, wobei zwischen Unicast- und Multicast-Protokollen unterschieden wird. Neben der Beschreibung des Protokolls BBQR, welches die Grundlage für RBBQR (s. Kapitel 5) bildet, liegt der Schwerpunkt auf Protokollen, die eine TDMA-basierte MAC-Schicht verwenden und bei der Ermittlung von QoS-Routen eine deterministische Reservierung von Ressourcen in Form von Zeitslots der MAC-Schicht vornehmen. Einige dieser Protokolle werden im Detail vorgestellt. Dabei wird auch untersucht, welche Probleme sie aufweisen bzw. welche Aspekte nicht berücksichtigt wurden. Die gewonnenen Erkenntnisse flossen in die Entwicklung der Protokolle RBBQR (Kapitel 5) und QMRP (Kapitel 6) ein.

Inhärente Probleme vieler existierender QoS-Routing- und Reservierungsprotokolle wurden bereits in den Abschnitten 3.1.1 und 3.1.2 allgemein erläutert und werden hier noch einmal kurz zusammengefasst. Die Kernprobleme beim Routing bestehen in parallelen Routensuchen (welche u. a. zu Fehlreservierungen führen können) und in Kollisionen von Kontrollpaketen. Protokolle, welche deterministische Reservierungen vornehmen, müssen das Hidden-Station- und das Exposed-Station-Problem berücksichtigen. Diejenigen der betrachteten Ansätze, die zur Eliminierung des Hidden-Station-Problems CDMA zusätzlich zu TDMA einsetzen, lassen das Exposed-Station-Problem außer Acht (s. u.). Die in diesem Kapitel vorgestellten reinen TDMA-Protokolle hingegen berücksichtigen beide Probleme. Eine genauere Betrachtung dieser Protokolle führte jedoch zu der Erkenntnis, dass bei vielen davon Fehlreservierungen aufgrund des Selbstinterferenzproblems auftreten. Eine weitere Schwierigkeit besteht darin, die Auswahl der für die einzelnen Links verwendeten Slots zu optimieren, um das SSSR- und das SSNR-Problem abzuschwächen. Während einige Protokolle diesen Aspekt nur rudimentär oder gar nicht berücksichtigen, definieren andere detaillierte Heuristiken für die Slotauswahl. Ein Nachteil aller hier genannten Protokolle besteht darin, dass diese lediglich simuliert und / oder analytisch betrachtet wurden, jedoch keine Implementierung auf realer Hardware existiert. Somit liegen keine Evaluationen unter realistischen Bedingungen vor.

Anmerkung: Für Protokolle, die von den Autoren nicht benannt wurden, wird hier ein auf dem Titel des zugehörigen Papers basierender Name vergeben.

4.1. Unicast-Protokolle

Es existieren zahlreiche QoS-Routing-Protokolle, die den Reservierungsaspekt allenfalls rudimentär berücksichtigen. Diese Protokolle nehmen entweder überhaupt keine oder lediglich statistische Reservierungen vor (vgl. Abschnitt 3.1.2). Statistische Reservierungen betreffen häufig die Bandbreite eines Knotens, welche als abstrakter Zahlenwert verwaltet wird. Wenn ein Knoten Bestandteil einer Route werden soll, wird geprüft, ob noch genügend Bandbreite zur Verfügung steht; diese wird dann von dem vorherigen Wert abgezogen. Die Auswirkungen von Bandbreitenreservierungen auf die Bandbreite der Nachbarknoten werden meist vernach-

lässigt.

In diese Kategorie fallen das *Ticket-Based Probing* (TBP) [CN99] und das *Multi-path QoS Routing Protocol* [LTWS01], welche eine Routensuche mit Hilfe sogenannter *Tickets* durchführen. Diese begrenzen die Anzahl der zur Routensuche verwendeten Pakete, um ein Fluten des Netzwerks zu vermeiden. TBP ermöglicht neben der Suche einer Route mit minimaler Bandbreite auch die einer Route mit maximaler Übertragungsverzögerung, reserviert jedoch keine Ressourcen. [LTWS01] verwendet lediglich die Bandbreite als Metrik, nimmt jedoch statistische Reservierungen vor und unterstützt die Aufteilung einer QoS-Route auf mehrere Pfade, welche zusammen die benötigte Bandbreite liefern (Multipfad-Routing).

Weitere Protokolle dieser Kategorie sind die positionsbasierten QoS-Routing-Protokolle *Predictive Location-Based QoS Routing* [SN02] und *Trigger-Based Distributed Routing* (TDR) [DDWQ02]. Predictive Location-Based QoS-Routing unterstützt verschiedene QoS-Metriken (z. B. Energie eines Knotens oder CPU-Auslastung), reserviert aber keine Ressourcen. TDR nimmt statistische Reservierungen von Bandbreite vor und unterstützt darüber hinaus die Hopanzahl einer Route als Metrik.

Auch das *Black-Burst-based QoS-Routing* (BBQR) [Bir09, BBCG11] nimmt ausschließlich statistische Reservierungen vor. Es bildet die Grundlage für das in Kapitel 5 entwickelte Protokoll RBBQR und wird in Abschnitt 4.1.1 detaillierter beschrieben.

Darüber hinaus gibt es Protokolle, deren Schwerpunkt darauf liegt, die Auswirkungen der Reservierung von Bandbreite in einem Knoten auf die Bandbreite der Interferenznachbarn zu ermitteln. Damit kann festgestellt werden, ob eine neue Übertragung zulässig ist. Beispiele hierfür sind das *Bandwidth Guaranteed Routing with Interference Consideration* [JGWV05] sowie das *Interference-aware QoS Routing* (IQRouting) [GJTW05]. Das verwendete Interferenzmodell entspricht dem beispielsweise in [JPPQ05] beschriebenen Konfliktgraphen auf Basis des Protokollinterferenzmodells (vgl. Abschnitt 2.8). Somit kann eine von der Kommunikationsreichweite verschiedene Interferenzreichweite verwendet werden. Das in [YK05] beschriebene *Contention-aware Admission Control Protocol* (CACP) geht davon aus, dass die Interferenzreichweite der Wahrnehmungsreichweite entspricht (d. h. $\forall\, a \in V : IN_1(a) = SN_1(a)$). Während [JGWV05] nicht auf die Etablierung von Reservierungen eingeht, sind diese bei [GJTW05] und [YK05] in statistischer Form möglich.

Da alle bisher genannten Protokolle keine exklusiven Reservierungen vornehmen, kann Kollisionsfreiheit nicht garantiert werden. Aus diesem Grund existieren auch Protokolle, die den Reservierungsaspekt detaillierter berücksichtigen und zusätzlich zur Ermittlung von QoS-Routen deterministische Reservierungen von Bandbreite in Form von Zeitslots der MAC-Schicht vornehmen. Da QoS-Routing-Protokolle für Ad-Hoc-Netze i. d. R. verteilt sind und die Slotauswahl von der gewählten Route abhängt, ist eine Verknüpfung von Routing- und Reservierungsprotokoll erforderlich.

Einige Protokolle verwenden zur Eliminierung des Hidden-Station-Problems CDMA zusätzlich zu TDMA. In diese Kategorie fallen beispielsweise das *Bandwidth Routing* [LL99], das *On-demand QoS Routing* [Lin01] sowie das *Link-state Multi-path QoS Routing* [CTSK04]. Beim CDMA-über-TDMA-Modell werden für Übertragungen, die sonst bei einem gemeinsamen Empfänger kollidieren würden, orthogonale Codes verwendet. Dazu werden jedoch entsprechende Codes sowie ein geeigneter Ansatz zu deren Verteilung benötigt. Eine Betrachtung dieses Problems ist in [Hu93] zu finden. Die genannten CDMA-über-TDMA-Protokolle berücksichtigen keine Interferenzen zwischen Knoten außerhalb der Kommunikationsreichweite, d. h. sie gehen von der 1-Hop-Interferenzannahme $\forall\, a \in V : IN_1(a) = CN_1(a)$ aus. Außerdem unterscheiden diese Protokolle nicht zwischen freien Sende- und Empfangsslots, sondern es werden nur allgemein freie und belegte Slots unterschieden. Dies kann zu einer

unnötigen Blockierung von Slots führen, da z. B. das Exposed-Station-Problem nicht beachtet wird. Dieses führt dazu, dass ein Knoten in einem Slot zwar nicht empfangen, aber trotzdem senden könnte, dies aber nicht tut (vgl. Abschnitt 3.1.2).

Daneben existieren reine TDMA-Ansätze, welche ebenfalls deterministische Reservierungen vornehmen. Im Gegensatz zu den genannten CDMA-über-TDMA-Protokollen unterscheiden diese auch zwischen freien Sende- und Empfangsslots. Für die Wiederverwendung von Slots werden jeweils Bedingungen formuliert, welche das Hidden-Station- sowie das Exposed-Station-Problem implizit berücksichtigen. Beispiele für derartige Protokolle sind der *Forward Algorithm* (FA) [ZC02], das *TDMA-based Bandwidth Reservation Protocol* [LTS02], das *Race-Free Bandwidth Reservation Protocol* [JW04], das *Distributed Slots Reservation Protocol* (DSRP) [SCCC06] sowie das *On-demand Bandwidth Reservation QoS Routing Protocol* [HST06]. Diese Protokolle gehen – analog zu den CDMA-über-TDMA-Ansätzen – von der 1-Hop-Interferenzannahme aus. Ein weiteres TDMA-basiertes Protokoll ist das in [SWW10] vorgestellte *Distributed QoS Routing*, welches die 2-Hop-Interferenzannahme verwendet. Es basiert auf dem Topology-Transparent Scheduling (TTS) [CF94], welches robust gegenüber Topologieänderungen ist. Bei TTS handelt es sich um ein TDMA-Verfahren, bei dem Slots i. Allg. mehrfach vergeben werden. Das Verfahren stellt inhärent sicher, dass jeder Knoten mit jedem seiner Nachbarn kommunizieren kann, wozu aber ggf. mehrere Sendevorgänge erforderlich sind. [SWW10] verwendet Prioritäten, um Datenslots exklusiv für Übertragungen reservieren zu können.

Die genannten Protokolle betrachten sowohl QoS-Routing als auch die Reservierung von Slots, wobei unterschiedlich viel Wert auf eine möglichst geschickte Slotverteilung gelegt wird. Während bei [ZC02] und [SCCC06] detailliertere Heuristiken zum Einsatz kommen, werden bei [LTS02] lediglich solche Slots bevorzugt wiederverwendet, die bereits von Nachbarknoten zum Senden verwendet werden (Exposed-Station-Problem). Bei [JW04] wird nicht auf die Slotauswahl eingegangen. Bei [HST06] werden Slots gewählt, die möglichst wenige neue Sende- und Empfangsblockierungen erzeugen. Bei [SWW10] werden Slots gewählt, die möglichst wenige neue Empfangsblockierungen erzeugen.

Die in [ZC02, LTS02, JW04, SCCC06] vorgestellten Protokolle werden in den Abschnitten 4.1.2 – 4.1.5 detaillierter betrachtet. Dabei wird auf inhärente Probleme eingegangen.

Weiterhin gibt es Arbeiten, die sich primär auf TDMA-Scheduling (Reservierungsaspekt) konzentrieren und das QoS-Routing ausklammern. Hierbei liegt der Fokus auf einer möglichst guten Verteilung der Zeitslots innerhalb des gesamten Netzes. Meist wird dabei die Verteilung von Zeitslots für einzelne Übertragungen anstatt für komplette Routen betrachtet. Ein umfassender Überblick ist in [SVV15] zu finden. Reine TDMA-Scheduling-Ansätze werden im Folgenden nicht detailliert betrachtet, da in dieser Arbeit der Schwerpunkt auf der Ermittlung von QoS-Routen sowie der kollisionsfreien Slotreservierung entlang der Routen liegt. Die Optimierung der Slotauswahl wird dabei als zusätzliches Kriterium berücksichtigt.

4.1.1. Black-Burst-based QoS-Routing (BBQR)

Black-Burst-based QoS-Routing (BBQR) [Bir09, BBCG11] ist ein verteiltes reaktives QoS-Routing-Protokoll, welches den Ausgangspunkt für die Entwicklung von RBBQR darstellt. Reservierungen sind nur in statistischer Form möglich. BBQR verwendet die in Abschnitt 3.3.2 vorgestellten Transferprotokolle Arbitrating und Cooperative Transfer (jedoch nicht den Bit-based Transfer). Deshalb müssen alle Knoten synchronisiert sein, wozu beispielsweise BBS [GK11] verwendet werden kann. Zudem muss es gemeinsame Referenzzeitpunkte geben (statisch konfiguriert oder dynamisch ermittelt), zu denen Routensuchen starten können. Da

die Black-Burst-basierten Übertragungen zur Ermittlung von Routen dienen, die anschließend für reguläre Übertragungen genutzt werden sollen, wird die Black-Burst-Reichweite auf die Kommunikationsreichweite eingeschränkt (s. Abschnitt 3.3.1). In BBQR bestehen Routensuchen aus vier Phasen mit konstanter oder zumindest beschränkter Dauer, welche im Folgenden detaillierter betrachtet werden.

Phase 1

Phase 1 dient dazu, parallele Routenanforderungen zu serialisieren und die zur Bearbeitung ausgewählte Routenanforderung im Netz bekanntzugeben. Alle Knoten, die eine QoS-Route anfordern wollen, senden ein RREQ-Paket (Route Request) per Arbitrating Transfer durch das gesamte Netz ($n_{hops} = n_{maxHops}$). Dieses Paket enthält die eindeutige ID des Senders, gefolgt von der ID des Ziels sowie der QoS-Anforderung. Diese unterteilt sich in QoS-Metrik und QoS-Wert (vgl. Abschnitt 3.1). Der Knoten mit der höchsten ID gewinnt die Arbitrierung und wird nach Abschluss von Phase 1 zum Quellknoten. Da das RREQ-Paket des Gewinners im gesamten Netz verteilt wird, kennen anschließend alle anderen Knoten (insbesondere auch der Zielknoten) die aktuelle Routenanforderung, d. h. Quelle, Ziel, QoS-Metrik und QoS-Wert.

Tabelle 4.1 (vgl. [BBCG11]) gibt einen Überblick über mögliche QoS-Metriken und deren Codierung. Da die Überprüfung, ob ein Knoten eine QoS-Anforderung erfüllen kann, in BBQR nur anhand lokal vorhandener Zahlenwerte erfolgt, sind nur konkave Metriken (s. Abschnitt 3.1) nutzbar. Somit kann beispielsweise die Metrik Latenz nicht verwendet werden.

QoS-Metrik	**Codierung**
Bandbreite	0
Latenz	1
Energie des Knotens	2

Tabelle 4.1.: QoS-Metriken und deren Codierung.

Tabelle 4.2 (vgl. [BBCG11]) zeigt die Abbildung zwischen QoS-Wert und der entsprechenden Codierung für die QoS-Metrik Bandbreite. Der QoS-Wert legt die Anzahl der pro Superslot zu reservierenden Slots für eine Route fest und ist die Zweierpotenz der jeweils zur Codierung verwendeten Zahl.

QoS-Wert	**Codierung**
1 Slot	0
2 Slots	1
4 Slots	2
8 Slots	3

Tabelle 4.2.: QoS-Werte und deren Codierung für die Metrik Bandbreite.

Anmerkung: Aus Gründen der Konsistenz wird hier die Einheit „Slots pro Superslot“ verwendet statt „Slots pro Sekunde“ wie in [BBCG11].

Phase 2

In Phase 2 wird ermittelt, ob mindestens eine Route existiert, die die QoS-Anforderung erfüllt. Dazu sendet der Zielknoten (nach Abschluss von Phase 1) per Cooperative Transfer mit $n_{hops} = 1$ ein RREP-Paket (Route Reply), welches einen mit 1 initialisierten Hopcounter enthält. Alle Knoten, die das Paket zum ersten Mal empfangen, überprüfen, ob sie die QoS-Anforderung erfüllen können.

Wenn ein Knoten die QoS-Anforderung erfüllen kann und die (konfigurierte) maximale Routenlänge noch nicht überschritten ist, so speichert er den empfangenen Hopcounter. Handelt es sich nicht um die Quelle, so wird der Wert um 1 erhöht und das Paket in gleicher Weise weitergeleitet. Empfängt die Quelle in Phase 2 ein RREP, so wurde mindestens eine hop-minimale Route gefunden, welche die QoS-Anforderung erfüllt. Der Wert des empfangenen Hopcounters entspricht der Hopanzahl der zugehörigen Route. Phase 2 kann jedoch nicht vorzeitig beendet werden, da noch weitere RREPs gesendet werden können. Nachfolgende RREPs werden von der Quelle verworfen, da sie nicht zu hop-minimalen Routen gehören. Empfängt die Quelle kein RREP, so existiert keine Route, welche die QoS-Anforderung erfüllt und die maximale Routenlänge nicht überschreitet.

Phase 3

In Phase 3 wird eine der in Phase 2 ermittelten hop-minimalen QoS-Routen verteilt ausgewählt. Zusätzlich werden die an der Route beteiligten Knoten über ihren jeweiligen Vorgänger informiert, da diese Information in Phase 4 benötigt wird. Die Quelle startet Phase 3 nach Abschluss von Phase 2 durch Versenden eines CREQ-Pakets (Construction Request) per Arbitrating Transfer mit $n_{hops} = 2$. Das Paket enthält die ID des Senders sowie einen Hopcounter, welcher mit dem in Phase 2 ermittelten Wert (um 1 dekrementiert) initialisiert wird. Jeder Knoten, der noch an der Routensuche teilnimmt und das CREQ-Paket im ersten Hop erhält (d. h. alle Bits werden in der jeweils ersten Burstrunde empfangen), vergleicht den Wert des empfangenen Hopcounters mit dem in Phase 2 gespeicherten. Ist der empfangene Hopcounter kleiner als der eigene, so ist der Knoten nicht Bestandteil einer hop-minimalen Route. Anderenfalls wird die ID des Vorgängers gespeichert und der Knoten sendet selbst wieder ein CREQ-Paket (mit seiner eigenen ID und einem um 1 dekrementierten Hopcounter).

Da ein gesendetes CREQ von mehreren 1-Hop-Nachfolgern gleichzeitig empfangen werden kann, welche dann selbst wieder ein CREQ senden, muss eine Arbitrierung erfolgen, um zu entscheiden, welcher der konkurrierenden Knoten Bestandteil der Route wird. Da sich alle Sender (aufgrund des gemeinsamen Vorgängers) in einem Abstand von maximal zwei Hops zueinander befinden, wird dies durch die Verwendung des Arbitrating Transfers mit $n_{hops} = 2$ gewährleistet. Abbildung 4.1 verdeutlicht dies.

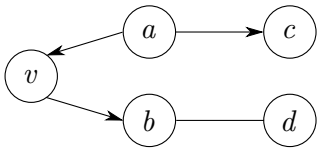

Abbildung 4.1.: Arbitrierung zwischen zwei potenziellen Routennachfolgern a und b.

Die Knoten a und b haben einen gemeinsamen Vorgänger v, dessen CREQ-Paket sie erhalten haben. Wenn a und b nun selbst ein CREQ senden, so gewinnt der Knoten mit der größten ID (dies sei hier a), da das CREQ-Paket an erster Stelle die ID des Senders enthält. Damit ist a der Routennachfolger von v. b verliert die Arbitrierung und hört daraufhin auf zu senden, was d anhand eines fehlenden End-Of-Frame (EOF)-Bits erkennt (vgl. Abschnitt 3.3.2.3).

Durch die Verwendung des Arbitrating Transfers mit $n_{hops} = 2$ wird zudem sichergestellt, dass der Gewinner der Arbitrierung im nächsten Knoten der Route eindeutig als Vorgänger eingetragen wird. Dies ist insbesondere relevant, wenn sich der nächste Knoten in Reichweite mehrerer konkurrierender Sender befindet. Abbildung 4.2 verdeutlicht dies.

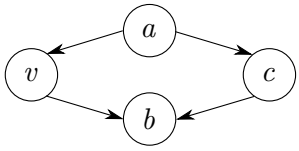

Abbildung 4.2.: Auswahl eines Vorgängers für c.

Die Knoten a und b haben beide das CREQ des Knotens v erhalten und senden nun jeweils selbst ein CREQ. Dabei sei die ID von a größer als die von b. Durch die Verwendung von $n_{hops} = 2$ verliert b die Arbitrierung. Damit empfängt c das CREQ-Paket von a und wählt a als seinen Vorgänger.

Wenn die letzten CREQ-Pakete von den Vorgängern des Ziels gesendet werden (diese haben einen Hopcounter von 0 und werden gleichzeitig gesendet), wird mit dem beschriebenen Mechanismus einer der Sender als Vorgänger des Ziels ausgewählt, womit die Route festgelegt ist.

Anmerkung: Im Allgemeinen kann es vorkommen, dass mehr als ein Gewinner existiert, wenn der Arbitrating Transfer nicht über das gesamte Netz ausgeführt wird. Dadurch empfängt ein Knoten, der sich im Umkreis von n_{hops} Hops um mehrere Gewinner befindet, die logische Veroderung des gesendeten Werte (vgl. Abschnitt 3.3.2.3). Hier tritt dieses Problem jedoch nicht auf, da alle Sender aufgrund des gemeinsamen Vorgängers einen Abstand von höchstens zwei Hops zueinander haben. Somit führt die Verwendung von $n_{hops} = 2$ dazu, dass es immer nur einen Gewinner gibt.

Phase 4

In Phase 4 wird die gewählte Route bekannt gegeben, indem jedem Knoten auf der Route die ID seines Nachfolgers mitgeteilt wird. Da es in dieser Phase keine konkurrierenden Sender gibt, können dazu reguläre Übertragungen verwendet werden. Der Zielknoten initiiert Phase 4 durch Versenden eines CREP-Pakets (Construction Reply), welches seine ID sowie die ID seines Vorgängers enthält. Jeder Knoten entlang der Route, der ein CREP erhält, speichert die ID seines Nachfolgers, passt die IDs entsprechend an und leitet das Paket weiter. Dabei werden die angeforderten Ressourcen reserviert und der zugehörige Eintrag in der Routentabelle wird als gültig markiert. Sobald das CREP die Quelle erreicht, ist die Route etabliert, und die benötigten Ressourcen sind reserviert.

Offene Aspekte

BBQR löst die beiden in Abschnitt 3.1.1 beschriebenen Kernprobleme existierender QoS-Routing-Protokolle. Das Problem paralleler Routenanforderungen wird durch deren Serialisierung in Phase 1 gelöst. Das Problem kollidierender Kontrollpakete wird durch die Verwendung der Black-Burst-basierten Transferprotokolle eliminiert.

BBQR nimmt jedoch eine rein statistische Reservierung von Ressourcen (z. B. Bandbreite) anhand lokal vorhandener Statusinformationen vor. Auch wird der Einfluss einer Reservierung auf andere Knoten nicht betrachtet. Aus diesem Grund sollte BBQR um ein (integriertes) Reservierungsprotokoll zur deterministischen Reservierung von Bandbreite in Form von Zeitslots erweitert werden. Diese Erweiterung erforderte jedoch erhebliche Änderungen, da mehr Status-

informationen ausgetauscht werden müssen. Dies führte zur Entwicklung des neuen Protokolls RBBQR, welches in Kapitel 5 beschrieben wird. Die Serialisierung paralleler Routenanforderungen in Phase 1 sowie die Verwendung der Black-Burst-basierten Transferprotokolle zur Vermeidung von Kollisionen bleiben auch in RBBQR erhalten, so dass auch dieses Protokoll die beiden Kernprobleme löst.

4.1.2. Forward Algorithm (FA)

Der *Forward Algorithm* (FA) wird in [ZC02] vorgestellt. Es handelt sich um ein verteiltes reaktives Protokoll für Slotreservierungen und QoS-Routing, welches auf AODV (Ad-hoc On-demand Distance Vector Routing) [PR99] basiert und einen Algorithmus zur effizienten Berechnung der Ende-zu-Ende-Bandbreite in dieses Protokoll integriert. Für FA wird explizit angenommen, dass sich immer nur eine Routensuche gleichzeitig in Bearbeitung befindet; dies wird jedoch nicht wie bei BBQR durch das Protokoll sichergestellt.

Prinzipiell kann für FA eine beliebige TDMA-MAC-Schicht eingesetzt werden. In [ZC02] wird zur Evaluation die von einem der Autoren des Papers entwickelte MAC-Schicht E-TDMA (Evolutionary-TDMA) [Zhu02] verwendet, welche reservierbare Zeitslots für Datenübertragungen bereitstellt. Slots für Kontrollpakete werden wettbewerbsbasiert reserviert, so dass Kollisionen prinzipiell möglich sind. Diese werden durch E-TDMA behandelt [Zhu02]; aus solchen Kollisionen resultieren jedoch indeterministische Verzögerungen.

In [ZC02] wird das *Bandwidth Calculation Problem* (BWC) definiert, welches darin besteht, die auf einer Route verfügbare Bandbreite durch geeignete Slotzuteilungen zu maximieren. In [Zhu02] wird gezeigt, dass dieses Problem NP-vollständig ist, weshalb FA eine Heuristik verwendet. Da aufgrund des Hidden-Station-Problems jeder Slot auf drei aufeinanderfolgenden Links einer Route nur einmal verwendbar ist (das Protokoll geht von der 1-Hop-Interferenzannahme aus), werden jeweils Mengen konfliktfreier Slots für drei adjazente Links berechnet, so dass die verfügbare Bandbreite lokal maximiert wird. Diese Berechnung wird dann auf dem Pfad zum Ziel propagiert. Damit wird das SSSR-Problem (vgl. Abschnitt 3.1.2) abgeschwächt. Das SSNR-Problem wird nicht betrachtet.

Zur Ermittlung der Routen wird AODV verwendet. Das RREQ-Paket (Route Request) von AODV wird um die für die letzten beiden Links ermittelten Slotmengen sowie die freien Sendeslots des weiterleitenden Knotens erweitert und per Broadcast gesendet. Die Empfänger ermitteln anhand ihrer freien Empfangsslots die für den letzten Link verwendbaren Slots und berechnen die Slotmengen für die letzten drei Links. Enthalten diese jeweils mindestens so viele Slots wie benötigt, so wird das RREQ weitergeleitet. Kann die Route bis zum Ziel fortgesetzt werden, so wird sie mittels eines RREP-Pakets (Route Reply) bestätigt. Dieses enthält die ausgewählten Sendeslots des Senders und des Vorgängers, so dass der Vorgänger seine Sendeslots reservieren und aus den übermittelten Slotmengen die Sendeslots für seinen Vorgänger berechnen kann. Die Freigabe von Routen und Reservierungen erfolgt durch Timeouts.

Es wird keine Aussage darüber getroffen, wie Sende- und Empfangsreservierungen an die entsprechenden Nachbarknoten propagiert werden. Es ist jedoch möglich, dass dies durch Mithören der RREP-Pakete erfolgt, da dort jeweils die Sende- und Empfangsslots eines Knotens enthalten sind.

Weiterhin wird zwar berücksichtigt, dass ein Slot auf drei aufeinanderfolgenden Links einer Route nur einmal verwendet werden kann, jedoch wird das in Abschnitt 3.1.2 beschriebene Selbstinterferenzproblem außer Acht gelassen. Dadurch können Fehlreservierungen auftreten. Das Selbstinterferenzproblem kommt dadurch zustande, dass die kürzeste Route zwischen

Quelle und Ziel nicht etabliert werden kann, weil auf einem Link nicht genügend freie Slots zur Verfügung stehen. In diesem Fall versucht das Protokoll, die Engstelle zu umgehen. Dadurch können topologisch benachbarte Knoten auf der Route weiter auseinander liegen.

Das Auftreten von Fehlreservierungen durch das Selbstinterferenzproblem bei der in [ZC02] verwendeten Heuristik wird in [SLC05] thematisiert (dort wird dies als *Shortcut Collision Problem* bezeichnet). [SLC05] stellt eine ähnliche Heuristik wie [ZC02] vor (*Bandwidth Reservation Algorithm*), um die auf einer Route verfügbare Bandbreite zu maximieren, wobei in diesem Fall das Selbstinterferenzproblem berücksichtigt wird. Die Entwicklung eines zugehörigen QoS-Routing-Protokolls wird jedoch offengelassen.

4.1.3. TDMA-based Bandwidth Reservation Protocol

Das *TDMA-based Bandwidth Reservation Protocol* wird in [LTS02] vorgestellt. Es handelt sich um ein quellbasiertes reaktives Protokoll für Slotreservierungen und QoS-Routing. Da es quellbasiert ist, wird der gesamte Pfad, den ein Paket durch das Netzwerk nimmt, von der Quelle festgelegt und ist im Paket enthalten.

Parallele Routensuchen werden weder explizit ausgeschlossen noch durch das Protokoll behandelt. Dadurch können Fehlreservierungen auftreten. Da die während der Routensuche vorgenommenen temporären Reservierungen nicht an die Interferenznachbarn propagiert werden und ein Knoten temporäre Reservierungen für eine Route bei anderen Routensuchen nicht berücksichtigt, können sowohl temporär reservierte als auch durch temporäre Reservierungen blockierte Slots für andere Routensuchen verwendet werden. Diese Probleme werden von [JW04] (s. Abschnitt 4.1.4) behandelt.

Es wird eine TDMA-basierte MAC-Schicht zugrunde gelegt, deren Zeitslots reserviert werden. Jedoch wird keine Aussage darüber getroffen, wie Kontrollpakete versendet werden. Somit kann nicht sichergestellt werden, dass dies kollisionsfrei erfolgt.

Zur Routensuche wird ein QREQ-Paket (QoS Request) verwendet. Jeder Knoten ermittelt die potenziell nächsten Knoten der Route mitsamt möglicher Slots zum Senden an diese und propagiert die Knoten und zugehörigen Slots mittels eines QREQ. Der bisherige Pfad mitsamt den gewählten Slots wird ebenfalls propagiert, wodurch das Hidden-Station-Problem vermieden werden kann.

Um die für einen Link nutzbaren Slots ermitteln zu können, speichert jeder Knoten a Informationen über die zum Senden und zum Empfangen reservierten Slots aller Knoten in $CN_{\leq 2}(a)$ (es gilt die 1-Hop-Interferenzannahme). Dabei gibt es jeweils eine Tabelle für die Sende- und eine für die Empfangsreservierungen. Eine zusätzliche Tabelle dient dazu, für jeden 1-Hop-Nachbarn wiederum dessen 1-Hop-Nachbarn zu speichern (d. h. $CN_1(CN_1(a))$). Dies wird beispielsweise benötigt, da ein 1-Hop-Nachbar eines Knotens nur dann empfangen kann, wenn keiner seiner 1-Hop-Nachbarn sendet (um dies zu ermitteln, werden die Sendereservierungen der 2-Hop-Nachbarn benötigt sowie die Information, welche der 2-Hop-Nachbarn 1-Hop-Nachbarn des potenziellen Routennachfolgers sind). Um diese Tabellen zu erstellen, senden alle Knoten ihre eigenen Statusinformationen periodisch per Broadcast an ihre 1- und 2-Hop-Nachbarn.

Die Slotreservierungen werden bei der Bestätigung einer gefundenen Route mittels eines QREP-Pakets (QoS Reply) vorgenommen. Dieses enthält die komplette Route mitsamt den zugehörigen Reservierungen und wird vom Ziel entlang der Route zur Quelle propagiert. Da jedoch keine asynchronen Statusaktualisierungen verwendet werden, um Reservierungen an die Nachbarknoten zu übermitteln, werden die reservierten Slots erst mit der nächsten peri-

odischen Aktualisierung propagiert. Durch diese Verzögerung kann es zu Fehlreservierungen kommen (d. h. es können bereits blockierte Slots für andere Routen reserviert werden).

Für die Slotauswahl werden keine detaillierten Heuristiken verwendet. Es werden lediglich solche Slots bevorzugt gewählt, die vom Exposed-Station-Problem betroffen sind (dies sind Slots, in denen Nachbarn des Senders bereits senden). Dies kann als eine Möglichkeit angesehen werden, das SSNR-Problem abzuschwächen (vgl. Abschnitt 4.1.5). Es werden jedoch keine Mechanismen verwendet, um beispielsweise das SSSR-Problem abzuschwächen. Auch Fehlreservierungen aufgrund des Selbstinterferenzproblems können bei diesem Protokoll auftreten.

4.1.4. Race-Free Bandwidth Reservation Protocol

Das *Race-Free Bandwidth Reservation Protocol* [JW04] ist eine Verbesserung von [LTS02]. Es handelt sich ebenfalls um ein quellbasiertes reaktives Protokoll. Routensuchen werden im Wesentlichen analog zu [LTS02] durchgeführt. Das Hauptziel besteht hier jedoch darin, parallele Routensuchen zu unterstützen und temporäre Reservierungen zu berücksichtigen, so dass Fehlreservierungen vermieden werden.

Das Protokoll unterteilt die Zeit in TDMA-Rahmen, welche aus einer Kontroll- und einer Datenphase bestehen. Im Gegensatz zu [LTS02] wird jedem Knoten des Netzes ein exklusiver Slot in der Kontrollphase zugeordnet, welcher zur dynamischen Reservierung von Datenslots in der Datenphase verwendet werden kann. Damit können Kontrollpakete kollisionsfrei gesendet werden; da jedoch alle Kontrollpakete eines Knotens serialisiert werden müssen, kann dies zu indeterministischen Verzögerungen führen.

Analog zu [LTS02] verwaltet jeder Knoten a die Slotzustände für Knoten aus $CN_{\leq 2}(a)$ in Sende- und Empfangstabellen (auch hier wird die 1-Hop-Interferenzannahme getroffen). Neben den Slotzuständen *free* und *reserved* wird ein zusätzlicher Zustand *allocated* für unbestätigte Reservierungen verwendet. Da diese vorläufigen Reservierungen in die Sende- und Empfangstabellen eingetragen werden, können sie bei anderen Routensuchen, die durch den Knoten verlaufen, berücksichtigt werden.

Wie bei [LTS02] werden periodisch Statusinformationen an die 1- und 2-Hop-Nachbarn propagiert. Zusätzlich werden jedoch auch asynchrone Statusaktualisierungen gesendet, wenn ein Slotzustand von *free* zu *allocated* oder von *allocated* zu *reserved* geändert wird. Grundsätzlich behebt die Verwendung des Zustands *allocated* sowie die asynchrone Aktualisierung der Statusinformationen das Problem von Fehlreservierungen durch nicht propagierte Vorreservierungen. Auch werden dadurch Fehlreservierungen aufgrund von Selbstinterferenz eliminiert. Die asynchronen Statusaktualisierungen können jedoch nur in den reservierten Kontrollslots der einzelnen Knoten gesendet werden, so dass Verzögerungen auftreten (allerdings geringere, als wenn auf die nächste periodische Aktualisierung gewartet werden muss). Erfolgt eine neue Routensuche, bevor die Statusaktualisierung propagiert werden kann, so sind wiederum Fehlreservierungen möglich.

Anmerkung: Die Unterscheidung zwischen bestätigten und unbestätigten Reservierungen ist nützlich für einen (verteilten) Reservierungsalgorithmus, wird aber für die Betrachtung des Interferenzproblems nicht benötigt. Deshalb wurde bei der in Kapitel 2 eingeführten Modellierung keine derartige Unterscheidung vorgesehen.

Um vorläufige Reservierungen freigeben zu können, wenn eine Route nicht zustande kommt (bzw. wenn eine andere Route gewählt wird), werden TTL-Timer (Time-To-Live) verwendet, welche den Status eines vorreservierten Slots von *allocated* zu *free* zurücksetzen, wenn der ent-

sprechende QoS-Request nicht innerhalb einer vordefinierten Zeitspanne bestätigt wird. Wird der Slotstatus jedoch zu früh auf *free* zurückgesetzt, so können ebenfalls Fehlreservierungen auftreten.

Die verwendete Strategie sieht vor, dass Routenanforderungen, welche aufgrund von zu wenigen freien Slots nicht erfüllbar sind, nicht direkt zurückgewiesen werden („wait-before-reject“). Statt dessen wird ermittelt, ob es eine entsprechende Anzahl vorreservierter Slots gibt, deren Timer innerhalb einer bestimmten Zeitspanne ablaufen. Ist dies der Fall, so wird gewartet und geprüft, ob genügend Slots frei werden (weil sie zu nicht gewählten Routen gehören).

4.1.5. Distributed Slots Reservation Protocol (DSRP)

Das in [SCCC06] vorgestellte *Distributed Slots Reservation Protocol* (DSRP) ist ein reaktives Protokoll zur Ermittlung von QoS-Routen und zur Reservierung von Slots entlang dieser Routen. Während das Routing quellbasiert erfolgt (analog zu [LTS02] und [JW04]), werden die für die Slotreservierungen benötigten Informationen verteilt ermittelt. Der Schwerpunkt des Protokolls liegt auf der Wiederverwendung von Datenslots.

Die Zeit wird in TDMA-Rahmen unterteilt, welche aus einem Kontroll- und einem Datenrahmen bestehen und jeweils in Slots unterteilt sind. Es wird jedoch keine Aussage über das Scheduling der Kontrollpakete getroffen. Somit ist nicht sichergestellt, dass dieses kollisionsfrei erfolgt.

Die benötigten Reservierungsinformationen werden in Form von *Slot Inhibited Policies* ermittelt. Da diese Informationen nicht proaktiv gesammelt werden, müssen sie im Bedarfsfall ermittelt werden. Ein potenzieller Sender x sammelt die benötigten Informationen von seinen Nachbarn (es wird von der 1-Hop-Interferenzannahme ausgegangen) und bestimmt seine zum Senden verwendbaren Slots. Diese werden im RREQ-Paket (Route Request) an den potenziellen Empfänger y weitergeleitet, welcher daraus und aus seinen zum Empfangen verwendbaren Slots die für den Link (x, y) nutzbaren Slots ermittelt. Die konkrete Slotauswahl erfolgt jeweils durch den 3-Hop-Nachfolger eines Knotens (s. u.). Da dieser Informationsaustausch in Kontrollrahmen erfolgt, können Kollisionen und / oder indeterministische Verzögerungen auftreten.

Ein weiteres Problem besteht darin, dass Slots während der Routensuche nicht vorreserviert werden. Die Reservierung erfolgt erst bei der Bestätigung der Route mittels eines RREP-Pakets (Route Reply). Da parallele Routensuchen zulässig sind, können die vorgesehenen Slots sowohl von den beteiligten Knoten selbst als auch von Interferenznachbarn für weitere Routensuchen allokiert werden. Dies führt zu Fehlreservierungen, wenn schließlich mehr als eine dieser Routen etabliert wird.

Zur Behebung von Fehlreservierungen dient ein *Slot Adjustment Protocol*. Dieses wird jedoch nur dann eingesetzt, wenn Slots bei einem Knoten selbst mehrfach reserviert werden. Hat jedoch statt dessen ein Interferenznachbar die entsprechenden Slots in der Zwischenzeit reserviert, so wird dies nicht bemerkt, da diese Information nur im reservierenden Knoten gespeichert wird. Somit können weiterhin Fehlreservierungen durch parallele Routensuchen auftreten. Auch Fehlreservierungen aufgrund des Selbstinterferenzproblems werden nicht verhindert.

Neben dem Hidden- und dem Exposed-Station-Problem werden das in [SCCC06] definierte *Slot Shortage for Self Route* (SSSR)- und das *Slot Shortage for Neighboring Routes* (SSNR)-Problem betrachtet (vgl. Abschnitt 3.1.2). Um das SSSR- und SSNR-Problem abzuschwächen, werden drei *Slot Decision Policies* verwendet (*3-Hop Backward Decision Policy* (3BDP), *Least*

Conflict First Policy (LCFP) und *Most Reuse First Policy* (MRFP)). 3BDF besteht darin, dass die Sendeslots eines Knotens jeweils von dessen 3-Hop-Nachfolger festgelegt werden, um eine möglichst geeignete Slotauswahl treffen zu können. LCFP verwendet für jeden Link diejenigen Slots, die am wenigsten Konflikte mit den nächsten beiden Links haben (3BDP und LCFP werden in Abschnitt 5.2.2.1 näher betrachtet). Das Ziel dieser beiden Policies besteht darin, die Slotauswahl so zu treffen, dass auf einer Route jeweils möglichst viele Slots für die nachfolgenden Links übrig bleiben. Sie können deshalb als eine Möglichkeit angesehen werden, das SSSR-Problem abzuschwächen. MRFP verwendet bevorzugt solche Slots, die bereits von Nachbarknoten verwendet werden. Damit wird eine möglichst gute Slotwiederverwendung ermöglicht, was dazu beiträgt, das SSNR-Problem abzuschwächen.

4.2. Multicast-Protokolle

Auch bei den Multicast-Protokollen gibt es solche, die sich auf das QoS-Routing konzentrieren und Reservierungen allenfalls in statistischer Form durchführen. In diese Kategorie fallen beispielsweise das *On-Demand QoS Multicast for MANETs* (ODQMM) [NLT04] und das *QoS Multicast Routing Protocol* (QMR) [SWB05], welche beide statistische Reservierungen von Bandbreite vornehmen und die Auswirkungen von Bandbreitenreservierungen auf Nachbarknoten vernachlässigen.

Es existieren jedoch auch Protokolle, welche den Reservierungsaspekt detailliert berücksichtigen und deterministische Reservierungen in Form von Zeitslots der MAC-Schicht vornehmen. Analog zu den Unicast-Protokollen sind hier zunächst Ansätze zu nennen, die das CDMA-über-TDMA-Modell verwenden. Beispiele für derartige Protokolle sind das *Lantern-Tree-Based Routing* [CKL02], das *QoS Multicast Routing Using Multiple Paths / Trees* [WJ07] sowie das *Multi-Path QoS Multicast Routing* (MQMR) [WL12]. Wie auch die CDMA-über-TDMA-Protokolle für Unicast-Routing gehen diese von der 1-Hop-Interferenzannahme aus und unterscheiden nicht zwischen freien Sende- und Empfangsslots.

Daneben existieren reine TDMA-Ansätze, welche zusätzlich zwischen freien Sende- und Empfangsslots unterscheiden. Dies sind jedoch im Vergleich zu den Unicast-Protokollen nur wenige. Im Wesentlichen sind hier das *Hexagonal-Tree QoS Multicast Protocol* [CLL07] und PSLCB [ZL13] zu nennen. Diese verwenden ebenfalls die 1-Hop-Interferenzannahme und werden in den folgenden beiden Abschnitten detaillierter betrachtet.

4.2.1. Hexagonal-Tree QoS Multicast Protocol

Das in [CLL07] vorgestellte *Hexagonal-Tree QoS Multicast Protocol* hat zum Ziel, mit Hilfe eines sogenannten hexagonalen Baums Multipfade zwischen der Quelle und den Zielen zu finden. Ein solcher Baum enthält hexagonale Strukturen, die auf Routenabschnitten Verwendung finden, auf denen die Bandbreite für einen einzelnen Pfad nicht ausreicht. Abbildung 4.3 zeigt ein Beispiel für eine solche hexagonale Struktur (hexagonaler Block).

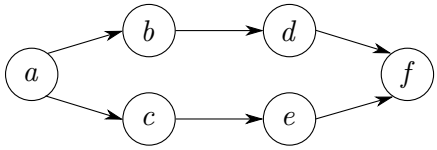

Abbildung 4.3.: Beispiel für einen hexagonalen Block.

Wird beispielsweise eine Route zwischen a und f mit einer Bandbreite von zwei Slots benötigt, und existiert keine solche, so können die Routen $a \to b \to d \to f$ und $a \to c \to e \to f$ verwendet werden, sofern diese jeweils eine Bandbreite von einem freien Slot haben.

Das Protokoll zielt darauf ab, die Slotwiederverwendung bei hexagonalen Strukturen zu optimieren. Dabei wird die 1-Hop-Interferenzannahme zugrunde gelegt. Bei dem hexagonalen Block aus Abbildung 4.3 können beispielsweise für die Links (a, b) und (c, e) sowie für (a, c) und (b, d) jeweils dieselben Slots verwendet werden. Dies kann als eine Möglichkeit angesehen werden, das SSSR- und SSNR-Problem abzuschwächen.

Bei diesem Protokoll wird kein lokaler Multicast verwendet. Bei einem solchen kann ein Knoten des Multicast-Baums, bei dem ein Abzweig beginnt (d. h. ein Sender mit mehreren Empfängern), dieselben Slots zum Senden an mehrere Empfänger nutzen, sofern diese in den entsprechenden Slots empfangen können. Der Vorteil eines lokalen Multicasts (Einsparung von Bandbreite) kommt daher bei diesem Protokoll nicht zum Tragen.

Routen werden reaktiv ermittelt, indem eine entsprechende Anfrage zu den Zielen gesendet wird. Die Ziele melden die Information über die gefunden Pfade (diese beinhalten ggf. hexagonale Strukturen) zur Quelle zurück, welche daraus anhand verschiedener Kriterien einen hexagonalen Baum erstellt. Dieser wird dann im Netz etabliert.

Es wird eine TDMA-basierte MAC-Schicht mit reservierbaren Datenslots zugrunde gelegt. Über das Scheduling der Kontrollpakete wird jedoch keine Aussage getroffen, so dass Kollisionen nicht ausgeschlossen werden können.

Parallele Routensuchen werden weder ausgeschlossen noch durch das Protokoll behandelt. Jeder Knoten speichert Informationen über seine 1-, 2- und 3-Hop-Nachbarn, da die Slots für eine hexagonale Struktur jeweils von einem Knoten festgelegt werden. Es wird jedoch keine Aussage darüber getroffen, wann diese Informationen jeweils aktualisiert werden und ob Vorreservierungen verwendet werden. Somit sind sowohl Fehlreservierungen durch parallele Routensuchen als auch aufgrund des Selbstinterferenzproblems denkbar.

4.2.2. PSLCB

PSLCB wird in [ZL13] vorgestellt (die Bedeutung des Akronyms wird nicht erklärt). Es handelt sich um ein reaktives Protokoll für den Aufbau von Multicast-Bäumen, deren Routen eine vorgegebene QoS-Anforderung erfüllen. Das Hauptziel besteht darin, die Anzahl der weiterleitenden Knoten und somit die insgesamt benötigte Bandbreite zu minimieren. Während das Routing quellbasiert erfolgt, werden die für Slotreservierungen benötigten Informationen verteilt ermittelt.

Es wird eine TDMA-basierte MAC-Schicht mit reservierbaren Datenslots verwendet. Über das Scheduling der Kontrollpakete wird keine Aussage getroffen, so dass Kollisionen nicht ausgeschlossen werden können.

Parallele Routensuchen werden weder ausgeschlossen noch durch das Protokoll behandelt. Zudem erfolgt die Reservierung von Zeitslots erst bei der Etablierung einer Route. Somit können Fehlreservierungen auftreten, da die für eine Route vorgesehenen Slots sowohl von den Knoten selbst als auch von deren Interferenznachbarn für weitere Routensuchen allokiert werden können.

Will ein Knoten z einem Multicast-Baum als neues Ziel beitreten, so sendet er ein JOIN-Paket, welches von allen Knoten, die noch nicht Bestandteil des Baums sind, weitergeleitet wird. Knoten des Multicast-Baums warten ab, bis sie eine bestimmte Anzahl an JOIN-Paketen erhalten haben, und wählen eins (von potenziell mehreren) mit der kleinsten Hopanzahl. Dieses wird mit einem REPLY-Paket beantwortet. z wartet wiederum ab, bis eine bestimmte Anzahl

an REPLY-Paketen empfangen wurde, und wählt ebenfalls eins mit der kleinsten Hopanzahl. Dadurch wird die Routenlänge von z zu dem Multicast-Baum minimiert. Die entsprechende Route wird mittels eines RESERVE-Pakets reserviert.

Die benötigte Reservierungsinformation (um zu entscheiden, ob ein Knoten, welcher ein JOIN-Paket erhält, die QoS-Anforderung erfüllen kann) wird reaktiv ermittelt. Dies kann zu Kollisionen mit anderen Kontrollpaketen führen. Es wird in [ZL13] zwar formuliert, dass die Interferenzreichweite der doppelten Kommunikationsreichweite entspricht; die verwendeten Theoreme für die Slotwiederverwendung legen jedoch die 1-Hop-Interferenzannahme zugrunde.

Die konkrete Slotauswahl erfolgt durch den 3-Hop-Vorgänger eines Knotens und wird so vorgenommen, dass jeweils möglichst viele Slots für die beiden vorausgehenden Links übrigbleiben. Dies entspricht im Wesentlichen den in [SCCC06] verwendeten Policies 3BDP und LCFP, so dass das SSSR-Problem abgeschwächt wird. Jedoch werden in [ZL13] die Slots für die Route vom Multicast-Baum zum jeweils neuen Ziel in der falschen Richtung ermittelt (d. h. es werden Slots zum Senden vom neuen Ziel zum Multicast-Baum reserviert anstatt umgekehrt). Zudem werden Fehlreservierungen, die durch das Selbstinterferenzproblem zustande kommen, nicht berücksichtigt. Weiterhin wird keine Aussage darüber getroffen, wie ein Knoten des Multicast-Baums, bei dem ein Abzweig beginnt, an seine jeweiligen Empfänger sendet. Es könnten entweder für jeden Empfänger eigene Slots reserviert oder nach Möglichkeit lokale Multicasts verwendet werden.

Kapitel 5.

5 Reservation-enhanced Black-Burst-based QoS-Routing (RBBQR)

In diesem Kapitel wird das im Rahmen dieser Arbeit entwickelte *Reservation-enhanced Black-Burst-based QoS-Routing* (RBBQR) beschrieben. Dabei handelt es sich um ein verteiltes reaktives QoS-Routing-Protokoll mit integriertem Reservierungsprotokoll, welches die beiden in Abschnitt 3.1.1 beschriebenen Kernprobleme (parallele Routensuchen sowie Kollisionen von Kontrollpaketen) löst und gleichzeitig deterministische Reservierungen in Form von wettbewerbsfreien virtuellen Slotregionen der MAC-Schicht (Datenslots) vornimmt. Da die Routensuche dezentralisiert stattfindet, ist eine Verknüpfung von Routing- und Reservierungsprotokoll erforderlich. Um Datenslots kollisionsfrei reservieren zu können, wird das in Kapitel 2 eingeführte Interferenzmodell verwendet.

Als Grundlage für RBBQR dient das in [Bir09, BBCG11] vorgestellte *Black-Burst-based QoS-Routing* (BBQR), welches ausschließlich statistische Reservierungen in Form von abstrakten Zahlenwerten vornimmt (s. Abschnitt 4.1.1). Die Integration von deterministischen Reservierungen erfordert umfangreiche Änderungen, so dass BBQR und RBBQR fundamentale Unterschiede aufweisen. Beibehalten wurde jedoch die Serialisierung von Routensuchen, da die Unterstützung paralleler Routensuchen aufgrund der zahlreichen dabei auftretenden Probleme (vgl. Abschnitt 3.1.1) schwierig zu realisieren ist.

Ein Schwerpunkt bei der Entwicklung von RBBQR bestand darin, den Versand der Kontrollpakete deterministisch und kollisionsfrei zu realisieren. Dazu dienen die in Abschnitt 3.3.2 beschriebenen Black-Burst-basierten Transferprotokolle Arbitrating Transfer, Cooperative Transfer und Bit-based Transfer (der Bit-based Transfer wurde eigens für RBBQR definiert). Zudem wird das Scheduling der Pakete auf MAC-Ebene detailliert betrachtet.

Während BBQR fast die gesamte Reservierungsinformation mittels Black-Burst-basierter Transferprotokolle versendet, bestand ein zusätzliches Ziel bei der Entwicklung von RBBQR darin, deren Verwendung so weit wie möglich zu reduzieren (da sie einen hohen Aufwand verursachen und RBBQR mehr Informationen propagieren muss als BBQR). Aus diesem Grund werden die Kontrollinformationen aufgespalten, so dass jeweils ein möglichst großer Teil in reguläre Pakete ausgelagert werden kann.

Zusätzlich wird für RBBQR berücksichtigt, dass etablierte Routen wieder gelöscht werden können (explizit oder implizit durch Nichtbenutzung) und dass Routenbrüche auftreten können. Dazu müssen zusätzliche Verwaltungsinformationen ausgetauscht werden. Um das Scheduling der Kontrollpakete möglichst effizient umsetzen zu können, wird eine eigens angepasste MAC-Schicht verwendet.

In diesem Kapitel werden zunächst einige für RBBQR erforderliche Voraussetzungen definiert. Danach werden die Routensuche mitsamt der Reservierung von Datenslots, die Datenübertragung sowie die Freigabe von Routen und Reservierungen betrachtet. Zudem werden Aufbau und Konfiguration einer mit RBBQR kompatiblen MAC-Schicht beschrieben, und es werden einige konkrete Berechnungen durchgeführt. Um das Protokoll zu evaluieren, wurde

dieses mittels SDL (s. Abschnitt 3.2) formal spezifiziert und funktional simuliert. Ausschnitte der Spezifikation sowie einer funktionalen Simulation werden ebenfalls vorgestellt.

Aus Gründen der Übersichtlichkeit sind alle von RBBQR verwendeten Paketformate in Anhang A zusammengefasst (sowohl die intern verwendeten Paketformate als auch die Schnittstellen zu Anwendung und MAC-Schicht). Die Tabellen, welche zur Speicherung von Routenanforderungen, Routen und Slotreservierungen dienen, sind in Anhang B zu finden. Anhang C beinhaltet weitere Details der SDL-Spezifikation; Anhang D enthält Ergänzungen zu der in diesem Kapitel vorgestellten sowie weitere funktionale Simulationen.

5.1. Voraussetzungen für RBBQR

Neben den in Abschnitt 2.2.1 formulierten allgemeinen Annahmen müssen für RBBQR einige weitere Voraussetzungen erfüllt werden:

1. Das Auftreten von Routenbrüchen (aufgrund von Knotenausfällen oder Linkbrüchen durch sich auseinander bewegende Knoten) wird behandelt (s. Abschnitt 5.4.3). Knotenausfälle und Linkbrüche dürfen jedoch nicht während einer Routensuche auftreten.

2. Knoten dürfen sich nicht so weit aufeinander zu bewegen, dass neue Interferenzen entstehen.

3. Das nachträgliche Hinzufügen neuer Knoten ist nicht erlaubt.

4. Die Uhren der einzelnen Knoten sind durch ein regelmäßig ausgeführtes Zeitsynchronisationsprotokoll (z. B. BBS [GK11]) hinreichend genau synchronisiert.

5. In Phasen, in denen RBBQR aktiv ist, ist kein weiteres Protokoll aktiv. Dies wird durch die Konfigurierung virtueller Slotregionen in der MAC-Schicht erreicht (vgl. Abschnitt 5.5).

6. Der maximale Netzwerkdurchmesser ($n_{maxHops}$) sowie die maximale Anzahl an Knoten im Netz ($n_{maxNodes}$) müssen allen Knoten bekannt sein.

7. Als QoS-Metrik wird die Bandbreite in Form von zu reservierenden Datenslots unterstützt, welche durch virtuelle Slotregionen der MAC-Schicht umgesetzt sind (vgl. Abschnitt 2.1).

8. Jeder Knoten hat eine global eindeutige ID, die statisch konfiguriert wird. Diese ID kann von allen Schichten genutzt werden (z. B. von der Anwendung zur Adressierung der Pakete oder von RBBQR zur Identifizierung des jeweiligen Nachfolgers auf der Route).

9. Es wird angenommen, dass die in Abschnitt 2.4.2 formulierte 2-Hop-Interferenzannahme $\forall\, a \in V : IN_1(a) = CN_1(a) \cup CN_2(a)$ gilt. Dies ist jedoch keine harte Anforderung, sondern dient als Abschätzung für die Wiederverwendung von Slots auf einer Route (vgl. Abschnitt 2.7). Ist $\forall\, a \in V : IN_1(a) \supseteq CN_1(a) \cup CN_2(a)$ nicht erfüllt, so bewirkt dies lediglich eine schlechtere Ausnutzung der Bandbreite. Ist $\forall\, a \in V : IN_1(a) \subseteq CN_1(a) \cup CN_2(a)$ nicht erfüllt, so ist es möglich, dass Slots zu früh für die Wiederverwendung auf der entsprechenden Route vorgesehen werden. Daraus entstehende Fehlreservierungen werden jedoch erkannt, bevor die Route in Betrieb genommen wird, und es werden entsprechende Gegenmaßnahmen getroffen (eine detaillierte Beschreibung ist

in Abschnitt 5.2.3.2 zu finden). Auf diese Weise werden keine Routen etabliert, die Fehlreservierungen aufgrund des Selbstinterferenzproblems aufweisen.

10. Es wird hier davon ausgegangen, dass Black-Burst-basierte Übertragungen stets zuverlässig sind, d. h. dass es keine False Positives (durch Hintergrundrauschen fälschlich erkannte Black Bursts) oder False Negatives (aufgrund von zu geringer Signalstärke nicht erkannte Black Bursts) gibt.

    Um feststellen zu können, ob ein Rahmen korrekt empfangen wurde, kann in der MAC-Schicht Redundanz (z. B. in Form von Prüfbits) hinzugefügt werden. Weiterhin kann man die Empfangssignalstärke sowie den Empfangszeitpunkt der einzelnen Bits als Kriterium verwenden. Wurde der Rahmen nicht korrekt empfangen, so wird er von der MAC-Schicht verworfen.

11. Auch bei regulären Übertragungen wird davon ausgegangen, dass diese stets korrekt empfangen werden, d. h. dass keine Rahmen verlorengehen oder Bits verfälscht werden. Dies kann mit geeigneten Mitteln (z. B. Verwendung von CRC) sichergestellt werden.

5.2. Routensuche und Reservierung von Datenslots

Während BBQR die Routensuche in vier Phasen durchführt, werden bei RBBQR nur drei Phasen benötigt. Der Standardablauf einer erfolgreichen Routensuche ist in Abbildung 5.1 dargestellt und wird im Folgenden kurz erläutert. Da einige der Pakete aus zwei Teilen bestehen, welche mit verschiedenen Übertragungsverfahren versendet werden, wird hier von den Übertragungsverfahren abstrahiert. In den Abschnitten 5.2.1, 5.2.2 und 5.2.3 werden die einzelnen Phasen im Detail beschrieben. Dabei wird auch auf die verwendeten Übertragungsverfahren, die Behandlung von Fehlerfällen sowie das Scheitern der Routensuche eingegangen.

Phase 1	Phase 2		Phase 3	
RREQ^l	RREP^m	RSTAT(RFND)	CREQ^n	RSTAT(FIN_SUCC)

Abbildung 5.1.: Standardablauf einer erfolgreichen Routensuche (die Parameter l, m und n beziehen sich darauf, dass jeweils mehr als ein Paket versendet wird).

In Phase 1 wird eine der vorliegenden Routenanforderungen mittels Arbitrierung ausgewählt und durch das Netz propagiert (mit Route Request (RREQ)-Paketen). Nach Abschluss dieser Phase kennen alle Knoten die gerade bearbeitete Routenanforderung (d. h. Quelle, Ziel und QoS-Anforderung) sowie ihre Kommunikationsdistanz zur Quelle. Die Codierung der QoS-Anforderung (QoS-Metrik und QoS-Wert) ist analog zu BBQR (s. Abschnitt 4.1.1). In Phase 2 werden (ausgehend vom Ziel) passende Routen mitsamt den entsprechenden Datenslots gesucht und zur Quelle propagiert (mit Route Reply (RREP)-Paketen). Jeder Knoten wählt dabei die Slots für seinen 3-Hop-Nachfolger aus. Zur effizienten Ermittlung der freien Slots entlang der Route (vom Ziel zur Quelle) wird die in Abschnitt 2.5 beschriebene Form der globalen Reservierungsbedingung verwendet. Sobald eine Route gefunden wurde, wird dies (mit Hilfe eines generischen Route Status (RSTAT)-Pakets) im gesamten Netz bekannt gegeben (Route Found (RFND)). Damit ist Phase 2 beendet. In Phase 3 wird die gefundene Route ausgehend von der Quelle allen beteiligten Knoten mitgeteilt (mit Construction Request (CREQ)-Paketen). Dabei werden jedem Knoten seine Slots mitgeteilt, und Reservierungen werden an die Interferenznachbarn propagiert. Treten dabei Probleme auf, so werden diese

behandelt. Anschließend wird das Ende von Phase 3 und damit das Ende der Routensuche (ebenfalls mit Hilfe eines RSTAT-Pakets) im gesamten Netz bekannt gegeben (Finish Success (FIN_SUCC)).

Anmerkung: In [Nis08,NG09] wurde gezeigt, dass die Verwendung von Multipfaden (d. h. die Aufspaltung einer Route auf mehrere Teilrouten, die zusammen die benötigte Bandbreite liefern) in zufälligen Topologien nur geringe Verbesserungen erzielt. Deshalb wurden für RBBQR keine Multipfade betrachtet.

Anmerkung: Für die Implementierung der verschiedenen von RBBQR verwendeten Pakete werden TypIDs definiert. Sie sind nicht für alle Pakete erforderlich, da bei RBBQR aus dem Empfangszeitpunkt meist eindeutig hervorgeht, welcher Pakettyp gerade empfangen wird. Sie dienen jedoch auch dazu, den Paketen Redundanz hinzuzufügen. Konzeptuell entspricht die TypID dem Pakettyp, weswegen sie im Folgenden nicht separat aufgeführt wird. Bei der Berechnung der Paketgrößen muss sie jedoch berücksichtigt werden.

5.2.1. Phase 1 der Routensuche

Phase 1 dient dazu, eine Routenanforderung netzweit auszuwählen und alle Knoten des Netzes über diese zu informieren (IDs von Quelle und Ziel sowie QoS-Metrik und QoS-Wert). Außerdem bestimmt in dieser Phase jeder Knoten seine Kommunikationsdistanz zur Quelle (QHopCount), welche für Phase 2 benötigt wird.

5.2.1.1. Übertragung und Paketformat

Das RREQ-Paket wird in zwei Teile zerlegt, RREQA (RREQ Arbitration) und RREQD (RREQ Data). Abbildung 5.2 zeigt den Ablauf von Phase 1.

Phase 1				
RREQA	RREQD	RREQD	...	RREQD

Abbildung 5.2.: Ablauf von Phase 1.

Das Paketformat hat folgende Form:
RREQA (QuellID : NodeId)
RREQD (ZielID : NodeId, QHopCount : HopCnt,
QoSMetrik : QosMetric, QoSWert : QosVal)

Da mehrere Knoten gleichzeitig versuchen könnten, eine Routensuche zu starten, wird zunächst mittels RREQA ein Arbitrating Transfer über $n_{maxHops}$ Hops durchgeführt, bei dem alle potenziellen Quellen ihre ID senden. Nachdem eine Quelle gewonnen hat, sendet diese ein RREQD, welches per $n_{maxHops}$ Mal wiederholtem Cooperative Transfer (jeweils über einen Hop) an alle Knoten im Netz übertragen wird. Hier kommt der Cooperative Transfer zum Einsatz, da dieser etwas weniger Aufwand verursacht als der Arbitrating Transfer und zu einem Zeitpunkt nur gleiche RREQDs gesendet werden (die Verwendung regulärer Übertragungen ist nicht möglich, da es jeweils mehrere Empfänger und damit mehrere gleichzeitig weiterleitende Knoten geben kann). RREQD enthält neben der ID des Ziels, der QoS-Metrik und dem QoS-Wert, welche gleich bleiben, auch den QHopCount, welcher bei jeder Weiterleitung um 1 erhöht wird.

Damit jeder Knoten auf diese Weise seine Distanz zur Quelle ermitteln kann, muss die Black-Burst-Reichweite für die Übertragung der RREQD-Pakete auf die Kommunikations-

reichweite eingeschränkt werden. Diese Einschränkung der Black-Burst-Reichweite wird nur für die RREQD-Pakete verwendet, da sie ansonsten für RBBQR nicht sinnvoll ist.

Eine Alternative (welche hier jedoch nicht umgesetzt wurde) besteht darin, mit dem *Automatic Topology Discovery Protocol* (ATDP) [Kra13,KCG15] die Topologie zu ermitteln. Dann kennt jeder Knoten die Hopdistanz zu allen anderen Knoten, so dass RBBQR diese Information nicht propagieren muss. In diesem Fall kann QHopCount aus RREQD entfernt und ein einziges RREQD-Paket per Cooperative Transfer über $n_{maxHops}$ Hops übertragen werden. Die Einschränkung der Black-Burst-Reichweite wird dann nicht benötigt.

5.2.1.2. Dauer von Phase 1

Die Dauer von Phase 1 hängt von der maximalen Anzahl Knoten sowie vom maximalen Netzwerkdurchmesser ab, ist für ein gegebenes Netz jedoch konstant. Sie berechnet sich als

$$d_{Phase1} = d_{RREQA} + n_{maxHops} \cdot d_{RREQD}$$

$$d_{RREQA} = n_{RREQABits} \cdot n_{maxHops} \cdot d_{arbBurst} \quad \text{(vgl. Formel (3.1))} \tag{5.1}$$

$$d_{RREQD} = n_{RREQDBits} \cdot d_{coopBurst} + d_{procCoop} \quad \text{(vgl. Formel (3.2))} \tag{5.2}$$

$$n_{RREQABits} = n_{typeIdBits} + n_{nodeIdBits}$$

$$n_{RREQDBits} = n_{typeIdBits} + n_{nodeIdBits} + n_{hopCntBits} + n_{qosMetricBits} + n_{qosValBits}\,.$$

Dabei ist $n_{typeIdBits}$ die Anzahl Bits, die zur Codierung der verschiedenen von RBBQR verwendeten Pakete benötigt wird. Da sich die Anzahl der zur Codierung einer natürlichen Zahl n benötigten Bits als $\lceil \log_2(n+1) \rceil$ berechnet und es für RBBQR 18 verschiedene Pakettypen (codiert durch die Werte 0 bis 17) gibt (vgl. Anhang A.1), gilt

$$n_{typeIdBits} = \lceil \log_2(18) \rceil = 5\,.$$

$n_{nodeIdBits}$ ist die Anzahl Bits zur Codierung einer Knoten-ID. Da das betrachtete Netz maximal $n_{maxNodes}$ Knoten hat (d. h. NodeId kann Werte von 0 bis $n_{maxNodes} - 1$ annehmen), gilt

$$n_{nodeIdBits} = \lceil \log_2(n_{maxNodes}) \rceil\,.$$

$n_{hopCntBits}$ ist die Anzahl Bits zur Codierung einer Hopdistanz. Da $n_{maxHops}$ der maximale Netzwerkdurchmesser ist (d. h. HopCnt kann Werte von 0 bis $n_{maxHops}$ annehmen), gilt

$$n_{hopCntBits} = \lceil \log_2(n_{maxHops} + 1) \rceil\,.$$

$n_{qosMetricBits}$ bzw. $n_{qosValBits}$ ist die Anzahl der zur Codierung einer QoS-Metrik bzw. eines QoS-Werts verwendeten Bits. Die Anzahl dieser Bits muss festgelegt werden.

Anmerkung: Der Einfluss der MAC-Schicht wird hier nicht berücksichtigt. Auf diese Aspekte wird in Abschnitt 5.5 eingegangen.

5.2.2. Phase 2 der Routensuche

In Phase 2 werden, ausgehend vom Zielknoten z, Routen zur Quelle q gesucht. Sofern z lokal die QoS-Anforderung erfüllen kann (d. h. wenn bei einer QoS-Anforderung von n Slots noch mindestens n freie Empfangsslots zur Verfügung stehen), legt er einen Eintrag in seiner

temporären Routentabelle an und sendet ein RREP-Paket mit seinen freien Empfangsslots ($F_{RX}(z)$) per Broadcast (das RREP-Paket ist wie RREQ zweigeteilt; der Grund dafür wird in Abschnitt 5.2.2.3 beschrieben). Jeder der empfangenden Nachbarknoten a berechnet daraus und aus seinen lokalen Reservierungsinformationen die freien Sendeslots für den entsprechenden Link ($F_{TX}(a, z)$) (zur Berechnung der freien Sende- und Empfangsslots s. Abschnitt 2.5). Kann die QoS-Anforderung weiterhin erfüllt werden, d. h. enthalten $F_{TX}(a, z)$ und $F_{RX}(a)$ jeweils mindestens n unterschiedliche Slots (notwendiges Kriterium für erfolgreiche Routensuche; vgl. Abschnitt 2.7), so legt a einen Eintrag in seiner temporären Routentabelle an und sendet selbst wieder ein RREP. Wenn ein Knoten lokal feststellt, dass die QoS-Anforderung auf der bisherigen Route nicht mehr erfüllt werden kann, so verwirft er das Paket.

5.2.2.1. Heuristiken zur Slotauswahl

Um aus den freien Slots eine Auswahl zu treffen, gibt es verschiedene Möglichkeiten. Beispielsweise kann diese Auswahl so erfolgen, dass eine möglichst gute Wiederverwendung erreicht wird. Alternativ könnte die Latenz optimiert werden, indem die Slots der einzelnen Knoten einer Route möglichst in aufsteigender Reihenfolge gewählt werden. Hier wird eine Mischung dieser beiden Kriterien verwendet, wobei die Slotwiederverwendung höhere Priorität hat. Dazu werden drei Heuristiken verwendet, von denen die ersten beiden der Slotwiederverwendung dienen und auf Heuristiken aus [SCCC06] basieren.

3FDP

In [SCCC06] wird die *3-Hop Backward Decision Policy* (3BDP) definiert, bei der die Sendeslots für einen Knoten jeweils von dessen 3-Hop-Nachfolger festgelegt werden. Dadurch können für die Auswahl der Sendeslots auch die für die nächsten beiden Knoten verfügbaren Slots berücksichtigt werden, so dass die Auswahl im Hinblick auf die Slotwiederverwendung optimiert werden kann. In [SCCC06] werden die Slots von der Quelle zum Ziel propagiert. Da bei RBBQR die Slots vom Ziel zur Quelle propagiert werden, wird 3BDP in eine *3-Hop Forward Decision Policy* (3FDP) umgewandelt, bei der die Sendeslots für einen Knoten jeweils von dessen 3-Hop-Vorgänger auf der Route festgelegt werden. Dazu werden die freien Sendeslots im RREP-Paket jeweils über drei Hops propagiert.

3FDP berücksichtigt die 2-Hop-Interferenzannahme, aufgrund derer ein Slot nur auf einem von vier aufeinanderfolgenden Links nutzbar ist (s. Abschnitt 2.7). In [SCCC06] wird die 1-Hop-Interferenzannahme zugrunde gelegt, weshalb ein Slot auf *drei* aufeinanderfolgenden Links einmal nutzbar ist. Die Abbildungen 5.3 und 5.4 verdeutlichen dies. Der unterbrochene Pfeil bedeutet, dass d die Sendeslots für a (3BDP) bzw. a die Slots für d (3FDP) festlegt.

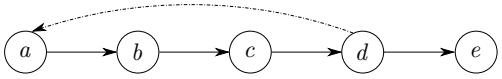

Abbildung 5.3.: 3BDP.

d legt die Slots für den Link (a, b) fest. Diese sind dann für (b, c) und (c, d) nicht nutzbar. Für (d, e) können sie wiederverwendet werden.

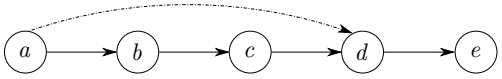

Abbildung 5.4.: 3FDP.

a legt die Slots für den Link (d, e) fest. Diese sind für (c, d), (b, c) und (a, b) nicht nutzbar (letzteres ist der Fall, da b und d sich in Interferenzreichweite zueinander befinden können).

Anmerkung: Da jeder Knoten die Sendeslots für seinen 3-Hop-Nachfolger berechnet, ken-

nen die Knoten ihre eigenen Slots zunächst noch nicht. Diese werden in Phase 3 bei der Etablierung der Route propagiert.

LCFP

Die *Least Conflict First Policy* (LCFP) wird von [SCCC06] fast unverändert übernommen. Gemäß LCFP werden für einen Link diejenigen Slots ausgewählt, welche die wenigsten Konflikte mit den nächsten beiden Links haben. Durch die Umwandlung von 3BDP in 3FDP sind dies hier die drei vorhergehenden Links. Auf diese Weise bleiben möglichst viele Slots für die anderen Links übrig.

Im obigen Beispiel würde bei 3BDP Knoten d für den Link (a, b) diejenigen Slots auswählen, die möglichst wenige Konflikte mit (b, c) und (c, d) haben. Bei 3FDP würde Knoten a diejenigen Slots für (d, e) wählen, die möglichst wenige Konflikte mit (a, b), (b, c) und (c, d) haben.

LCFP bewirkt eine Abschwächung des SSSR-Problems (s. Abschnitt 3.1.2), was durch das Beispiel in Abbildung 5.5 gezeigt wird.

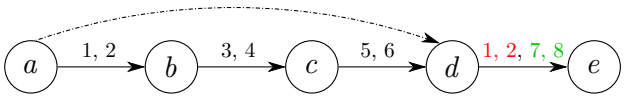

Abbildung 5.5.: Abschwächung von SSSR durch LCFP.

Angenommen, der QoS-Wert betrage zwei Slots. Da 3FDP verwendet wird, legt a die Slots für (d, e) fest. Würden die Slots 1 und 2 gewählt, so wären für (a, b) keine Slots mehr übrig. Aufgrund von LFCP werden aber stattdessen die Slots 7 und 8 gewählt, da diese – im Gegensatz zu 1 und 2 – keine Konflikte mit den drei vorhergehenden Links haben. Auf diese Weise wird die Route erfolgreich etabliert.

Um neben dem SSSR-Problem auch das SSNR-Problem (s. Abschnitt 3.1.2) abzuschwächen, könnte man beispielsweise die ebenfalls in [SCCC06] vorgestellte *Most Reuse First Policy* (MRFP) implementieren. MRFP verwendet eine sogenannte Utilization Rate, um nach Anwendung von LCFP diejenigen Slots auszuwählen, die bereits von 1-Hop-Nachbarn verwendet werden. Da die Sendeslots eines Knotens jedoch nicht von diesem selbst festgelegt werden, muss die Utilization Rate für die einzelnen Slots in den Routingpaketen mit übertragen werden. Um diesen zusätzlichen Kommunikationsaufwand zu sparen, wurde MRFP in RBBQR bisher nicht umgesetzt.

SDFP

Die *Shortest Delay First Policy* (SDFP) dient dazu, die Latenz zu optimieren. Dazu werden (nach Anwendung von LCFP) die Slots möglichst in absteigender Reihenfolge gewählt (da die Route vom Ziel aus gesucht wird). Auf diese Weise wird in jedem Knoten die Zeitspanne zwischen dem Empfang eines Pakets und dessen Weiterleitung minimiert. Jeder Knoten sendet dazu den kleinsten Sendeslot (für den 3-Hop-Nachfolger) im RREP-Paket an seinen Vorgänger, so dass dieser nach Anwendung von LCFP die Slotauswahl für seinen 3-Hop-Nachfolger beim größten Slot, der kleiner als der vom Nachfolger mitgeteilte kleinste Sendeslot ist, beginnen kann.

5.2.2.2. Identifikation der Routen

Um Routen global eindeutig identifizieren zu können, wird in jedem Knoten für jede Route eine lokal eindeutige RouteID vergeben, welche im RREP zusammen mit der ID des Knotens

übertragen wird. Der Vorgänger auf der Route vergibt eine eigene lokal eindeutige RouteID und zeichnet diese zusammen mit ID und RouteID des Nachfolgers auf. Die Auswahl einer RouteID in einem Knoten legt jeweils den Rest der Route eindeutig fest (insbesondere gilt das auch für die Quelle). Auf diese Weise kann jede Route global eindeutig identifiziert werden.

5.2.2.3. Übertragung und Paketformat

Da in Phase 2 – ausgehend vom Ziel – Routen gesucht werden, muss jeder Knoten freie Sende- und / oder Empfangsslots propagieren. Somit senden alle Knoten unterschiedliche Pakete. Da die Pakete zudem jeweils nur über einen Kommunikationshop propagiert werden müssen, ist es hier sinnvoll, die effizienteren regulären Übertragungen statt der Black-Burst-basierten Transferprotokolle zu verwenden. Zudem hat dies den Vorteil, dass (auch ohne eine Einschränkung der Reichweite) nur die Kommunikationsnachbarn erreicht werden, was für die Routensuche erforderlich ist.

Es muss jedoch sichergestellt werden, dass keine Kollisionen auftreten. Dazu wird das RREP-Paket (analog zu RREQ) in zwei Teile, RREPA (RREP Arbitration) und RREPD (RREP Data), zerlegt. Zusätzlich wird ein RSTAT-Paket (Route Status) verwendet, um das Ende von Phase 2 im Netz bekanntzugeben. Abbildung 5.6 zeigt den Ablauf von Phase 2 im Fall einer erfolgreichen Routensuche.

Phase 2					
RREPA	RREPD	...	RREPA	RREPD	RSTAT(RFND)

Abbildung 5.6.: Ablauf von Phase 2 (Routensuche erfolgreich).

Das Paketformat hat folgende Form:
RREPA(QHopCountInv : HopCnt, SenderID : NodeId)
RREPD(SenderID : NodeId, SenderRouteID : RouteId,
RouteLenZiel : RouteLen, MaxRouteLength : RouteLen,
NumberOfSlots(TX2HopNachfolger)[1] *: SlotCnt*, TXSlots(2HopNachfolger) : SlotSet,
NumberOfSlots(TX1HopNachfolger) : SlotCnt, TXSlots(1HopNachfolger) : SlotSet,
NumberOfSlots(TXSender) : SlotCnt, TXSlots(Sender) : SlotSet,
NumberOfSlots(RXSender) : SlotCnt, RXSlots(Sender) : SlotSet,
LowestUsedSlot : SlotNr)
RSTAT(Type : RType) mit RType = {RFND, FIN_SUCC, FIN_FAIL}

RREPA wird per Arbitrating Transfer über zwei Interferenzhops übertragen. Das Paket enthält das logische Inverse der aus Phase 1 bekannte Kommunikationsdistanz zur Quelle, da eine geringere Entfernung zur Quelle bei der Arbitrierung bevorzugt werden soll. Dieses ist definiert als QHopCountInv $=_{Df}$ $n_{maxHops}$ - QHopCount ($n_{maxHops}$ ist die Obergrenze für QHopCount). QHopCountInv ist jedoch nur für die Umsetzung relevant; für die folgenden konzeptuellen Betrachtungen und Beispiele wird der Einfachheit halber QHopCount verwendet. Neben QHopCountInv enthält RREPA die ID des Senders.

Der Gewinner der Arbitrierung überträgt sein RREPD als reguläres Paket an alle Kommunikationsnachbarn. Durch den Arbitrating Transfer über zwei (Interferenz)Hops wird das Hidden-Station-Problem gelöst. Es ist jedoch möglich, dass ein Phantomrahmen empfangen wurde, der keinem tatsächlich gesendeten entspricht (vgl. Abschnitt 3.3.2.3). Deshalb muss

[1] Die Kardinalitäten von Slotmengen werden kursiv dargestellt, da sie auf konzeptueller Ebene nicht benötigt werden, jedoch für die Implementierung erforderlich sind.

die im RREPA-Paket enthaltene ID des Senders zusätzlich im RREPD gesendet werden. Die Distanz zur Quelle wird dagegen nur zur Arbitrierung benötigt und muss somit nicht erneut gesendet werden.

Anmerkung: Es wird hier kein EOF-Bit benötigt, da die Knoten nicht wissen müssen, ob sie sich in einem Umkreis von zwei Hops um einen Gewinner befinden. Gewinner der Arbitrierung senden ein RREPD, während alle anderen Knoten abwarten, ob sie ein RREPD empfangen.

RREPD enthält neben der ID und RouteID des Senders die aktuelle Routenlänge zum Ziel (RouteLenZiel), welche nicht notwendigerweise der Distanz des Knotens zum Ziel entspricht. RouteLenZiel wird verwendet, um zu entscheiden, ob ein Paket weitergeleitet wird (dies wird in Abschnitt 5.2.2.4 detaillierter betrachtet). Weiterhin ist die maximale Routenlänge (MaxRouteLength) enthalten, welche die maximale Anzahl Hops auf der Route angibt und von der Distanz zwischen Quelle und Ziel abhängt. Diese Distanz ist dem Zielknoten in Form von QHopCount bekannt, woraus er MaxRouteLength berechnet und im RREPD-Paket propagiert. MaxRouteLength kann z. B. als MaxRouteLength $=_{Df}$ QHopCount + 2 oder als MaxRouteLength $=_{Df}$ 2 $\cdot$ QHopCount definiert werden.

Zusätzlich sind im RREPD-Paket die freien Sendeslots des 2-Hop-Nachfolgers, des 1-Hop-Nachfolgers und des Senders (zum Senden an den jeweiligen Routennachfolger) sowie die freien Empfangsslots des Senders (zum Empfangen von einem beliebigen Kommunikationsnachbarn) enthalten. Dies ist erforderlich, da die Sendeslots für einen Knoten gemäß 3FDP jeweils von dessen 3-Hop-Vorgänger festgelegt werden. Die Kardinalitäten der Slotmengen werden zwar konzeptuell nicht benötigt (da sie anhand der Mengen ermittelt werden können), müssen aber bei der Umsetzung des Protokolls berücksichtigt werden, da sich daraus die Länge des RREPD-Pakets ergibt. Zudem müssen die einzelnen Slots so ins RREPD-Paket geschrieben werden, dass rekonstruiert werden kann, zu welcher Slotmenge diese jeweils gehören.

Weiterhin wird der kleinste für den 3-Hop-Nachfolger vergebene Sendeslot (LowestUsedSlot) übertragen, damit die Slots entlang der Route mittels SDFP möglichst in aufsteigender Reihenfolge vergeben werden können.

Erhält die Quelle ein RREPD, so wurde eine Route gefunden, und die Quelle sendet ein RSTAT-Paket mit dem Parameter RFND (Route Found). Nach Empfang von RSTAT(RFND) wissen alle Knoten, dass Phase 2 beendet ist (sie senden also keine RREPs mehr) und Phase 3 beginnt. Auf diese Weise wird immer die erste gefundene Route gewählt und die Dauer von Phase 2 minimiert. RSTAT ist ein generisches Paketformat für Statusinformationen und wird per Cooperative Transfer über $n_{maxHops}$ Hops gesendet. In Abschnitt 5.2.2.5 wird beschrieben, wie RSTAT gegenüber weiteren RREPAs priorisiert wird.

5.2.2.4. Weiterleitung der RREP-Pakete

Die Verwendung von QHopCount (bzw. QHopCountInv) zur Arbitrierung dient dazu, eine möglichst kurze Route zu wählen (sofern mindestens eine hop-minimale Route existiert, welche die QoS-Anforderung erfüllt, wird eine solche gewählt). Zudem sorgt dies dafür, dass im Falle mehrerer konkurrierender Sender eine der Routen bevorzugt propagiert wird (da einer der Empfänger einen geringeren QHopCount als die übrigen Sender hat, so dass im nächsten Schritt dieselbe Route wieder gewählt wird). Auf diese Weise endet Phase 2 möglichst schnell, da ein „Gleichschritt“ mehrerer benachbarter Routen (bei dem jeweils abwechselnd einer der Knoten sendet) vermieden wird. Dies wird anhand des Netzes in Abbildung 5.7 verdeutlicht.

Für das Netz gelten die folgenden allgemeinen Annahmen:

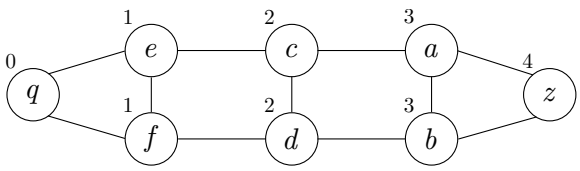

Abbildung 5.7.: Beispiel für die Verwendung von QHopCount zur Arbitrierung.

- Es gelte $\forall\, a \in V : SN_1(a) = IN_1(a) = CN_1(a) \cup CN_2(a)$. Somit erfolgt die Arbitrierung über vier Kommunikationshops (Abbildung 5.7 zeigt nur die Kommunikationslinks).
- Die Zahlen an den Knoten geben jeweils den QHopCount an.
- Für die IDs gelte $a > b > c > d > e > f$.

In diesem Beispiel wird davon ausgegangen, dass jeder Knoten die QoS-Anforderung erfüllen kann. Zunächst sendet z sein RREPD (keine konkurrierenden Sender). Anschließend findet eine Arbitrierung zwischen a und b statt. Da der QHopCount gleich ist, gewinnt die höhere ID, also a. Im nächsten Schritt konkurrieren dann b und c. c gewinnt aufgrund des geringeren QHopCount. Analoges gilt im nächsten Schritt für b und e. Auf diese Weise wird die Route $q \rightarrow e \rightarrow c \rightarrow a \rightarrow z$ ohne Unterbrechungen durch Sendevorgänge anderer Routen bis zur Quelle propagiert.

Anmerkung: In diesem Beispiel ist die gefundene Route zu kurz für parallele Sendevorgänge. Bei längeren Routen sind solche möglich; jedoch findet spätestens bei der Quelle wieder eine Serialisierung statt.

Um zu entscheiden, ob ein RREPD weitergeleitet wird, wird die aktuelle Routenlänge zum Ziel (RouteLenZiel) verwendet. Ein RREPD wird verworfen, wenn seine Routenlänge MaxRouteLength überschreitet oder wenn der entsprechende Knoten bereits ein RREPD mit kleinerer Routenlänge weitergeleitet hat. Letzteres sorgt für die Vermeidung von Routingschleifen. Das spätere Auftreten eines RREPDs mit gleicher Routenlänge tritt ein, wenn es mehrere gleich lange Routen gibt, die als Kandidaten in Frage kommen. Das spätere Auftreten einer kleineren Routenlänge ist jedoch nicht möglich, da optimale Routen immer zuerst gewählt werden. Für diese ist die Routenlänge minimal.

Anmerkung: Hat ein Knoten ein RREPD zu senden, so tritt er solange in den Wettbewerb (mittels RREPA), bis er dieses gesendet hat. Liegen mehrere RREPDs vor, so werden diese in FIFO-Reihenfolge bearbeitet. Im obigen Beispiel hat b zwei RREPDs zu senden, nachdem z und a gesendet haben. Das Versenden des RREPDs für die Route, die z als Nachfolger hat, führt jedoch dazu, dass das andere RREPD anschließend aufgrund der größeren Routenlänge verworfen wird.

Prinzipiell könnte auch QHopCount für die Weiterleitungsentscheidung verwendet werden, jedoch wäre es dann nicht mehr möglich, eine Route zu finden, wenn keine der hop-minimalen Routen die QoS-Anforderung erfüllt. Dies wird anhand des Netzes in Abbildung 5.8 verdeutlicht. Für dieses gelten dieselben allgemeinen Annahmen wie für das Netz in Abbildung 5.7.

In diesem Beispiel können alle Knoten mit Ausnahme von b die QoS-Anforderung erfüllen. Zunächst sendet z sein RREPD und anschließend a. Da b die QoS-Anforderung nicht erfüllen kann, kommt die hop-minimale Route nicht zustande. Würde QHopCount für die Weiterleitungsentscheidung verwendet, so könnte keine Route gefunden werden, da c einen QHopCount von 4 und a einen von 3 hat, d. h. c würde ein Paket mit einem kleineren QHopCount als sein

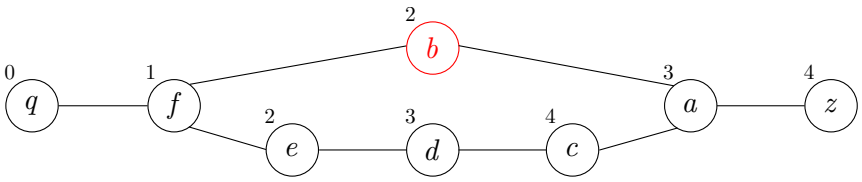

Abbildung 5.8.: Gegenbeispiel für die Verwendung von QHopCount zur Weiterleitung.

eigener erhalten und dieses verwerfen. Bei Verwendung der Routenlänge als Kriterium kann c das RREPD von a jedoch erfolgreich weiterleiten.

5.2.2.5. Arbitrierung zwischen RREPA und RSTAT

Um ein RREPD-Paket kollisionsfrei versenden zu können, wird mittels RREPA eine Arbitrierung über zwei Interferenzhops vorgenommen. Da parallel zu RREPA jedoch ein RSTAT-Paket gesendet werden könnte (per Cooperative Transfer über $n_{maxHops}$ Hops), ist eine netzweite Arbitrierung zwischen RREPA und RSTAT erforderlich. Eine Lösung bestünde darin, sowohl RREPA als auch RSTAT per Arbitrating Transfer über $n_{maxHops}$ Hops zu versenden. Jedoch würde dann netzweit immer nur ein RREPA die Arbitrierung gewinnen, solange kein RSTAT gesendet wird. Dies würde die mögliche Parallelität bei der Routensuche einschränken. Ein noch größerer Nachteil besteht darin, dass die zum Versenden von RREPA benötigte Zeitspanne stark ansteigen würde, da es über $n_{maxHops}$ anstatt zwei Hops übertragen werden müsste. Um dies zu vermeiden, wird ein zusätzliches – wesentlich kürzeres – Paket PreArb (Preliminary Arbitration) verwendet, welches nur ein Bit (FinishPh2) enthält.

Das Paketformat hat folgende Form:
PreArb(FinishPh2 : Bit)

PreArb wird von jedem Knoten, der ein RREPA oder RSTAT senden will, zuvor per Arbitrating Transfer über $n_{maxHops}$ Hops gesendet (das Scheduling der Pakete durch die MAC-Schicht wird in Abschnitt 5.5 beschrieben). Dabei wird FinishPh2 auf 1 gesetzt, wenn ein RSTAT gesendet werden soll, und auf 0, wenn ein RREPA gesendet werden soll. Somit wird das Ende der Routensuche gegenüber der Ermittlung weiterer Routen priorisiert.

5.2.2.6. Dauer von Phase 2

Während bei BBQR alle Phasen eine konstante oder zumindest beschränkte Dauer haben, ist dies bei RBBQR nicht der Fall. Die Dauer von Phase 2 hängt einerseits von der Länge der gefundenen Route ab und andererseits davon, wie viel Parallelität beim Versenden der RREPDs genutzt werden kann (letzteres hat nur dann Auswirkungen, wenn die als erstes propagierte Route scheitert). Zwar lässt sich eine Obergrenze für die Dauer berechnen (s. u.); um jedoch nicht bei jeder Routensuche die maximale Zeitspanne abwarten zu müssen, wird das erfolgreiche Ende von Phase 2 mittels RSTAT(RFND) bekannt gegeben (s. Abschnitt 5.2.2.3).

Um die Dauer von Phase 2 nach oben zu beschränken, wird festgelegt, dass jeder Knoten nur eine bestimmte Anzahl an RREPDs weiterleitet ($n_{maxRREPs}$) und alle weiteren verwirft. Damit nimmt jeder Knoten außer Quelle und Ziel höchstens an $n_{maxRREPs}$ Übertragungen teil (das Ziel sendet immer nur einmal und die Quelle gar nicht). In einem Netz mit $n_{maxNodes}$ Knoten finden somit maximal

$$n_{maxTransPhase2} = (n_{maxNodes} - 2) \cdot n_{maxRREPs} + 1 \tag{5.3}$$

Übertragungen statt. Im ungünstigsten Fall ist keine Parallelität möglich, so dass alle RREPDs nacheinander gesendet werden müssen. Da RREPD als reguläres Paket gesendet wird, wird zur Berechnung der maximalen Dauer von Phase 2 die Dauer einer regulären Übertragung benötigt (neben den in den Formeln (3.1) bzw. (3.2) beschriebenen Zeitspannen für Arbitrating und Cooperative Transfer). Für die Übertragung von n_{bits} Bits berechnet sich die Dauer d_{reg} als

$$d_{reg} = d_{frameOverhead} + \lceil n_{bits}/8 \rceil \cdot d_{byteTrans} \tag{5.4}$$

(vgl. [BBCG11]), wobei $d_{frameOverhead}$ den Overhead zur Übertragung eines regulären Rahmens und $d_{byteTrans}$ die Dauer zur Übertragung eines Bytes innerhalb eines regulären Rahmens darstellt. Formel (5.4) berücksichtigt jedoch nicht, dass reguläre Rahmen eine Maximalgröße haben, d. h. dass n_{bits} Bits u. U. nicht innerhalb eines Rahmens übertragen werden können.

Damit berechnet sich die maximale Dauer von Phase 2 als

$$d_{maxPhase2} = n_{maxTransPhase2} \cdot (d_{PreArb} + d_{RREPA} + d_{maxRREPD}) + d_{PreArb} + d_{RSTAT}$$

$$d_{PreArb} = n_{PreArbBits} \cdot n_{maxHops} \cdot d_{arbBurst} \quad \text{(vgl. Formel (3.1))} \tag{5.5}$$

$$d_{RREPA} = n_{RREPABits} \cdot 2 \cdot d_{arbBurst} \quad \text{(vgl. Formel (3.1))} \tag{5.6}$$

$$d_{maxRREPD} = d_{frameOverhead} + \lceil n_{maxRREPDBits}/8 \rceil \cdot d_{byteTrans} \quad \text{(vgl. Formel (5.4))} \tag{5.7}$$

$$d_{RSTAT} = n_{maxHops} \cdot (n_{RSTATBits} \cdot d_{coopBurst} + d_{procCoop}) \quad \text{(vgl. Formel (3.2))} \tag{5.8}$$

$$n_{PreArbBits} = n_{typeIdBits} + 1$$

$$n_{RREPABits} = n_{typeIdBits} + n_{hopCntBits} + n_{nodeIdBits}$$

$$n_{maxRREPDBits} = n_{typeIdBits} + n_{nodeIdBits} + n_{routeIdBits} + 2 \cdot n_{routeLenBits} + 4 \cdot n_{slotCntBits} + 4 \cdot (n_{maxSlot} + 1) \cdot n_{slotNrBits} + n_{slotNrBits}$$

$$n_{RSTATBits} = n_{typeIdBits} + 2 \quad \text{(Codierung von RFND, FIN_SUCC, FIN_FAIL)}.$$

Die Berechnung von $n_{typeIdBits}$, $n_{hopCntBits}$ und $n_{nodeIdBits}$ wurde bereits in Abschnitt 5.2.1.2 beschrieben. $n_{routeIdBits}$ bzw. $n_{routeLenBits}$ ist die Anzahl der zur Codierung einer RouteID bzw. einer Routenlänge benötigten Bits. Der Typ RouteId kann Werte von 0 bis $n_{maxSlot} - 1 + n_{maxRREPs}$ annehmen, da im ungünstigsten Fall für jeden Slot bis auf den letzten eine eigene Route reserviert wurde und sich zusätzlich $n_{maxRREPs}$ noch nicht gewählte Routen (die zur aktuellen Routensuche gehören) in der temporären Routentabelle befinden (mindestens ein Slot muss noch frei sein, sonst würde der Knoten nicht an der aktuellen Routensuche teilnehmen). Daraus ergibt sich

$$n_{routeIdBits} = \lceil \log_2(n_{maxSlot} + n_{maxRREPs}) \rceil .$$

Da die Berechnung der maximalen Routenlänge nicht festgelegt ist, existiert für $n_{routeLenBits}$ keine allgemeine Formel. Für MaxRouteLength $=_{Df}$ QHopCount + 2 gilt jedoch

$$n_{routeLenBits} = \lceil \log_2(n_{maxHops} + 2 + 1) \rceil ,$$

denn $n_{maxHops}$ ist die Obergrenze für QHopCount.

$n_{slotNrBits}$ ist die Anzahl der zur Codierung einer *Slotnummer* verwendeten Bits und $n_{slotCntBits}$ die Anzahl Bits zur Codierung einer *Slotanzahl*. Der Typ SlotNr kann Werte von 0 bis $n_{maxSlot}$ annehmen und der Typ SlotCnt Werte von 0 bis $n_{maxSlot} + 1$. Damit gilt

$$n_{slotNrBits} = \lceil \log_2(n_{maxSlot} + 1) \rceil \text{ sowie}$$

$$n_{slotCntBits} = \lceil \log_2(n_{maxSlot} + 2) \rceil .$$

Die Anzahl Bits und damit auch die Dauer von RREPD muss nach oben abgeschätzt werden, da im ungünstigsten Fall jede der im RREPD enthaltenen Mengen TXSlots(2HopNachfolger), TXSlots(1HopNachfolger), TXSlots(Sender) und RXSlots(Sender) die maximale Anzahl an Slots ($n_{maxSlot} + 1$) enthalten kann, von denen jeder durch $n_{slotNrBits}$ Bits codiert wird. Damit ergibt sich der Term $4 \cdot (n_{maxSlot} + 1) \cdot n_{slotNrBits}$.

Anmerkung: Die *minimale* Dauer von Phase 2 berechnet sich als

$$d_{minPhase2} = \text{QHopCount} \cdot (d_{PreArb} + d_{RREPA} + d_{maxRREPD}) + d_{PreArb} + d_{RSTAT} .$$

Diese wird erreicht, wenn die erste hop-minimale Route erfolgreich bis zur Quelle fortgesetzt werden kann. QHopCount ist die Distanz zwischen Quelle und Ziel und damit eine Untergrenze für die Routenlänge. Für RREPD muss auch hier jeweils von der maximalen Dauer ausgegangen werden, da die Länge der RREPD-Pakete im Voraus unbekannt ist und das nachfolgende – per Arbitrating Transfer versendete – PreArb-Paket zu einen definierten Zeitpunkt beginnen muss.

5.2.2.7. Scheitern der Routensuche

Kann die QoS-Anforderung bereits im Zielknoten nicht erfüllt werden, so sendet dieser ein RSTAT-Paket mit dem Parameter FIN_FAIL (Finish Failure), um das Scheitern der Routensuche allen Knoten des Netzes bekanntzugeben. Phase 2 besteht dann nur aus diesem Paket. Abbildung 5.9 zeigt den Ablauf (die vorgeschaltete Arbitrierung mittels PreArb ist hier nicht dargestellt).

Phase 2
RSTAT(FIN_FAIL)

Abbildung 5.9.: Ablauf von Phase 2 (QoS-Anforderung im Zielknoten nicht erfüllt).

Wenn keine Route gefunden wird (weil für alle potenziellen Routen die QoS-Anforderung in einem Zwischenknoten nicht erfüllt werden kann), so stellt die Quelle dies dadurch fest, dass sie nach Ablauf von $n_{maxTransPhase2}$ virtuellen RREPD-Slots (diese werden von der MAC-Schicht erzeugt; vgl. Abschnitt 5.5) kein RREPD erhalten hat. In diesem Fall sendet die Quelle ein RSTAT(FIN_FAIL)-Paket. Abbildung 5.10 zeigt den Ablauf (auch hier ist die vorgeschaltete Arbitrierung nicht dargestellt).

Phase 2					
RREPA	RREPD	...			RSTAT(FIN_FAIL)

Abbildung 5.10.: Ablauf von Phase 2 (Timeout bei Quelle).

Darüber hinaus kann der Fall eintreten, dass die Quelle zwar ein oder mehrere RREPD-Pakete erhält, jedoch für keins davon eine zulässige Slotverteilung für die letzten Links der

jeweiligen Route ermitteln kann. Auch in diesem Fall tritt nach Ablauf von $n_{maxTransPhase2}$ virtuellen RREPD-Slots ein Timeout auf.

5.2.3. Phase 3 der Routensuche

Während bei BBQR die Route in Phase 3 durch Arbitrierung ermittelt wird, steht diese bei RBBQR nach einem erfolgreichen Ende von Phase 2 bereits fest. In Phase 3 wird die gefundene Route im Netz etabliert, d. h. den beteiligten Knoten (Zwischenknoten und Ziel) wird mitgeteilt, dass sie Bestandteil der Route sind, die zugehörigen Slots werden reserviert, und die Blockierungen werden den jeweiligen Interferenznachbarn mitgeteilt.

Um die Route bekanntzugeben, werden CREQ-Pakete verwendet, welche (analog zu RREQ und RREP) zweigeteilt sind. Da in Phase 2 jeder Knoten die Slots für seinen 3-Hop-Nachfolger berechnet hat, werden diese mit Hilfe von mehreren CREQ-Paketen zu den jeweiligen Knoten weitergeleitet. Zusätzlich zu den auch vom Routennachfolger benötigten eigenen Sendeslots müssen in jedem CREQ-Paket die eigenen Empfangsslots propagiert werden. Beide Slotmengen werden von den Interferenznachbarn benötigt, um die Slots entsprechend als zum Senden blockiert (Empfangsslots) bzw. als zum Empfangen blockiert (Sendeslots) markieren zu können. Dabei wird ausgenutzt, dass mit Hilfe Black-Burst-basierter Übertragungen die Interferenznachbarn eines Knotens direkt erreicht werden können.

5.2.3.1. Übertragung und Paketformat

In Phase 3 propagiert jeder Knoten seine Sende- und Empfangsslots an alle Interferenznachbarn und die Sendeslots für den 1-, 2- und 3-Hop-Nachfolger an den 1-Hop-Nachfolger (die Sendeslots für den 3-Hop-Nachfolger hat der Knoten selbst berechnet; seine Sende- und Empfangsslots sowie die für 1- und 2-Hop-Nachfolger wurden an ihn gesendet). Während für die eigenen Sende- und Empfangsslots eine Black-Burst-basierte Übertragung erforderlich ist (um die Interferenznachbarn zu erreichen), genügt für die restlichen Slots eine reguläre Übertragung. Das CREQ-Paket wird deshalb in zwei Teile, CREQI (CREQ Information) und CREQD (CREQ Data), zerlegt. Zusätzlich wird ein RSTAT-Paket verwendet, um das Ende der Routensuche im Netz bekanntzugeben (im Gegensatz zu Phase 2 wird in Phase 3 keine vorgeschaltete Arbitrierung zum Versenden von RSTAT benötigt, da es zu jedem Zeitpunkt nur einen Sender gibt). Abbildung 5.11 zeigt den Ablauf von Phase 3, wenn die Route erfolgreich etabliert werden kann und keine Fehlreservierungen aufgrund des Selbstinterferenzproblems auftreten (s. Abschnitt 5.2.3.2).

Phase 3					
CREQI	CREQD	...	CREQI	CREQD	RSTAT(FIN_SUCC)

Abbildung 5.11.: Ablauf von Phase 3 (Route erfolgreich etabliert).

Da es – im Gegensatz zu Phase 2 – in Phase 3 zu jedem Zeitpunkt nur einen Sender gibt und das CREQI-Paket nur über einen Interferenzhop propagiert werden muss, kann der Bit-based Transfer verwendet werden. Dieser wird benötigt, da anstelle des CREQI auch andere Pakete gesendet werden können (s. Abschnitt 5.2.3.2), so dass die Länge eines empfangenen Rahmens nicht im Voraus bekannt ist. Das CREQD-Paket wird im Anschluss an CREQI mittels einer regulären Übertragung versendet. Nachdem QREQI und CREQD auf der gefundenen Route bis zum Ziel propagiert wurden, ist die Routensuche erfolgreich beendet, und das Ziel sendet ein RSTAT-Paket mit dem Parameter FIN_SUCC (Finish Success).

Das Paketformat hat folgende Form:
CREQI(SenderID : NodeId, EmpfängerID : NodeId,
TXSlots(Vorgänger) : SlotSet, TXSlots(Sender) : SlotSet)
CREQD(SenderRouteID : RouteId, EmpfängerRouteID : RouteId,
RouteLenQuelle : RouteLen, TXSlots(1HopNachfolger) : SlotSet,
TXSlots(2HopNachfolger) : SlotSet, TXSlots(3HopNachfolger) : SlotSet)

CREQI enthält neben den IDs von Sender und Empfänger die eigenen Sende- und Empfangsslots (die Empfangsslots entsprechen den Sendeslots des Vorgängers), während CREQD die RouteIDs von Sender und Empfänger, die Routenlänge zur Quelle sowie die Sendeslots für den 1-, 2- und 3-Hop-Nachfolger enthält. Die SenderID wird benötigt, da die Interferenznachbarn aufzeichnen müssen, welcher Knoten für die Blockierung verantwortlich ist. SenderID und SenderRouteID werden außerdem benötigt, da jeder an der Route beteiligte Knoten die ID und RouteID seines Vorgängers kennen muss, um Fehlermeldungen (z. B. bei einem Routenbruch) zur Quelle weiterleiten zu können. Dabei ist jedoch zu beachten, dass der Vorgänger in den zum Senden an den Routennachfolger reservierten Slots möglicherweise nicht empfangen kann, so dass diese nicht verwendet werden können. Anhand der EmpfängerID wird der Routennachfolger identifiziert. Die übrigen Interferenznachbarn können anhand der EmpfängerID erkennen, dass sie nicht der Routennachfolger sind. Die konkrete Route wird mittels der zugehörigen EmpfängerRouteID identifiziert. Die TXSlots des Vorgängers werden sowohl beim Routennachfolger als auch bei den restlichen Interferenznachbarn als zum Senden blockiert eingetragen. Die TXSlots des Senders werden beim Routennachfolger als zum Empfangen reserviert und bei allen anderen Interferenznachbarn als zum Empfangen blockiert eingetragen. Die Routenlänge zur Quelle wird benötigt, um den RouteTimer des jeweiligen Knotens korrekt stellen zu können (s. Anhang B.3).

Auch der Zielknoten muss ein CREQI senden, da dessen Interferenznachbarn seine Empfangsslots als zum Senden blockiert markieren müssen. Er muss jedoch kein CREQD senden.

Die Kardinalität der Slotmengen für die CREQI- und CREQD-Pakete ist stets gleich (sofern die entsprechenden Mengen nicht leer sind) und allen Knoten aus Phase 1 bekannt. Somit muss sie nicht in den Paketen übertragen werden. In manchen Paketen sind eine oder mehrere der Slotmengen leer, da beispielsweise die Quelle keinen Vorgänger bzw. das Ziel keinen Nachfolger hat. Zur Vereinfachung wird dies bei der Länge jedoch nicht unterschieden, sondern es wird ggf. eine entsprechende Anzahl ungültiger Slotnummern übermittelt. Da die Anzahl der ausgewählten Slots i. d. R. gering ist und nur wenige Pakete betroffen sind, ist der dadurch verursachte Overhead unbedeutend.

5.2.3.2. Behebung von Fehlreservierungen durch Selbstinterferenzproblem

Es ist möglich, dass ein sich nahe an der Quelle befindlicher Knoten mit einem nahe am Ziel gelegenen interferiert, obwohl beide auf der gewählten Route durch hinreichend viele Links getrennt sind (dieses Selbstinterferenzproblem wurde bereits in Abschnitt 3.1.2 beschrieben). Dadurch können zusätzliche Slotblockierungen entstehen, die bei der Ermittlung der möglichen Routen in Phase 2 noch nicht berücksichtigt werden können, da in dieser Phase Blockierungen noch nicht zu den Interferenznachbarn propagiert wurden. Ein Beispiel für eine solche Selbstinterferenz ist in Abbildung 5.12 dargestellt.

Auf der Route $q \to a \to b \to c \to d \to e \to f \to g \to h \to i \to j \to k \to l \to m \to z$ befinden sich e und j in Interferenzreichweite zueinander. Deshalb sind die Sendeslots von e in j zum Empfangen und die Empfangsslots zum Senden blockiert.

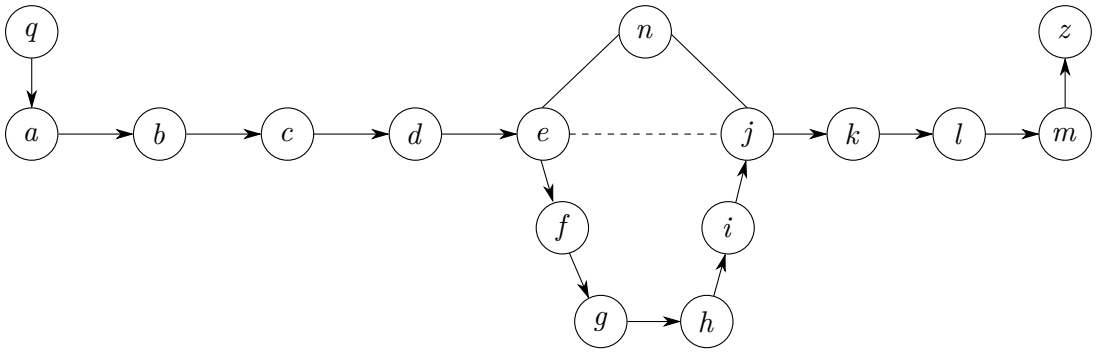

Abbildung 5.12.: Beispiel für Selbstinterferenzproblem.

Es gibt zwei Möglichkeiten für das Zustandekommen einer Route mit Selbstinterferenz. Eine Möglichkeit besteht darin, dass die 2-Hop-Interferenzannahme nicht erfüllt ist. Dies wäre beispielsweise der Fall, wenn im obigen Beispiel der Knoten n fehlen würde. Die „regulären“ Interferenzen, die sich gemäß der 2-Hop-Interferenzannahme über zwei Kommunikationshops erstrecken, werden bei der Auswahl der Slots in Phase 2 berücksichtigt (vgl. Abschnitte 2.7 und 5.2.2.1).

Ist die 2-Hop-Interferenzannahme erfüllt, so gibt es für zwei Knoten e und j mit $j \in IN_1(e) \setminus CN_1(e)$ immer einen Zwischenknoten n mit $n \in CN_1(e)$ und $n \in CN_1(j)$. Da kürzere Routen bevorzugt werden, wird die durch n führende Route gewählt, sofern n die QoS-Anforderung erfüllen kann. Somit kann bei erfüllter 2-Hop-Interferenzannahme nur dann eine Route mit Selbstinterferenz gewählt werden, wenn keine hop-minimale Route die QoS-Anforderung erfüllen kann.

Es ist nicht zu erwarten, dass Fehlreservierungen aufgrund des Selbstinterferenzproblems häufig auftreten, denn selbst wenn eine Route mit Selbstinterferenz gewählt wird, werden für die interferierenden Knoten (im Beispiel e und j) nicht zwangsläufig dieselben Slots verwendet.

In Phase 3 kann festgestellt werden, ob eine Fehlreservierung auftritt, da dort jeweils die Sende- und Empfangsslots eines Knotens mittels CREQI an dessen Interferenznachbarn übertragen werden, bevor die weitere Route fixiert wird. Im Beispiel würde also Knoten j feststellen, dass Slots, welche zum Senden (Empfangen) reserviert werden sollen, bereits durch e zum Senden (Empfangen) blockiert sind. Tritt eine solche Fehlreservierung auf, so wird versucht, alternative Slots zu wählen. Da jeder Knoten aktuelle Informationen über die blockierten Slots hat und weiß, welche Sendeslots für seinen 1-, 2- und 3-Hop-Nachfolger ausgewählt wurden, ist dies möglich, ohne auf den nachfolgenden Links zusätzliche Probleme zu verursachen.

Anmerkung: Eine Fehlreservierung kann erst ab dem fünften Knoten einer Route auftreten, da jeder Slot auf vier aufeinanderfolgenden Links nur einmal verwendet werden kann (d. h. auf den ersten vier Links sind die Slots disjunkt; vgl. Abschnitt 2.7). Insbesondere kann damit bei der Quelle keine Fehlreservierung auftreten. Dies wird anhand des Beispiels in Abbildung 5.13 verdeutlicht.

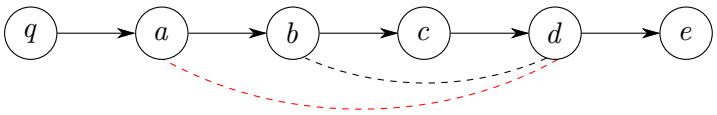

Abbildung 5.13.: Beispiel für Selbstinterferenzproblem beim fünften Knoten einer Route.

Die auf (q, a) verwendeten Slots können normalerweise auf (d, e) wiederverwendet werden, da die Interferenz von d nur bis zu b reicht. Befindet d sich jedoch in Interferenzreichweite zu a (d. h. $IN_1(a) \subseteq CN_1(a) \cup CN_2(a)$ ist nicht erfüllt), so führt eine solche Wiederverwendung zu einer Fehlreservierung bei d.

Da man davon ausgehen kann, dass Fehlreservierungen selten auftreten, wurden die im Folgenden beschriebenen Verfahren zu deren Behebung so entworfen, dass sie keinen Kommunikationsaufwand verursachen, solange dieses Problem nicht auftritt. Im Falle einer Fehlreservierung ist dagegen ein etwas höherer Aufwand vertretbar.

Fehlreservierungen bei Sendeslots

Wird bei den Sendeslots eines Knotens j eine Fehlreservierung festgestellt, so ist dies relativ einfach zu beheben, da die Reservierung noch nicht an die Interferenznachbarn propagiert wurde. Stehen noch genügend freie Sendeslots zur Verfügung (die weder für j selbst noch für seinen 1-, 2- oder 3-Hop-Nachfolger als Sendeslots vorgesehen sind), so werden aus diesen alternative Slots ausgewählt und im CREQI-Paket von j anstelle der ursprünglich gewählten Slots propagiert. Dies ist möglich, da die Route bis zu j bereits fixiert ist (die Blockierungen also bereits eingetragen sind) und die Information über die Sendeslots des 1-, 2- und 3-Hop-Nachfolgers lokal vorliegt.

Für das Beispiel aus Abbildung 5.12 lässt sich dies folgendermaßen formal beschreiben: $\forall\, s \in LK(j,k) : (\neg\, F^s_{TX}(j,k) \implies \text{wähle } s' \in F_{TX}(j,k) \setminus (LK(j,k) \cup LK(k,l) \cup LK(l,m) \cup LK(m,z)))$.

Anmerkung: Zur (Neu-)Berechnung von $F_{TX}(j,k)$ werden die Slots aus $F_{RX}(k)$ benötigt, welche in der temporären Routentabelle aufgezeichnet wurden. Eigentlich würden aktualisierte Werte für $F_{RX}(k)$ benötigt, da die aus Phase 2 vorhandenen aufgrund des Selbstinterferenzproblems veraltet sein können. Wenn sich weiter vorne auf der Route befindliche Knoten in Interferenzreichweite zu k befinden, so können in k zusätzliche Slots zum Empfangen blockiert sein, was im $F_{RX}(k)$-Eintrag von Knoten j noch nicht enthalten ist. Dadurch können bei k Fehlreservierungen bei den Empfangsslots auftreten. Um jedoch prophylaktischen Kommunikationsaufwand zu vermeiden, werden für $F_{RX}(k)$ die aufgezeichneten Werte verwendet und eine ggf. auftretende Fehlreservierung wird entsprechend behandelt (s. nächster Abschnitt).

Fehlreservierungen bei Empfangsslots

Wird bei den Empfangsslots eines Knotens j eine Fehlreservierung festgestellt, so erfordert dies eine Anpassung der Sendeslots des Vorgängers. Dies macht eine nachträgliche Änderung aufwendiger, da zum einen die Slots bereits an die (übrigen) Interferenznachbarn des Vorgängers propagiert wurden und zum anderen die freien Sendeslots des Vorgängers unbekannt sind.

Tritt eine solche Fehlreservierung auf, so sendet j ein CREQC-Paket (CREQ Collision) an den Vorgänger, welches von diesem mit einem CREQU-Paket (CREQ Update) beantwortet wird, sofern die Routenfindung nicht scheitert (das Scheitern der Routenfindung wird in Abschnitt 5.2.3.3 behandelt). Der Ablauf von Phase 3 in diesem Fall ist in Abbildung 5.14 dargestellt.

Phase 3							
...	CREQI	CREQD	CREQC	CREQU	CREQI	CREQD	...

Abbildung 5.14.: Ablauf von Phase 3 (Fehlreservierung bei den Empfangsslots).

Das Paketformat hat folgende Form:

CREQC(EmpfängerID : NodeId,
NumberOfSlots(CollSender) : SlotCnt, CollSlots(Sender) : SlotSet,
NumberOfSlots(RXSender) : SlotCnt, RXSlots(Sender) : SlotSet)
CREQU(SenderID : NodeId,
NumberOfSlots(NewTXSender) : SlotCnt, NewTXSlots(Sender) : SlotSet)

Da beim Senden eines CREQC kein anderer Knoten sendet und auch kein Knoten außerhalb der Kommunikationsreichweite das Paket empfangen muss, kann eine reguläre Übertragung verwendet werden. Das CREQC-Paket enthält (neben der ID des Vorgängers) die fehlreservierten Slots sowie die freien Empfangsslots von j, welche aufgrund der Fehlreservierung beim Vorgänger nicht mehr aktuell sind. Damit wird eine Neuberechnung der freien Sendeslots und somit eine alternative Slotauswahl beim Vorgänger ermöglicht.

Anmerkung: Da CREQI mittels Bit-based Transfer und CREQC regulär übertragen wird, muss ein Knoten beim Empfang eines MAC-Rahmens feststellen können, um welchen Typ es sich handelt (da zu einem Zeitpunkt sowohl ein CREQI als auch ein CREQC gesendet werden kann). Dies ist möglich, da die zur Codierung von Black Bursts verwendeten Rahmen stets kürzer sind als der kürzeste reguläre (vgl. Abschnitt 3.3.1).

Der Vorgänger berechnet mit Hilfe der übermittelten freien Empfangsslots die Menge der freien Sendeslots neu, wählt für jeden fehlreservierten Slot einen alternativen Sendeslot aus und passt seine Einträge in der permanenten Routentabelle und der Slottabelle entsprechend an. Die alternativen Slots müssen dem Routennachfolger sowie den restlichen Interferenznachbarn mitgeteilt werden. Dazu wird das CREQU-Paket verwendet, welches mittels Bit-based Transfer übertragen wird. Der Bit-based Transfer muss verwendet werden, da die Länge des Pakets unbekannt ist.

Anmerkung: Da zu einem Zeitpunkt sowohl ein CREQU als auch das CREQI des nächsten Knotens auf der Route gesendet werden kann, ist die Länge für beide unbekannt, d. h. es muss sowohl für CREQI als auch für CREQU der Bit-based Transfer verwendet werden.

Das CREQU-Paket enthält die SenderID sowie die neu zum Senden reservierten Slots. Der Nachfolger auf der Route erkennt die ID seines Vorgängers (SenderID), da diese aus dem CREQI-Paket bekannt ist. Er ersetzt die fehlreservierten Empfangsslots durch die neuen. Gleichzeitig können die übrigen Interferenznachbarn des Senders die Blockierungen anhand der SenderID dem richtigen Knoten zuordnen.

Die neu zum Senden reservierten Slots müssen in den Interferenznachbarn des Senders zum Empfangen blockiert werden. Die Blockierungen für die ursprünglich reservierten Sendeslots werden nicht explizit entfernt, da dies die Propagierung zusätzlicher Informationen erfordern würde. Sie werden bei der nächsten Statusaktualisierung automatisch freigegeben, da die Slots nun nicht mehr zum Senden reserviert sind (analog zur Freigabe einer Route; s. Abschnitt 5.4.2).

5.2.3.3. Scheitern der Routenfindung

Stehen im Falle einer Fehlreservierung nicht genügend alternative Sende- oder Empfangsslots zur Verfügung, so ist die Routenfindung gescheitert. Möglicherweise könnte zwar trotzdem noch eine Route etabliert werden, wenn die (bereits ausgewählten) Slots für die nachfolgenden Links geändert würden, oder durch Anpassung der Slotauswahl für die vorhergehenden Links. Aufgrund des erhöhten Aufwands wird diese Möglichkeit hier jedoch nicht betrachtet.

Wenn ein Knoten feststellt, dass die Routenfindung gescheitert ist, so wurde die Route

bereits in dessen Vorgängern in die permanente Routentabelle eingetragen. Um diesen Knoten das Scheitern der Routenfindung bekanntzugeben, wird ein CREQF-Paket (CREQ Failure) verwendet, welches von jedem Knoten an seinen Vorgänger auf der Route gesendet wird. Damit wird verhindert, dass eine unvollständige Route in Betrieb genommen wird. Wenn die Quelle ein CREQF erhält, so sendet sie ein RSTAT(FIN_FAIL), womit die Routensuche (erfolglos) beendet ist.

Abbildung 5.15 zeigt den Ablauf von Phase 3, wenn ein Knoten eine Fehlreservierung feststellt und nicht genügend alternative Slots hat.

Phase 3							
...	CREQI	CREQD	CREQF	CREQF	...	CREQF	RSTAT(FIN_FAIL)

Abbildung 5.15.: Ablauf von Phase 3 (Routenfindung gescheitert, da Knoten nicht genug alternative Slots hat).

Abbildung 5.16 zeigt den Ablauf von Phase 3, wenn ein Knoten eine Fehlreservierung bei seinen Empfangsslots feststellt und der Vorgänger nicht genügend alternative Sendeslots hat.

Phase 3							
...	CREQI	CREQD	CREQC	CREQF	...	CREQF	RSTAT(FIN_FAIL)

Abbildung 5.16.: Ablauf von Phase 3 (Routenfindung gescheitert, da Vorgänger nicht genug alternative Slots hat).

Das Paketformat hat folgende Form:
CREQF(EmpfängerID : NodeId)

Das CREQF-Paket kann als reguläres Paket übertragen werden, da es nur vom jeweiligen Vorgänger auf der Route empfangen werden muss und es keine konkurrierenden Sender gibt. Es enthält lediglich die EmpfängerID, da jeder Knoten die RouteID der bei dieser Routensuche etablierten Route kennt. Somit muss diese nicht im Paket übertragen werden.

Der Empfänger sendet ein entsprechendes CREQF an seinen Vorgänger und entfernt anschließend den zur aktuellen RouteID gehörenden Eintrag aus seiner permanenten Routentabelle und die zugehörigen Sende- und Empfangsreservierungen aus der Slottabelle. Die Blockierungen der Interferenznachbarn werden bei der nächsten Statusaktualisierung automatisch freigegeben (s. Abschnitt 5.4.2).

5.2.3.4. Dauer von Phase 3

Die Dauer von Phase 3 hängt sowohl von der Länge der gefundenen Route ab als auch davon, ob Fehlreservierungen aufgrund des Selbstinterferenzproblems auftreten. Somit hat Phase 3 keine konstante Dauer. Um das Ende dieser Phase und damit das der Routensuche im gesamten Netz bekanntzugeben, wird deshalb ein RSTAT-Paket mit dem Parameter FIN_SUCC oder FIN_FAIL versendet. Anschließend kann eine neue Routensuche starten.

Ohne Berücksichtigung von zusätzlichen Paketen aufgrund von Fehlreservierungen hat Phase 3 die maximale Dauer

$$d_{maxPhase3} = \text{MaxRouteLength} \cdot (d_{maxCREQI} + d_{maxCREQD}) + d_{maxCREQI} + d_{RSTAT}$$

$$d_{maxCREQI} = n_{maxCREQIBits} \cdot d_{coopBurst} + d_{procCoop} \quad \text{(vgl. Formel (3.3))}$$

$$d_{maxCREQD} = d_{frameOverhead} + \lceil n_{maxCREQDBits}/8 \rceil \cdot d_{byteTrans} \quad \text{(vgl. Formel (5.4))}$$

$$n_{maxCREQIBits} = n_{typeIdBits} + 2 \cdot n_{nodeIdBits} + 2 \cdot n_{maxNrChosenSlots} \cdot n_{slotNrBits}$$

$$n_{maxCREQDBits} = n_{typeIdBits} + 2 \cdot n_{routeIdBits} + n_{routeLenBits} + 3 \cdot n_{maxNrChosenSlots} \cdot n_{slotNrBits} \,.$$

Dabei ist MaxRouteLength die maximal erlaubte Routenlänge. d_{RSTAT} wurde bereits in Abschnitt 5.2.2.6 berechnet. Der zusätzliche $d_{maxCREQI}$-Term wird benötigt, da der Zielknoten ebenfalls ein CREQI versenden muss, um seine Interferenznachbarn über die Empfangsblockierungen zu informieren. $n_{maxNrChosenSlots}$ ist die maximale Anzahl auswählbarer Slots. Diese ist die Zweierpotenz der maximalen zur Codierung des QoS-Werts verwendeten Zahl (vgl. Tabelle 4.2 auf Seite 60). Letztere wird als $n_{maxQoSWertCode}$ bezeichnet und ergibt sich wiederum aus der Anzahl ihrer Bits:

$$n_{maxNrChosenSlots} = 2^{n_{maxQoSWertCode}}$$

$$n_{maxQoSWertCode} = 2^{n_{qosValBits}} - 1$$

$$\Rightarrow n_{maxNrOfChosenSlots} = 2^{(2^{n_{qosValBits}} - 1)} \,.$$

Die tatsächliche Dauer der CREQI- und CREQD-Pakete lässt sich im Voraus berechnen (im Gegensatz zur Dauer der RREPD-Pakete, welche nicht im Voraus bekannt ist), da die Anzahl $n_{nrChosenSlots}$ der ausgewählten Slots anhand des in Phase 1 netzweit übermittelten QoS-Werts ermittelt werden kann. Sie berechnet sich als

$$d_{CREQI} = n_{CREQIBits} \cdot d_{coopBurst} + d_{procCoop} \tag{5.9}$$

$$d_{CREQD} = d_{frameOverhead} + \lceil n_{CREQDBits}/8 \rceil \cdot d_{byteTrans} \tag{5.10}$$

$$n_{CREQIBits} = n_{typeIdBits} + 2 \cdot n_{nodeIdBits} + 2 \cdot n_{nrChosenSlots} \cdot n_{slotNrBits}$$

$$n_{CREQDBits} = n_{typeIdBits} + 2 \cdot n_{routeIdBits} + n_{routeLenBits} + 3 \cdot n_{nrChosenSlots} \cdot n_{slotNrBits} \,.$$

Die *minimale* Dauer von Phase 3 berechnet sich demnach als

$$d_{minPhase3} = \text{RouteLength} \cdot (d_{CREQI} + d_{CREQD}) + d_{CREQI} + d_{RSTAT} \,,$$

wobei RouteLength die tatsächliche Routenlänge bezeichnet. Sofern keine Fehlreservierungen auftreten, entspricht die minimale Dauer von Phase 3 der tatsächlichen.

Nach Ablauf von Phase 2 lässt sich die maximale Dauer der CREQC- und CREQU-Pakete folgendermaßen nach oben abschätzen:

$$d_{maxCREQC} = d_{frameOverhead} + \lceil n_{maxCREQCBits}/8 \rceil \cdot d_{byteTrans}$$

$$d_{maxCREQU} = n_{maxCREQUBits} \cdot d_{coopBurst} + d_{procCoop}$$

$$n_{maxCREQCBits} = n_{typeIdBits} + n_{nodeIdBits} + 2 \cdot n_{slotCntBits} + (n_{nrChosenSlots} + (n_{maxSlot} + 1 - n_{nrChosenSlots})) \cdot n_{slotNrBits}$$

$$= n_{typeIdBits} + n_{nodeIdBits} + 2 \cdot n_{slotCntBits} + (n_{maxSlot} + 1) \cdot n_{slotNrBits}$$

$$n_{maxCREQUBits} = n_{typeIdBits} + n_{nodeIdBits} + n_{slotCntBits} + n_{nrChosenSlots} \,.$$

Diese Abschätzung ist erforderlich, da im Voraus nicht bekannt ist, wie viele Slots fehlreserviert sind und wie viele freie Empfangsslots der Empfänger noch hat. Im ungünstigsten Fall sind alle der $n_{nrChosenSlots}$ gewählten Slots fehlreserviert. Für die freien Empfangsslots wurde hier als obere Schranke angenommen, dass alle der $n_{maxSlot} + 1$ Slots mit Ausnahme der zum Empfangen vorgesehenen noch frei sind.

Die Dauer eines CREQF-Pakets ist konstant und berechnet sich als

$$d_{CREQF} = d_{frameOverhead} + \lceil n_{CREQFBits}/8 \rceil \cdot d_{byteTrans}$$

$$n_{CREQFBits} = n_{typeIdBits} + n_{nodeIdBits} \,.$$

Anmerkung: Da CREQI mittels Bit-based Transfer übertragen wird, wohingegen es sich sowohl bei CREQC als auch bei CREQF um reguläre Pakete handelt, gilt $d_{CREQI} > d_{maxCREQC}$ und $d_{CREQI} > d_{CREQF}$. Weiterhin gilt $d_{CREQI} > d_{maxCREQU}$, da beide mittels Bit-based Transfer übertragen werden und $n_{CREQIBits} > n_{maxCREQUBits}$ gilt. Analog gilt $d_{CREQD} > d_{CREQF}$, da beide mittels regulärer Rahmen übertragen werden und $n_{CREQDBits} > n_{CREQFBits}$ gilt. Diese Abschätzungen werden für die virtuellen Slots der MAC-Schicht benötigt (s. Abschnitt 5.5.2).

5.3. Datenübertragung in RBBQR

Nach dem erfolgreichen Abschluss einer Routensuche kann die etablierte Route für Datenübertragungen verwendet werden. Dazu dienen DATA-Pakete, die in den reservierten Datenslots mittels regulärer Übertragungen gesendet werden.

Das Paketformat hat folgende Form:
DATA(EmpfängerID : NodeId, EmpfängerRouteID : RouteId, DataValue : DataVal)

DATA-Pakete enthalten die ID und RouteID des jeweiligen Empfängers, d. h. des nächsten Knotens auf der Route, sowie den zu übertragenden Wert, welcher vom Typ DataVal ist. Die Datenslots müssen so dimensioniert sein, dass ein DATA-Paket, das einen Wert des Typs DataVal enthält, innerhalb eines Datenslots versendet werden kann. Die ID des Ziels wird nicht benötigt, da ein Knoten anhand der Tatsache, dass er zu einer RouteID keinen Nachfolger in seiner permanenten Routentabelle aufgezeichnet hat, erkennen kann, dass er das Ziel ist. Anstatt das Paket weiterzuleiten, wird es dann an höhere Protokollschichten übergeben. Die ID der Quelle wird ebenfalls nicht benötigt, da sie im Ziel anhand der permanenten Routentabelle ermittelt werden kann.

Anmerkung: Wenn eine Route gebrochen ist, kann dies beim Versenden von Daten nicht festgestellt werden, da keine ACKs versendet werden. Die Behandlung von Routenbrüchen ist in Abschnitt 5.4.3 beschrieben.

Die Dauer eines Datenpakets ist konstant und berechnet sich folgendermaßen:

$$d_{DATA} = d_{frameOverhead} + \lceil n_{DATABits}/8 \rceil \cdot d_{byteTrans} \tag{5.11}$$

$$n_{DATABits} = n_{typeIdBits} + n_{nodeIdBits} + n_{routeIdBits} + n_{dataValBits} \,.$$

5.4. Freigabe von Routen und Reservierungen

Für die Freigabe von Routen existieren verschiedene Varianten. Einerseits können nicht mehr benötigte Routen explizit freigegeben werden, andererseits können die einzelnen Knoten diese nach einem Timeout implizit freigeben, wenn keine Daten mehr darüber versendet werden.

Wird eine Route in einem Knoten freigegeben, so werden die zugehörigen Sende- und Empfangsreservierungen gelöscht. Jedoch müssen auch die dadurch verursachten Blockierungen bei den Interferenznachbarn freigegeben werden (dies erfolgt allerdings mit einer gewissen Verzögerung).

Darüber hinaus müssen auch Routenbrüche behandelt werden. Diese treten auf, wenn zwei auf einer Route benachbarte Knoten sich aus der gegenseitigen Kommunikationsreichweite bewegt haben. Der Teil zwischen Bruchstelle und Ziel wird implizit mittels Timeouts freigegeben, da bei dem Knoten hinter der Bruchstelle keine Daten mehr ankommen. Der Teil zwischen Quelle und Bruchstelle muss explizit freigegeben werden. Dies ist jedoch aufwendiger als die explizite Freigabe einer intakten Route, da in diesem Fall die Freigabe rückwärts, d. h. von der Bruchstelle zur Quelle, erfolgen muss.

5.4.1. Explizite und implizite Routenfreigabe

Für die explizite Freigabe einer Route wird ein RDEL-Paket (Route Delete) verwendet, welches – analog zu einem Datenpaket – mittels einer regulären Übertragung versendet wird.

Das Paketformat hat folgende Form:
RDEL(EmpfängerID : NodeId, EmpfängerRouteID : RouteId)

RDEL enthält die ID und RouteID des jeweiligen Routennachfolgers. Es wird entlang der Route propagiert, sobald alle noch gepufferten Daten versendet sind. Dazu können die zum Versenden von Daten reservierten Slots genutzt werden. Sobald dies erfolgt ist, wird der zugehörige Eintrag aus der permanenten Routentabelle entfernt (zusammen mit diesem Eintrag werden auch die entsprechenden Sende- und Empfangsreservierungen aus der Slottabelle entfernt). Blockierungen der Interferenznachbarn werden implizit bei der nächsten Statusaktualisierung freigegeben (s. Abschnitt 5.4.2). Die explizite Freigabe einer (intakten) Route verursacht wenig Kommunikationsaufwand, da dazu jeder Knoten nur ein reguläres Paket entlang der Route versenden muss.

Anmerkung: Es handelt sich hier um einen Sonderfall, da ein Kontrollpaket innerhalb eines eigentlich für Daten reservierten Slots versendet wird.

Die Dauer eines RDEL-Pakets ist konstant und berechnet sich folgendermaßen:

$$d_{RDEL} = d_{frameOverhead} + \lceil n_{RDELBits}/8 \rceil \cdot d_{byteTrans}$$

$$n_{RDELBits} = n_{typeIdBits} + n_{nodeIdBits} + n_{routeIdBits}\,.$$

Da $n_{RDELBits} < n_{DATABits}$ gilt und beide Pakete mittels regulärer Übertragungen versendet werden, passt ein RDEL-Paket in die zum Versenden von DATA-Paketen vorgesehenen Slots.

Zur impliziten Freigabe einer Route dienen die RouteTimer der permanenten Routentabelle, deren Berechnung in Anhang B.3 beschrieben ist. Ein solcher läuft ab, wenn keine Daten mehr über die entsprechende Route versendet werden. Analog zur expliziten Freigabe werden beim Ablaufen eines RouteTimers der zugehörige Eintrag aus der permanenten Routentabelle

und die Sende- und Empfangsreservierungen aus der Slottabelle entfernt. Auch hier werden Blockierungen der Interferenznachbarn bei der nächsten Statusaktualisierung freigegeben.

5.4.2. Propagierung von Slotreservierungen

Wird eine Route freigegeben, so sind die durch deren Sende- und Empfangsreservierungen verursachten Blockierungen noch bei den Interferenznachbarn aufgezeichnet. Um diese freigeben zu können, sendet jeder Knoten regelmäßig ein SPROP-Paket (Slot Propagation) mit seinen Slotreservierungen (damit die SPROP-Pakete nicht mit Routingpaketen kollidieren, wird eine entsprechend konfigurierte MAC-Schicht verwendet; s. Abschnitt 5.5). Um Kollisionen durch gleichzeitig gesendete SPROP-Pakete mehrerer Knoten zu vermeiden, wird das Paket (analog zu RREQ und RREP) in zwei Teile, SPROPA (SPROP Arbitration) und SPROPD (SPROP Data), zerlegt. Zusätzlich wird ein SCOUNT-Paket (Slot Count) verwendet, um die maximale Länge von SPROPD im Voraus abschätzen zu können.

Das Paketformat hat folgende Form:
SCOUNT(MaxNumberOfReservedSlots : SlotCnt)
SPROPA(SenderID : NodeId)
SPROPD(SenderID : NodeId,
NumberOfSlots(TXSender) : SlotCnt, TXSlots(Sender) : SlotSet,
NumberOfSlots(RXSender) : SlotCnt, RXSlots(Sender) : SlotSet)

SPROPA enthält die ID des Senders und wird per Arbitrating Transfer über zwei Interferenzhops übertragen. SPROPD enthält die zum Senden sowie die zum Empfangen reservierten Slots und wird vom Gewinner der Arbitrierung mittels Bit-based Transfer übertragen (da alle Interferenznachbarn erreicht werden müssen und die Länge des Pakets unbekannt ist). Zudem muss die SenderID nochmals übertragen werden, da SPROPA lediglich zur Arbitrierung dient und nicht notwendigerweise von allen Knoten, die anschließend das SPROPD empfangen, korrekt empfangen wird (vgl. Abschnitte 3.3.2.3 und 5.2.2.3).

Jeder Knoten kennt zwar seine zum Senden und Empfangen reservierten Slots, jedoch steht dieses Wissen den anderen Knoten des Netzes nicht zur Verfügung. Somit ist die maximale Anzahl der reservierten Sende- und Empfangsslots im Voraus nicht bekannt. Der zum Versenden von SPROPD vorgesehene virtuelle Slot der MAC-Schicht (s. Abschnitt 5.5) müsste deshalb so dimensioniert werden, dass ein SPROPD mit maximaler Länge versendet werden kann. Da jedoch im Extremfall alle Slots reserviert sein könnten und ein Black-Burst-basiertes Übertragungsverfahren für SPROPD verwendet wird, ist dies nicht praktikabel.

Deshalb wird von jedem Knoten, der im aktuellen Superslot ein SPROPD versenden möchte, vor der Arbitrierung mittels SPROPA ein SCOUNT per Arbitrating Transfer über $n_{maxHops}$ Hops gesendet. SCOUNT enthält die jeweils größte (entweder zum Senden oder zum Empfangen) reservierte Slotanzahl des Knotens. Insgesamt wird damit netzweit die größte Slotanzahl ermittelt, die im nächsten SPROPD potenziell gesendet wird. Diese wird sowohl für die Anzahl der Sende- als auch Empfangsslots als Obergrenze angenommen. Dadurch entsteht ein gewisser Verschnitt; da man aber davon ausgeht, dass i. Allg. eher wenige Slots reserviert sind, ist dies vertretbar.

Bei Empfang eines SPROPD-Pakets werden alle für SenderID aufgezeichneten Blockierungen aktualisiert. Für jeden im Parameter TXSlots(Sender) enthaltenen Slot wird eine Blockierung zum Empfangen hinzugefügt, sofern für diesen keine Empfangsreservierung besteht. Für alle in RXSlots(Sender) enthaltenen Slots wird eine Blockierung zum Senden hinzugefügt, sofern keine Sendereservierung besteht. Bestehende Blockierungen, deren Slots nicht in

TXSlots(Sender) bzw. RXSlots(Sender) enthalten sind, werden entfernt.

Anmerkung: Blockierungen, die denselben Slot betreffen, aber durch unterschiedliche Knoten verursacht werden, werden dadurch unterschieden, dass es für jeden Knoten eine eigene Zeile in der Slottabelle gibt (vgl. Anhang B.4).

Anmerkung: Da Blockierungen erst nach einiger Zeit freigegeben werden (nämlich wenn der entsprechende Knoten sein SPROPD versendet), kann es vorkommen, dass ein Knoten Slots bereits wieder für andere Routen vergibt, während die Interferenznachbarn für diese Slots noch Blockierungen aufgezeichnet haben. Dies ist jedoch unproblematisch, da beim Reservieren einer Route die entsprechenden Blockierungen an die Interferenznachbarn übertragen werden. Haben diese bereits eine äquivalente Blockierung (d. h. zum Senden oder Empfangen) für diesen Slot und diesen Knoten, so wissen sie, dass diese durch eine neue ersetzt wurde (der Eintrag in der Tabelle bleibt dabei gleich). Es ist jedoch zu beachten, dass Knoten, welche noch nicht freigegebene Blockierungen für einen Slot haben, diesen noch nicht wieder zum Senden bzw. Empfangen nutzen können, obwohl er eigentlich frei ist.

Die Dauer der Pakete berechnet sich folgendermaßen:

$$d_{SCOUNT} = n_{SCOUNTBits} \cdot n_{maxHops} \cdot d_{arbBurst} \tag{5.12}$$

$$d_{SPROPA} = n_{SPROPABits} \cdot 2 \cdot d_{arbBurst} \tag{5.13}$$

$$d_{maxSPROPD} = n_{maxSPROPDBits} \cdot d_{coopBurst} + d_{procCoop} \tag{5.14}$$

$$n_{SCOUNTBits} = n_{typeIdBits} + n_{slotCntBits}$$

$$n_{SPROPABits} = n_{typeIdBits} + n_{nodeIdBits}$$

$$n_{maxSPROPDBits} = n_{typeIdBits} + n_{nodeIdBits} + 2 \cdot n_{slotCntBits} + 2 \cdot \text{MaxNumberOfReservedSlots} \cdot n_{slotNrBits} .$$

Die Anzahl Bits (und damit die Dauer) des SPROPD-Pakets ist nicht konstant, kann jedoch nach dem Empfang von SCOUNT mit Hilfe von MaxNumberOfReservedSlots nach oben abgeschätzt werden.

Versenden der SPROPD-Pakete

Es muss sichergestellt werden, dass jeder Knoten innerhalb einer definierten Zeitspanne die Arbitrierung gewinnt und sein SPROPD-Paket versenden kann (anderenfalls könnten die Blockierungsinformationen beliebig veralten). Um eine definierte Zeitspanne zwischen zwei SPROPDs eines Knotens zu erreichen, wird die maximale Anzahl $n_{maxCompNodes}$ an Knoten, die bei der Arbitrierung konkurrieren, ermittelt. Dazu kann man entweder Kenntnisse über die Topologie ausnutzen, oder man verwendet $n_{maxNodes}$ als obere Schranke.

Während initial alle Knoten konkurrieren, nimmt jeder Knoten nach Versenden seines SPROPD-Pakets erst nach $n_{maxCompNodes}$ Superslots wieder an der Arbitrierung teil. Dazu wird ein SpropTimer verwendet, welcher folgenden Ablaufzeitpunkt hat:

$$\text{SpropTimer} = t_{superslotBegin} + n_{maxCompNodes} \cdot d_{superslot} .$$

Hier bezeichnet $t_{superslotBegin}$ den Anfang des aktuellen (0-ten) Superslots und $d_{superslot}$ die Dauer eines Superslots. SpropTimer läuft jeweils zum Ende des $(n_{maxCompNodes} - 1)$-ten Superslots ab, so dass der Knoten im $n_{maxCompNodes}$-ten Superslot wieder sendet. Damit

konkurrieren nach den ersten $n_{maxCompNodes}$ Superslots keine Knoten mehr (es können parallele Sendevorgänge im Netz stattfinden, aber nur solche, die bei der Arbitrierung nicht in Konkurrenz stehen). Somit ist gewährleistet, dass alle Blockierungsinformationen maximal $n_{maxCompNodes}$ Superslots alt sind.

5.4.3. Routenbrüche

Ein Routenbruch wird dadurch verursacht, dass sich ein Knoten auf der Route aus der Kommunikationsreichweite seines Vorgängers bewegt hat. Dies ist nicht unmittelbar zu erkennen, da der Knoten, hinter dem die Route gebrochen ist, nicht merkt, dass der Empfänger der Daten nicht mehr da ist. Ein Knoten muss also darüber informiert werden, wenn sich ein Nachfolger für eine seiner Routen aus seiner Kommunikationsreichweite bewegt. Für den vorderen Teil der Route muss dann der Routenbruch explizit signalisiert werden. Der hintere Teil wird implizit freigegeben, da keine Daten mehr darüber versendet werden.

Anmerkung: Der Ausfall eines Knotens entspricht dem Bruch aller Kommunikations- und Interferenzlinks zu diesem Knoten.

Erkennung eines Routenbruchs

SPROPD-Pakete dienen nicht nur zur Propagierung von Sende- und Empfangsreservierungen, sondern auch zum Ermitteln der Interferenznachbarn. Für jeden Interferenznachbarn b eines Knotens a wird eine entsprechende Zeile in der Slottabelle angelegt (vgl. Anhang B.4). Diese kommt entweder durch das erste SPROPD- oder das erste CREQI-Paket von b zustande. Mit jeder solchen Zeile wird ein SlotTabTimer assoziiert. Da jeder Knoten alle $n_{maxCompNodes}$ Superslots sein SPROPD versendet, tritt ein Timeout auf, sobald in mehr als $n_{maxCompNodes}$ Superslots kein SPROPD von b empfangen wurde. In diesem Fall hat b sich aus der Interferenzreichweite von a bewegt, weshalb die entsprechende Zeile aus der Slottabelle von a entfernt wird. War b ein Kommunikationsnachbar, so werden alle Routen, bei denen b der Nachfolger ist, freigegeben (s. nächster Abschnitt). Die Information, ob b ein Kommunikationsnachbar ist oder nicht, wird ebenfalls in der Slottabelle aufgezeichnet. Sie kann anhand der per Bit-based Transfer übertragenen SPROPD-Pakete ermittelt bzw. aktualisiert werden, denn beim Bit-based Transfer teilt die MAC-Schicht den höheren Schichten in einem separaten Bit mit, ob sich der Sender in Kommunikationsreichweite befindet (vgl. Abschnitt 3.3.2.4).

Wird mit Hilfe dieses Bits festgestellt, dass sich ein Kommunikationsnachbar b aus der Kommunikationsreichweite von a bewegt hat, so werden alle Routen, bei denen b der Nachfolger ist, freigegeben (dies ist analog zu dem Fall, dass sich ein Kommunikationsnachbar b komplett aus der Reichweite von a bewegt hat). Die entsprechende Zeile in der Slottabelle bleibt jedoch bestehen, da b immer noch ein Interferenznachbar ist.

Anmerkung: Die Aktualisierung der Information, ob ein Interferenznachbar eines Knotens auch ein Kommunikationsnachbar ist, erfolgt neben den SPROPD-Paketen auch durch CREQI und CREQU, da diese Pakete ebenfalls per Bit-based Transfer übertragen werden.

Behandlung eines Routenbruchs

Wurde ein Routenbruch erkannt, so muss der noch bestehende Teil der Route (von der Quelle bis zu dem Knoten, dessen Nachfolger sich aus der Kommunikationsreichweite bewegt hat) freigegeben werden. Da sowohl ID als auch RouteID des Vorgängers in der permanenten Routentabelle aufgezeichnet sind, ist dies möglich. Für die Freigabe werden RBRK-Pakete (Route Break) verwendet, die jeder Knoten an seinen Vorgänger sendet.

Im Gegensatz zur expliziten Freigabe muss die Route hier in umgekehrter Richtung freigegeben werden, d. h. von der Bruchstelle zur Quelle. Somit können die für die Route reservierten

Datenslots nicht genutzt werden. Deshalb wird die MAC-Schicht so konfiguriert, dass pro Superslot ein RBRK versendet werden kann (analog zum Versenden der SPROP-Pakete). Da mehrere Knoten gleichzeitig versuchen könnten, Routenbrüche weiterzuleiten, wird RBRK in zwei Teile, RBRKA (RBRK Arbitration) und RBRKD (RBRK Data), zerlegt.

Das Paketformat hat folgende Form:
RBRKA(SenderID : NodeId)
RBRKD(VorgängerID : NodeId, VorgängerRouteID : RouteId)

RBRKA enthält die ID des Senders und wird per Arbitrating Transfer über zwei Interferenzhops übertragen. RBRKD enthält die ID des Vorgängers sowie dessen RouteID für die freizugebende Route und wird als reguläres Paket gesendet (da jeweils nur der Vorgänger auf der Route erreicht werden muss). Da die SenderID hier ausschließlich zur Arbitrierung benötigt wird, muss sie nicht zusätzlich im RBRKD übertragen werden.

Anmerkung: Da Routenbrüche die Folge davon sind, dass sich Knoten auseinander bewegt haben, werden i. d. R. bei einem Knoten mehrere Routenbrüche gleichzeitig auftreten. Diese müssen dann nacheinander, d. h. in verschiedenen Superslots, signalisiert werden.

Die Dauer der Pakete ist konstant und berechnet sich folgendermaßen:

$$d_{RBRKA} = n_{RBRKABits} \cdot 2 \cdot d_{arbBurst} \tag{5.15}$$

$$d_{RBRKD} = d_{frameOverhead} + \lceil n_{RBRKDBits}/8 \rceil \cdot d_{byteTrans} \tag{5.16}$$

$$n_{RBRKABits} = n_{typeIdBits} + n_{nodeIdBits}$$

$$n_{RBRKDBits} = n_{typeIdBits} + n_{nodeIdBits} + n_{routeIdBits}\,.$$

5.5. Aufbau und Konfiguration der MAC-Schicht

Ein wesentliches Ziel von RBBQR besteht darin, Kontrollpakete kollisionsfrei zu schedulen. Bei der Routensuche wird dies durch die in Abschnitt 5.2 beschriebenen Arbitrierungen zwischen den Paketen gewährleistet (dies beinhaltet auch die Serialisierung paralleler Routensuchen). Zusätzlich müssen jedoch Slotinformationen propagiert und Routenbrüche weitergeleitet werden, ohne dass die dafür verwendeten Pakete miteinander oder mit den für Routensuchen genutzten kollidieren. Zudem sollen sich die verschiedenen Kontrollpakete nicht gegenseitig blockieren (z. B. sollen Slotinformationen auch dann noch innerhalb einer definierten Zeitspanne gesendet werden können, wenn gerade eine Routensuche stattfindet). Außerdem werden Datenslots benötigt, die durch RBBQR reservierbar sind. Diese Anforderungen können nur mittels einer konfigurierbaren MAC-Schicht umgesetzt werden. Ein grundsätzliches Konzept für eine solche, welches auf der Verwendung virtueller Slotregionen basiert, wurde in Abschnitt 2.1 vorgestellt. Dieses wird nun erweitert.

Zur Übertragung von Daten wird eine bestimmte Anzahl reservierbarer virtueller Slotregionen vorgesehen, deren Dauer gleich ist und in denen jeweils ein Datenpaket gesendet werden kann. Die einzelnen (Daten)Slots werden von 0 bis $n_{maxSlot}$ konsekutiv nummeriert (vgl. Abschnitt 2.1). Die Dauer eines Datenslots berechnet sich als

$$d_{DATASlot} = d_{DATA} + 2 \cdot d_{guard} \quad \text{(Berechnung von } d_{DATA} \text{ s. Formel (5.11))}\,,$$

wobei es sich bei d_{guard} um ein Schutzintervall handelt, das zur Kompensation von Synchronisationsungenauigkeiten benötigt wird und jeweils am Anfang und am Ende eines Slots

freigehalten werden muss.

Für die Kontrollpakete werden ebenfalls virtuelle Slotregionen vorgesehen. Da diese Pakete jedoch unterschiedliche Längen haben, würde die Verwendung generischer Slotregionen, welche für beliebige Kontrollpakete nutzbar sind, zu einem erheblichen Verschnitt führen. Eine naheliegende Lösungsidee besteht darin, für jeden Pakettyp eigene virtuelle Slotregionen vorzusehen. Es tritt jedoch das zusätzliche Problem auf, dass bei einigen Paketen die Länge erst während der Laufzeit ermittelt werden kann (dies ist beispielsweise bei SPROPD der Fall; vgl. Abschnitt 5.4.2). Die Verwendung statisch konfigurierter virtueller Slotregionen würde hier also ebenfalls zu einem erheblichen Verschnitt führen, da derer Dauer so gewählt werden müsste, dass sie für Pakete mit maximaler Länge ausreicht. Ein weiteres Problem besteht darin, dass bei Routensuchen nicht statisch bekannt ist, wie viele Übertragungen z. B. in Phase 2 stattfinden, d. h. wie viele Slotregionen eines Typs jeweils benötigt würden.

Zur Behebung dieser Probleme werden *virtuelle Slots* für die unterschiedlichen Pakettypen definiert, welche eine flexible Dauer haben können. Die Dauer mancher virtueller Slots kann anhand von Konstanten des Netzes (z. B. $n_{maxHops}$) statisch berechnet werden; bei anderen muss sie zur Laufzeit ermittelt werden (z. B. wird die Dauer des SPROPD-Slots nach Empfang von SCOUNT berechnet). Entscheidend dabei ist, dass die Berechnung der Dauer *vor* Beginn des virtuellen Slots erfolgen kann und nur auf Konstanten oder netzweit übermittelten Parametern basiert, damit die ermittelte Dauer in allen Knoten identisch ist.

Anmerkung: Der wesentliche Unterschied zwischen virtuellen Slots und virtuellen Slotregionen besteht darin, dass letztere immer eine konstante Dauer haben.

Die virtuellen Slots werden dynamisch innerhalb der für Kontrollpakete vorgesehenen virtuellen Slotregionen (im Folgenden zur Abkürzung als Kontrollregionen bezeichnet) platziert. Es werden dabei solange virtuelle Slots innerhalb einer Kontrollregion untergebracht, bis der nächste virtuelle Slot nicht mehr hineinpasst. Dieser wird dann in die nächste Kontrollregion verschoben.

Der Beginn eines virtuellen Slots (bei den Datenslots der Beginn der virtuellen Slotregion) wird jeweils mit einem von der MAC-Schicht an RBBQR gesendeten Signal bekannt gegeben. Die Signale sind in Anhang A.3.2 zu finden. In umgekehrter Richtung teilt RBBQR der MAC-Schicht beispielsweise mit, dass Phase 2 zu Ende ist und Phase 3 beginnt, oder dass nach Abschluss einer Routensuche wieder virtuelle Slots für RREQA erzeugt werden sollen. Diese Signale sind in Anhang A.3.1 aufgeführt.

Im Folgenden werden nur die für RBBQR reservierten virtuellen Slotregionen betrachtet und alle anderen Bereiche ausgeblendet. Es wird davon ausgegangen, dass sich alle Kontrollregionen und ebenso alle Datenslots direkt hintereinander befinden. Der Aufbau eines Superslots aus Sicht von RBBQR ist in Abbildung 5.17 dargestellt, wobei die Unterteilung in Mikroslots vernachlässigt wird.

<table>
<tr><td>virtuelle Slots
für SPROP</td><td>virtuelle Slots
für RBRK</td><td>virtuelle Slots
für Routensuche</td><td>0</td><td>1</td><td>...</td><td>$n_{maxSlot}$</td></tr>
<tr><td colspan="3">zusammengefasste Kontrollregionen</td><td colspan="4">Datenslots</td></tr>
</table>

Abbildung 5.17.: Aufbau eines Superslots (Sicht von RBBQR).

In jedem Superslot werden zunächst virtuelle Slots für die Propagierung eines SPROP sowie eines RBRK vorgesehen, damit diese Informationen auch dann innerhalb einer definierten Zeitspanne propagiert werden können, wenn gerade parallel eine Routensuche stattfindet. Der verbleibende Teil der zusammengefassten Kontrollregionen wird für Routensuchen verwendet.

5.5.1. Virtuelle Slots für SPROP und RBRK

Abbildung 5.18 zeigt die virtuellen Slots für SCOUNT, SPROPA und SPROD (mitsamt den entsprechenden Paketen), wobei die Dauer jeweils unterhalb der Slots in Form von Variablen angegeben ist. Eine gestrichelte Linie am Ende eines virtuellen Slots kennzeichnet eine dynamisch berechnete Slotdauer, d. h. diese kann nicht bei der Initialisierung des Systems aus Konstanten berechnet werden.

SCOUNT	SPROPA	SPROPD
$d_{SCOUNTSlot}$	$d_{SPROPASlot}$	$d_{SPROPDSlot}$

Abbildung 5.18.: Virtuelle Slots für SCOUNT, SPROPA, SPROPD.

Die Dauer dieser virtuellen Slots berechnet sich als

$d_{SCOUNTSlot} = d_{SCOUNT} + 2 \cdot d_{guard}$ (Berechnung von d_{SCOUNT} s. Formel (5.12))

$d_{SPROPASlot} = d_{SPROPA} + 2 \cdot d_{guard}$ (Berechnung von d_{SPROPA} s. Formel (5.13))

$d_{SPROPDSlot} = d_{maxSPROPD} + 2 \cdot d_{guard}$ (Berechnung von $d_{maxSPROPD}$ s. Formel (5.14)) .

Nach der netzweiten Übertragung von SCOUNT (der zugehörige Slot hat eine konstante Dauer) kann jeder Knoten $d_{maxSPROPD}$ und damit die Dauer des SPROPD-Slots berechnen. Dieser schließt sich an den SPROPA-Slot (ebenfalls konstante Dauer) an. Wenn kein Knoten ein SPROPD zu senden hat, wird dies daran erkannt, dass kein SCOUNT empfangen wird. In diesem Fall entfallen die virtuellen Slots für SPROPA und SPROPD.

Anmerkung: Am Anfang und am Ende jedes virtuellen Slots wird ein Schutzintervall d_{guard} zur Kompensation von Synchronisationsungenauigkeiten freigehalten. Es wird hier der Einfachheit halber angenommen, dass dieses für alle virtuellen Slots gleich ist und dem Schutzintervall der Datenslots entspricht.

Es folgen die virtuellen Slots für RBRKA und RBRKD, welche beide eine konstante Dauer haben und in Abbildung 5.19 mit den zugehörigen Paketen dargestellt sind.

RBRKA	RBRKD
$d_{RBRKASlot}$	$d_{RBRKDSlot}$

Abbildung 5.19.: Virtuelle Slots für RBRKA und RBRKD.

Die Dauer der virtuellen Slots berechnet sich als

$d_{RBRKASlot} = d_{RBRKA} + 2 \cdot d_{guard}$ (Berechnung von d_{RBRKA} s. Formel (5.15))

$d_{RBRKDSlot} = d_{RBRKD} + 2 \cdot d_{guard}$ (Berechnung von d_{RBRKD} s. Formel (5.16)) .

Anmerkung: Der Slot für RBRKD kann nicht wegoptimiert werden, wenn kein RBRKA gesendet wird. Da ein RBRKA immer nur über zwei Interferenzhops propagiert wird, kann nicht netzweit ermittelt werden, ob ein solches gesendet wurde oder nicht.

5.5.2. Virtuelle Slots für die Routensuche

Nach den virtuellen Slots für SPROP und RBRK kann der übrige Teil der Kontrollregionen für Routensuchen verwendet werden. Die MAC-Schicht erzeugt zunächst so lange virtuelle

Slots zum Senden von RREQA, bis (mindestens) ein Knoten eine Routensuche beginnt (oder bis der Superslot beendet ist). Wird eine Routensuche gestartet, so folgt der reguläre Ablauf von Phase 1 (bestehend aus RREQD-Slots), gefolgt von den Phasen 2 und 3. Kann eine Routensuche in einem Superslot nicht beendet werden, so wird sie im nächsten fortgesetzt (nach den virtuellen Slots für SPROP und RBRK).

Phase 1

Abbildung 5.20 zeigt die virtuellen Slots für Phase 1 mit den zugehörigen Paketen. Sobald ein Knoten in einem RREQA-Slot eine Routensuche startet, schließen sich an diesen $n_{maxHops}$ RREQD-Slots an.

...	RREQA	RREQA	RREQD	...	RREQD
	$d_{RREQASlot}$	$d_{RREQASlot}$	$d_{RREQDSlot}$		$d_{RREQDSlot}$

Abbildung 5.20.: Virtuelle Slots für Phase 1.

Die Dauer der virtuellen Slots ist jeweils konstant und berechnet sich als

$$d_{RREQASlot} = d_{RREQA} + 2 \cdot d_{guard} \quad \text{(Berechnung von } d_{RREQA} \text{ s. Formel (5.1))}$$

$$d_{RREQDSlot} = d_{RREQD} + 2 \cdot d_{guard} \quad \text{(Berechnung von } d_{RREQD} \text{ s. Formel (5.2))}\,.$$

Phase 2

Nach Abschluss von Phase 1 startet Phase 2, welche eine flexible Dauer hat. Abbildung 5.21 zeigt die virtuellen Slots für Phase 2 für den Fall einer erfolgreichen Routensuche inklusive der zugehörigen Pakete (bei PreArb und RSTAT sind jeweils die Parameter des Pakets mit angegeben, d. h. 0 bzw. 1 bei PreArb und RFND bei RSTAT).

PreArb(0)	RREPA	RREPD	PreArb(0)	RREPA	RREPD	...
$d_{PreArbSlot}$	$d_{RREPASlot}$	$d_{RREPDSlot}$	$d_{PreArbSlot}$	$d_{RREPASlot}$	$d_{RREPDSlot}$	

...	PreArb(1)	RSTAT(RFND)
	$d_{PreArbSlot}$	$d_{RSTATSlot}$

Abbildung 5.21.: Virtuelle Slots für Phase 2 (Routensuche erfolgreich).

Wurde eine Route gefunden, so wird Phase 2 durch ein netzweit gesendetes RSTAT(RFND)-Paket beendet. Anschließend ist das Ende von Phase 2 allen Knoten bekannt. Zur Arbitrierung zwischen RREPA und RSTAT dient das ebenfalls netzweit gesendete PreArb, nach dessen Übertragung jeweils der nachfolgende virtuelle Slot (für RREPA oder RSTAT) netzweit eindeutig ausgewählt werden kann.

Die Dauer der virtuellen Slots für Phase 2 ist jeweils konstant (bei RREPD wird für die Paketlänge eine obere Schranke verwendet, da diese vor Beginn des Slots nicht ermittelt werden kann) und berechnet sich als

$$d_{PreArbSlot} = d_{PreArb} + 2 \cdot d_{guard} \quad \text{(Berechnung von } d_{PreArb} \text{ s. Formel (5.5))}$$

$$d_{RREPASlot} = d_{RREPA} + 2 \cdot d_{guard} \quad \text{(Berechnung von } d_{RREPA} \text{ s. Formel (5.6))}$$

$$d_{RREPDSlot} = d_{maxRREPD} + 2 \cdot d_{guard} \quad \text{(Berechnung von } d_{maxRREPD} \text{ s. Formel (5.7))}$$

$d_{RSTATSlot} = d_{RSTAT} + 2 \cdot d_{guard}$ (Berechnung von d_{RSTAT} s. Formel (5.8)).

Hat die Quelle nach $n_{maxTransPhase2} = (n_{maxNodes} - 2) \cdot n_{maxRREPs} + 1$ (vgl. Formel (5.3)) RREPD-Slots kein RREPD erhalten, für das sie für die letzten Links der Route eine zulässige Slotverteilung ermitteln kann, so kann die QoS-Anforderung nicht erfüllt werden. In diesem Fall ist die Routensuche beendet und die Quelle sendet RSTAT(FIN_FAIL). Die zugehörigen virtuellen Slots sind in Abbildung 5.22 dargestellt. Sie entsprechen denen, die im Fall einer erfolgreichen Routensuche erzeugt werden, jedoch werden i. d. R. mehr Slots für PreArb, RREPA und RREPD erzeugt, bevor ein Timeout bei der Quelle eintritt.

PreArb(0)	RREPA	RREPD	...
$d_{PreArbSlot}$	$d_{RREPASlot}$	$d_{RREPDSlot}$	

...	—	—	—	PreArb(1)	RSTAT(FIN_FAIL)
	$d_{PreArbSlot}$	$d_{RREPASlot}$	$d_{RREPDSlot}$	$d_{PreArbSlot}$	$d_{RSTATSlot}$

Abbildung 5.22.: Virtuelle Slots für Phase 2 (Timeout bei Quelle).

Anmerkung: Es ist auch möglich, dass in allen virtuellen Slots von Phase 2 Pakete gesendet werden, dass die Quelle jedoch für keins der empfangenen RREPDs eine zulässige Slotverteilung für die letzten Links der jeweiligen Route ermitteln kann.

Anmerkung: Die Verwendung eines Timers (mit der maximalen Dauer von Phase 2) ist hier nicht sinnvoll, da während der abzuwartenden virtuellen Slots ein Superslotwechsel stattfinden kann.

Weiterhin kann der Fall eintreten, dass die QoS-Anforderung bereits im Zielknoten nicht erfüllt werden kann. Die virtuellen Slots und Pakete für diesen Fall sind in Abbildung 5.23 dargestellt.

PreArb(1)	RSTAT(FIN_FAIL)
$d_{PreArbSlot}$	$d_{RSTATSlot}$

Abbildung 5.23.: Virtuelle Slots für Phase 2 (QoS-Anforderung im Zielknoten nicht erfüllt).

In dem in Abbildung 5.23 gezeigten Fall sendet der Zielknoten – nach entsprechender Arbitrierung – direkt ein RSTAT(FIN_FAIL)-Paket, was die Routensuche beendet.

Phase 3

Nach Abschluss von Phase 2 beginnt Phase 3. Diese hat ebenfalls eine flexible Dauer und wird durch ein RSTAT beendet. Abbildung 5.24 zeigt die virtuellen Slots für Phase 3 sowie die Pakete für den Fall einer erfolgreichen Routensuche.

In Phase 3 kann es aufgrund einer Fehlreservierung durch das Selbstinterferenzproblem vorkommen, dass statt des nächsten CREQI / CREQD ein CREQC, gefolgt von CREQU, gesendet werden muss (s. Abschnitt 5.2.3.2). Da das Versenden Black-Burst-basierter Pakete länger dauert als das regulärer, können die für diese vorgesehenen Slots auch für reguläre Pakete verwendet werden (jedoch nicht umgekehrt). Somit kann für CREQC der nächste für CREQI vorgesehene Slot genutzt werden. Für CREQU kann anschließend der nächste CREQI-Slot genutzt werden. Da beide mittels Bit-based Transfer übertragen werden und CREQU eine kürzere Länge besitzt (vgl. Abschnitt 5.2.3.4), erfordert dies keine Anpassung des CREQI-Slots. Zum Senden von RSTAT wird ebenfalls einer der für CREQI vorgesehenen

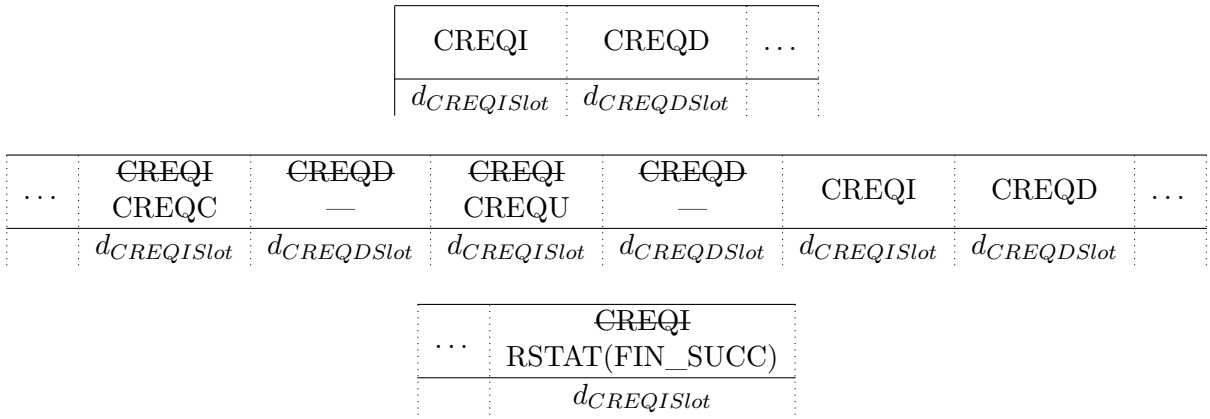

Abbildung 5.24.: Virtuelle Slots für Phase 3 (Route erfolgreich etabliert; Fehlreservierung durch Selbstinterferenz).

Slots verwendet. Hierbei ist zu beachten, dass das RSTAT-Paket zwar kürzer ist, jedoch per Cooperative Transfer durch das gesamte Netz gesendet werden muss, während CREQI nur über einen (Interferenz)Hop propagiert wird. Damit der CREQI-Slot auch für RSTAT verwendet werden kann, muss dieser bei einem großen Netz u. U. vergrößert werden.

Darüber hinaus hängt die Dauer der virtuellen Slots für Phase 3 von der Anzahl der zu reservierenden Datenslots ab, denn daraus ergibt sich die Größe der CREQI- und CREQD-Pakete. Da die Anzahl der zu reservierenden Datenslots jedoch nach Phase 1 netzweit bekannt ist, kann jeder Knoten vor Beginn von Phase 3 die Dauer der virtuellen Slots ermitteln. Diese berechnet sich als

$$d_{CREQISlot} = \max(d_{CREQI}, d_{RSTAT}) + 2 \cdot d_{guard}$$

(Berechnung von d_{CREQI} und d_{RSTAT} s. Formel (5.9) und (5.8))

$$d_{CREQDSlot} = d_{CREQD} + 2 \cdot d_{guard} \quad \text{(Berechnung von } d_{CREQD} \text{ s. Formel (5.10))}\,.$$

Stellt ein Knoten eine Fehlreservierung fest und hat nicht genügend alternative Slots, so ist die Routenfindung gescheitert (vgl. Abschnitt 5.2.3.3). Die virtuellen Slots sind die gleichen wie bei einer erfolgreichen Routensuche; lediglich die gesendeten Pakete unterscheiden sich. Abbildung 5.25 zeigt dies.

Jeder Knoten sendet ein CREQF an seinen Vorgänger, und die Quelle sendet schließlich RSTAT(FIN_FAIL). Da CREQF ein reguläres Paket mit einer kürzeren Länge als CREQD ist, kann es sowohl in den für CREQI als auch in den für CREQD vorgesehenen Slots gesendet werden. RSTAT(FIN_FAIL) wird in einem für CREQI vorgesehenen Slot gesendet.

Stellt ein Knoten eine Fehlreservierung bei den Empfangsslots fest und sendet ein CREQC, so kann der Fall eintreten, dass der Vorgänger nicht genügend alternative Sendeslots hat. Der Ablauf ist dann fast der gleiche wie im vorherigen Fall. Abbildung 5.26 zeigt dies.

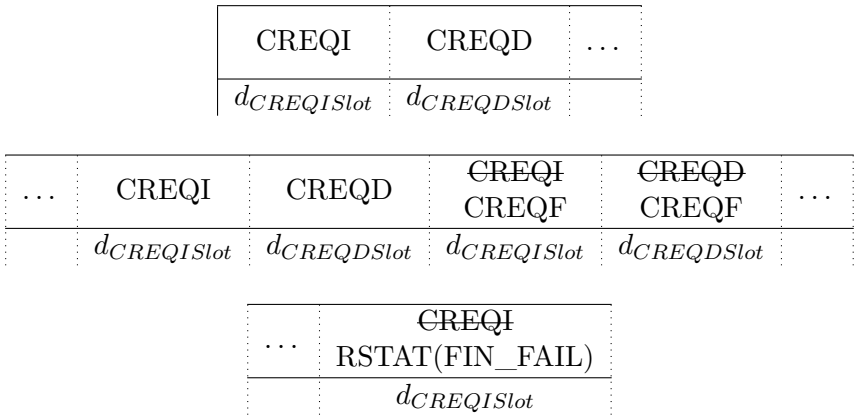

Abbildung 5.25.: Virtuelle Slots für Phase 3 (Route gescheitert, da Knoten nicht genug alternative Slots hat).

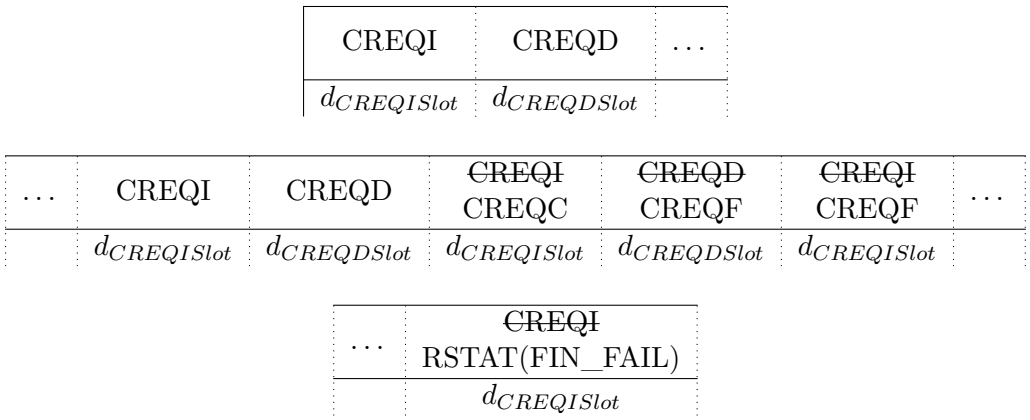

Abbildung 5.26.: Virtuelle Slots für Phase 3 (Route gescheitert, da Vorgänger nicht genug alternative Slots hat).

5.6. Konkrete Berechnungsbeispiele

Im Folgenden werden einige Berechnungen zur Dauer der RBBQR-Phasen für konkrete Netze durchgeführt. Dazu werden die in den Abschnitten 5.2.1.2, 5.2.2.6 und 5.2.3.4 hergeleiteten Formeln verwendet. Effekte der MAC-Schicht werden hier außer Acht gelassen. Insbesondere werden das für jeden virtuellen Slot zweifach zu berücksichtigende Schutzintervall d_{guard}, die Zeit zwischen den virtuellen Slots für die Routensuche und die von der MAC-Schicht zu den Rahmen hinzugefügten Bits nicht berücksichtigt. Weiterhin wurde davon abstrahiert, dass bei regulären Übertragungen u. U. nicht alle zu versendenden Daten in einen Rahmen passen (in diesem Fall würde der Overhead zur Übertragung eines regulären Rahmens mehrfach anfallen). Für Phase 3 wurde zudem davon ausgegangen, dass keine Fehlreservierungen durch das Selbstinterferenzproblem auftreten.

Im Folgenden werden zwei Berechnungen durchgeführt, eine für ein eher kleines und eine für ein größeres Netz. Als Hardware wird ein Atmel AT86RF230 Transceiver [Atm09] zugrunde gelegt, welcher eine Übertragungsrate von 250 kBit/s hat. Die resultierenden Hardwareparameter wurden aus [BBCG11] entnommen. Sie basieren auf den in [Chr10] vorgestellten Berechnungen und sind in Tabelle 5.1 zusammengefasst.

Parameter	Wert
$d_{arbBurst}$	337 μs
$d_{coopBurst}$	304 μs
$d_{procCoop}$	300 μs
$d_{frameOverhead}$	356 μs
$d_{byteTrans}$	32 μs

Tabelle 5.1.: Hardwareparameter für Atmel AT86RF230 Transceiver [BBCG11].

Der Parameter $d_{procCoop}$ ist in [BBCG11] nicht explizit aufgeführt, jedoch beispielsweise in [Chr10] zu finden. Analog zu [BBCG11] wurde auch hier eine Synchronisation mit BBS [GK11] und einem Resynchronisationsintervall von 1 s angenommen (die Synchronisationsungenauigkeit geht in die Berechnung von $d_{arbBurst}$ und $d_{coopBurst}$ mit ein; vgl. [Chr10]).

Die maximale Routenlänge wird als MaxRouteLength $=_{Df}$ QHopCount + 2 definiert, d. h. eine Route darf maximal zwei Hops länger sein als die Distanz zwischen Quelle und Ziel. Zudem müssen einige szenariospezifische Parameter global festgelegt werden. Diese sind in Tabelle 5.2 zusammengefasst.

Parameter	Wert
$n_{maxSlot}$	99
$n_{maxRREPs}$	3
$n_{qosMetricBits}$	2
$n_{qosValBits}$	2

Tabelle 5.2.: Szenariospezifische Parameter.

Für das erste Netzwerk wird $n_{maxHops} = 4$ und $n_{maxNodes} = 10$ angenommen. Damit ergibt sich $d_{Phase1} = 32.79$ ms für die Dauer von Phase 1 und $d_{maxPhase2} = 717.5$ ms für die maximale Dauer von Phase 2. Weiterhin wird angenommen, dass Quelle und Ziel eine Hopdistanz von 3 Kommunikationshops haben, d. h. QHopCount = 3. Die minimale Dauer von Phase 2 beträgt damit $d_{minPhase2} = 101.76$ ms. Aufgrund der Definition von MaxRouteLength kann die gefundene Route maximal 5 Hops lang sein, was für Phase 3 zu einer maximalen Dauer von $d_{maxPhase3} = 245.13$ ms führt. Wird weiter angenommen, dass die tatsächliche Routenlänge 4 beträgt (RouteLength = 4) und $n_{nrChosenSlots} = 4$ gilt, so ergibt sich $d_{minPhase3} = 119.31$ ms für die minimale Dauer von Phase 3. Da angenommen wurde, dass keine Fehlreservierungen auftreten, entspricht dies der tatsächlichen Dauer.

Für das zweite Netzwerk wird $n_{maxHops} = 10$ und $n_{maxNodes} = 100$ angenommen. Damit beträgt die Dauer von Phase 1 $d_{Phase1} = 104.24$ ms und die maximale Dauer von Phase 2 $d_{maxPhase2} = 12675.22$ ms. Auch hier wird QHopCount = 3 angenommen, weshalb Phase 2 eine minimale Dauer von $d_{minPhase2} = 172.95$ ms hat. Auch hier kann die gefundene Route maximal 5 Hops lang sein, so dass Phase 3 eine maximale Dauer von $d_{maxPhase3} = 270.64$ ms hat. Wenn man analog zum ersten Beispiel RouteLength = 4 und $n_{nrChosenSlots} = 4$ annimmt, so ergibt sich für Phase 3 eine minimale Dauer von $d_{minPhase3} = 143$ ms. Dies entspricht

auch hier der tatsächlichen Dauer, da davon ausgegangen wird, dass keine Fehlreservierungen auftreten.

Bei den Berechnungen fällt auf, dass sich für Phase 2 ein großer Unterschied zwischen minimaler und maximaler Dauer ergibt. Dies ist darauf zurückzuführen, dass bei der Berechnung der maximalen Dauer davon ausgegangen wurde, dass jeder Knoten (außer Quelle und Ziel) an $n_{maxRREPs}$ Übertragungen teilnimmt und keine Parallelisierung möglich ist. Im Optimalfall wird dagegen eine hop-minimale Route bis zur Quelle propagiert, ohne dass eine Verzögerung durch Pakete weiterer Routen erfolgt. Es ist zu erwarten, dass die Dauer von Phase 2 bei tatsächlichen Routensuchen näher an der minimalen Dauer liegt, sofern das Netz nicht zu stark ausgelastet ist.

5.7. Funktionale Simulation

Um die Funktionalität des Protokolls RBBQR nachzuweisen, wurde dieses mittels SDL (s. Abschnitt 3.2) formal spezifiziert und mit dem internen Simulator der Rational SDL Suite [IBM15] simuliert. Im Folgenden wird der Aufbau eines Simulationssystems dargestellt, und es wird eine funktionale Simulation beschrieben. Weitere Details zur SDL-Spezifikation sind in Anhang C zu finden. Ergänzungen zu der hier vorgestellten sowie eine Beschreibung weiterer funktionaler Simulationen sind in Anhang D zu finden.

5.7.1. Aufbau eines Simulationssystems

Der allgemeine Aufbau eines Simulationssystems ist in Abbildung 5.27 dargestellt.

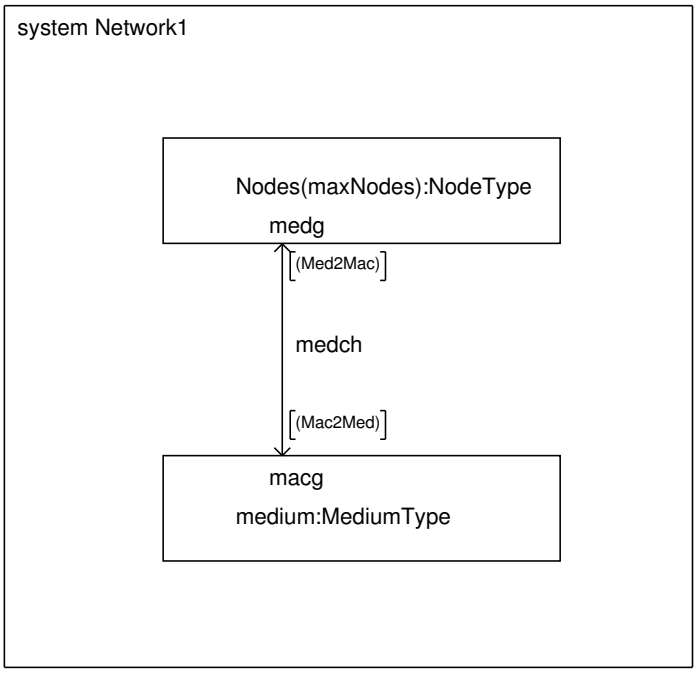

Abbildung 5.27.: Aufbau eines Simulationssystems.

Das System besteht aus einem Medium sowie aus $n_{maxNodes}$ Knoten, die mit diesem verbunden sind. Sowohl das Medium als auch die Knoten sind durch Blöcke (Instanzen von

Blocktypen) repräsentiert. Der Medium-Block enthält lediglich einen Prozess. Dieser spezifiziert die Kommunikations- und Interferenzlinks zwischen den einzelnen Knoten und stellt gesendete Rahmen entsprechend verzögert zu. Knoten unterteilen sich hingegen in weitere Blöcke. Der Aufbau eines Knotens ist in Abbildung 5.28 dargestellt.

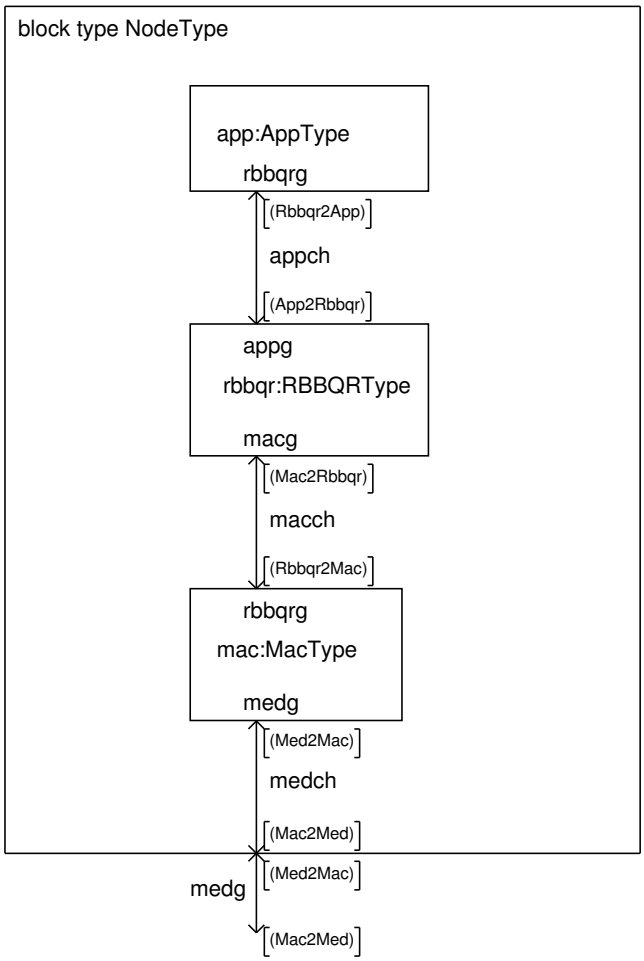

Abbildung 5.28.: Aufbau eines Knotens.

Ein Knoten besteht aus jeweils einem Block für die MAC-Schicht, die Routingschicht und die Anwendungsschicht. Der Block für die MAC-Schicht enthält genau einen Prozess. Dieser realisiert die in Abschnitt 5.5 beschriebene Slotaufteilung und leitet Daten von der Routingschicht an das Medium weiter (und umgekehrt). Der Block für die Anwendungsschicht enthält ebenfalls genau einen Prozess. Dieser dient hier lediglich zum Anfordern und Löschen von Routen sowie zum Versenden von Daten über diese. Der Block für die Routingschicht enthält neben dem Hauptprozess (dieser setzt das RBBQR-Protokoll um) einen Codex, der die von RBBQR gesendeten Signale für die MAC-Schicht verpackt und empfangene Signale auspackt.

Der RBBQR-Prozess ist in mehrere Services unterteilt, welche verschiedene Aufgaben ausführen und dabei auf gemeinsame Datenstrukturen (z. B. Routing- und Slottabellen) zugreifen. Eine detailliertere Beschreibung einiger dieser Services ist in Anhang C zu finden.

5.7.2. Beispielhafte Simulation

In diesem Abschnitt wird eine funktionale Simulation beschrieben, die mit Hilfe der SDL-Spezifikation und dem Simulator der Rational SDL Suite durchgeführt wurde. Ergänzungen sind in Anhang D.1 zu finden. Die zugrunde gelegten Hardwareparameter und szenariospezifischen Parameter entsprechen den in Abschnitt 5.6 verwendeten (insbesondere gilt damit $n_{maxSlot} = 99$). Für die Dauer der Superslots wird $d_{superslot} = 1\,\text{s}$ verwendet.

Für die Simulation wird das Netzwerk aus Abbildung 5.29 verwendet, für das $n_{maxNodes} = 10$ und $n_{maxHops} = 4$ gilt. Die Bedingung $\forall\, a \in V : IN_1(a) \subseteq CN_1(a) \cup CN_2(a))$ ist erfüllt; jedoch ist $\forall\, a \in V : IN_1(a) \supseteq CN_1(a) \cup CN_2(a)$ nicht erfüllt, so dass die Bandbreite bei der Routensuche nicht optimal genutzt werden kann. In diesem Netz werden fünf Routen etabliert, welche ebenfalls in Abbildung 5.29 zu finden sind. Die jeweils verwendeten Slots sind an den entsprechenden Links angegeben. Anschließend werden die Routen wieder freigegeben. Im Folgenden wird die Etablierung und Freigabe der Routen in chronologischer Reihenfolge beschrieben. Die Freigabe wird hier nur kurz zusammengefasst; eine ausführliche Beschreibung ist in Anhang D.1 zu finden.

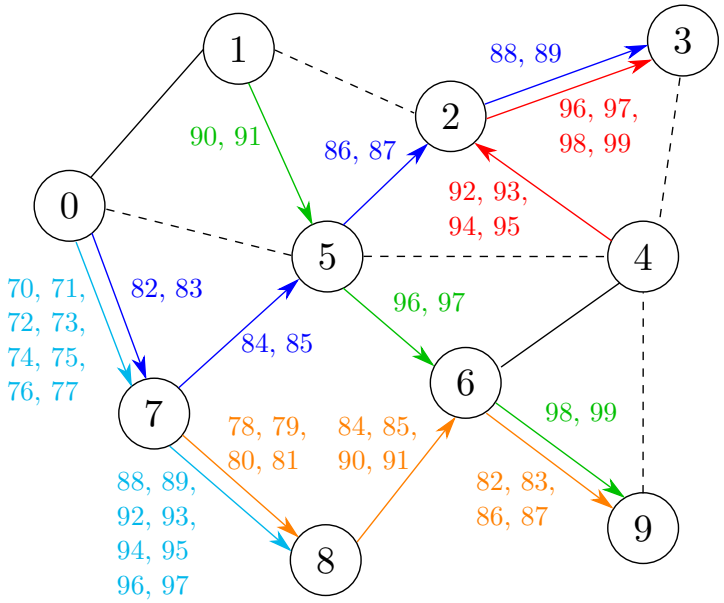

Abbildung 5.29.: Netzwerk für funktionale Simulation von RBBQR.

Anmerkung: Die in diesem Beispiel verwendeten Routen umfassen maximal vier Links, deshalb werden die Sendeslots für alle Knoten einer Route von der Quelle festgelegt (da die Sendeslots vom 3-Hop-Vorgänger eines Knotens zugewiesen werden; vgl. Abschnitt 5.2.2.1).

Etablierung von Routen

Zum Zeitpunkt 0 s fordern die Knoten 0, 1 und 4 gleichzeitig eine Route an. Im ersten RREQA-Slot wird eine entsprechende Arbitrierung durchgeführt, welche Knoten 4 gewinnt (da er die höchste ID hat). Daraufhin sendet Knoten 4 seine Routenanforderung mit Knoten 3 als Ziel und vier Slots (in Form eines RREQD-Pakets), wodurch die einzige hop-minimale Route $4 \rightarrow 2 \rightarrow 3$ etabliert wird. Da noch alle Slots frei sind, hat LCFP hier keinen Einfluss auf deren Vergabe. SDFP sorgt dafür, dass für den Link $(2, 3)$ die höchsten Slotnummern 96 – 99 verwendet werden und für $(4, 2)$ die direkt davor liegenden 92 – 95, so dass die Latenz minimiert wird (zu LCFP und SDFP s. Abschnitt 5.2.2.1). Listing 5.1 zeigt (ausschnittsweise)

die Routenfindung mittels RREPD-Paketen (ausgehend vom Ziel) sowie die Etablierung der Route mit CREQI- und CREQD-Paketen (ausgehend von der Quelle).

```
*** TRANSITION START
*       PId     : <<Block Nodes:3/Block rbbqr>> MainProc:3
*       Service: RouteSearch
*       Now     : 0.1806
*   OUTPUT of RREPD to <<Block Nodes:3/Block rbbqr>> CoDex:3
*       Parameter(s) : 3, 0, 0, 4, [  ], [  ], [  ], [ 0, 1, ..., 98, 99 ],
    -1
...
*** TRANSITION START
*       PId     : <<Block Nodes:2/Block rbbqr>> MainProc:2
*       Service: RouteSearch
*       Now     : 0.2239
*   OUTPUT of RREPD to <<Block Nodes:2/Block rbbqr>> CoDex:2
*       Parameter(s) : 2, 0, 1, 4, [  ], [  ], [ 0, 1, ..., 98, 99 ], [ 0,
    1, ..., 98, 99 ], -1
...
*** TRANSITION START
*       PId     : <<Block Nodes:4/Block rbbqr>> MainProc:4
*       Service: RouteSearch
*       Now     : 0.2826
*   OUTPUT of CREQI to <<Block Nodes:4/Block rbbqr>> CoDex:4
*       Parameter(s) : 4, 2, [  ], [ 92, 93, 94, 95 ]
...
*** TRANSITION START
*       PId     : <<Block Nodes:4/Block rbbqr>> MainProc:4
*       Service: RouteSearch
*       Now     : 0.3077
*   OUTPUT of CREQD to <<Block Nodes:4/Block rbbqr>> CoDex:4
*       Parameter(s) : 0, 0, 0, [ 96, 97, 98, 99 ], [  ], [  ]
...
*** TRANSITION START
*       PId     : <<Block Nodes:2/Block rbbqr>> MainProc:2
*       Service: RouteSearch
*       Now     : 0.3088
*   OUTPUT of CREQI to <<Block Nodes:2/Block rbbqr>> CoDex:2
*       Parameter(s) : 2, 3, [ 92, 93, 94, 95 ], [ 96, 97, 98, 99 ]
...
*** TRANSITION START
*       PId     : <<Block Nodes:2/Block rbbqr>> MainProc:2
*       Service: RouteSearch
*       Now     : 0.3340
*   OUTPUT of CREQD to <<Block Nodes:2/Block rbbqr>> CoDex:2
*       Parameter(s) : 0, 0, 1, [  ], [  ], [  ]
...
```

Listing 5.1.: Ermittlung und Etablierung der Route $4 \rightarrow 2 \rightarrow 3$.

Die RREPD-Pakete enthalten jeweils ID und RouteID des Senders, die aktuelle Routenlänge zum Ziel, die maximale Routenlänge, die freien Sendeslots des 2-Hop-Nachfolgers, des 1-Hop-Nachfolgers und des Senders (zum Senden an den jeweiligen Routennachfolger), die freien Empfangsslots des Senders (zum Empfangen von einem beliebigen Kommunikationsnachbarn) sowie den kleinsten vergebenen Sendeslot für den 3-Hop-Nachfolger (vgl. Abschnitt 5.2.2.3). Die Kardinalitäten der Slotmengen werden hier nicht explizit verwendet. Die CREQI-Pakete

enthalten die IDs von Sender und Empfänger sowie die ausgewählten Sendeslots des Vorgängers und des Senders; die CREQD-Pakete enthalten die RouteIDs von Sender und Empfänger, die Routenlänge zur Quelle sowie die Sendeslots für den 1-, 2- und 3-Hop-Nachfolger (vgl. Abschnitt 5.2.3.1).

Anmerkung: Zur besseren Lesbarkeit wurden in Listing 5.1 die vom SDL-Laufzeitsystem vergebenen PIds (s. Abschnitt 3.2) angepasst, so dass sie den IDs der jeweiligen Knoten entsprechen.

Nachdem die erste Route etabliert ist, findet erneut eine Arbitrierung zwischen 0 und 1 statt, welche 1 gewinnt. Knoten 1 sendet anschließend seine Routenanforderung mit Knoten 9 als Ziel und zwei Slots, was zur Etablierung der einzigen hop-minimalen Route $1 \rightarrow 5 \rightarrow 6 \rightarrow 9$ führt. Für den Link $(1, 5)$ sind die Slots 92 – 99 nicht verwendbar, für die Links $(5, 6)$ sowie $(6, 9)$ sind die Slots 92 – 95 nicht verwendbar. Gemäß LCFP werden die Slots 96 – 99 bevorzugt für $(6, 9)$ ausgewählt, da sie auf einem der anderen Links nicht verfügbar sind. Da nur zwei Slots benötigt werden, wählt SDFP die höchsten, also 98 und 99. Für $(5, 6)$ wählt LCFP anschließend 96 und 97. Für $(1, 5)$ schließlich hat LCFP keinen Einfluss, da es keine weiteren Links mehr gibt (diese Heuristik zielt darauf ab, möglichst viele Slots für die übrigen Links einer Route freizulassen). SDFP wählt die nächstkleineren verfügbaren Slots 90 und 91.

Nach der Etablierung dieser Route gewinnt 0 als einziger verbleibender Knoten die Arbitrierung und fordert anschließend eine Route mit 3 als Ziel und zwei Slots an. Dadurch wird die Route $0 \rightarrow 7 \rightarrow 5 \rightarrow 2 \rightarrow 3$ etabliert. Es gibt noch eine weitere hop-minimale Route, die durch 1 statt durch 7 führt. Da diese beiden Knoten jedoch in Phase 2 konkurrieren und die gleiche Hopdistanz zur Quelle haben, gewinnt 7 aufgrund der größeren ID. Für den Link $(0, 7)$ sind die Slots 90, 91, 96, 97 nicht verwendbar, für $(7, 5)$, $(5, 2)$ und $(2, 3)$ sind die Slots 90 – 99 nicht verwendbar. Somit hat LCFP keinen Einfluss, und SDFP wählt die höchsten verfügbaren Slots 88 und 89 für $(2, 3)$ und für die restlichen Links die jeweils nächstkleineren.

Zum Zeitpunkt 1.3 s benötigen die Knoten 0 und 7 jeweils eine Route. 7 gewinnt als erstes die Arbitrierung und fordert eine Route mit vier Slots zu Knoten 9 an, woraufhin die Route $7 \rightarrow 8 \rightarrow 6 \rightarrow 9$ etabliert wird. Auch hier gibt es eine alternative Route durch Knoten 5 anstelle von 8, jedoch wird 8 aufgrund der größeren ID gewählt. Für den Link $(7, 8)$ sind die Slots 82 – 85, 90, 91, 98, 99 nicht verwendbar; für $(8, 6)$ die Slots 82, 83, 86, 87, 92 – 99 und für $(6, 9)$ die Slots 84, 85, 90 – 99. Damit wählt LCFP zunächst die Slots 82 und 83 für $(6, 9)$ aus, da diese auf den beiden anderen Links nicht verfügbar sind, und anschließend 86 und 87 (auf einem anderen Link nicht verfügbar). Danach wählt LCFP die Slots 84, 85, 90, 91 für $(8, 6)$ aus, da diese auf $(7, 8)$ blockiert sind. Für $(7, 8)$ hat LCFP keinen Einfluss, so dass SDFP die nächstkleineren verfügbaren Slots 78, 79, 80, 81 wählt.

Nach der Etablierung dieser Route nimmt nur noch Knoten 0 an der Arbitrierung teil und fordert anschließend eine Route zu Knoten 8 mit acht Slots an. Dies liefert die einzige hop-minimale Route $0 \rightarrow 7 \rightarrow 8$. Für den Link $(0, 7)$ sind die Slots 78 – 87, 90, 91, 96, 97 nicht nutzbar und für $(7, 8)$ die Slots 78 – 87, 90, 91, 98, 99. Damit wählt LCFP zunächst 96 und 97 für $(7, 8)$, da diese Slots auf $(0, 7)$ nicht verfügbar sind. Anschließend können für $(7, 8)$ nur noch Slots verwendet werden, die auch für $(0, 7)$ nutzbar wären, so dass SDFP die nächstkleineren Slots 88, 89 und 92 – 95 wählt. Für $(0, 7)$ wählt SDFP dann die nächstkleineren verwendbaren Slots 70 – 77.

Freigabe von Routen

In dem gewählten Szenario werden alle etablierten Routen wieder freigegeben, was im Folgenden kurz zusammengefasst wird. Eine ausführliche Beschreibung mit entsprechenden Listings

ist in Anhang D.1 zu finden.

Die Route $1 \rightarrow 5 \rightarrow 6 \rightarrow 9$ wird zum Zeitpunkt 5 s durch die Anwendung explizit freigegeben. Die Route $4 \rightarrow 2 \rightarrow 3$ wird nach einem Bruch des Kommunikationslinks zwischen den Knoten 2 und 4 zum Zeitpunkt 6.9999 s freigegeben. In den Knoten 2 und 3 erfolgt die Freigabe implizit durch Timeouts (Ablaufen der entsprechenden RouteTimer). Knoten 4 gibt die Route frei, nachdem bei Erhalt des nächsten SPROPD-Pakets von Knoten 2 festgestellt wird, dass 2 und 4 keine Kommunikationsnachbarn mehr sind. Über die Route $0 \rightarrow 7 \rightarrow 8$ werden ab dem Zeitpunkt 9.8 s keine Daten mehr versendet, so dass die einzelnen Knoten diese ebenfalls nach Ablaufen ihres zugehörigen RouteTimers freigeben. Die Route $0 \rightarrow 7 \rightarrow 5 \rightarrow 2 \rightarrow 3$ wird nach einem Linkbruch (Kommunikations- und Interferenzlink) zwischen den Knoten 5 und 7 zum Zeitpunkt 14.9999 s freigegeben. Die Freigabe in den Knoten 5, 2 und 3 erfolgt auch hier durch Ablaufen des jeweiligen RouteTimers. Bei Knoten 7 läuft der SlotTabTimer für Knoten 5 ab, da kein SPROPD mehr von diesem empfangen wird. Anschließend sendet Knoten 7 ein RBRKD an Knoten 0. Die letzte Route $7 \rightarrow 8 \rightarrow 6 \rightarrow 9$ bricht, nachdem zum Zeitpunkt 24.9999 s Knoten 8 ausfällt. Sie wird analog zur Route $0 \rightarrow 7 \rightarrow 5 \rightarrow 2 \rightarrow 3$ freigegeben.

Kapitel 6.

6 QoS Multicast Routing Protocol (QMRP)

In diesem Kapitel wird das *QoS Multicast Routing Protocol* (QMRP) vorgestellt, welches in [Geb15] entwickelt wurde. Es führt Routensuchen reaktiv und zentralisiert durch und nimmt deterministische Reservierungen von Zeitslots vor. Eine detailliertere Beschreibung des Protokolls mit den zugehörigen Algorithmen, Beispielen sowie Evaluationen ist in [Geb15] zu finden. Das Protokoll wird auch in [GGIK15] publiziert.

Im Gegensatz zu RBBQR, welches für beliebige Netze entwickelt wurde, handelt es sich bei QMRP um ein eher spezielles Protokoll. Es wurde für ein Produktionsnetz im Kontext eines größeren Industrieprojekts entwickelt, bei dem ein Großteil der Knoten stationär ist (z. B. Sensoren und Aktuatoren) und es zusätzlich einige mobile Knoten (z. B. Roboter) gibt.

Das Ziel bestand in der Entwicklung eines Multicast-Protokolls, welches QoS-Routing unterstützt und die Eigenschaften der stationären Knoten ausnutzt. So werden beispielsweise Routen von und zu mobilen Knoten so weit wie möglich durch das stationäre Netz geführt.

Die Topologie des stationären Netzes ist allen Knoten bekannt. Aus diesem Grund wurde ein zentralisierter Ansatz verwendet, um den Kommunikationsaufwand zu verringern. Dieser ermittelt zunächst eine angeforderte Route und reserviert anschließend Slots für diese. Somit findet sich hier eine Entkopplung von Routing- und Reservierungsprotokoll. Im Gegensatz dazu wird bei RBBQR das Reservierungsprotokoll mit der Routensuche kombiniert, da dort Routen verteilt gesucht werden. Ferner ist bei QMRP keine Verwendung von Black Bursts erforderlich, da aufgrund des zentralisierten Ansatzes keine Reservierungen an Interferenznachbarn propagiert werden müssen. Auch wird keine Arbitrierung benötigt, da jeder Knoten einen eigenen Kontrollslot zum Versenden von Management-Paketen hat.

Im Folgenden werden zunächst einige für QMRP erforderliche Voraussetzungen definiert. Anschließend wird die Erzeugung stationärer QoS-Multicast-Bäume sowie die Reservierung der zugehörigen Zeitslots betrachtet. Danach wird auf die zur Unterstützung mobiler Knoten erforderlichen Erweiterungen eingegangen.

6.1. Voraussetzungen für QMRP

Neben den in Abschnitt 2.2.1 formulierten allgemeinen Annahmen werden für QMRP einige weitere Voraussetzungen benötigt, die im Folgenden formuliert werden.

1. Das Netz besteht überwiegend aus stationären und sowie aus einigen mobilen Knoten.

2. Das nachträgliche Hinzufügen neuer Knoten sowie der Ausfall von Knoten werden momentan nicht betrachtet.

3. Für das stationäre Netz gilt die Single Network Property. Dies impliziert insbesondere, dass es zwischen zwei stationären Knoten stets einen (zuverlässigen) Kommunikationspfad gibt, welcher keine mobilen Knoten beinhaltet.
4. Die Topologie des stationären Netzes wird vorab mit dem *Automatic Topology Discovery Protocol* (ATDP) [Kra13, KCG15] ermittelt und ist allen stationären Knoten bekannt.
5. In Phasen, in denen QMRP aktiv ist, ist kein weiteres Protokoll aktiv. Dies muss durch die MAC-Schicht entsprechend unterstützt werden. Der Aufbau der MAC-Schicht sowie das genaue Scheduling der Kontrollpakete werden hier nicht im Detail betrachtet (z. B. die Übertragung von Reservierungen vom Master an die entsprechenden Knoten). Es wird jedoch davon ausgegangen, dass es Kontroll- und Datenslots gibt (durch virtuelle Slotregionen realisiert; vgl. Abschnitt 2.1). Jeder Knoten hat einen netzweit reservierten Kontrollslot, in dem Kontrollpakete an alle Knoten in Kommunikationsreichweite gesendet werden können. Es wird vorausgesetzt, dass die Dauer dieser Kontrollslots so gewählt ist, dass sie für alle in diesen potenziell zu versendenden Pakete ausreichend ist.
6. Für jeden mobilen Knoten m lässt sich eine minimale Teilmenge $V_{acc}(m) \subseteq V_{stat}$ der stationären Knoten definieren (diese werden als *Access Nodes* von m bezeichnet), so dass m sich stets in Reichweite von mindestens einem Knoten aus $V_{acc}(m)$ befindet. Sowohl die Access Nodes als auch m wissen stets, in Reichweite welches Knotens m sich zu einem Zeitpunkt befindet, und wenn dies mehrere sind, welcher Knoten gerade für m zuständig ist (dieser wird auch als aktiver Access Node bezeichnet). Der aktive Access Node kann durch den Austausch periodischer Beacon-Rahmen in den reservierten Kontrollslots der entsprechenden Knoten ermittelt werden und ist immer eindeutig bestimmt.
7. Die Uhren der einzelnen Knoten sind durch die Verwendung des Zeitsynchronisationsprotokolls BBS [GK11], welches regelmäßig ausgeführt wird, hinreichend genau synchronisiert.
8. Als QoS-Metrik wird momentan nur Bandbreite in Form zu reservierender Datenslots unterstützt. Diese entsprechen virtuellen Slotregionen der MAC-Schicht (vgl. Abschnitt 2.1). Es wird angenommen, dass die QoS-Anforderung stets genau einem Datenslot entspricht. Die QoS-Metrik Verzögerung wird zwar nicht direkt unterstützt; für das Scheduling der Slots steht jedoch eine Heuristik zur Verfügung, welche die Verzögerung optimiert.
9. Jeder Knoten hat eine eindeutige ID.
10. Alle Übertragungen werden im selben Zeitslot durch ein Acknowledgement (ACK) bestätigt. Soll ein Slot s zum Senden von einem Knoten a an einen Knoten b reserviert werden, muss deshalb die wechselseitige globale Reservierungsbedingung $F^s(a, b)$ (s. Definition 2.9 auf Seite 22) erfüllt sein. Da Multicast-Übertragungen verwendet werden, können mehrere Knoten gleichzeitig eine Übertragung eines Knotens a empfangen. Es wird deshalb davon ausgegangen, dass zusätzlich zu der eigentlichen Übertragung bis zu drei ACKs in einen Slot passen.

6.2. Stationäre QoS-Routing-Bäume

QMRP wurde für ein auf einer Service-Architektur basierendes Anwendungsszenario entwickelt. Bei einer solchen bieten Knoten Dienste an (beispielsweise Alarmbenachrichtigungen

oder periodische Temperaturwerte), welche in einer verteilten Service Registry veröffentlicht werden. Andere Knoten können diese Dienste zur Laufzeit abonnieren. Für jedes solche Abonnement wird eine QoS-Route vom Diensterbringer zum Dienstnutzer benötigt. Jedoch wird ein Dienst i. d. R. von mehreren Knoten abonniert, so dass die Erstellung eines Multicast-Baums vom Diensterbringer zu allen Dienstnutzern sinnvoll ist. Im Vergleich zu einzelnen Unicast-Routen kann dies die Auslastung des Netzes deutlich reduzieren, da oft Teile der Routen gemeinsam genutzt werden können.

Die QoS-Routing-Bäume werden dynamisch erzeugt und erweitert. Wenn ein Knoten als erster einen Dienst abonniert, wird ein aus einem einzelnen Pfad bestehender Baum erzeugt. Bei nachfolgenden Reservierungen wird dieser Baum dann entsprechend erweitert. Dabei ist zu beachten, dass ein Baum stets an einen bestimmten Dienst gebunden ist. Bietet beispielsweise ein Diensterbringer q mehrere Dienste an, so muss für jeden Dienst ein eigener Multicast-Baum erstellt werden, der q als Quelle hat.

Für QMRP ist ein zentralisierter Ansatz vorteilhaft, da beim Starten des Netzes das Protokoll ATDP ausgeführt wird, welches die Kommunikations- und Interferenztopologie der stationären Knoten ermittelt und netzweit verteilt. Diese Information kann zur zentralen Ermittlung von Routen durch einen Masterknoten genutzt werden. Da bei einem zentralisierten Ansatz auch die gesamte Slotinformation vom Master verwaltet wird, entfällt zudem der Kommunikationsaufwand, um Blockierungen an Interferenznachbarn weiterzuleiten. Da diese sich außerhalb der Kommunikationsreichweite der entsprechenden Knoten befinden können, würde ihre Benachrichtigung die Verwendung Black-Burst-basierter Übertragungsprotokolle (analog zu RBBQR) oder eine Weiterleitung über mehrere Hops erfordern. Ein zentralisierter Ansatz hat jedoch den Nachteil, dass zusätzlicher Kommunikationsaufwand anfällt, um die Sende- und Empfangsreservierungen an die entsprechenden Knoten zu übermitteln. Ein weiterer Nachteil besteht darin, dass der Ausfall des Masters das gesamte Netz lahmlegt. Dies ist hier jedoch weniger problematisch, denn da ohnehin eine Master-basierte Zeitsynchronisation mit BBS [GK11] durchgeführt wird, gibt es bereits einen Single Point of Failure. Wird der für die Zeitsynchronisation verwendete Masterknoten auch für die Routensuchen als Master verwendet, so ergibt sich keine zusätzliche Schwachstelle.

Wenn ein Knoten einen Dienst abonnieren will, wird eine entsprechende Anfrage an den Masterknoten gesendet, welcher entweder einen neuen Routing-Baum anlegt oder einen vorhandenen erweitert. Um diese Anfrage an den Master weiterzuleiten, wird die mittels ATDP ermittelte (und an alle Knoten verteilte) Kommunikationstopologie genutzt. Zum Senden nutzt jeder Knoten seinen exklusiv reservierten Kontrollslot, so dass keine Kollisionen auftreten. Eine kurze Beschreibung dieser *Route Request Phase* sowie des verwendeten Paketformats ist in [Geb15] zu finden. Diese Aspekte werden hier jedoch nicht vertieft betrachtet.

6.2.1. Ermittlung der Routing-Bäume

Bei QMRP erfolgt die initiale Erzeugung eines Baums sowie die Erweiterung eines existierenden Baums, bevor Slots für die entsprechende Route reserviert werden. Die dazu verwendeten Heuristiken werden in den folgenden Abschnitten beschrieben. Durch diese Entkopplung ist es möglich, dass eine Routenanforderung scheitert, weil ein Pfad gewählt wird, für den anschließend nicht genügend freie Slots verfügbar sind. Für einen anderen Pfad wäre es jedoch u. U. möglich, die Routenanforderung zu erfüllen. In diesem Fall könnte man durch Wiederholung der Routensuche einen anderen Pfad ermitteln und versuchen, für diesen Slots zuzuweisen. Für QMRP wurde dies jedoch nicht vorgesehen, da in Fällen, in denen die Slotzuweisung nicht erfolgreich ist, die Netzwerklast bereits hoch und somit die Wahrscheinlichkeit gering ist, dass

für einen anderen Pfad die Slotzuweisung erfolgreich verläuft.

6.2.1.1. Erzeugung eines Routing-Baums

Abonniert ein Knoten einen Dienst, der noch von keinem anderen Knoten abonniert wurde, so wird ein neuer Routing-Baum erzeugt, der zunächst nur aus einer Unicast-Route zwischen dem Diensterbringer und dem neuen Dienstnutzer besteht. Zur Erzeugung des Baums stehen zwei verschiedene Heuristiken zur Auswahl. Die erste zielt darauf ab, durch den neuen Baum möglichst wenige zusätzliche Blockierungen zu verursachen. Die zweite dagegen hat zum Ziel, den Baum so zu wählen, dass spätere Erweiterungen möglichst wenige neue Links hinzufügen.

Betrachtung des existierenden Schedule
Bei dieser Heuristik wird zunächst die Menge der kürzesten Kommunikationspfade zwischen Quelle q (Diensterbringer) und Ziel z (Dienstnutzer) ermittelt (diese wird hier mit $P(q,z)$ bezeichnet). Existieren mehrere solche Kommunikationspfade, so wird ein Pfad $p \in P(q,z)$ gewählt, dessen minimale Anzahl an freien Slots am kleinsten (jedoch größer als eine vorgegebene Schranke thr) ist. Die minimale Anzahl $m_{p'}$ an freien Slots eines Pfads p' ist definiert als die minimale Anzahl freier Slots für alle Links von p'. Da jede Übertragung im selben Zeitslot mit einem ACK bestätigt wird, wird die in Definition 2.9 (Seite 22) vorgestellte wechselseitige globale Reservierungsbedingung verwendet, um freie Slots zu bestimmen. Dies lässt sich folgendermaßen formalisieren (zu $P_{G_{cl}}$ und $d_{G_{cl}}$ vgl. Definition 2.2 auf Seite 16):

$$P(q,z) =_{Df} \{p' : p' \in P_{G_{cl}}(q,z) \wedge |p'| = d_{G_{cl}}(q,z)\}$$

$$m_{p'} =_{Df} \min\{|F(a,b)| : (a,b) \in p'\}$$

$$p \in \{p' : p' \in P(q,z) \wedge m_{p'} = \min\{m_{p''} : m_{p''} > thr \wedge p'' \in P(q,z)\}\}\,.$$

Das in Abbildung 6.1 dargestellte Beispiel verdeutlicht diese Heuristik.

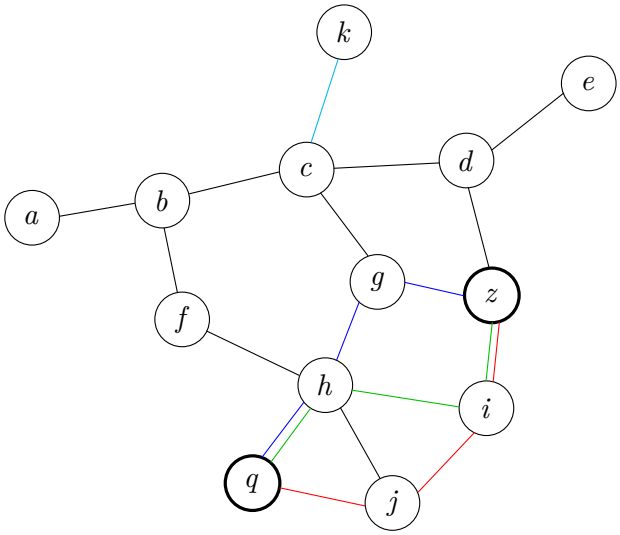

Abbildung 6.1.: Pfade $p_1, p_2, p_3 \in P(q,z)$ und ein existierender Baum t von c zu k.

In diesem Netzwerk existiert bereits ein Baum t von c zu k, der lediglich aus einer Route besteht. Es soll nun ein weiterer Baum etabliert werden. Zwischen der neuen Quelle q und

dem neuen Ziel z existieren drei kürzeste Kommunikationspfade p_1, p_2 und p_3, die jeweils eine Länge von drei Hops haben (d. h. $P(q,z) = \{p_1, p_2, p_3\}$ und $|p_1| = |p_2| = |p_3| = d_{G_{cl}}(q,z) = 3$). Aus Gründen der Übersichtlichkeit wird für dieses Beispiel die 1-Hop-Interferenzannahme (s. Abschnitt 2.3) getroffen. Angenommen, es gebe fünf Slots und für die Übertragung von c zu k werde Slot 1 verwendet. Dann kann g in Slot 1 nicht empfangen und (aufgrund des ACKs von k zu c) nicht senden. Die restlichen Knoten von p_1, p_2 und p_3 können alle Slots verwenden, so dass $m_{p_1} = 4$ und $m_{p_2} = m_{p_3} = 5$ gilt. Wird beispielsweise $thr = 3$ gesetzt, so wird der Pfad p_1 gewählt.

Diese Heuristik liegt darin begründet, dass der neu erstellte Baum einen möglichst geringen Einfluss auf den Rest des Netzes haben soll. Pfade mit einer geringen Anzahl freier Slots verlaufen durch Regionen des Netzes, in denen bereits viele Routen verlaufen. Dadurch sind bereits viele Slots blockiert, so dass durch das Hinzufügen einer weiteren Route wenige neue Blockierungen auftreten. Somit wird die Wahrscheinlichkeit erhöht, für spätere Routenanforderungen noch Pfade mit genügend freien Slots zu finden.

Ein Nachteil dieser Heuristik besteht darin, dass die gewählte Route scheitern kann, wenn die Schranke thr zu niedrig gewählt wird. Es könnte dann der Fall eintreten, dass nicht für jeden Link der Route ein freier Slot zugewiesen werden kann.

Betrachtung der Pfadnachbarschaft

Bei dieser Heuristik wird zunächst ebenfalls die Menge $P(q,z)$ der kürzesten Kommunikationspfade zwischen q und z ermittelt. Existiert mehr als ein solcher Pfad, so wird die 1-Hop-Kommunikationsnachbarschaft der einzelnen Pfade betrachtet (vgl. Definition 2.6 auf Seite 20). Es wird dann ein Pfad $p \in P(q,z)$ gewählt, dessen Nachbarschaft die meisten Knoten enthält. Dies lässt sich folgendermaßen formalisieren:

$$p \in \{p' : p' \in P(q,z) \wedge |CN_1(p')| = \max\{|CN_1(p'')| : p'' \in P(q,z)\}\}\,.$$

Abbildung 6.2 zeigt ein Beispiel.

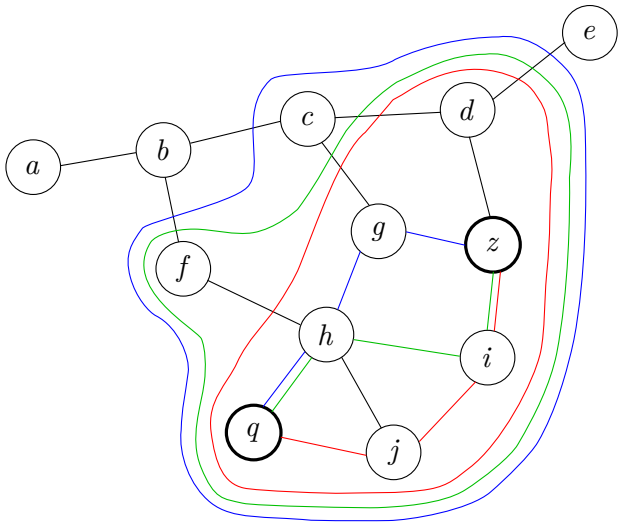

Abbildung 6.2.: Pfade $p_1, p_2, p_3 \in P(q,z)$ mitsamt den zugehörigen 1-Hop-Kommunikationsnachbarschaften (vgl. [GGIK15]).

Analog zu dem in Abbildung 6.1 dargestellten Beispiel existieren auch hier drei kürzeste Pfade p_1, p_2 und p_3 zwischen der Quelle q und dem Ziel z, die jeweils eine Länge von drei

Hops haben. Da $|CN_1(p_1)| > |CN_1(p_2)| > |CN_1(p_3)|$ gilt, wird der Pfad p_1 gewählt.

Diese Heuristik liegt darin begründet, dass ein Pfad mit vielen Nachbarn durch einen dichten Teil des Netzes verläuft. Somit ist es wahrscheinlich, dass beim Hinzufügen neuer Ziele nur geringe Erweiterungen des Baums erforderlich sind.

Ein Nachteil dieser Heuristik besteht darin, dass in dichten Regionen des Netzes möglicherweise weniger freie Zeitslots zur Verfügung stehen, so dass die gewählte Route scheitern könnte. Zudem ist zu beachten, dass ein Pfad mit vielen Kommunikationsnachbarn i. d. R. auch viele Interferenznachbarn hat und somit eine durch einen dichten Teil des Netzes verlaufende Route mehr Slotblockierungen verursacht.

6.2.1.2. Erweiterung eines Routing-Baums

Abonniert ein Knoten einen Dienst, der bereits von anderen Knoten abonniert wurde, so wird der bereits bestehende Baum t zwischen dem Diensterbringer und den bisherigen Dienstnutzern erweitert. Dabei werden zwei Ziele verfolgt: zunächst wird die Gesamtzahl der Links des Baums minimiert und anschließend die Länge des Pfads von der Quelle q (Diensterbringer) zu dem neuen Ziel z (Dienstnutzer). Durch die Minimierung der Linkanzahl wird die Gesamtzahl der benötigten Zeitslots minimiert und durch die Minimierung der Pfadlänge die Verzögerung. Es ist jedoch zu beachten, dass die Verzögerung neben der Länge des Pfads zusätzlich auch von der Lage der reservierten Slots innerhalb eines Superslots abhängt (vgl. Abschnitt 6.2.2).

Um den Baum t zu erweitern, wird zunächst die Menge der kürzesten Kommunikationspfade von allen Knoten $a' \in t$ zu z ermittelt (dies dient dazu, die Gesamtzahl der Links des resultierenden Baums zu minimieren). Diese Menge wird hier mit $P(t, z)$ bezeichnet und ein Pfad der Menge, der in $a' \in t$ startet und in z endet, mit $p_{a',z}$ (gibt es mehr als einen Pfad von a' zu z in $P(t, z)$, so wird der zweite Pfad mit $p'_{a',z}$ bezeichnet, usw.). Aus $P(t, z)$ werden zunächst alle Pfade $p_{a',z}$ gewählt, für die die Länge des Pfads $p_{a'} =_{Df} p_{q,a'} \bullet p_{a',z}$ minimal ist. $\bullet$ steht dabei für die Konkatenation von Pfaden; der Pfad $p_{q,a'}$ ist bereits Bestandteil von t (gibt es mehr als einen Pfad von a' zu z in $P(t, z)$, so wird die Notation $p'_{a'}$ für $p_{q,a'} \bullet p'_{a',z}$ verwendet, usw.). Aus der Menge dieser Pfade wird ein Pfad p_a gewählt, dessen Nachbarschaft die meisten Knoten enthält, und $p_{a,z}$ wird zu t hinzugefügt. Dies lässt sich folgendermaßen formalisieren:

$$P(t, z) =_{Df} \{p_{a',z} : a' \in t \wedge p_{a',z} \in P(a', z) \wedge |p_{a',z}| = \min\{|p_{a'',z}| : a'' \in t \wedge p_{a'',z} \in P(a'', z)\}\}$$

$$p_a \in \{p_{a'} : p_{a',z} \in P(t, z) \wedge |p_{a'}| = \min\{|p_{a''}| : p_{a'',z} \in P(t, z)\} \wedge$$

$$|CN_1(p_{a'})| = \max\{|CN_1(p_{a''})| : p_{a'',z} \in P(t, z) \wedge$$

$$|p_{a''}| = \min\{|p_{a'''}| : p_{a''',z} \in P(t, z)\}\}\}\,.$$

Abbildung 6.3 zeigt ein Beispiel für die beschriebenen Heuristiken. In diesem Netz existiert bereits ein Baum t von der Quelle q zu den Zielen z_1 und z_2. Es soll nun ein weiteres Ziel z_n zu t hinzugefügt werden. Für dieses gilt $P(t, z_n) = \{p_{h,z_n}, p'_{h,z_n}, p_{g,z_n}, p_{d,z_n}\}$ (die Pfade p_{q,z_n}, p_{z_1,z_n} und p_{z_2,z_n} sind länger). Von den Pfaden aus $P(t, z_n)$ liefern p_{h,z_n} und p'_{h,z_n} Pfade p_h und p'_h mit minimaler Gesamtlänge, da $|p_h| = |p'_h| = 4$, $|p_g| = 5$ und $|p_d| = 7$. Für p_h und p'_h gilt $|CN_1(p_h)| > |CN_1(p'_h)|$, so dass p_h gewählt wird.

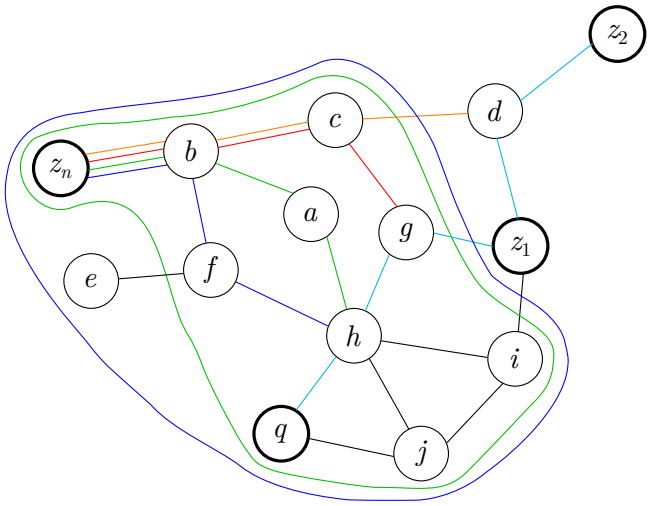

Abbildung 6.3.: Existierender Baum t von q zu z_1 und z_2 sowie Pfade $p_{h,z_n}, p'_{h,z_n}, p_{g,z_n}, p_{d,z_n} \in P(t, z_n)$ und die zu p_h und p'_h gehörenden 1-Hop-Kommunikationsnachbarschaften.

6.2.2. Reservierung von Zeitslots

Aufgrund des zentralisierten Ansatzes werden alle Reservierungen vom Masterknoten berechnet. Da jede Übertragung im selben Zeitslot mit einem ACK bestätigt wird, wird die in Definition 2.9 vorgestellte wechselseitige globale Reservierungsbedingung verwendet. Diese kann vom Master direkt berechnet werden, da er die Kommunikations- und Interferenztopologie sowie alle Reservierungen (und damit auch die Blockierungen) kennt. Somit kann der Master die Menge der freien Slots für einen Link ermitteln.

Die reservierten Slots müssen den betreffenden Knoten mitgeteilt werden. Dazu werden die für jeden Knoten zur Verfügung stehenden Kontrollslots verwendet. Damit der Master nicht an jeden Knoten ein separates Paket versenden muss, und um lokalen Multicast zu unterstützen (s. Abschnitt 6.2.2.2), wird der gesamte Schedule versendet. Eine kurze Beschreibung dieser *Route Confirm Phase* sowie des verwendeten Paketformats ist in [Geb15] zu finden. Diese Aspekte werden hier jedoch nicht näher betrachtet.

Da Reservierungen nur vom Master vorgenommen werden, ist es nicht erforderlich, die dadurch verursachten Blockierungen an die entsprechenden Interferenznachbarn zu übertragen.

6.2.2.1. Reservierung von Zeitslots bei Erzeugung eines Routing-Baums

Für die Reservierung von Zeitslots entlang eines Pfads (ein neuer Baum besteht zunächst immer aus einem einzelnen Pfad) stehen zwei Heuristiken zur Verfügung. Die erste zielt darauf ab, die Verzögerung gering zu halten, während die zweite die Nutzung der Slots optimiert.

Minimierung der Verzögerung

Eine untere Schranke für die Verzögerung eines Pfads (gemessen in Slots) ist die Anzahl seiner Links. Diese Schranke wird erreicht, wenn für die einzelnen Knoten des Pfads aufeinanderfolgende Slots reserviert werden können (ein Beispiel für eine besonders ungünstige Reservierung ist Slot 5 für den ersten Knoten und Slot 4 für den nächsten, da die Verzögerung dann fast einen ganzen Superslot beträgt). Allgemein besteht das Ziel darin, die Lücken zwischen den reservierten Slots möglichst gering zu halten. Um dies zu erreichen, wird jeweils der nächste

freie Slot gewählt. Diese Heuristik minimiert die Verzögerung; ein Nachteil besteht jedoch darin, dass die Slotwiederverwendung nicht berücksichtigt wird. Somit könnte der Fall eintreten, dass für nachfolgende Routensuchen nicht mehr genügend freie Slots gefunden werden können, was bei Verwendung einer anderen Heuristik möglich wäre.

Maximierung der Slotnutzung

Um die Slotnutzung zu maximieren, werden die Slots so ausgewählt, dass einerseits die Wahrscheinlichkeit möglichst groß ist, die aktuelle Route erfolgreich etablieren zu können und andererseits möglichst viele Slots für nachfolgende Routensuchen nutzbar bleiben. Auf diese Weise werden das SSSR- und das SSNR-Problem (s. Abschnitt 3.1.2) abgeschwächt. Diese Heuristik kann verwendet werden, wenn die Minimierung der Verzögerung weniger relevant ist, jedoch eine hohe Netzauslastung vorliegt.

Um dies umzusetzen, werden mehrere Policies verwendet. Als erstes kommt die *Fewest Possibilities First Policy* (FPFP) zum Einsatz. Diese besteht darin, zunächst den Link (a, b) auszuwählen, für den am wenigsten freie Slots zur Verfügung stehen. Die Verwendung von FPFP erhöht die Wahrscheinlichkeit, dass für die gesamte Route freie Slots gefunden werden können.

Um den Slot für den mittels FPFP gewählten Link (a, b) zu bestimmen, wird zunächst die in [SCCC06] beschriebene *Least Conflict First Policy* (LCFP) verwendet (s. Abschnitt 5.2.2.1). Es wird für (a, b) also ein Slot zugewiesen, der am wenigsten Konflikte mit den übrigen Links des Pfads hat. Aufgrund der globalen Reservierungsinformation des Masterknotens kann hier der gesamte Pfad betrachtet werden (in [SCCC06] werden immer nur drei Links betrachtet, da dort ein verteilter Ansatz verwendet wird).

Stehen nach Anwendung von LCFP noch mehrere Slots für (a, b) zur Auswahl, so wird die – ebenfalls in [SCCC06] beschriebene – *Most Reuse First Policy* (MRFP) verwendet. Hier wird diese geringfügig abgewandelt. Es wird ein Slot s ausgewählt, für den die meisten Knoten in Interferenzreichweite von a und b bereits blockiert sind. Aufgrund der globalen Information des Masterknotens kann lokal ermittelt werden, auf welche Slots dies zutrifft.

Anmerkung: Da aufgrund der ACKs jede Sende- auch eine Empfangsreservierung beinhaltet, muss nicht zwischen Sende- und Empfangsblockierungen unterschieden werden. Aus dem gleichen Grund kann es hier nicht vorkommen, dass Interferenznachbarn von a und b bereits Reservierungen für s haben, denn in diesem Fall würde zwangsläufig $\neg F^s(a, b)$ gelten.

Anmerkung: Die ebenfalls in [SCCC06] beschriebene *3-Hop Backward Decision Policy* (3BDP) wird hier nicht benötigt, da der Masterknoten alle Slots festlegt.

6.2.2.2. Reservierung von Zeitslots bei Erweiterung eines Routing-Baums

Wenn ein neues Ziel zu einem Baum t hinzugefügt wird, müssen Slots für die Knoten des entsprechenden Pfads gewählt werden. Dazu können im Wesentlichen die bereits beschriebenen Heuristiken verwendet werden. Eine Ausnahme bildet jedoch der Knoten $a \in t$, bei dem der neue Abzweig startet. Hier wird nach Möglichkeit ein *lokaler Multicast* verwendet. Da eine Übertragung stets von allen Knoten in Kommunikationsreichweite empfangen wird, kann dies ausgenutzt werden, um mit einer Übertragung mehrere Knoten zu erreichen. Jedoch ist diese Zahl nicht beliebig, da die Empfänger im selben Slot ein ACK senden. Deshalb wurde angenommen, dass mit einer Übertragung bis zu drei Knoten erreicht werden können.

Für das Scheduling der ACKs können die IDs der Knoten verwendet werden. Beispielsweise können die Knoten in aufsteigender Reihenfolge ihrer IDs senden. Nach Ablauf der *Route Confirm Phase* kennt jeder Knoten den gesamten Schedule. Somit kennt jeder Empfänger

eines lokalen Multicasts die übrigen Empfänger und kann die eindeutige Reihenfolge der ACKs ermitteln.

Sendet a in Slot s bisher an höchstens zwei Knoten, so kann ein lokaler Multicast verwendet werden, um den nächsten Knoten b des neuen Pfads zu erreichen, sofern die wechselseitige globale Reservierungsbedingung für den Link (a, b) erfüllt ist (hierbei ist zu beachten, dass Blockierungen, die durch den Sendevorgang von a verursacht werden, keine Rolle spielen). Kann kein lokaler Multicast verwendet werden (entweder weil b in Slot s durch andere Übertragungen blockiert ist oder weil a bereits drei Empfänger hat), so wird für (a, b) ein weiterer Slot benötigt.

Nach der Auswahl eines Slots für (a, b) werden die restlichen Slots gemäß der verwendeten Heuristik (Minimierung der Verzögerung oder Maximierung der Slotnutzung) ermittelt. Zur Minimierung der Verzögerung wird für den nächsten Link der erste freie Slot nach dem für (a, b) verwendeten zugewiesen.

6.3. Erweiterung auf teilweise mobile Netze

Im Folgenden werden die in QMRP integrierten Erweiterungen beschrieben, um neben den stationären auch mobile Knoten (beispielsweise autonome Roboter) zu unterstützen. Mobile Knoten können sowohl Diensterbringer als auch Dienstnutzer sein. Sind sie Diensterbringer, so können neben stationären auch andere mobile Knoten die angebotenen Dienste nutzen. Im Umkehrschluss können mobile Knoten sowohl die Dienste stationärer als auch anderer mobiler Knoten nutzen.

Eine Möglichkeit besteht darin, mobile Knoten bei der Routensuche wie stationäre zu behandeln. Dies impliziert jedoch häufige Routenbrüche aufgrund von Positionsänderungen. Ein weiteres, noch größeres Problem ergibt sich dadurch, dass durch die Positionsänderungen andere Übertragungen gestört werden können, wenn ein mobiler Knoten sich in die Interferenzreichweite anderer Knoten bewegt.

Um diese Probleme zu vermeiden, wurde für mobile Knoten eine eigene Strategie entwickelt, welche jedoch mehr Ressourcen (in Form von reservierten Zeitslots) verbraucht. Um diese Strategie umzusetzen, werden die bereits in Abschnitt 6.1 erwähnten Access Nodes $V_{acc}(m)$ verwendet, welche für jeden mobilen Knoten m existieren. Diese dienen dazu, Daten von m oder an m weiterzuleiten.

6.3.1. Ermittlung der Routing-Bäume in teilweise mobilen Netzen

Ist ein mobiler Knoten Bestandteil eines Multicast-Baums, so kann er sowohl als Ziel (Dienstnutzer) als auch als Quelle (Diensterbringer) auftreten. Im Folgenden werden diese beiden Fälle gesondert betrachtet. Da sich die beschriebenen Verfahren kombinieren lassen, sind auch Bäume möglich, bei denen sowohl die Quelle als auch ein oder mehrere Ziele mobile Knoten sind. Es können auch stationäre und mobile Knoten gleichzeitig als Ziele auftreten. Mobile Knoten können jedoch keine Zwischenknoten sein, da Routen immer durch das stationäre Netz geführt werden und jeder mobile Knoten nur mit seinem jeweils aktiven Access Node kommuniziert (dieser ist immer eindeutig bestimmt; vgl. Abschnitt 6.1).

6.3.1.1. Mobiler Knoten als Ziel

Abonniert ein mobiler Knoten m einen Dienst, so muss eine Route von der Quelle q (Diensterbringer) zu m aufgebaut werden (zur Vereinfachung wird zunächst davon ausgegangen,

dass noch kein anderer Knoten den betreffenden Dienst von q abonniert hat und dass q ein stationärer Knoten ist). Um dies zu erreichen, wird ein Multicast-Baum von q zu allen Knoten aus $V_{acc}(m)$ aufgebaut, welcher dann um einen letzten Hop zu m erweitert wird. Dieser letzte Hop wird jedoch nur von dem jeweils aktiven Access Node aus $V_{acc}(m)$ verwendet. Der einzige Unterschied zum stationären Routing besteht darin, dass nun ein kompletter Baum anstelle eines einzelnen Pfads ermittelt werden muss. Dazu wird zunächst eine Route von q zu einem Access Node $n \in V_{acc}(m)$ ermittelt, dessen Distanz zu q am kleinsten ist, d. h.

$$n \in \{n' : n' \in V_{acc}(m) \wedge d_{G_{cl}}(q, n') = \min\{d_{G_{cl}}(q, n'') : n'' \in V_{acc}(m)\}\}\,.$$

Die Ermittlung dieser Route erfolgt analog zur initialen Ermittlung eines stationären Baums (s. Abschnitt 6.2.1.1). Anschließend werden die restlichen Access Nodes aufsteigend nach ihrer Entfernung zu q geordnet und der erzeugte Baum wird sukzessive um Pfade zu diesen Access Nodes erweitert (s. Abschnitt 6.2.1.2). Abbildung 6.4 zeigt ein Beispiel für einen solchen Multicast-Baum.

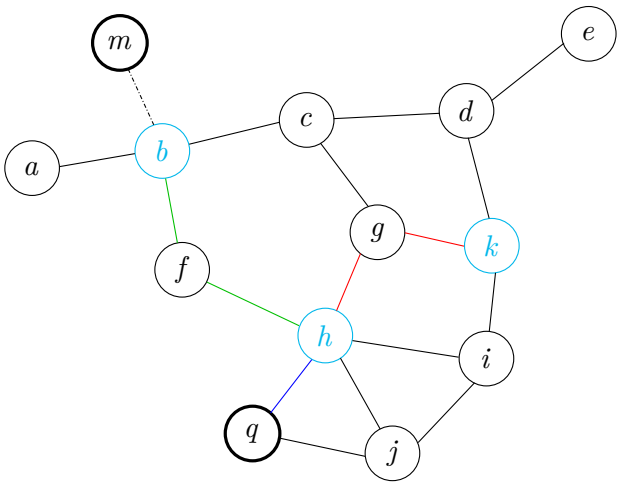

Abbildung 6.4.: Multicast-Baum von einer Quelle q zu den Access Nodes b, h und k des mobilen Ziels m, bestehend aus den Pfaden p_1, p_2 und p_3 (vgl. [GGIK15]).

Um den Multicast-Baum von der Quelle q zu den Access Nodes b, h und k des mobilen Knotens m aufzubauen, wird zunächst Knoten h gewählt, da dieser die kürzeste Distanz zu q hat (Pfad p_1). Der Baum wird anschließend um Zweige zu den Knoten b und k erweitert, die beide die gleiche Distanz zu q haben (Pfade p_2 und p_3). In Abbildung 6.4 ist gerade der Access Node b aktiv, was mit einer unterbrochenen Linie zwischen m und b dargestellt wird.

Soll ein mobiler Knoten als Ziel zu einem bereits bestehenden Multicast-Baum hinzugefügt werden, so entspricht dies dem separaten Hinzufügen seiner Access Nodes (wiederum aufsteigend nach ihrer Distanz zur Quelle geordnet).

6.3.1.2. Mobiler Knoten als Quelle

Abonniert der erste Knoten z einen Dienst eines mobilen Knotens m, so muss eine Route von m zu z aufgebaut werden (zur Vereinfachung wird zunächst davon ausgegangen, dass z ein stationärer Knoten ist). Auch hier werden die Access Nodes verwendet, um die Daten von m ins stationäre Netzwerk zu übertragen. Ein einfacher Ansatz bestünde nun darin, einen

Baum von allen Knoten $n \in V_{acc}(m)$ zu z aufzubauen. Jedoch wäre in diesem Fall für jedes neu hinzuzufügende Ziel (d. h. für jeden weiteren Dienstnutzer) ein eigener Baum erforderlich. Um dies zu vermeiden, wird ein *Distributor Node* festgelegt, an den der jeweils aktive Access Node die Daten von m übermittelt und der sie dann an die Dienstnutzer weiterleitet. Zu diesem Zweck wird ein Concast-Baum (d. h. ein Baum mit mehreren Quellen und einem Ziel) von den Access Nodes zum Distributor Node aufgebaut. Für jeden Zweig dieses Baums wird ein erster Hop von m zu dem entsprechenden Access Node hinzugefügt. Zusätzlich wird ein Multicast-Baum vom Distributor Node zu allen Zielen ermittelt. Da z der erste Knoten ist, der den entsprechenden Dienst von m abonniert, besteht dieser zunächst nur aus einem Pfad zum Ziel z. Der Aufbau dieses Baums sowie dessen Erweiterung beim Hinzufügen weiterer Ziele (sowohl stationäre als auch mobile) erfolgt wie in den vorangegangenen Abschnitten beschrieben, da der Multicast-Baum sich nicht von den bereits betrachteten Fällen (stationäre Bäume und Bäume, die mobile Knoten als Ziele enthalten) unterscheidet.

Um einen möglichst guten Distributor Node zu wählen, kommen verschiedene Strategien in Betracht. Die einfachste besteht darin, den Access Node zu verwenden, welcher zum Zeitpunkt der Anforderung aktiv ist. Diese Strategie ist jedoch nur dann optimal, wenn m sich hauptsächlich in der Reichweite dieses Access Nodes aufhält. Sobald m sich aus dessen Reichweite bewegt, werden die Routen ineffizient. Da der Distributor Node potenziell Daten an alle Knoten des Netzwerks weiterleiten muss, besteht eine andere Strategie darin, diesen so zu wählen, dass er eine möglichst kurze Entfernung zu den meisten Knoten des Netzes hat. Für QMRP wurde diese Strategie verwendet, und es wird ein Distributor Node k gewählt, welcher die kürzeste durchschnittliche Entfernung $d_{avg}(k)$ zu allen anderen stationären Knoten des Netzes hat, d. h.

$$d_{avg}(k') =_{Df} \frac{\sum_{n \in V_{stat}} d_{G_{cl}}(k', n)}{|V_{stat}|}$$

$$k \in \{k' : k' \in V_{stat} \wedge d_{avg}(k') = \min\{d_{avg}(k'') : k'' \in V_{stat}\}\}\,.$$

Anmerkung: Diese Strategie impliziert insbesondere, dass der Distributor Node für alle mobilen Knoten gleich ist.

Um den Concast-Baum von den Access Nodes aus $V_{acc}(m)$ zum Distributor Node k zu erhalten, wird ein Multicast-Baum von k zu den Access Nodes ermittelt und in umgekehrter Richtung verwendet. Dies ist möglich, da generell nur bidirektionale Links betrachtet werden (vgl. Abschnitt 2.2.1). Dazu wird zunächst ein Baum zwischen k und einem der Access Nodes erzeugt (hier könnte man beispielsweise zunächst den Access Node mit der geringsten Entfernung zu k verwenden). Bei der Erweiterung des Concast-Baums um Pfade zu den restlichen Access Nodes ist es im Gegensatz zu Multicast-Bäumen nicht erforderlich, die Gesamtzahl der Links zu minimieren, da jeweils nur ein Zweig gleichzeitig verwendet werden kann (da immer nur ein Knoten aus $V_{acc}(m)$ aktiv ist). Somit besteht hier das Optimierungsziel lediglich darin, die Gesamtlänge der einzelnen Pfade zu minimieren. Auch die Verwendung eines Pfads mit einer möglichst großen Anzahl an Knoten in der Nachbarschaft ist nicht von Vorteil, da dieses Kriterium darauf abzielt, bei späteren Erweiterungen des Baums möglichst wenige Links hinzufügen zu müssen. Dies spielt hier jedoch keine Rolle. Somit wird beim Hinzufügen eines Knotens $n \in V_{acc}(m)$ ein beliebiger Pfad $p \in P(k, n)$ gewählt. Daraus folgt ebenfalls, dass es für die Erzeugung des Baums sinnvoller ist, die Heuristik zu verwenden, welche den existierenden Schedule berücksichtigt, da die Betrachtung der Pfadnachbarschaft hier keinen Vorteil bietet.

Abbildung 6.5 zeigt ein Beispiel für einen Concast-Baum und den zugehörigen Multicast-Baum.

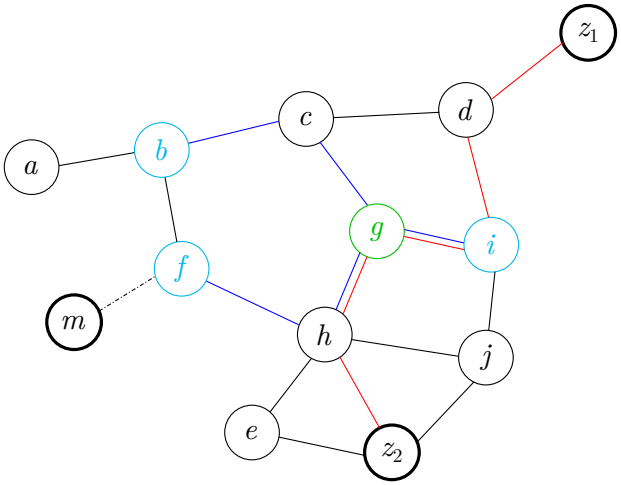

Abbildung 6.5.: Concast-Baum von den Access Nodes b, f und i einer mobilen Quelle m zum Distributor Node g und Multicast-Baum von diesem zu den Zielen z_1 und z_2 (vgl. [GGIK15]).

Für dieses Netz gilt $d_{avg}(g) = d_{avg}(h) = 1.75$, während der Wert für alle anderen Knoten größer ist. Hier wurde Knoten g als Distributor Node gewählt. Wenn die mobile Quelle m an die Ziele z_1 und z_2 senden will, werden die Daten zunächst zum aktiven Access Node übertragen (in Abbildung 6.5 ist dies f; dargestellt mit einer unterbrochenen Linie). Dieser leitet sie dann auf dem entsprechenden Zweig des Concast-Baums an den Distributor Node g weiter, welcher den zugehörigen Multicast-Baum verwendet, um die Daten an z_1 und z_2 zu übermitteln. Dabei ist zu beachten, dass manche Links in beide Richtungen verwendet werden. Ist der Access Node f aktiv, so ist dies beispielsweise für den Link zwischen g und h der Fall. Das ist hier zwar ineffizient, kann jedoch nicht vermieden werden, da die Route auch dann noch funktionieren soll, wenn ein anderer Access Node (z. B. b) aktiv ist.

6.3.2. Reservierung von Zeitslots für mobile Knoten

Bei der Reservierung von Zeitslots sind für mobile Knoten einige Besonderheiten zu beachten. Auch hier wird danach unterschieden, ob der mobile Knoten als Quelle oder als Ziel auftritt.

6.3.2.1. Mobiler Knoten als Ziel

Ist der mobile Knoten m das Ziel, so wurde ein Multicast-Baum von der Quelle zu allen Access Nodes erstellt. Dieser besteht lediglich aus stationären Knoten, so dass die Slotreservierungen wie in Abschnitt 6.2.2 beschrieben vorgenommen werden können (es wird hier angenommen, dass die Quelle stationär ist; der Fall einer mobilen Quelle wird im nächsten Abschnitt behandelt). Zusätzlich muss ein Slot für den letzten Hop von den Access Nodes zu m reserviert werden. Da zu jedem Zeitpunkt jeweils nur einer der Access Nodes die Daten für den mobilen Knoten weiterleitet, ist es ausreichend, hier insgesamt einen Slot zu reservieren. Es ist jedoch zu beachten, dass diese Reservierung netzweit erfolgen muss (d. h. keine Verwendung von SDMA), da der mobile Knoten seine Position ändern und dadurch potenziell mit allen

Knoten des Netzes interferieren kann (da er ACKs sendet). Eine Wiederverwendung des Slots könnte deshalb dazu führen, dass andere Übertragungen gestört werden.

6.3.2.2. Mobiler Knoten als Quelle

Ist der mobile Knoten m die Quelle, so muss zunächst ein Slot für die Übertragung von m zum jeweils aktiven Access Node netzweit reserviert werden. Da immer nur ein Access Node gleichzeitig aktiv ist, reicht ein Slot aus.

Weiterhin müssen Slots für den Concast-Baum von den Access Nodes zum Distributor Node reserviert werden. Da jedoch nur ein Zweig des Concast-Baums gleichzeitig aktiv ist, blockieren die Reservierungen der einzelnen Zweige sich nicht gegenseitig. Somit können beim Scheduling der Slots für einen Zweig die auf anderen Zweigen dieses Baums reservierten Slots wiederverwendet werden. Beim Scheduling eines Zweigs werden daher Blockierungen, die lediglich durch andere Zweige desselben Concast-Baums verursacht werden, nicht berücksichtigt. Das kann dazu führen, dass für alle Zweige dieselben Slots verwendet werden; dies ist jedoch nicht zwangsläufig der Fall.

Zusätzlich werden Reservierungen für den Multicast-Baum vom Distributor Node zu den Zielen benötigt. Wie diese getroffen werden, wurde in Abschnitt 6.2.2 für stationäre Knoten und in Abschnitt 6.3.2.1 für mobile Knoten beschrieben.

Kapitel 7.

7 Entwicklung von Simulationskomponenten

Die Entwicklung verteilter eingebetteter Systeme erfordert die gemeinsame Simulation von Komponenten auf unterschiedlichen Abstraktionsebenen. Zudem kommen für die Entwicklung solcher Komponenten unterschiedliche Techniken zum Einsatz (beispielsweise Matlab Simulink für die Spezifikation von Funktionen oder SDL für die Definition des Verhaltens von Knoten). Aus diesem Grund wurde das Simulationsframework FERAL entwickelt, welches die Integration verschiedenster Simulatoren unterstützt. Mehrere Simulatoren wurden dort bereits integriert bzw. eigens für FERAL entwickelt.

Um realistische Netzwerke simulieren zu können, wurden im Rahmen dieser Arbeit einige von dem Netzwerksimulator ns-3 bereitgestellte Kommunikationstechnologien in FERAL integriert. Außerdem wurde ein Modul zur Simulation des im Bereich eingebetteter Systeme gebräuchlichen CC2420-Transceivers entwickelt (eine erste Version dieses Moduls wurde in [Gro12] vorgestellt). Das entwickelte Modul kann sowohl für eigenständige ns-3-Simulationen als auch für Simulationen mit FERAL verwendet werden. Ergebnisse dieses Kapitels wurden auch in [BCG+13, IG13, BCG+14] publiziert.

In diesem Kapitel wird zunächst ein Überblick über das Simulationsframework FERAL gegeben. Anschließend werden einige integrierte Komponenten kurz vorgestellt, die nicht im Rahmen dieser Arbeit entwickelt wurden, die jedoch in umfangreicheren Simulationssystemen Verwendung finden (s. Abschnitt 7.7.2). Weiterhin wird die generelle Konstruktion eines Simulationssystems kurz beschrieben. Danach werden ns-3 sowie dessen Integration in FERAL vorgestellt; im Anschluss daran wird das CC2420-Modul für ns-3 und FERAL beschrieben. Zudem werden einige Simulationssysteme präsentiert und die Ergebnisse der damit durchgeführten Simulationen beleuchtet. Abschließend wird ein Vergleich mit anderen Simulationsansätzen vorgenommen.

7.1. Simulationsframework FERAL

FERAL (*Framework for the Efficient simulator coupling on Requirements and Architecture Level*) ist ein Java-basiertes Simulationsframework zur Evaluierung funktionaler und nichtfunktionaler Anforderungen vernetzter Systeme. Es wurde in [BCG+13, BCG+14] vorgestellt. Eine genauere Beschreibung seiner Semantik ist in [KFBG13] zu finden. FERAL ist eine Weiterentwicklung älterer Simulationsansätze (s. z. B. [KGGR05, KB06, KG08, Kuh09]) und verwendet einige Konzepte des Ptolemy-Frameworks [EJL+03].

Aufgrund seiner modularen Struktur und der losen Kopplung der Simulationskomponenten unterstützt FERAL die schnelle Entwicklung holistischer Simulationsumgebungen. Dies wird beispielsweise für virtuelles Prototyping benötigt, um Entwurfsalternativen (z. B. verschiedene Kommunikationstechnologien) in frühen Entwicklungsphasen zu evaluieren. Aufgrund der

Integration spezialisierter, domänenspezifischer und maßgeschneiderter Simulatoren sowie der Erweiterbarkeit von FERAL können die damit erzeugten Simulationssysteme Komponenten unterschiedlicher Abstraktionsebenen beinhalten. Diese Komponenten können dann sukzessive verfeinert werden. Dadurch ermöglicht FERAL kontinuierliches Testen und kontinuierliche Integration während des gesamten Entwicklungsprozesses.

Das Metamodell eines Simulationssystems für FERAL ist in Abbildung 7.1 dargestellt.

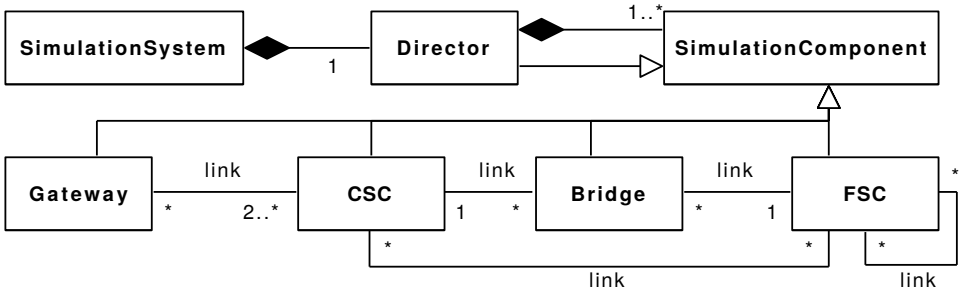

Abbildung 7.1.: Metamodell für FERAL-Simulationssysteme [BCG+13, BCG+14].

Jedes Simulationssystem (*SimulationSystem*) enthält einen einzelnen Root-Direktor (*Director*), welcher die Ausführung von mindestens einer Simulationskomponente (*SimulationComponent*) kontrolliert. Simulationskomponenten haben eindeutige Namen und interagieren über benannte Ports, welche durch unidirektionale Links verbunden sind.

Simulationskomponenten sind entweder steuernde Komponenten (*Directors*), verhaltensbasierte Komponenten (*CSCs* und *FSCs*; *Communication-based Simulation Components* und *Functional Simulation Components*) oder konvertierende Komponenten (*Bridges* und *Gateways*). Da die Direktoren selbst als Simulationskomponenten konzipiert sind, können sie beliebig verschachtelt werden.

Verhaltensbasierte Komponenten simulieren Kommunikationstechnologien (CSCs) oder Anwendungsfunktionalitäten (FSCs). Sie werden durch spezialisierte Simulatoren realisiert, welche Verhaltensmodelle ausführen. Konvertierende Simulationskomponenten dienen der Kopplung verhaltensbasierter Komponenten und spielen eine zentrale Rolle für deren Austauschbarkeit. Sie verbinden entweder FSCs mit CSCs (Bridges) – dies ermöglicht eine von Kommunikationsaspekten unabhängige Beschreibung von FSCs – oder CSCs mit CSCs (Gateways), wodurch mehrere (auch unterschiedliche) Kommunikationsmedien verbunden werden können.

Die Integration von spezialisierten Simulatoren für CSCs und FSCs erfordert die Implementierung des *SimulationComponent*-Interface, welches es FERAL ermöglicht, ihre Ausführung zu kontrollieren. Zu diesem Zweck definiert das Interface die Methoden *init*, *preFire*, *fire* und *postFire*. Die Methode *init* wird nur einmal – während der Initialisierung des Frameworks – ausgeführt und kann zur Durchführung komponentenspezifischer Initialisierungen verwendet werden. In jedem Zeitschritt ruft ein Direktor zunächst die Methode *preFire* aller von ihm kontrollierten Simulationskomponenten auf, dann *fire* und schließlich *postFire* (hier wird nur die Semantik zeitgetriggerter Direktoren betrachtet; das Verhalten anderer Typen (beispielsweise ereignisgetriggerter Direktoren) ist in [KFBG13] beschrieben). Diese Aufrufe sind Bestandteil der *fire*-Methode des Direktors. Die *fire*-Methode des Root-Direktors wird direkt von FERAL ausgeführt. Die Methode *preFire* leitet Nachrichten vorheriger Zeitschritte an die entsprechenden Zielkomponenten weiter. Berechnungen und Simulationen für einen Zeitschritt werden als Bestandteil der *fire*-Methode der entsprechenden Simulationskomponente ausgeführt. Der Direktor ruft *postFire* als letzte Methode eines Zeitschritts auf, um alle erzeugten Nachrichten

der von ihm kontrollierten Simulationskomponenten einzusammeln, bevor er die Simulationszeit weiterschaltet.

Eine wesentliche Herausforderung bei der Kombination mehrerer Simulatoren besteht darin, ihre MOCCs (Models of Computation and Communication) anzugleichen, ihre Ausführung zu kontrollieren und ihre Kommunikationsmuster zu definieren. Abhängig vom MOCC einer Simulationskomponente muss ein passender übergeordneter Direktor gewählt werden (FERAL unterstützt zeitgetriggerte, ereignisgetriggerte und datenflussorientierte Semantiken). Dieser Direktor führt die zugeordneten Simulationskomponenten aus und leitet Nachrichten unter Berücksichtigung der entsprechenden MOCCs weiter.

7.2. Integration von Simulatoren in FERAL

Der Aufwand zur Integration existierender Simulatoren in FERAL hängt von deren Architektur, Schnittstelle, MOCC sowie der Implementierungssprache ab. Die Integration eines Simulators erfolgt stets in Form einer Simulationskomponente, welche dessen Ansteuerung übernimmt. Dazu muss das von FERAL bereitgestellten *SimulationComponent*-Interface implementiert werden (s. Abschnitt 7.1). Aus Sicht von FERAL unterscheiden sich damit integrierte Simulatoren nicht von direkt implementierten (nativen) Simulationskomponenten (diese werden in Abschnitt 7.3 erläutert).

Im Rahmen dieser Arbeit wurden u. a. Komponenten des Netzwerksimulators ns-3 (dieser dient der Simulation von CSCs) in FERAL integriert, was durch die in Abschnitt 7.5 beschriebene Simulationskomponente ns-3 realisiert wird. Hier wird jedoch zunächst ein kurzer Überblick über die Integration anderer Simulatoren in FERAL gegeben, welche zusammen mit der Simulationskomponente ns-3 zur Realisierung des in Abschnitt 7.7.2 beschriebenen Adaptive Cruise Control-Szenarios verwendet werden. Detailliertere Erläuterungen zu den im Folgenden beschriebenen Komponenten sind in [BCG+13, BCG+14] zu finden.

7.2.1. Simulatoren für FSCs

Um die modellbasierte Spezifikation von Funktionalitäten oder Knoten eines verteilten Systems zu unterstützen, wurden FSC-Simulatoren in FERAL integriert, beispielsweise ein Simulator zur Spezifikation von Funktionen mit Matlab Simulink [Mat15] sowie ein Simulator für SDL [Int12a].

Integration von Simulink

Das von MathWorks entwickelte Matlab Simulink [Mat15] ist im Bereich der modellbasierten Entwicklung und Simulation von dynamischen und eingebetteten Systemen weit verbreitet [NL02]. Simulink unterstützt die Modellierung von zeitdiskreten Kontrollalgorithmen und dynamischen Systemen, aber auch die Entwicklung zustandsbasierter Systeme mit Hilfe des Werkzeugs Stateflow. Ferner ermöglicht es die Spezifikation und Entwicklung der Funktionalitäten eingebetteter Systeme auf unterschiedlichen Abstraktionsebenen, welche von unvollständigen Prototypen bis zu kompletten Algorithmen reichen.

Durch die Integration von Simulink in FERAL können Simulink-Modelle als FSCs in Simulationssystemen ausgeführt werden. Jedes Simulink-System wird dabei zu einer unabhängigen Simulationskomponente. Die Integration von Simulink basiert auf C-Code, welcher von MathWorks' *Embedded Coder* generiert und unter der Kontrolle des Simulink-Simulators ausgeführt wird. Zur Kopplung von FERAL mit dem generierten C-Code wird das *Java Native*

Interface (JNI) verwendet. Simulink-Simulationskomponenten werden von einem zeitgetriggerten Direktor ausgeführt, welcher die zeitgetriggerte Ausführungs- und Kommunikationssemantik der Simulink-Modelle direkt umsetzt. Während der Ausführung werden die Werte der Eingangs- und Ausgangsports des Simulink-Modells auf entsprechende Datentypen von FERAL und die zugehörigen Ports der Simulationskomponente abgebildet (und umgekehrt).

Integration von SDL

Die *Specification and Description Language* (SDL) [Int12a] wurde in Abschnitt 3.2 beschrieben. Zur Integration von SDL in FERAL wird die Rational SDL Suite von IBM [IBM15] verwendet. Diese dient dazu, das Verhalten der entsprechenden FSCs zu spezifizieren, welches anschließend in die textuelle Repräsentation SDL-PR überführt und mit dem *Configurable Transpiler for SDL to C++ Translation* (ConTraST) [FGW06, Fli09] in C++-Code übersetzt wird. Zusammen mit dem *SDL Runtime Environment* (SdlRE) [FGW06, Fli09] sowie dem *SDL Environment Framework* (SEnF) [FGJ+05] lässt sich ein ausführbares System erzeugen. SdlRE implementiert die virtuelle Maschine für SDL, welche die Ausführung eines SDL-Systems gemäß der SDL-Semantik steuert. Wird eine SDL-basierte FSC ausgeführt, so führt SdlRE alle ausführbereiten SDL-Transitionen aus. SEnF stellt plattformspezifische Hardwaretreiber zur Verfügung, um auf die Peripherie der jeweiligen Plattform zuzugreifen. Um SDL-Spezifikationen mit FERAL zu verbinden, wurden spezielle virtuelle Treiber für CAN und FlexRay sowie für verschiedene von ns-3 bereitgestellte Kommunikationstechnologien in SEnF integriert. Die Verbindung zwischen FERAL und der C++-Implementierung eines SDL-Systems wird durch JNI realisiert. Für SDL wird sowohl ein zeit- als auch ein ereignisgetriggertes MOCC unterstützt.

7.2.2. Simulatoren für CSCs

Um Entwurfsalternativen zu evaluieren, Laufzeitverhalten vorherzusagen und Integrationstests bei verteilten Systemen durchzuführen, werden realistische Simulatoren für aktuelle Kommunikationstechnologien benötigt. Der Schwerpunkt liegt hier auf der Evaluierung nichtfunktionaler Anforderungen (z. B. bezüglich Latenzen) sowie auf einem Performanzvergleich zwischen verschiedenen Kommunikationstechnologien. Neben der Integration von ns-3 wurden Simulatoren für CAN und FlexRay eigens für FERAL entwickelt [BCG+13, BCG+14].

Integration von Simulatoren für CAN und FlexRay

CAN (Controller Area Network) und FlexRay sind Feldbustechnologien, die im Automobil- und Automatisierungsbereich weit verbreitet sind. Für beide wurden realistische Simulatoren entwickelt, welche in Java umgesetzt wurden. Implementierungsdetails sind in [BCG+13, BCG+14] zu finden.

CAN ist ein ereignisgetriggerter Kommunikationsbus, welcher von der Robert Bosch GmbH seit 1983 entwickelt und insbesondere im Automobilbereich für sicherheitskritische ebenso wie für komfortbezogene Funktionen verwendet wird [Ets01]. Er hat eine Datenrate von bis zu 1 MBit/s. CAN basiert auf dem CSMA/CA-Mechanismus. Jede Rahmenübertragung beginnt mit einem CAN-Nachrichtenidentifier, welcher die Priorität des Rahmens repräsentiert. Starten mehrere Knoten gleichzeitig eine Übertragung, so gewinnt der Knoten die Arbitrierung, dessen Rahmen die höchste Priorität hat. Die Übertragungen anderer Knoten werden verzögert, bis die nächste Arbitrierung beginnt.

FlexRay ist ein deterministischer, zeitgetriggerter Feldbus, der speziell für die Automobilindustrie entwickelt wurde [Fle05, Fle10]. Er bietet zwei unabhängige physikalische Kanäle, welche jeweils eine Datenrate von bis zu 10 MBit/s bereitstellen. Als Medienzugriffsverfahren

kommt TDMA im statischen Segment und FTDMA (Flexible TDMA) im optionalen dynamischen Segment zum Einsatz. Das statische Segment ist in statische Slots gleicher Dauer unterteilt und ermöglicht deterministische Garantien bezüglich Latenz und Jitter eines Rahmens. Der FTDMA-Mechanismus des dynamischen Segments erlaubt prioritätsbasierte Übertragungen mit dynamischer Akquisition von Bandbreite. Der entwickelte Simulator unterstützt die Versionen 2.1a und 3.0.1 der FlexRay-Protokollspezifikation.

7.3. Konstruktion eines Simulationssystems

Die Konstruktion eines Simulationssystems erfolgt in zwei Schritten: Zunächst wird ein abstraktes Simulationssystem definiert, welches die abstrakte Struktur und Topologie beschreibt. Die konkrete Realisierung der Komponenten wird dort nicht betrachtet. Anschließend wird ein konkretes Simulationssystem instanziiert, indem Simulatoren und entsprechende Funktions- bzw. Kommunikationsmodelle für die Komponenten des abstrakten Simulationssystems ausgewählt werden.

Ein abstraktes Simulationssystem beschreibt Topologie und Simulationskomponenten eines Simulationsaufbaus sowie die Verbindungsstruktur. Zur graphischen Darstellung erhalten Simulationskomponenten Annotationen in Form von Stereotypen, welche ihren jeweiligen Typ spezifizieren (z. B. FSC, CSC, Bridge, Gateway). Die Interaktion zwischen Simulationskomponenten wird mit Hilfe von Links in abstrakter Form beschrieben. Eindeutige Namen dienen zur Identifikation der Komponenten und um ihre jeweilige Rolle innerhalb des Simulationssystems zu dokumentieren. Beispiele für abstrakte Simulationssysteme sind in den Abschnitten 7.7.1.2 und 7.7.2 zu finden.

Durch die Auswahl von Verhaltensmodellen für die FSCs und Kommunikationstechnologien für die CSCs (zusammen mit den entsprechenden Simulatoren) wird ein abstraktes Simulationssystem zu einem konkreten (in diesem werden konkrete Simulationskomponenten instanziiert). Für die Spezifikation des Verhaltens von FSCs kann beispielsweise Simulink oder SDL genutzt werden. Für die Spezifikation von CSCs können neben Kommunikationsmedien, die von ns-3 bereitgestellt werden, die in FERAL integrierten Simulatoren für CAN und FlexRay verwendet werden.

Neben Simulationskomponenten, die zur Ansteuerung externer Simulatoren dienen, können zudem FSCs in Form von nativen Simulationskomponenten erstellt werden, welche eigens für FERAL implementiert werden. Diese ermöglichen beispielsweise die schnelle Erstellung von Prototypen für Systemfunktionalitäten, indem sie das externe Verhalten nachbilden, jedoch intern ein vereinfachtes Verhalten haben. Native Simulationskomponenten sind direkt in Java implementiert und kombinieren die erforderliche Unterstützung der Simulationskontrolle (z. B. Implementierung der Methoden von *SimulationComponent*) mit einem entsprechenden Verhaltensmodell. Für die Interaktion mit CSCs können Bridges verwendet werden. Native Simulationskomponenten können zwar prinzipiell auch zur Spezifikation von CSCs verwendet werden, sind jedoch primär für FSCs gedacht, da CSCs meist eher komplex sind (z. B. weil das simulierte Medium ein komplexes Verhalten hat), so dass für letztere die Verwendung spezieller Simulatoren sinnvoller ist.

Um die Austauschbarkeit von Simulationskomponenten zu gewährleisten, wurden Bridges und Gateways in FERAL integriert, welche kommunikationsrelevante Aspekte von den Modellen der FSCs trennen. Ziel ist es, verschiedene Kommunikationstechnologien evaluieren zu können, ohne dass dazu Modifikationen des abstrakten Simulationssystems und der FSCs erforderlich sind.

Bridges verbinden FSCs mit CSCs, indem kommunikationsspezifische Eigenschaften (z. B. CAN-Nachrichtenprioritäten oder FlexRay-Puffer) gekapselt werden. Dadurch können FSCs ohne Berücksichtigung des verwendeten Kommunikationsmediums spezifiziert werden. Die Verwendung von Bridges beschränkt sich auf FSCs basierend auf Simulink und nativen Implementierungen, da diese oft als Prototypen, Black-Box-Komponenten oder Platzhalter in frühen Entwicklungsphasen verwendet werden. Entwurfsalternativen (beispielsweise verschiedene Kommunikationstechnologien) werden typischerweise in diesen Phasen evaluiert.

Gateways dienen der Verbindung von CSCs untereinander. Sie leiten Nachrichten zwischen verschiedenen Medien weiter und bieten damit Schnittstellen zwischen unterschiedlichen Kommunikationstechnologien. Auf diese Weise können CSCs ausgetauscht und verschiedene Kombinationen von Kommunikationstechnologien evaluiert werden.

7.4. Netzwerksimulator ns-3

Der Simulator ns-3 [NS-15] ist ein zeitdiskreter, ereignisbasierter Simulator für Netzwerke aller Art. Er kann beispielsweise für Internetsysteme verwendet werden, ist aber nicht darauf beschränkt. ns-3 ist der Nachfolger des bekannten Simulators ns-2 [USC15], welcher im akademischen Bereich immer noch weit verbreitet ist, jedoch nicht mehr aktiv entwickelt wird. Die Hauptvorteile von ns-3 gegenüber ns-2 sind das verständliche, strukturierte Design und die aktive Weiterentwicklung. ns-2 basiert auf einer Kombination von C++ und TCL (*Tool Command Language*), wohingegen ns-3 komplett in C++ umgesetzt wurde. Ein detaillierter Vergleich zwischen ns-2 und ns-3 ist in [FID+10] zu finden. Abbildung 7.2 zeigt die grundsätzliche Struktur des ns-3-Quellcodes.

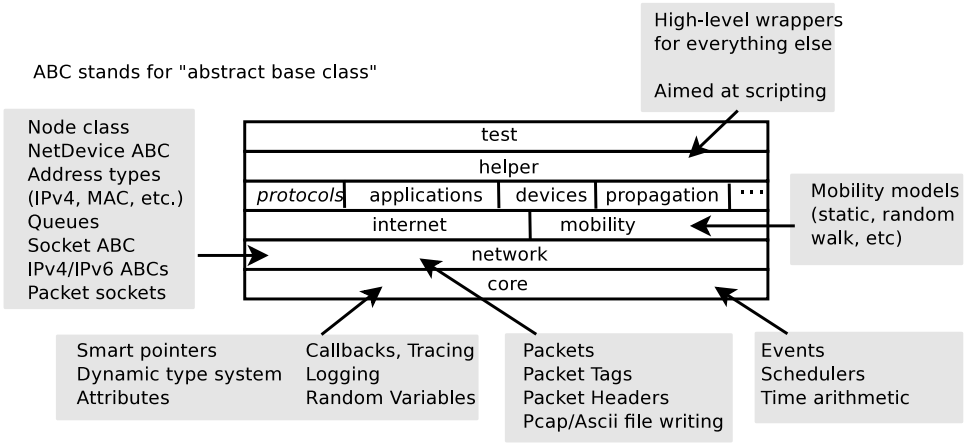

Abbildung 7.2.: Struktur des ns-3-Quellcodes [NS-15].

ns-3-Simulationen können in Form einer *main*-Funktion in C++ erstellt werden, in welcher das Simulationssystem aufgebaut und der Simulator gestartet wird. Die benötigten ns-3-Bibliotheken werden (statisch oder dynamisch) zu dem Programm dazugelinkt (bei ns-3 wird jedes Modul in einer eigenen Bibliothek abgelegt). Zusätzlich wird ein Großteil der Schnittstelle in Python exportiert, so dass Simulationssysteme auch in Form von Python-Programmen erstellt werden können.

In einem Simulationssystem werden Knoten definiert, die über Kanäle miteinander verbunden sind. Auf diesen Knoten können dann NetDevices, Protokollstapel (z.B. UDP / IP) und Applikationen installiert werden. Abbildung 7.3 zeigt den grundsätzlichen Aufbau.

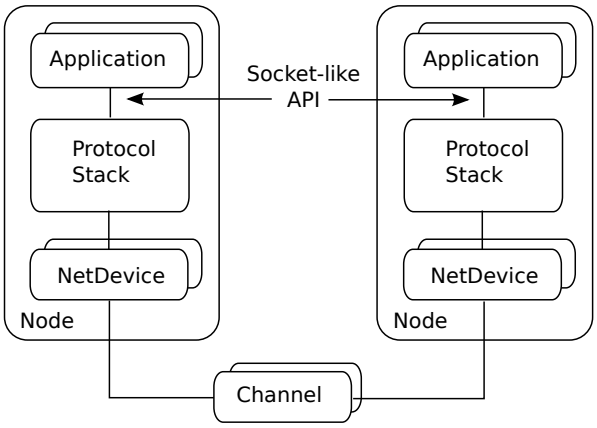

Abbildung 7.3.: Grundsätzlicher Aufbau einer Simulation in ns-3 (vgl. [NS-15]).

NetDevices stellen MAC-Funktionalität bereit und definieren eine Schnittstelle, über die die Netzwerkschicht auf ein physikalisches Gerät zugreifen kann. Der Protokollstapel kann weggelassen werden, d. h. es ist auch möglich, eine Applikation zu definieren, die direkt mit einem NetDevice interagiert. Applikationen dienen dazu, Traffic zu erzeugen und zu verarbeiten. Auf drahtlosen Knoten wird zusätzlich ein Mobilitätsmodell installiert, welches die Positionen und Bewegungen der einzelnen Knoten festlegt. Weiterhin stellt ns-3 diverse Verlust- und Verzögerungsmodelle zur Verfügung, welche für Kanäle verwendet werden können, um deren Übertragungseigenschaften zu modellieren.

Die eigentliche Simulation wird über Ereignisse gesteuert, welche einem Scheduler übergeben werden. Initial werden solche Ereignisse typischerweise durch die auf den Knoten laufenden Applikationen erzeugt. Weitere Ereignisse werden entweder ebenfalls durch die Applikationen erzeugt oder ergeben sich aus dem Programmablauf (beispielsweise löst ein Sendeereignis bei den empfangenden Knoten ein Empfangsereignis aus).

7.5. Anbindung von ns-3 an FERAL

Um ns-3 aus FERAL-Simulationssystemen heraus nutzen zu können, wird eine Simulationskomponente ns-3 definiert, welche von FERAL ausgeführt wird und die ihrerseits den ns-3-Simulator ausführt. Es ist zu beachten, dass in einem FERAL-Simulationssystem stets nur eine Instanz dieser Simulationskomponente erzeugt wird, die jedoch mehrere Medien mittels ns-3 simulieren kann.

Da FERAL in Java geschrieben ist und ns-3 in C++, besteht die Integration aus einem Java-Teil und einem C++-Teil. Die Verbindung zwischen beiden Teilen wird mittels JNI realisiert. Die JNI-Bibliothek dient dazu, den Aufbau des für ns-3 relevanten Teils des FERAL-Simulationssystems an ns-3 zu übermitteln, dessen Ausführung zu steuern und Nachrichten in beide Richtungen zu übertragen. Die Integration modifiziert den existierenden Code von ns-3 nicht. Dadurch können zukünftige Versionen von ns-3 einfach übernommen werden, so dass die Simulationskomponente wartbar und aktuell bleibt. Die Verwendung der Simulationskomponente in einem Simulationssystem für FERAL ist in Anhang E zu finden.

Obwohl ns-3 ein ereignisbasierter Simulator ist, hat die FERAL-Komponente ein zeitgetriggertes Ausführungsmodell. Dies liegt darin begründet, dass ns-3 die Möglichkeit bietet, die Simulation für eine spezifizierte Zeitspanne auszuführen. Damit ist es nicht erforderlich,

die Uhr und den internen Scheduler von ns-3 zu modifizieren, was bei einer ereignisbasierten Ausführung unumgänglich wäre (in diesem Fall müsste man selbst einen Mechanismus implementieren, um die Simulationskontrolle wieder an FERAL zu übergeben).

7.5.1. Aufbau des Java-Teils der Simulationskomponente ns-3

Abbildung 7.4 zeigt den Java-Teil der Simulationskomponente ns-3, wobei einige Hilfsklassen aus Gründen der Lesbarkeit nicht dargestellt sind. Auch zeigt die Abbildung nicht alle Attribute und Methoden der entsprechenden Klassen; beispielsweise sind Getter und Setter für private Attribute nicht aufgeführt (dies gilt auch für die folgenden Abbildungen dieses Kapitels). Die Vererbungshierarchie ist vereinfacht dargestellt, denn *NS3TimeTriggeredComponent* implementiert nicht direkt das von FERAL bereitgestellte Interface *SimulationComponent*, sondern erbt von der – ebenfalls von FERAL bereitgestellten – Klasse *TimeTriggeredSimulationComponent*.

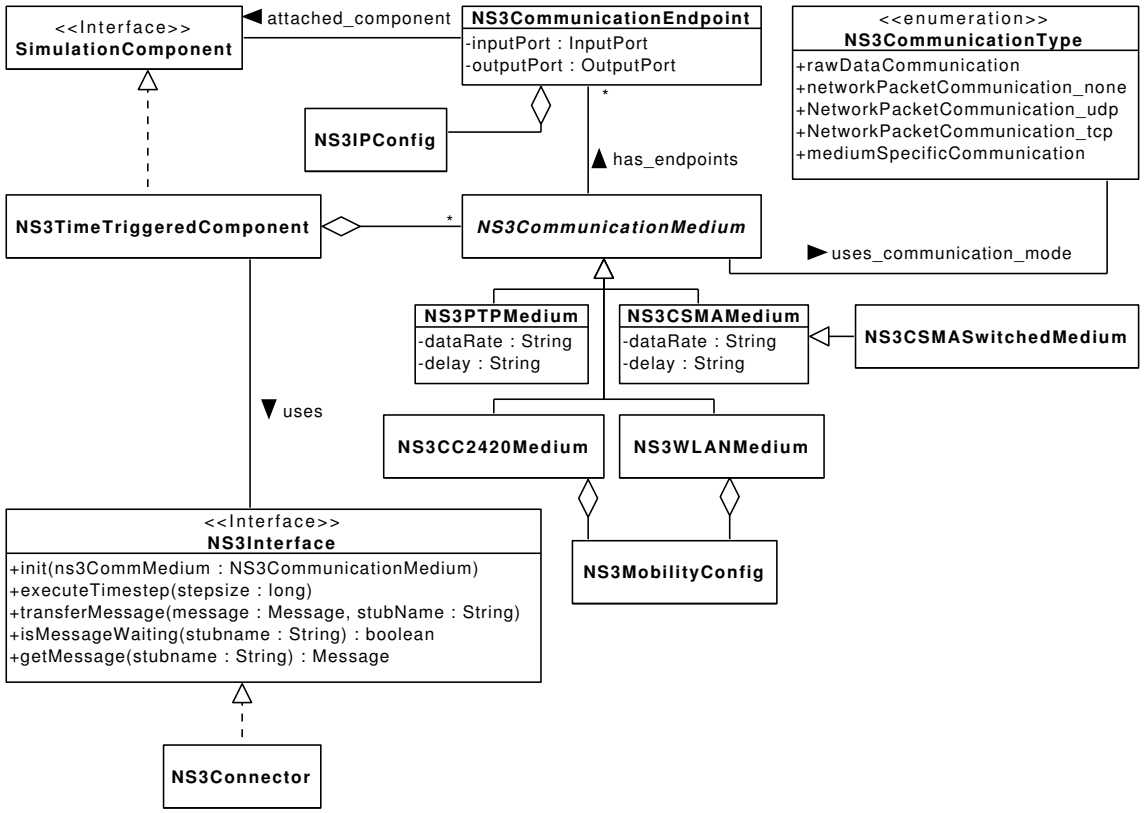

Abbildung 7.4.: Aufbau der Simulationskomponente ns-3 – Java-Teil.

Die Verbindung des Java-Teils zum C++-Teil wird durch *NS3Interface* und *NS3Connector* realisiert. *NS3Interface* stellt die entsprechende Schnittstelle dar, und *NS3Connector* deklariert die Methoden aus *NS3Interface* als `native`. Diese werden dann in C++ implementiert und von einer JNI-Bibliothek bereitgestellt.

Für jedes Kommunikationsmedium, welches mit ns-3 simuliert werden soll, muss ein entsprechendes *NS3CommunicationMedium* instanziiert werden. Für drahtlose Medien gibt es zusätzlich eine *NS3MobilityConfig*, welche das Mobilitätsmodell, d. h. die initialen Positionen sowie die Bewegungen (Richtung und Geschwindigkeit) der zu diesem Medium gehörenden

Knoten festlegt. Standardmäßig werden die Knoten initial auf einem zweidimensionalen Gitter angeordnet, und es wird ein *ConstantPositionMobilityModel* verwendet, d. h. diese bewegen sich nicht. Näheres ist in Anhang E zu finden.

Momentan werden fünf Medien unterstützt. Eines davon ist das CC2420-Medium, welches neu entwickelt wurde. Dieses wird in Abschnitt 7.6 beschrieben. Die restlichen Medien werden von ns-3 bereitgestellt. Bei einem handelt es sich um ein Punkt-zu-Punkt-Medium, welches genau zwei Knoten mit einem Vollduplexkanal verbindet. Zudem wird WLAN (IEEE 802.11) [Ins12a] unterstützt, wobei hier nur der Ad-Hoc-Modus verwendet werden kann. Daneben existiert eine vereinfachte Variante von Ethernet (IEEE 802.3) [Ins12b] in Form des CSMA-Mediums. Abweichend von Ethernet implementiert diese keine Kollisionserkennung, da Kollisionen a priori vermieden werden. Zusätzlich gibt es noch eine geswitchte Variante des CSMA-Mediums, bei der alle Knoten mit einem zentralen Switch verbunden sind, welcher die Nachrichten weiterleitet.

Die zu simulierenden Medien werden von einer *NS3TimeTriggeredComponent* verwaltet. *NS3CommunicationEndpoints* verbinden FSCs, welche das Verhalten der Knoten simulieren, mit dem entsprechenden Kommunikationsmedium. Bei diesen Endpunkten handelt es sich um „passive“ Simulationskomponenten, die wie „aktive“ Simulationskomponenten Ports besitzen, jedoch kein Verhalten implementieren und deshalb auch nicht von einem Direktor ausgeführt werden. Die Ports sind notwendig, um Nachrichten von ns-3 an die mit dem Endpunkt verbundene Simulationskomponente weiterleiten zu können (und umgekehrt). Jeder Endpunkt hat einen eindeutigen Namen und besitzt genau einen *InputPort* sowie genau einen *OutputPort*, mit dem jeweils die *OutputPorts* bzw. *InputPorts* der zugehörigen Simulationskomponente verschaltet werden. Der Ablauf einer solchen Nachrichtenübertragung ist in Abschnitt 7.5.1.1 für den Java-Teil beschrieben. Ein Beispiel für die Verschaltung der Ports ist in Anhang E zu finden.

In ns-3 werden Endpunkte durch eigens definierte *virtuelle Applikationen* repräsentiert (vgl. Abschnitt 7.5.2). Während Applikationen in einer eigenständigen ns-3-Simulation dazu dienen, Traffic zu erzeugen und zu verarbeiten, haben die virtuellen Applikationen kein eigentliches Verhalten, sondern werden nur für die Kommunikation verwendet und dienen auf C++-Seite als Gegenstück für die Java-Endpunkte. Die Kommunikation zwischen Endpunkten und den zugehörigen virtuellen Applikationen erfolgt über den Austausch von Nachrichten. Das Gegenstück zum *NS3CommunicationMedium* ist der ns-3-Kanal, welcher das eigentliche Kommunikationsmedium simuliert.

Weiterhin wird ein *Kommunikationsmodus* eingeführt, welcher das für ein Medium zu verwendende Nachrichtenformat sowie ggf. den entsprechenden Protokollstapel festlegt. Für jeden Modus gibt es eine eigene generische virtuelle Applikation, welche die entsprechenden Nachrichten in ns-3-Pakete umwandelt und sie gemäß des gewählten Kommunikationsmodus sendet. Deshalb muss dieser beim Aufbau des Simulationssystems festgelegt werden und ist nachträglich nicht änderbar. Die verwendeten Java-Nachrichtenformate für die einzelnen Modi sind in Abschnitt 7.5.1.2 näher beschrieben.

Zunächst gibt es den standardmäßig verwendeten Modus *rawDataCommunication*, welcher zum Versenden von Nutzdaten ohne irgendeine Form der Adressierung dient. Er ist hauptsächlich für Testzwecke gedacht. In ns-3 wird kein Protokollstapel auf den Knoten installiert, d. h. die Nachrichten werden von der virtuellen Applikation direkt an das NetDevice des Knotens weitergeleitet und von diesem per Broadcast versendet.

Zusätzlich gibt es die *networkPacketCommunication*-Modi, welche die Protokolle UDP (Modus *networkPacketCommunication_udp*) und TCP (*networkPacketCommunication_tcp*) sowie eine Kommunikation ohne Protokollstapel (*networkPacketCommunication_none*) unter-

stützen. Bei Verwendung von UDP kann die Nachricht entweder adressiert oder per Broadcast versendet werden. Bei Verwendung von TCP kann die Nachricht nicht per Broadcast versendet werden, da TCP verbindungsorientiert ist. Sie muss also in jedem Fall adressiert werden. Sobald ein Knoten a zum ersten Mal per TCP eine Nachricht an einen Knoten b sendet, baut a eine Verbindung zu b auf. Diese Verbindung bleibt dann für die restliche Dauer der Simulation offen. Bei der Kommunikation ohne Protokollstapel werden die Nachrichten vom entsprechenden NetDevice als Broadcast versendet (ähnlich zu *rawDataCommunication*).

Wird TCP oder UDP verwendet, so muss für jeden Endpunkt eine *NS3IPConfig* definiert werden. Diese enthält die IP-Adresse des Knotens, die Subnetzmaske und den Port, auf dem auf Verbindungsanfragen (TCP) bzw. ankommende Pakete (UDP) gelauscht wird. Während der Initialisierung wird ein entsprechender Protokollstapel auf den ns-3-Knoten installiert.

Neben diesen generischen Modi existiert ein mediumspezifischer Kommunikationsmodus (*mediumSpecificCommunication*), welcher momentan nur für das CC2420-Medium unterstützt wird und spezielle CC2420-Nachrichten verwendet (dies wird in Abschnitt 7.6 näher erläutert). In diesem Modus wird kein Protokollstapel verwendet. Die Nachrichten werden an ein *CC2420InterfaceNetDevice* weitergeleitet und von diesem per Broadcast versendet.

Mit Ausnahme von *mediumSpecificCommunication* können alle Kommunikationsmodi für jedes Medium verwendet werden, da die Nachrichtenformate generisch sind und die Kommunikation unabhängig vom Medium erfolgt.

Die Methode *init* des *NS3Interface* wird bei der Initialisierung der *NS3TimeTriggeredComponent* aufgerufen. Sie legt für das übergebene Kommunikationsmedium ein entsprechendes Medium in ns-3 an und setzt die zugehörigen Attribute (z. B. Datenrate, Verzögerung oder Mobilitätsmodell). Für jeden mit dem Medium verbundenen Endpunkt muss in ns-3 ein entsprechender Repräsentant angelegt werden. Dazu wird ein Knoten mit einem zu dem Medium passenden NetDevice und einer zu dem Kommunikationsmodus passenden virtuellen Applikation erzeugt (wird für den Kommunikationsmodus ein Protokollstapel benötigt, so wird dieser ebenfalls auf den Knoten installiert). Diese virtuelle Applikation wird mit dem eindeutigen Namen des *NS3CommunicationEndpoint* in das Dictionary von ns-3 eingetragen und dient als Schnittstelle zu FERAL.

Die Methode *executeTimestep* wird bei der Ausführung eines Zeitschritts der *NS3TimeTriggeredComponent* aufgerufen und führt ns-3 für die als Parameter übergebene Zeitspanne aus.

Die Methode *transferMessage* wird bei der Ausführung der *preFire*-Methode der *NS3TimeTriggeredComponent* aufgerufen und überträgt jeweils eine Nachricht von einem *NS3CommunicationEndpoint* in die Nachrichtenqueue der entsprechenden virtuellen Applikation in ns-3.

Die Methoden *isMessageWaiting* und *getMessage* werden bei der Ausführung von *postFire* der *NS3TimeTriggeredComponent* aufgerufen. *isMessageWaiting* prüft, ob für den entsprechenden *NS3CommunicationEndpoint* eine Nachricht in der zugehörigen virtuellen Applikation von ns-3 vorliegt, und *getMessage* liefert die erste solche Nachricht.

7.5.1.1. Nachrichtenübertragung

Im Folgenden wird die Nachrichtenübertragung, beschränkt auf den Java-Teil der Simulationskomponente ns-3, betrachtet. Um Nachrichten zwischen Simulationskomponenten auszutauschen, verwendet FERAL einen Port-Mechanismus. Ausgehende Nachrichten werden in *OutputPorts* einer Simulationskomponente abgelegt und über *Links* zu *InputPorts* der Ziel-

komponente weitergeleitet. *Receiver* steuern den Empfang der Nachrichten (sie können z. B. Nachrichten filtern oder verzögern). Abbildung 7.5 zeigt den Java-Teil eines konkreten Simulationssystems, in dem zwei native Simulationskomponenten über ein mittels ns-3 simuliertes CC2420-Medium kommunizieren.

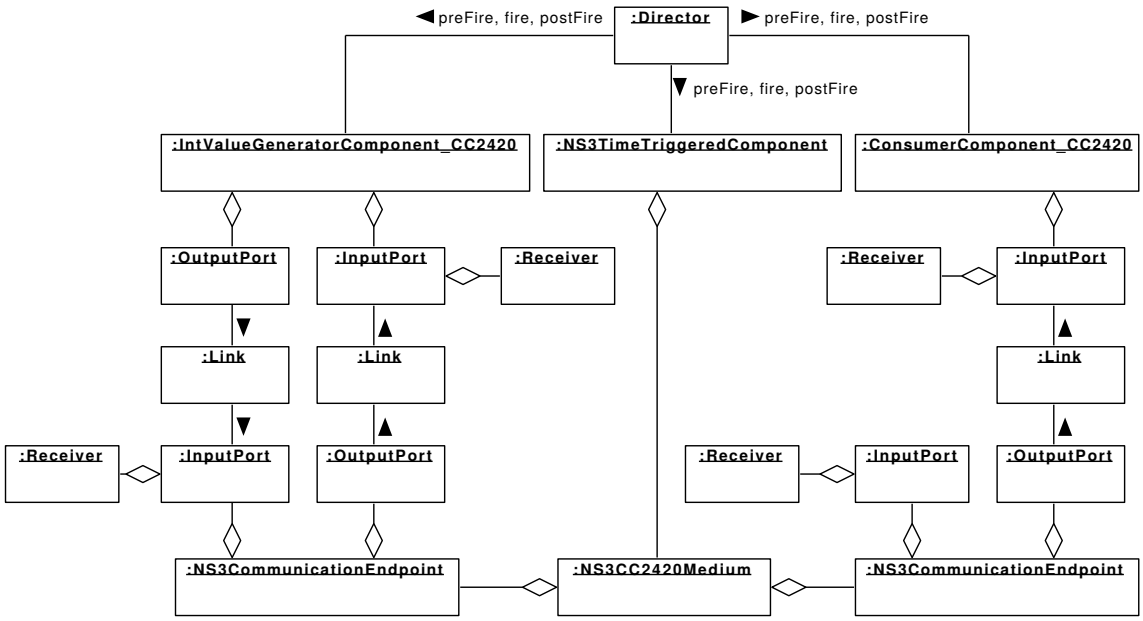

Abbildung 7.5.: Kommunikation zwischen Simulationskomponenten – Java-Teil.

Die *IntValueGeneratorComponent_CC2420* besitzt einen *InputPort* und einen *OutputPort*, da diese Komponente einerseits Nachrichten erzeugt und andererseits Statusnachrichten des CC2420-Simulationsmoduls verarbeitet (beispielsweise *CC2420Sending*, *CC2420SendFinished* oder *CC2420Cca*; s. Abschnitt 7.6). Da die *ConsumerComponent_CC2420* nur Nachrichten konsumiert, besitzt sie lediglich einen *InputPort*, jedoch keinen *OutputPort*. Deshalb wird der *InputPort* des zugehörigen *NS3CommunicationEndpoint* nicht verschaltet. Vor der Ausführung eines Simulationsschritts wird vom *Director* die Methode *preFire* der Simulationskomponenten ausgeführt. In der *preFire*-Methode der *NS3TimeTriggeredComponent* wird über alle enthaltenen Medien und darüber über alle Endpunkte iteriert. Es werden jeweils alle Nachrichten von dem *Receiver*, der dem *InputPort* eines solchen Endpunkts zugeordnet ist, geholt und per JNI an ns-3 übermittelt. Ein Simulationsschritt wird mittels der *fire*-Methode ausgeführt. *IntValueGeneratorComponent_CC2420* erzeugt während der Ausführung eines Simulationsschritts Nachrichten, welche im *OutputPort* gespeichert werden. Nach Ausführung eines Simulationsschritts wird die Methode *postFire* auf allen Simulationskomponenten aufgerufen. Bei *IntValueGeneratorComponent_CC2420* werden die Nachrichten vom jeweiligen *OutputPort* über den entsprechenden *Link* zum *InputPort* des zugehörigen Endpunkts weitergeleitet. In der *postFire*-Methode der *NS3TimeTriggeredComponent* werden anstehende Nachrichten von ns-3 geholt, in den *OutputPort* des zugehörigen Endpunkts geschrieben und über den *Link* zum entsprechenden *InputPort* von *IntValueGeneratorComponent_CC2420* oder *ConsumerComponent_CC2420* weitergeleitet.

7.5.1.2. Nachrichtenformate

Im Folgenden werden die im Java-Teil der Simulationskomponente verwendeten Nachrichtenformate beschrieben. Welche Nachrichten über ein simuliertes Medium versendet werden können, wird durch dessen Kommunikationsmodus festgelegt. Tabelle 7.1 gibt einen Überblick über die Kommunikationsmodi und die zugehörigen Nachrichtenformate zum Senden und Empfangen.

Kommunikationsmodus	**Nachrichtenformat TX / RX**
rawDataCommunication	*PTP_TX* / *PTP_RX*
mediumSpecificCommunication	*CC2420Config*, *CC2420Setup*, *CC2420Send*, *CC2420StatusReq* / *CC2420Sending*, *CC2420SendFinished*, *CC2420Recv*, *CC2420Cca*, *CC2420StatusResp* (nur für CC2420-Medium)
networkPacketCommunication_none	*NS3NetworkPacketSend* / *NS3NetworkPacketReceive* mit Protokoll *ns3_None*
networkPacketCommunication_udp	*NS3NetworkPacketSend* / *NS3NetworkPacketReceive* mit Protokoll *ns3_UDP*
networkPacketCommunication_tcp	*NS3NetworkPacketSend* / *NS3NetworkPacketReceive* mit Protokoll *ns3_TCP*

Tabelle 7.1.: Nachrichtenformate für die verschiedenen Kommunikationsmodi.

Die Nachrichtenschnittstelle für den Modus *rawDataCommunication* ist in Abbildung 7.6 dargestellt.

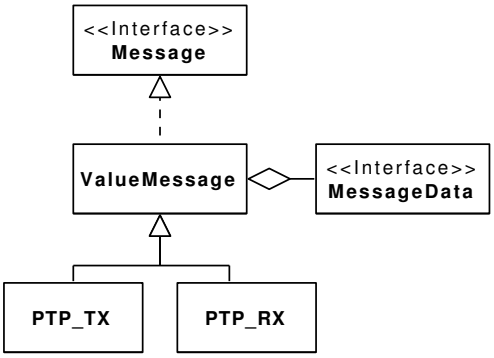

Abbildung 7.6.: Nachrichtenschnittstelle für *rawDataCommunication*.

In diesem Modus werden Nachrichten von Typ *PTP_TX* versendet und auf der Gegenseite als *PTP_RX* empfangen. Diese (ursprünglich für ein Punkt-zu-Punkt-Medium vorgesehenen) Klassen sind im FERAL-Framework bereits implementiert und dienen der Übertragung von Daten ohne weitere Parameter, weshalb sie hier wiederverwendet werden können. Beide Klassen sind von *ValueMessage* abgeleitet, welche das *Message*-Interface implementiert

und die Nutzdaten als *MessageData* kapselt. Diese Elemente werden ebenfalls vom FERAL-Framework zur Verfügung gestellt.

Die Nachrichtenformate des momentan nur für das CC2420-Medium unterstützten Modus *mediumSpecificCommunication* werden in Abschnitt 7.6 beschrieben.

Die Modi *networkPacketCommunication_udp* und *networkPacketCommunication_tcp* dienen der Realisierung einer Kommunikation über UDP bzw. TCP. Zusätzlich wurde der Modus *networkPacketCommunication_none* definiert, um eine Kommunikation ohne Protokollstapel zu ermöglichen, bei der aber beispielsweise die Adresse des Senders als Parameter mitgegeben werden kann (im Gegensatz zu *rawDataCommunication*). Für diese Kommunikationsmodi wurde ein einheitliches Nachrichtenformat definiert, welches aus den Klassen *NS3NetworkPacketSend* und *NS3NetworkPacketReceive* besteht. Es ist in Abbildung 7.7 dargestellt.

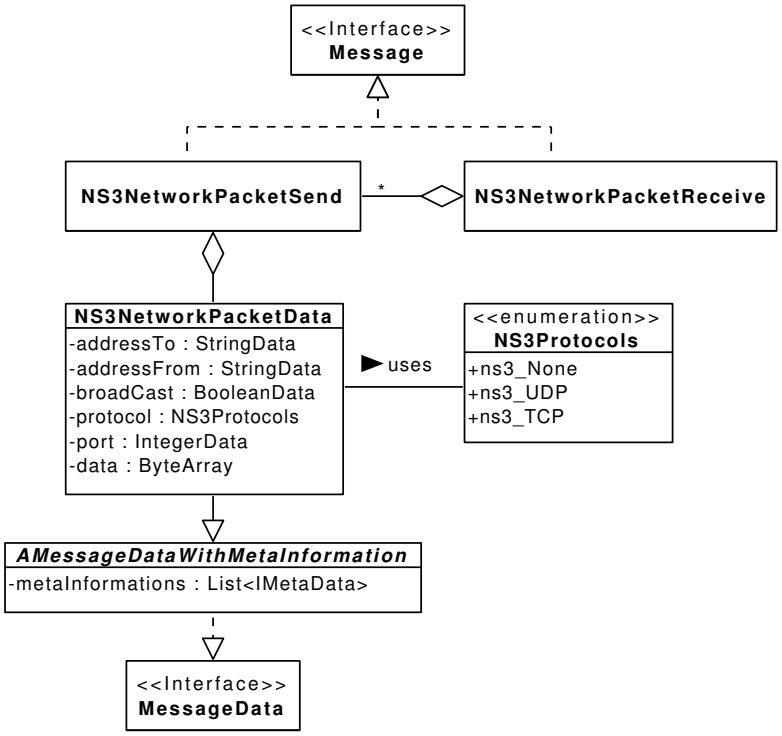

Abbildung 7.7.: Nachrichtenschnittstelle für *networkPacketCommunication*-Modi.

Die Verwendung eines einheitlichen Nachrichtenformats für alle drei Kommunikationsmodi hat den Vorteil, dass beispielsweise TCP gegen UDP ausgetauscht werden kann, ohne dass dies Änderungen an der Schnittstelle der Simulationskomponenten erfordert.

Um Parameter kapseln zu können, wird eine eigene Datenklasse *NS3NetworkPacketData* bereitgestellt, welche (indirekt) *MessageData* implementiert. Durch Vererbung von einer entsprechenden abstrakten Klasse bietet *NS3NetworkPacketData* zusätzlich die Möglichkeit, Metadaten zu verwalten. Dieser Aspekt wird hier jedoch nicht näher betrachtet. Die Klasse *NS3NetworkPacketReceive* enthält komplette *NS3NetworkPacketSend*-Nachrichten anstelle von *NS3NetworkPacketData*-Elementen, da vorgegebene Schnittstellen eingehalten werden müssen, welche hier nicht dargestellt sind.

7.5.2. Aufbau des C++-Teils der Simulationskomponente ns-3

Abbildung 7.8 zeigt den C++-Teil der Simulationskomponente ns-3, welcher durch eine JNI-Bibliothek mit dem Java-Teil verbunden ist.

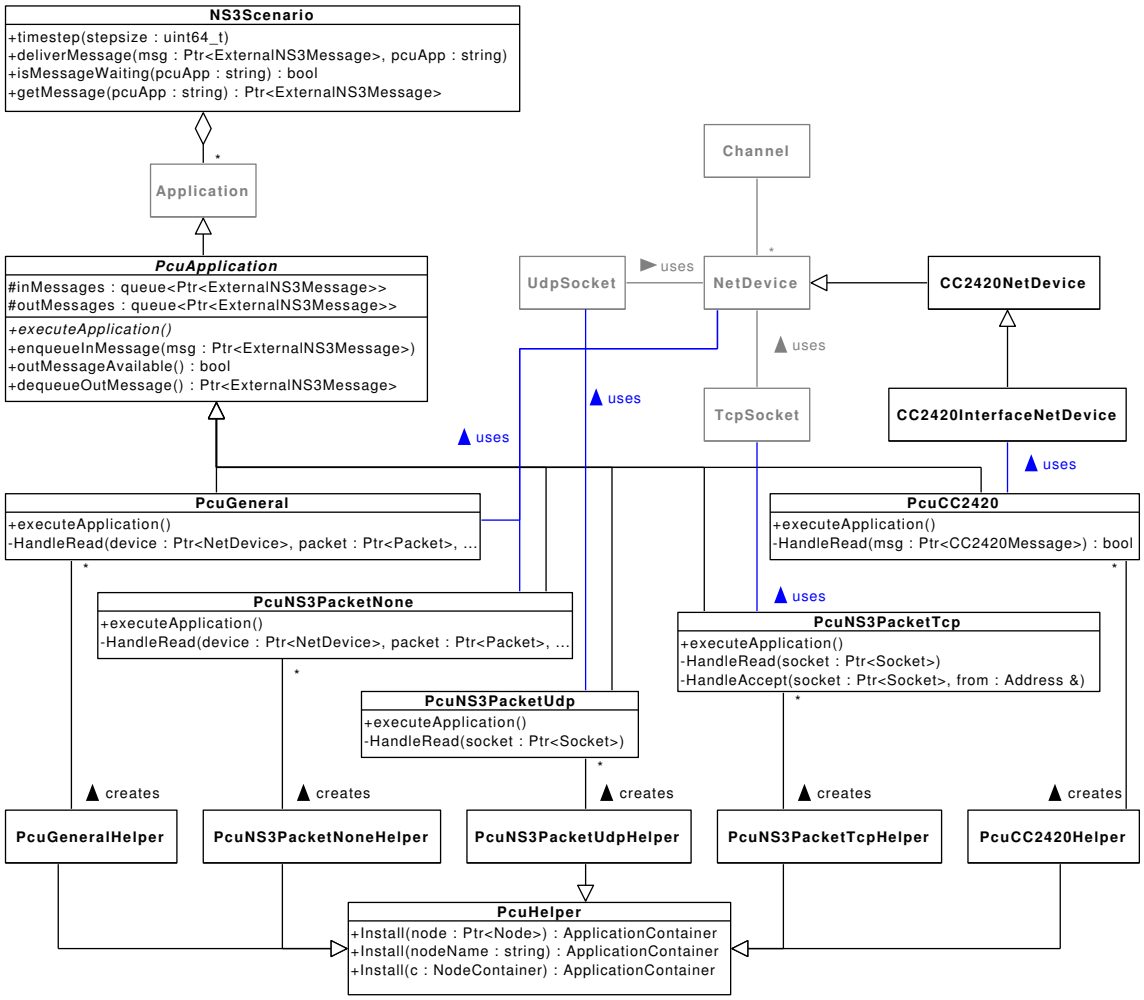

Abbildung 7.8.: Aufbau der Simulationskomponente ns-3 – C++-Teil.

Die (hier grau gezeichneten) Klassen *Channel*, *NetDevice* und *Application* entsprechen Abbildung 7.3 (Seite 131) und werden von ns-3 bereitgestellt. Ebenso werden *TcpSocket* und *UdpSocket* von ns-3 bereitgestellt.

Die Subklassen der abstrakten Klasse *PcuApplication* repräsentieren die virtuellen Applikationen, die als Gegenstück der *NS3CommunicationEndpoints* in Java fungieren (PCU steht für Protocol Control Unit; diese Bezeichnung wurde aus Gründen der Konsistenz zu anderen FERAL-Komponenten gewählt). Diese Klassen haben jeweils eine Queue für eingehende und eine für ausgehende Nachrichten und stellen für den jeweiligen ns-3-Knoten die Schnittstelle zu FERAL dar. Während *PcuGeneral*, *PcuNS3PacketNone*, *PcuNS3PacketUdp* und *PcuNS3PacketTcp* unabhängig vom verwendeten Medium sind, kann *PcuCC2420* nur mit dem CC2420-Medium verwendet werden. Die *uses*-Beziehungen (zur jeweils darunterliegenden Schicht) sind zur besseren Übersicht blau gezeichnet. Der Ablauf einer Nachrichtenübertragung ist in Abschnitt 7.5.2.1 gezeigt.

Die Klasse *NS3Scenario* repräsentiert das Simulationssystem auf ns-3-Seite. Sie enthält einen *ApplicationContainer* mit den einzelnen Applikationen der Simulation. Die Knoten, auf denen diese laufen, sind implizit gespeichert, da jede Applikation eindeutig einem Knoten zugeordnet ist. Die Methode *timestep* wird von der *executeTimestep*-Methode der JNI-Bibliothek aufgerufen und führt ns-3 für die als Parameter übergebene Zeitspanne aus. Die Methode *deliverMessage* wird von der *transferMessage*-Methode der JNI-Bibliothek aufgerufen, nachdem die als Parameter übergebene Nachricht in eine C++-Repräsentation überführt wurde. Sie überträgt diese an die entsprechende *PcuApplication*, deren eindeutiger Name ebenfalls als Parameter übergeben und deren Methode *enqueueInMessage* aufgerufen wird. Analog werden die Methoden *isMessageWaiting* und *getMessage* genutzt. Die in C++ verwendeten Nachrichtenformate und ihre Beziehungen zu den Java-Nachrichtenformaten sind in Abschnitt 7.5.2.2 beschrieben.

Die Methode *executeApplication* der Klasse *PcuApplication* ist abstrakt und wird jeweils von den Subklassen bereitgestellt. Die *executeApplication*-Methode von *PcuGeneral* erzeugt für jede Nachricht in der Eingangsqueue – die vom Typ *RawDataMessage* sein muss – ein ns-3-Paket und versendet dieses über das NetDevice des Knotens. Die entsprechende Methode von *PcuNS3PacketNone* realisiert dasselbe Verhalten für Nachrichten vom Typ *PacketCommunicationNoneMessage*. Die *executeApplication*-Methode von *PcuNS3PacketUdp* erzeugt ns-3-Pakete für Nachrichten von Typ *PacketCommunicationUdpMessage* und leitet diese an ein Udp-Socket weiter. Bei *PcuNS3PacketTcp* wird in *executeApplication* zunächst geprüft, ob für die entsprechende Ziel-IP bereits eine Verbindung existiert. Ist dies der Fall, so wird das (aus einer *PacketCommunicationTcpMessage* erzeugte) Paket an das entsprechende Socket weitergeleitet. Anderenfalls wird zunächst eine Verbindung aufgebaut und anschließend das Paket gesendet. Die *executeApplication*-Methode von *PcuCC2420* reicht die Nachrichten in der Eingangsqueue – welche in diesem Fall vom Typ *CC2420Message* sein müssen – unverändert an das *CC2420InterfaceNetDevice* weiter. Das *CC2420InterfaceNetDevice* verarbeitet diese Nachrichten und verpackt ausgehende Nachrichten entsprechend. Die *PcuCC2420*-Applikation wird genau dann auf den Knoten installiert, wenn für das entsprechende CC2420-Medium der Kommunikationsmodus *mediumSpecificCommunication* verwendet wird. In diesem Fall kann man mit dem CC2420-Modul nicht nur durch das Versenden und Empfangen von Paketen kommunizieren, sondern beispielsweise auch Statusnachrichten austauschen. Auch dazu dient die Nachrichtenschnittstelle *CC2420Message*. Diese wird zusammen mit der Integration des CC2420-Moduls in ns-3 und FERAL in Abschnitt 7.6 beschrieben.

Der Empfang von Nachrichten wird über einen Callback realisiert, der beim Erstellen der jeweiligen Applikation gesetzt wird. Als Callback dient die Methode *HandleRead*. Bei *PcuNS3PacketTcp* wird beim Erstellen der Applikation lediglich ein Socket erzeugt, welches auf Verbindungsanfragen lauscht. Eingehende Verbindungsanfragen werden ebenfalls durch einen Callback behandelt, wozu die Methode *HandleAccept* verwendet wird. Für jede Verbindung wird ein eigenes Socket erzeugt, für das wiederum die Methode *HandleRead* als Callback zum Empfang von Nachrichten verwendet wird.

Die Klasse *PcuHelper* bzw. die entsprechenden Subklassen dienen lediglich dazu, Applikationen des entsprechenden Typs zu erzeugen und auf den Knoten des Simulationssystems zu installieren.

7.5.2.1. Nachrichtenübertragung

Im Folgenden wird die Nachrichtenübertragung, welche bereits in Abschnitt 7.5.1.1 für den Java-Teil dargestellt wurde, um den C++-Teil erweitert. Abbildung 7.9 greift das Simulati-

onssystem aus Abbildung 7.5 (Seite 135) auf und ergänzt dieses entsprechend (der Java-Teil ist hier verkürzt dargestellt).

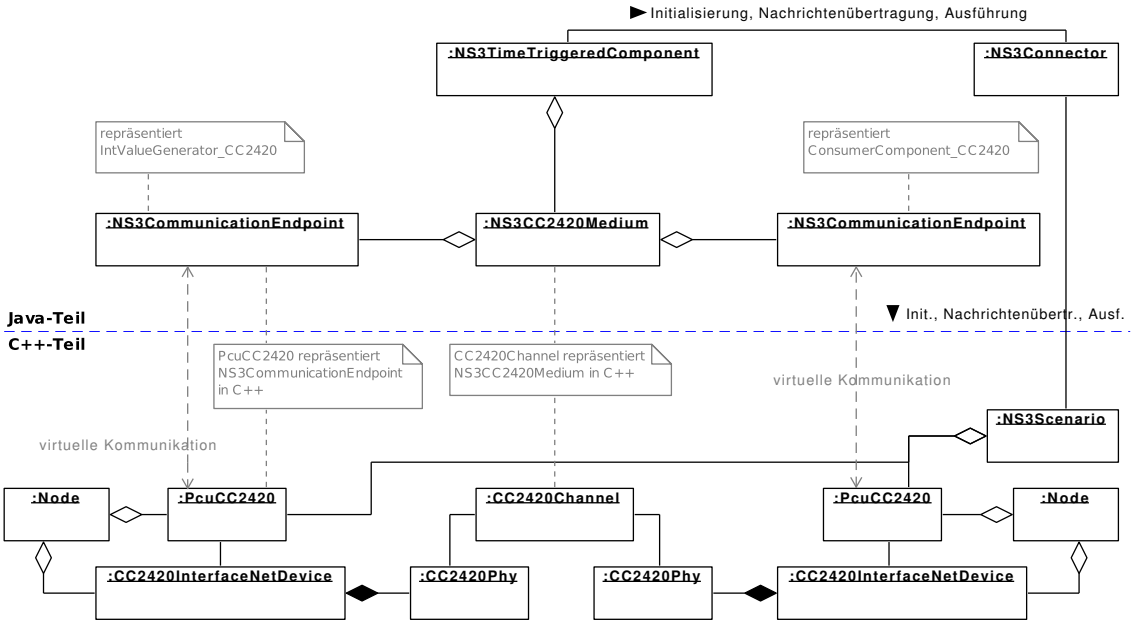

Abbildung 7.9.: Kommunikation zwischen Simulationskomponenten.

NS3Scenario repräsentiert das Simulationssystem auf ns-3-Seite und dient als Container, um die virtuellen Applikationen – hier vom Typ *PcuCC2420* – zu speichern. Nachrichten werden virtuell von einem *NS3CommunicationEndpoint* an die zugehörige *PcuCC2402*-Applikation übertragen und dann über einen mittels ns-3 simulierten CC2420-Kanal versendet. Die Zuordnung zwischen *NS3CommunicationEndpoint* und *PcuCC2420* erfolgt über die Verwendung eindeutiger Namen. Da die *NS3CommunicationEndpoints* nicht ausgeführt werden, sondern nur die *NS3TimeTriggeredComponent* (Methode *fire*), erfolgt die reale Kommunikation über diese. Sie überträgt Nachrichten an und von ns-3 und übernimmt die Initialisierung und Ausführung des Simulators. Die für die Kommunikation zwischen Java und C++ verwendete JNI-Bibliothek wird durch *NS3Connector* realisiert.

Die Nachrichten des Endpunkts, der *IntValueGeneratorComponent_CC2420* repräsentiert, werden in der *preFire*-Methode von *NS3TimeTriggeredComponent* an *NS3Scenario* übertragen, von wo aus sie an die entsprechende *PcuCC2420*-Instanz zugestellt werden. Diese versendet die Nachrichten dann bei Ausführung eines Zeitschritts (*fire*-Methode) über das zugeordnete *CC2420InterfaceNetDevice*. In der *postFire*-Methode der *NS3TimeTriggeredComponent* werden auf demselben Weg die Nachrichten von den beiden *PcuCC2420*-Instanzen geholt und an die zugehörigen Endpunkte weitergeleitet.

7.5.2.2. Nachrichtenformate

Für die in Abschnitt 7.5.1.2 beschriebenen Java-Nachrichtenformate gibt es Entsprechungen im C++-Teil der Simulationskomponente, um die Nachrichten in ns-3 verarbeiten zu können. Tabelle 7.2 gibt einen Überblick über die Zuordnung zwischen den Nachrichtenformaten.

Während es in Java für das Versenden und Empfangen jeweils unterschiedliche Nachrichtenformate gibt, wurde in C++ jeweils nur eines verwendet, da diese Unterscheidung in ns-3

Java-Nachrichtenformat (TX / RX)	**C++-Nachrichtenformat**
PTP_TX / *PTP_RX*	*RawDataMessage*
CC2420Config, *CC2420Setup*, *CC2420Send*, *CC2420StatusReq* / *CC2420Sending*, *CC2420SendFinished*, *CC2420Recv*, *CC2420Cca*, *CC2420StatusResp*	identisch zu den Java-Formaten
NS3NetworkPacketSend / *NS3NetworkPacketReceive* mit Protokoll *ns3_None*	*PacketCommunicationNoneMessage*
NS3NetworkPacketSend / *NS3NetworkPacketReceive* mit Protokoll *ns3_UDP*	*PacketCommunicationUdpMessage*
NS3NetworkPacketSend / *NS3NetworkPacketReceive* mit Protokoll *ns3_TCP*	*PacketCommunicationTcpMessage*

Tabelle 7.2.: Zuordnung zwischen Java- und C++-Nachrichtenformaten.

nicht erforderlich ist. Auf Java-Seite wurde sie getroffen, um konsistenter zum übrigen FERAL-Framework zu sein. Die CC2420-Datentypen sind in Java und C++ gleich benannt und werden in Abschnitt 7.6.3 betrachtet.

Die Nachrichtenschnittstellen für *rawDataCommunication* und die *networkPacketCommunication*-Modi sind in Abbildung 7.10 dargestellt. Basisklasse für alle Nachrichten, die von oder an ns-3 gesendet werden, ist die Klasse *ExternalNS3Message*, welche von der ns-3-Klasse *Object* abgeleitet ist, um das Smart-Pointer-System von ns-3 nutzen zu können. Von *ExternalNS3Message* abgeleitet ist die Klasse *RawDataMessage*, die einen Puffer für Nutzdaten sowie dessen Größe enthält und als Repräsentation für Nachrichten vom Typ *PTP_TX* bzw. *PTP_RX* verwendet wird.

Nachrichten vom Typ *NS3NetworkPacketSend* bzw. *NS3NetworkPacketReceive* werden – je nach Protokolltyp – auf *PacketCommunicationNoneMessage*, *PacketCommunicationUdpMessage* oder *PacketCommunicationTcpMessage* abgebildet. Im Gegensatz zum Java-Teil wurden hier je nach Protokoll unterschiedliche Nachrichtenformate verwendet, um protokollspezifische Eigenschaften besser abbilden zu können (beispielsweise wird das Attribut *broadcast* bei TCP nicht benötigt; im *networkPacketCommunication_none*-Modus wird keines der Attribute benötigt). Die Verwendung von drei separaten Formaten trägt deshalb zu einer besseren Übersicht bei. Im Java-Teil wurde auf diese Aufteilung verzichtet, da dort der Fokus darauf liegt, Systeme mit möglichst geringem Anpassungsbedarf mit unterschiedlichen Protokollen testen zu können.

Die in den *networkPacketCommunication*-Modi zusätzlich verwendeten Metadaten sind zwar Bestandteil der entsprechenden *PacketCommunicationMessages*, werden aber beim Versenden des ns-3-Pakets über das simulierte Medium in einen *Tag* verpackt. Dieses Konzept wird von ns-3 bereitgestellt, um Daten transportieren zu können, die lediglich für die Verwaltung der Simulation benötigt werden, die Größe der simulierten Nachricht jedoch nicht beeinflussen sollen.

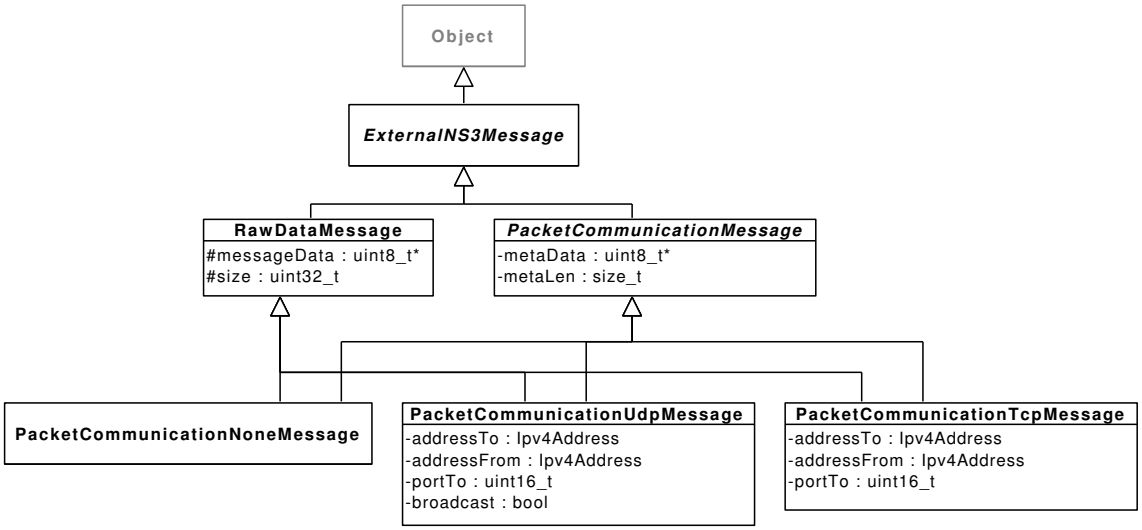

Abbildung 7.10.: Nachrichtenschnittstellen für *rawDataCommunication* und *networkPacketCommunication*-Modi.

7.6. Entwicklung eines CC2420-Simulationsmoduls für ns-3 und FERAL

Der CC2420-Transceiver von TexasInstruments [Tex13] operiert im unlizenzierten 2.4-GHz-Band, hat eine Datenrate von bis zu 250 kBit/s, 16 Kanäle und ist konform zum IEEE 802.15.4-Standard [Ins11]. Ferner ist er kostengünstig, konfigurierbar und energieeffizient, weshalb er sich beispielsweise zum Aufbau von Sensornetzen eignet. In diesen sind niedrige Datenraten oft ausreichend; den Knoten steht jedoch meist nur wenig Energie zur Verfügung. Um solche Sensornetze simulieren zu können, wurde ein Simulationsmodul für den CC2420-Transceiver entwickelt, welches sowohl für ns-3 als auch für FERAL nutzbar ist. Dieses wurde in [IG13] vorgestellt. Der Vorteil eines solchen Moduls besteht darin, dass man einerseits komplexe Simulationen mit FERAL durchführen kann, bei denen die einzelnen Komponenten durch verschiedene integrierte Simulatoren realisiert werden. Andererseits kann man auch auf eigenständige ns-3-Simulationen zurückgreifen, wenn die Simulationsmöglichkeiten von ns-3 ausreichend sind, und vermeidet so den Simulationsoverhead von FERAL.

7.6.1. Funktionalität des CC2420-Moduls

Das CC2420-Simulationsmodul adaptiert den im Datenblatt des Transceivers [Tex13] beschriebenen Zustandsautomaten. Dabei wird von einigen Aspekten abstrahiert, beispielsweise Ein- und Ausschalten des Transceivers, Reset, Acknowledgement-Mechanismus und Überlauf- bzw. Unterlauf-Erkennung. Abbildung 7.11 zeigt den für die Simulation verwendeten Teil des Zustandsautomaten. Der vollständige Zustandsautomat aus dem Datenblatt ist in Anhang F abgebildet.

Der Transceiver startet im Zustand RX_CALIBRATE. Diese Phase hat eine Dauer von 12 Symbolperioden; anschließend wird der Zustand RX_SFD_SEARCH erreicht. Da jedes Symbol 4 Bit encodiert und die Datenrate des Transceivers 250 kBit/s beträgt, hat jede Symbolperiode eine Dauer von 16 μs. Der Transceiver verwendet einen Synchronisationsheader, um sich auf die Symbole eines empfangenen Rahmens synchronisieren zu können. Dieser Synchro-

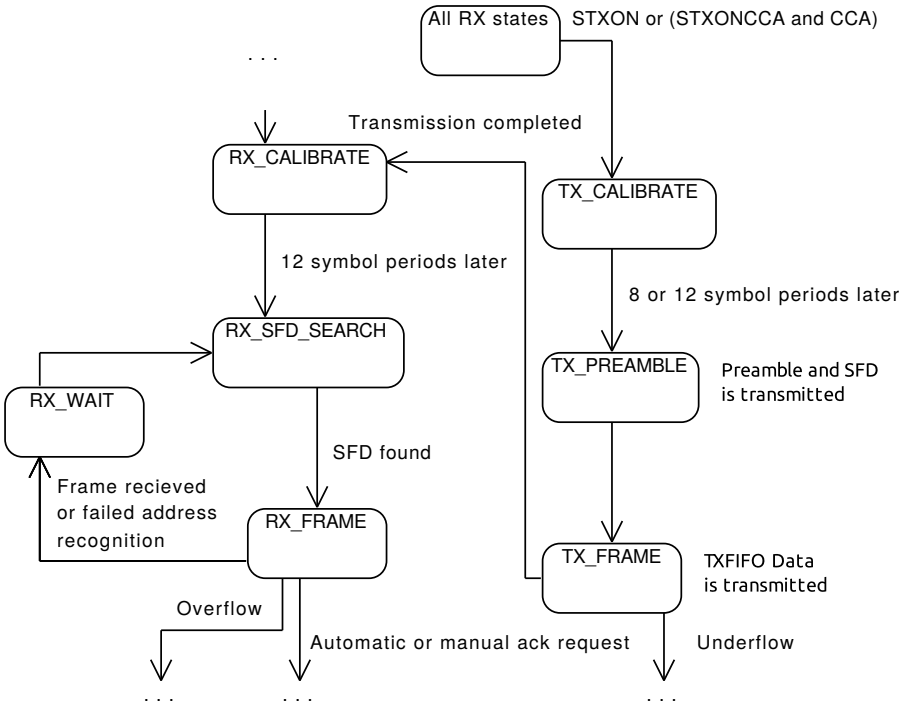

Abbildung 7.11.: Simulierter Teil des CC2420-Zustandsautomaten [Tex13].

nisationsheader besteht aus einer Präambel und einem sogenannten Start of Frame Delimiter (SFD) (das Rahmenformat ist in Abbildung 7.14 auf Seite 146 dargestellt). Die standardmäßig verwendete Präambel besteht aus 4 Bytes mit dem Wert 0x00, und der Standard-SFD ist 0xA7. Beides ist konform zum IEEE 802.15.4-Standard. Das sogenannte *Sync Word* besteht aus dem letzten Byte der Präambel (um konform mit IEEE 802.15.4 zu sein, muss dieses Null sein) sowie dem SFD-Byte. Die Länge der Präambel (bzw. die Anzahl führender Null-Bytes) und das Sync Word können über spezielle Register konfiguriert werden, jedoch ist der Transceiver nicht mehr konform zu IEEE 802.15.4, wenn die Standardwerte geändert werden. Wird ein Rahmen empfangen, so synchronisiert sich der Transceiver auf die Null-Symbole der Präambel und sucht die SFD-Sequenz. Die Länge der Präambel ist in diesem Fall nicht relevant; sie wird nur beim Senden berücksichtigt. Wird ein SFD empfangen, so wechselt der Transceiver in den Zustand RX_FRAME. Sobald der Rahmen vollständig empfangen wurde, wird der Zustand RX_WAIT angenommen. Wenn der Transceiver wieder bereit ist, einen neuen Rahmen zu empfangen, geht er zurück in den Zustand RX_SFD_SEARCH.

Sendeanforderungen können in allen RX-Zuständen verarbeitet werden. Es kann entweder mit Clear Channel Assessment (CCA) übertragen werden (Signal STXONCCA) oder ohne (Signal STXON). IEEE 802.15.4 definiert drei verschiedene CCA-Modi, welche alle vom CC2420-Transceiver unterstützt werden. In Modus 1 wird der Kanal als frei angesehen, wenn die Empfangssignalstärke (Energie auf dem Medium) unterhalb eines konfigurierbaren Grenzwerts liegt. In Modus 2 ist der Kanal frei, wenn der Transceiver gerade keinen gültigen IEEE 802.15.4-Rahmen empfängt. Modus 3 ist eine Kombination der Modi 1 und 2, d. h. der Kanal ist frei, wenn die Empfangssignalstärke unterhalb des Grenzwerts liegt *und* der Transceiver keinen gültigen IEEE 802.15.4-Rahmen empfängt. Zusätzlich wird eine Hysterese definiert, welche zu häufige CCA-Wechsel in den Modi 1 und 3 verhindern soll. Da sich der CCA-Wert (in Modus 1) nur dann von belegt zu frei ändert, wenn die Energie unter den Grenzwert

minus der Hysterese fällt, und nur dann von frei zu belegt, wenn die Energie den Grenzwert überschreitet, führen geringfügige Fluktuationen nicht zu CCA-Wechseln.

Eine Sendeanforderung verursacht einen Zustandswechsel zum Zustand TX_CALIBRATE. Diese Phase hat eine Dauer von 8 oder 12 Symbolperioden; welcher Wert verwendet wird, wird durch den Parameter *TX_TURNAROUND* konfiguriert. Der Grund für das Vorhandensein von zwei unterschiedlichen Werten besteht darin, dass eine Dauer von 12 Symbolperioden konform zu IEEE 802.15.4 ist, der Transceiver jedoch schneller kalibrieren kann. Nach Ablauf der Kalibrierungsphase werden Präambel und SFD übertragen (Zustand TX_PREAMBLE), und schließlich der Rest des Rahmens, welcher aus einem speziellen FIFO-Speicher ausgelesen wird (Zustand TX_FRAME). Nach Abschluss des Sendevorgangs geht der Transceiver wieder in den Zustand RX_CALIBRATE über.

Das Simulationsmodul für den CC2420-Transceiver enthält ebenfalls einen Zustandsautomaten, der den Zustand des Transceivers kombiniert mit dem des Mediums implementiert. Dieser ist in Anhang F zu finden.

7.6.2. Integration des CC2420-Moduls in ns-3

Eine erste Version des CC2420-Moduls für ns-3 wurde in [Gro12] entwickelt. Diese basiert auf einem existierenden CC2420-Modul für ns-2, welches im Rahmen der Vorläufer von FERAL entwickelt wurde (vgl. [Kuh09]). Der Teil eines Simulationssystems, welcher durch das CC2420-Simulationsmodul abgedeckt wird, ist in Abbildung 7.12 blau markiert.

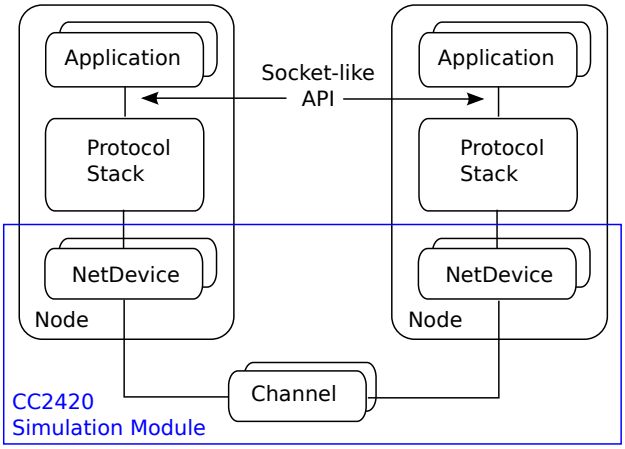

Abbildung 7.12.: Vom CC2420-Modul abgedeckter Teil einer ns-3-Simulation.

Da das Modul als Bestandteil von ns-3 entwickelt wurde, kann vom Vorhandensein existierender ns-3-Komponenten profitiert werden. Für den CC2420-Kanal werden existierende Verlust- und Verzögerungsmodelle verwendet, und auf Knoten, welche ein CC2420-NetDevice verwenden, können vordefinierte Mobilitätsmodelle sowie existierende Applikationen und Protokollstapel (z. B. UDP / IP) installiert werden.

Die Struktur des CC2420-Moduls ist in Abbildung 7.13 dargestellt. Die Klassen *NetDevice*, *Channel*, *Packet*, *Tag*, *Header* und *Trailer* sowie die entsprechenden Relationen zwischen diesen werden von ns-3 bereitgestellt.

Für jeden Knoten, welcher über CC2420 kommunizieren soll, erzeugt *CC2420Helper* ein *CC2420NetDevice* oder ein *CC2420InterfaceNetDevice*. Zusammen mit dem NetDevice wird eine physikalische Schicht (*CC2420Phy*) kreiert, welche entweder mit einem existierenden oder

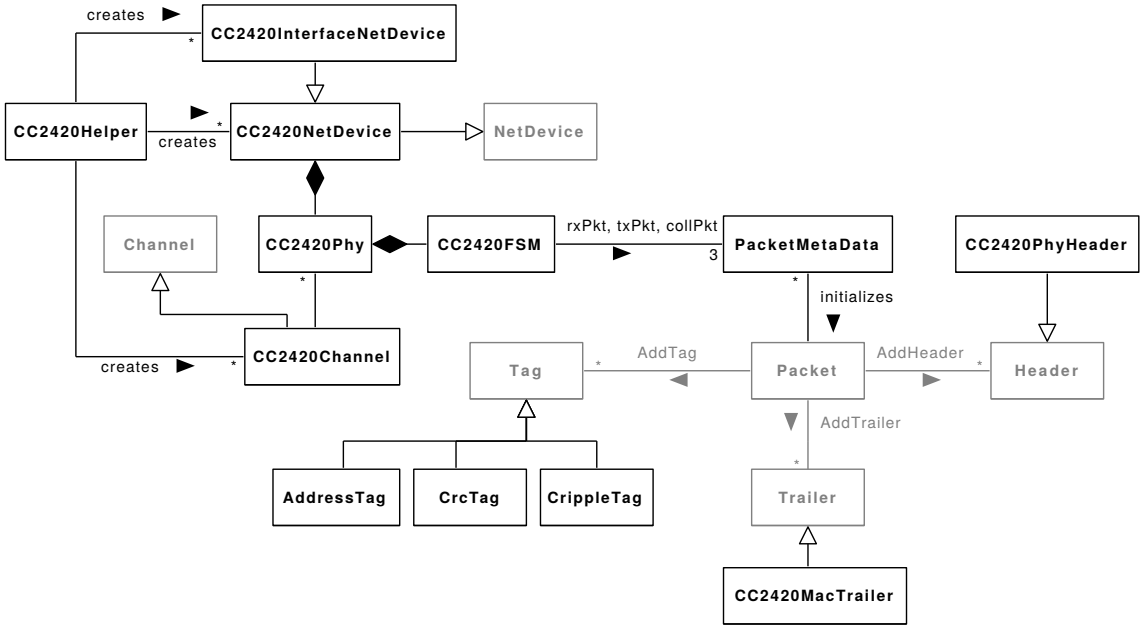

Abbildung 7.13.: Struktur des CC2420-Simulationsmoduls.

mit einem neu erzeugten *CC2420Channel* verbunden wird.

CC2420NetDevice stellt ein Interface bereit, über welches das Simulationsmodul mit höheren Protokollschichten interagiert. *CC2420InterfaceNetDevice* definiert ein erweitertes Interface, über das es nicht nur möglich ist, Rahmen zu senden und zu empfangen, sondern beispielsweise auch den Transceiver zu konfigurieren (s. Abschnitt 7.6.3). Zum Weiterleiten empfangener Rahmen an höhere Protokollschichten nutzt *CC2420NetDevice* einen *ReceiveCallback*, dessen Interface in der *NetDevice*-Klasse definiert ist. *CC2420InterfaceNetDevice* stellt zusätzlich einen *MessageCallback* bereit. Während der *ReceiveCallback* nur den Empfang regulärer Rahmen erlaubt (als ns-3-Pakete repräsentiert), dient der *MessageCallback* zum Empfang spezifischer CC2420-Nachrichten gemäß des erweiterten Interface.

CC2420Phy verwaltet die konfigurierbaren Parameter des Transceivers (z. B. Präambellänge, Sync Word, Sendesignalstärke und Kanalnummer) und fügt einen entsprechenden Header und Trailer (s. u.) zu den ns-3-Paketen hinzu, welche die Rahmen des Transceivers repräsentieren. Zusätzlich kontrolliert die physikalische Schicht den Zustandsautomaten des Simulationsmoduls (realisiert durch *CC2420FSM*). Dieser setzt im Wesentlichen die in Abschnitt 7.6.1 beschriebene Funktionalität um und kombiniert dabei den Zustand des Simulationsmoduls mit dem des simulierten Mediums. Eine detailliertere Beschreibung dieses Automaten ist in Anhang F zu finden. Dies ist der einzige Teil, der von dem für die Vorgänger von FERAL existierenden Simulationsmodul nahezu unverändert übernommen werden konnte.

PacketMetaData ist eine Hilfsklasse, die zur Speicherung des gerade empfangenen, gerade kollidierenden bzw. gerade gesendeten Rahmens (jeweils repräsentiert durch ein ns-3-Paket) dient, zusammen mit dessen Empfangs- bzw. Sendesignalstärke sowie dem Anfangszeitpunkt.

CC2420Channel verwaltet die 16 Kanäle des 2.4-GHz-Bands als Subkanäle und berechnet, wann ein gesendeter Rahmen bei welchem Knoten ankommt und mit welcher Signalstärke. Dazu werden sowohl die Mobilitätsmodelle von Sender und Empfänger als auch die Sendesignalstärke berücksichtigt. Die eigentlichen Berechnungen werden vom Verzögerungs- bzw. Verlustmodell vorgenommen.

Tags dienen dazu, Simulationsinformationen zu einem ns-3-Paket hinzuzufügen, ohne dessen simulierte Länge zu vergrößern. Quell- und Zieladresse eines CC2420-Rahmens werden in einem *AddressTag* gespeichert. Diese Adressen werden nicht direkt im zugehörigen Paket codiert, da das CC2420-NetDevice primär für Broadcastübertragungen entworfen wurde. Die eigentliche Adressierung wird von einer höheren Protokollschicht vorgenommen und ist deshalb bereits in der MAC Protocol Data Unit (MPDU) enthalten. Da die Adressinformation jedoch zur Kompatibilität mit der *NetDevice*-Schnittstelle von ns-3 benötigt wird, wird sie als Tag bereitgestellt.

CrippleTag dient zur Markierung von Paketen, welche nicht korrekt empfangen werden können, entweder aufgrund eines Kanalwechsels oder weil sie ein anderes Sync Word enthalten und deshalb vom Transceiver nicht erkannt werden.

Das Rahmenformat des CC2420-Transceivers ist in Abbildung 7.14 dargestellt.

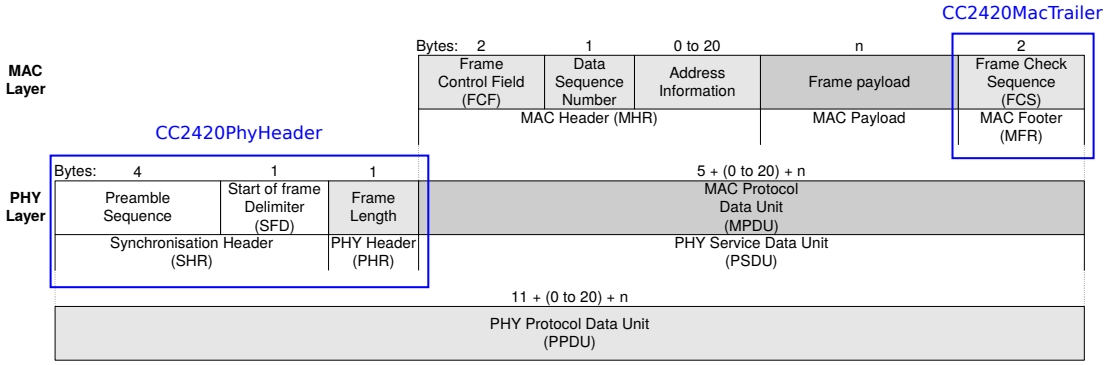

Abbildung 7.14.: Rahmenformat des CC2420-Transceivers [Tex13].

Präambel, SFD und Frame Length werden von der Klasse *CC2420PhyHeader* gekapselt und zum ns-3-Paket hinzugefügt. Frame Length beschreibt die Länge der MPDU in Bytes; da das höchste Bit reserviert ist [Tex13], ist der maximale Wert 127. Somit kann die MPDU höchstens 127 Bytes enthalten. Der MAC Header (Frame Control Field, Data Sequence Number und Address Information) wird nicht simuliert. Da in der Simulation stets 2 Bytes für die Frame Check Sequence (FCS) reserviert sind, ist die maximale Payloadgröße bzw. MTU (Maximum Transmission Unit) 125 Bytes. Die FCS ist eine CRC-Prüfsumme, welche durch die Klasse *CC2420MacTrailer* umgesetzt wird. Für das Simulationsmodul kann konfiguriert werden, ob diese Prüfsumme zu den gesendeten Paketen hinzugefügt werden soll oder nicht. Wird sie hinzugefügt, so wird jedes Paket mit einem *CrcTag* markiert, da der Empfänger anderenfalls nicht feststellen könnte, ob die Prüfsumme enthalten ist oder nicht.

Der Ablauf einer erfolgreichen Nachrichtenübertragung mit dem *CC2420NetDevice* ist in Abbildung 7.15 dargestellt. Nachrichten werden von einer höheren Protokollschicht in Form von ns-3-Paketen an das *CC2420NetDevice* gesendet, welches diese an die physikalische Schicht weiterleitet. Diese registriert die Sendeanforderung im Zustandsautomaten, welcher – nach Ablauf der Kalibrierungszeit – das Weiterleiten des Pakets von der physikalischen Schicht an den Kanal auslöst. Der Kanal speichert das Paket im entsprechenden Subkanal und schedult ein Empfangsereignis für alle Empfänger, welche mit diesem Subkanal verbunden sind. Sobald der zugehörige Timer abläuft (dieser modelliert die Übertragungsverzögerung), löst der Scheduler den Empfang des Pakets in der physikalischen Schicht aus. Diese sendet eine Empfangsanforderung an den Zustandsautomaten, welcher – nach Ablauf der entsprechenden Dauer zum

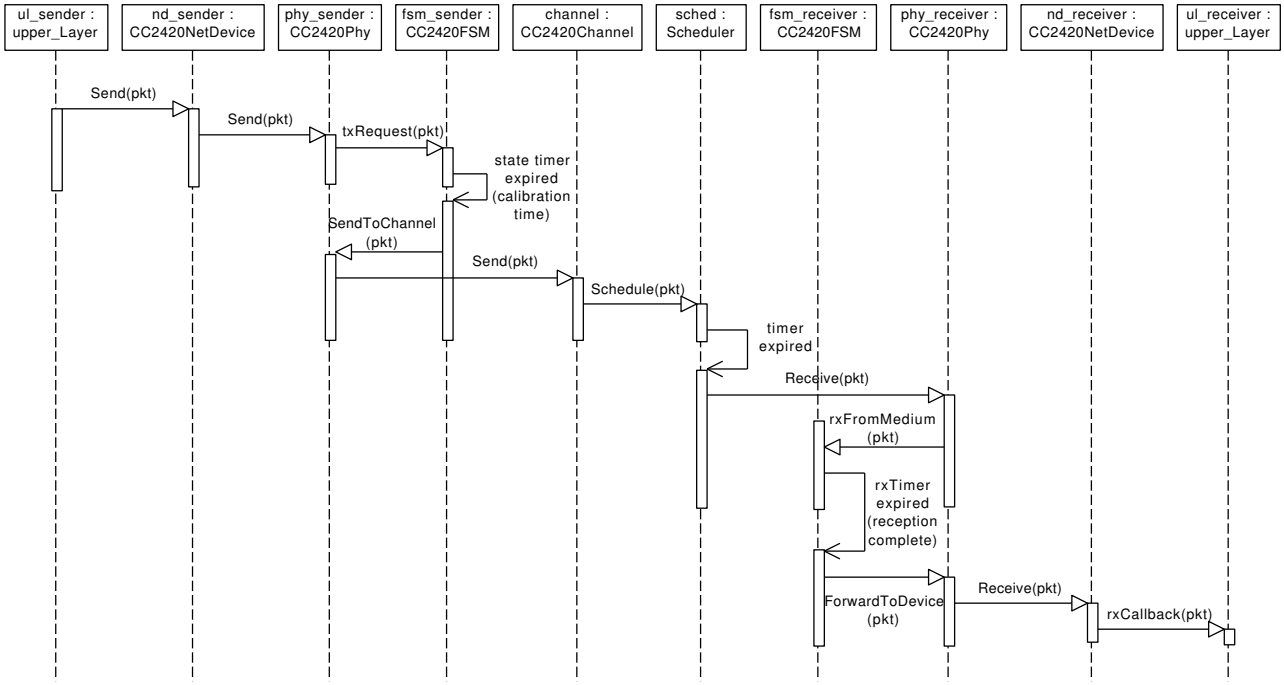

Abbildung 7.15.: Ablauf einer erfolgreichen Nachrichtenübertragung.

Empfang – die Weiterleitung des Pakets von der physikalischen Schicht an das NetDevice auslöst. Schließlich wird das Paket mit dem *ReceiveCallback* an eine höhere Protokollschicht übergeben.

7.6.3. Erweiterungen

Ein NetDevice bietet standardmäßig nur die Möglichkeit, ns-3-Pakete zu senden und zu empfangen. Für das CC2420-Modul sollen jedoch weitere Signale für Konfigurations- und Informationszwecke bereitgestellt werden. Beispielsweise soll ein empfangenes Paket seine Empfangssignalstärke beinhalten, CCA-Wechsel sowie das Ende einer Übertragung sollen signalisiert werden, und es soll möglich sein, den Kanal zu wechseln oder die Sendesignalstärke zu ändern. Weiterhin soll es möglich sein, Informationen über die aktuelle Konfiguration des Transceivers zu erhalten. Aus diesen Gründen wurde das *CC2420InterfaceNetDevice* definiert, welches vom Standard-*CC2420NetDevice* erbt und die in Abbildung 7.16 gezeigte erweiterte Nachrichtenschnittstelle verwendet.

Die bereits in Abschnitt 7.5.2.2 vorgestellte Klasse *RawDataMessage* kapselt die Nutzdaten. *CC2420Message* stellt eine einheitliche Schnittstelle für alle Nachrichten dar, welche an das *CC2420InterfaceNetDevice* gesendet oder von diesem empfangen werden. Die Nachrichten *CC2420Send*, *CC2420Setup*, *CC2420Config* und *CC2420StatusReq* werden an das Modul gesendet, während *CC2420Recv*, *CC2420Cca*, *CC2420Sending*, *CC2420SendFinished* und *CC2420StatusResp* vom Simulationsmodul an höhere Protokollschichten gesendet werden.

Das *CC2420NetDevice* sendet und empfängt reguläre ns-3-Pakete. Deshalb kann es mit existierenden, von ns-3 bereitgestellten Applikationen und Protokollstapeln verwendet werden, ohne dass Modifikationen erforderlich sind. Das *CC2420InterfaceNetDevice* erfordert jedoch Anpassungen in mindestens einer höheren Schicht (Applikation und / oder Protokollstapel),

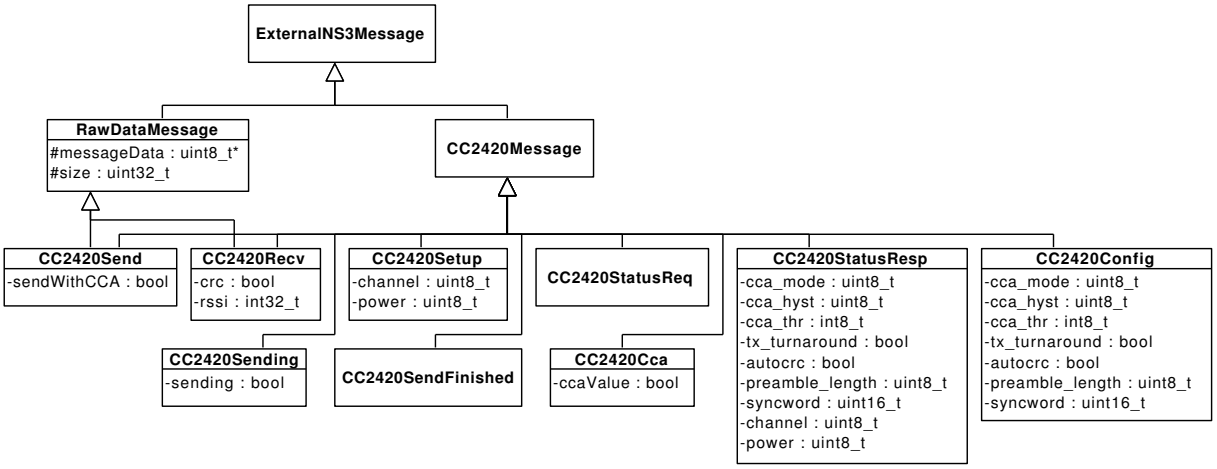

Abbildung 7.16.: Erweiterte Nachrichtenschnittstelle für das CC2420-Simulationsmodul.

da *CC2420Messages* erzeugt und verarbeitet werden müssen.

CC2420Send dient zum Versenden eines Pakets. Es kann explizit konfiguriert werden, ob CCA verwendet werden soll oder nicht, d. h. ob das Medium vor dem Senden überprüft werden soll (die Sendemethode des *CC2420NetDevice* sendet implizit mit CCA). *CC2420Recv* enthält die empfangenen Daten und stellt zusätzlich Informationen zur Korrektheit der CRC-Prüfsumme sowie den Received Signal Strength Indicator (RSSI) bereit. *CC2420Sending* stellt Informationen über den Beginn einer Übertragung zur Verfügung. Der Parameter ist *true*, wenn die Übertragung erfolgreich gestartet werden konnte, und *false*, wenn dies nicht der Fall ist (z. B. wenn das Medium belegt ist und mit CCA gesendet werden soll). *CC2420SendFinished* hat keine Parameter und wird an eine höhere Protokollschicht gesendet, wenn der Transceiver eine Übertragung beendet hat. *CC2420Cca* stellt Informationen über den aktuellen CCA-Status bereit und wird immer dann nach oben gesendet, wenn dieser sich ändert. *CC2420Setup* und *CC2420Config* werden verwendet, um den Transceiver von höheren Protokollschichten aus zu konfigurieren. *CC2420Setup* wird zum Einstellen von Kanal und Sendesignalstärke verwendet, während *CC2420Config* Werte für CCA-Modus, CCA-Hysterese, CCA-Grenzwert, TX Turnaround, automatisches CRC, Präambellänge und Sync Word enthält. Für den CCA-Modus gibt es jedoch momentan keine Prüfung auf gültige IEEE 802.15.4-Daten, d. h. die Verwendung der drei unterschiedlichen Modi ist noch nicht vollständig umgesetzt. Mit Hilfe der *CC2420StatusReq*-Nachricht kann man Informationen über die aktuelle Konfiguration des Transceivers anfordern. Es wird dann eine *CC2420StatusResp*-Nachricht nach oben gesendet, welche die Werte aller konfigurierbaren Parameter enthält.

Verglichen mit dem CC2420-Simulationsmodul für ns-2 wurde die Abfrage der aktuell konfigurierten Parameterwerte und die Unterstützung von Subkanälen ergänzt. Weiterhin wird jetzt die Konfiguration von CCA-Hysterese, TX Turnaround, automatischem CRC, Präambellänge und Sync Word unterstützt, und es werden Informationen über CRC und RSSI bereitgestellt. Schließlich wurde die Verwendung verschiedener CCA-Modi vorgesehen.

7.6.4. Integration des CC2420-Moduls in FERAL

Neben der bereits beschriebenen Integration von ns-3 in FERAL sind zwei weitere Schritte erforderlich, um das CC2420-Simulationsmodul mit FERAL verwenden zu können. Zunächst

muss eine entsprechende Mediumklasse bereitgestellt werden, um ein CC2420-Medium aus einem FERAL-Simulationssystem heraus erzeugen zu können. Damit ist es bereits möglich, das CC2420-Modul mit der generischen Nachrichtenschnittstelle zu verwenden.

Um die in Abschnitt 7.6.3 beschriebene erweiterte Nachrichtenschnittstelle nutzen zu können, wird der bereits erwähnte mediumspezifische Kommunikationsmodus benötigt, welcher momentan nur für CC2420 verwendet wird. Zudem wird eine Repräsentation der Nachrichtenschnittstelle im FERAL-Framework benötigt. Dazu dient eine entsprechende, (fast) identische Java-Nachrichtenschnittstelle. Diese ist in Abbildung 7.17 dargestellt.

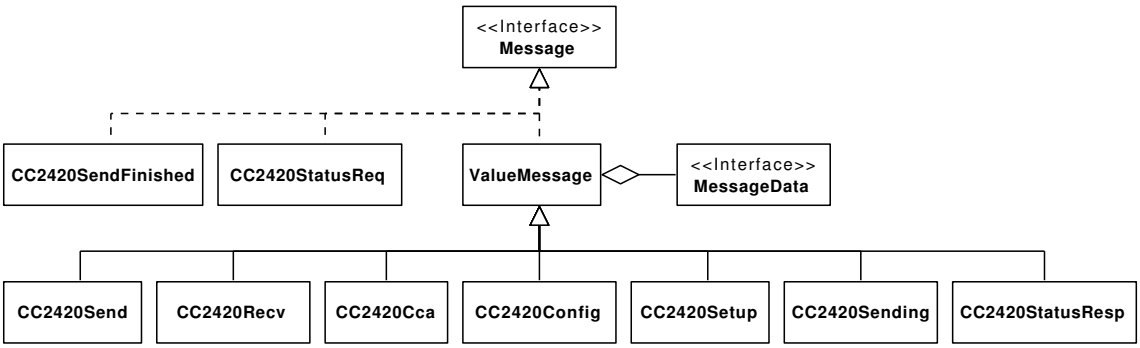

Abbildung 7.17.: Java-Repräsentation der erweiterten CC2420-Nachrichtenschnittstelle.

Die Nachrichten *CC2420SendFinished* und *CC2420StatusReq* haben keine Parameter, weshalb sie das *Message*-Interface direkt implementieren. Die restlichen Nachrichten sind von *ValueMessage* abgeleitet, welche die Parameter als *MessageData* kapselt.

Da die Kommunikation zwischen dem FERAL-Framework und dem ns-3-Simulator per JNI erfolgt, musste außerdem Code bereitgestellt werden, um Java-Nachrichten in die entsprechenden C++-Nachrichten umzuwandeln.

Wird der mediumspezifische Kommunikationsmodus in Kombination mit dem CC2420-Medium verwendet, so wird eine CC2420-spezifische virtuelle Applikation (*PcuCC2420*; s. Abbildung 7.8 auf Seite 138) auf den ns-3-Knoten installiert. Diese Applikation leitet die Nachrichten direkt an ein *CC2420InterfaceNetDevice* weiter, welches sie verarbeitet und die entsprechenden Methoden aufruft. Da die Applikation direkt mit dem NetDevice kommuniziert, ist die Verwendung eines Protokollstapels in diesem Fall nicht möglich.

7.7. Simulationsszenarien

Im Folgenden werden einige Simulationsszenarien vorgestellt, die mit Hilfe der in den vorangegangenen Abschnitten vorgestellten Simulationskomponenten evaluiert wurden. Diese dienen primär dazu, die Funktionalität der Komponenten zu zeigen, weshalb die simulierten Abläufe eher von untergeordneter Bedeutung sind. Zunächst werden einige Simulationen mit dem CC2420-Modul durchgeführt. Anschließend wird die Simulationskomponente ns-3 bei der Simulation eines komplexeren Systems eingesetzt, wobei von ns-3 bereitgestellte Medien anstelle des CC2420-Moduls verwendet werden.

7.7.1. Simulationen mit dem CC2420-Modul

In diesem Abschnitt werden die Ergebnisse einiger mit dem CC2420-Modul durchgeführter Simulationen vorgestellt. Dabei wird das CC2420-Modul sowohl in eigenständigen ns-3-

Simulationen verwendet als auch in Simulationen, bei denen ns-3 als Simulationskomponente innerhalb des FERAL-Frameworks zum Einsatz kommt.

7.7.1.1. Eigenständige ns-3-Simulationen

In den im Folgenden durchgeführten eigenständigen ns-3-Simulationen bestehen die Simulationssysteme aus zwei Knoten, welche als Sender bzw. Empfänger agieren. Auf dem Senderknoten wird eine *OnOffApplication* installiert, welche innerhalb konfigurierbarer Zeitintervalle Werte sendet. Auf dem Empfängerknoten wird eine *PacketSink*-Applikation installiert, welche Pakete empfängt. Beide Applikationen werden von ns-3 bereitgestellt. Abbildung 7.18 zeigt die Struktur.

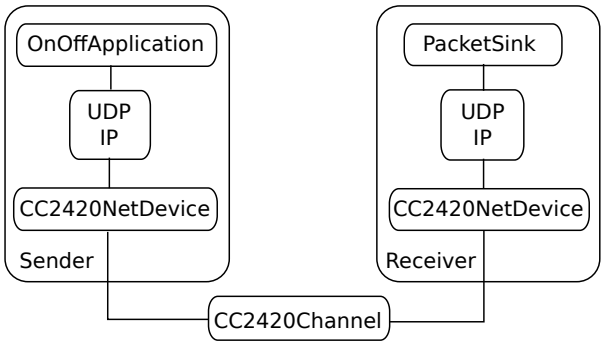

Abbildung 7.18.: Struktur eines eigenständigen ns-3-Simulationssystems.

Standardmäßig pausiert *OnOffApplication* für 1 s und sendet anschließend für 1 s Pakete. Dies wird für die Dauer der Simulation wiederholt. *OnOffApplication* wird bei 2 s Simulationszeit gestartet, und es werden insgesamt 5 s simuliert, d. h. in dem Intervall zwischen 3 s und 4 s werden Pakete gesendet. Durch Variation von Datenrate und Paketgröße der Applikation erhält man drei Simulationssysteme (diese wurden alle in C++ erstellt). Ein Auszug aus dem ersten dieser Simulationssysteme ist in Listing 7.1 gezeigt.

Zum Senden und Empfangen werden UDP-Sockets verwendet (Zeilen 12 und 17), und die IP-Adresse von *PacketSink* wird als Zieladresse für *OnOffApplication* verwendet (Zeile 12; *OnOffApplication* wird auf Knoten 0 (Zeile 15) und *PacketSink* auf Knoten 1 (Zeile 18) des *NodeContainers* installiert). Die erste Simulation verwendet eine Applikationsdatenrate von 70 kBit/s und eine Paketgröße von 20 Bytes (Zeilen 13 und 14). Mit diesen Werten erreichen alle Pakete ihr Ziel, was durch Listing 7.2 verdeutlicht wird.

Die Anzahl der von *OnOffApplication* gesendeten Bytes entspricht der Anzahl der von *PacketSink* empfangenen Bytes, d. h. alle Pakete wurden korrekt empfangen.

In der zweiten Simulation wird eine Applikationsdatenrate von 90 kBit/s anstatt 70 kBit/s konfiguriert. Da UDP- und IP-Header sowie CC2420-PhyHeader und -MacTrailer zur Applikationsdatenrate hinzukommen und außerdem die Kalibrierungszeit des Transceivers beachtet werden muss, ist diese Datenrate zu hoch für den Transceiver. Deshalb können nicht alle Pakete gesendet werden, was in Listing 7.3 gezeigt wird.

Die Applikation sendet alle Pakete entlang des Protokollstapels an den CC2420-Transceiver, aber da dieser eine neue Sendeanforderung erst dann verarbeiten kann, wenn die aktuelle beendet ist, werden einige der Pakete vom Transceiver verworfen. Auf diese Weise wurde ein Engpass im System identifiziert. Dieses Verhalten kann auch anhand weiterer Logausgaben

```
NodeContainer nodes; nodes.Create(2);
CC2420Helper cc2420;
NetDeviceContainer devices;
devices = cc2420.Install(nodes, ...); // CC2420NetDevice
InternetStackHelper stack; stack.Install(nodes);
MobilityHelper mobility;
...
Ipv4AddressHelper addr;
addr.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = addr.Assign(devices);

OnOffHelper onoff ("ns3::UdpSocketFactory",
    InetSocketAddress(interfaces.GetAddress(1), 9));
onoff.SetAttribute("DataRate", StringValue("70kbps"));
onoff.SetAttribute("PacketSize", StringValue("20"));
ApplicationContainer senderApps = onoff.Install(nodes.Get(0));

PacketSinkHelper pktSink("ns3::UdpSocketFactory", ...);
ApplicationContainer receiverApps = pktSink.Install(nodes.Get(1));
```

Listing 7.1.: Auszug aus dem Simulationssystem für die erste Simulation.

```
At time 3.00229s on-off application sent 20 bytes to 10.1.1.2 port 9
    total Tx 20 bytes
At time 3.00427s packet sink received 20 bytes from 10.1.1.1 port 49153
    total Rx 20 bytes
...
At time 3.99886s on-off application sent 20 bytes to 10.1.1.2 port 9
    total Tx 8740 bytes
At time 4.00084s packet sink received 20 bytes from 10.1.1.1 port 49153
    total Rx 8740 bytes
```

Listing 7.2.: Ausgabe der ersten Simulation (alle Pakete empfangen).

```
At time 3.00178s on-off application sent 20 bytes to 10.1.1.2 port 9
    total Tx 20 bytes
...
At time 3.99911s on-off application sent 20 bytes to 10.1.1.2 port 9
    total Tx 11240 bytes
At time 3.99932s packet sink received 20 bytes from 10.1.1.1 port 49153
    total Rx 5620 bytes
```

Listing 7.3.: Ausgabe der zweiten Simulation.

des CC2420-Moduls verifiziert werden, welche in Listing 7.3 aus Gründen der Lesbarkeit nicht enthalten sind.

In der dritten Simulation wird die Applikationsdatenrate analog zur ersten auf 70 kBit/s gesetzt, aber die Paketgröße wird auf 150 Bytes erhöht. Dies würde die MTU des *CC2420NetDevice* überschreiten, die nur 125 Bytes beträgt (UDP- und IP-Header sowie CC2420-PhyHeader und -MacTrailer vergrößern die Paketgröße der Applikation sogar noch). Um dies zu vermeiden, findet eine IP-Fragmentierung statt, d. h. das IP-Paket wird in mehrere hinreichend kleine Unterpakete aufgespalten. Dies funktioniert mit dem CC2420-Transceiver jedoch nicht,

da dieser eine neue Sendeanforderung erst dann verarbeiten kann, wenn die aktuelle beendet ist. Aus diesem Grund kann nur das erste Unterpaket jedes IP-Pakets erfolgreich übertragen werden. Da unvollständige IP-Pakete auf der Netzwerkschicht verworfen werden, empfängt *PacketSink* überhaupt keine Pakete. Listing 7.4 zeigt dies.

```
At time 3.01714s on-off application sent 150 bytes to 10.1.1.2 port 9
    total Tx 150 bytes
...
At time 3.99429s on-off application sent 150 bytes to 10.1.1.2 port 9
    total Tx 8700 bytes
```

Listing 7.4.: Ausgabe der dritten Simulation.

Anschließend wurden die Simulationen mit geringfügig modifizierter *OnOffApplication* und *PacketSink* wiederholt, um die Verwendung der erweiterten CC2420-Nachrichtenschnittstelle zu zeigen. Anstatt einen Protokollstapel auf den Knoten zu installieren, werden die Nachrichten nun von der Applikation direkt zum NetDevice weitergeleitet (und umgekehrt), welches jetzt ein *CC2420InterfaceNetDevice* ist. Damit sind auch die übertragenen Pakete kleiner, da nun keine UDP- und IP-Header mehr hinzugefügt werden.

In den modifizierten Simulationssystemen wurde die erweiterte Nachrichtenschnittstelle verwendet, um den Kanal – sowohl in *OnOffApplication* als auch in *PacketSink* – mit einer *CC2420Setup*-Nachricht vom Standardwert 11 auf 12 zu ändern, bevor die eigentliche Nachrichtenübertragung gestartet wird. Um sicherzustellen, dass der Kanalwechsel durchgeführt wurde, werden eine *CC2420StatusReq*- sowie die zugehörige *CC2420StatusResp*-Nachricht verwendet.

Die erste dieser modifizierten Simulationen liefert nahezu dieselbe Ausgabe wie die mit der ursprünglichen *OnOffApplication* und *PacketSink*. Da die Pakete kleiner sind, werden sie etwas früher empfangen. Außerdem empfangen die Applikationen nun weitere Nachrichten, welche von den *CC2420InterfaceNetDevices* gesendet werden. Listing 7.5 zeigt dies. Die Nachricht *CC2420Sending* mit Wert *true*, welche sofort nach Beginn einer Übertragung an die höheren Protokollschichten gesendet wird, zeigt beispielsweise deren erfolgreichen Beginn an.

```
At time 3.00229s on-off-cc2420 application sent 20 bytes total Tx 20 bytes
At time 3.00229s on-off-cc2420 application received CC2420Sending message
    with value true
At time 3.00257s packet sink cc2420 received CC2420Cca message with value
    false
At time 3.00337s on-off-cc2420 application received CC2420SendFinished
    message
At time 3.00337s packet sink cc2420 received 20 bytes with CRC=true and
    RSSI=-67; total Rx 20 bytes
At time 3.00341s packet sink cc2420 received CC2420Cca message with value
    true
...
```

Listing 7.5.: Ausgabe der modifizierten ersten Simulation.

In der zweiten modifizierten Simulation können nun alle Pakete erfolgreich übertragen werden. Da keine UDP- und IP-Header verwendet werden, kann die Applikationsdatenrate von 90 kBit/s vom Transceiver verarbeitet werden.

In der dritten modifizierten Simulation überschreitet die Paketgröße von 150 Bytes die MTU des NetDevice, da keine IP-Fragmentierung stattfindet. Aus diesem Grund wird kein Paket übertragen.

7.7.1.2. Simulationen mit FERAL und SDL

In diesem Abschnitt wird FERAL verwendet, um die Kommunikation zwischen zwei Sendern und einem Empfänger über ein gemeinsames Medium abzubilden. Die Topologie ist so gewählt, dass der Empfänger zwischen den Sendern positioniert ist und zu beiden jeweils den gleichen Abstand hat.

Im Folgenden wird zunächst ein abstraktes Simulationssystem für FERAL definiert. Dieses wird dann mit konkreten Simulatoren und Modellen instanziiert, woraus sich konkrete Simulationssysteme ergeben. Anschließend werden die Simulationsergebnisse vorgestellt.

Abstraktes Simulationssystem

Das abstrakte Simulationssystem ist in Abbildung 7.19 dargestellt.

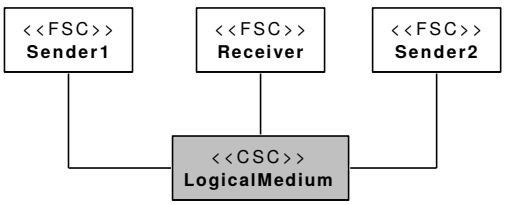

Abbildung 7.19.: Abstraktes Simulationssystem für das betrachtete Szenario.

Der erste Sender beginnt seine erste Übertragung bei 2,0001 s Simulationszeit und sendet dann alle 100 ms einen Wert. Der zweite Sender beginnt bei 3 s Simulationszeit und sendet alle 200 ms einen Wert.

Konkrete Simulationssysteme

Tabelle 7.3 gibt einen Überblick über die konkreten Simulatoren, die zur Erzeugung von konkreten Simulationssystemen verwendet werden.

Komponente	Typ	Simulator
Sender1, Sender2, Receiver	FSC	SDL
LogicalMedium	CSC	ns-3 (CC2420)

Tabelle 7.3.: Simulatoren in den konkreten Simulationssystemen.

Um das Verhalten der FSCs auf einem hohen Abstraktionslevel zu spezifizieren, wird SDL verwendet, während die CSC durch das in ns-3 integrierte CC2420-Simulationsmodul realisiert wird. Für die Kommunikation mit dem CC2420-Modul wird die erweiterte Nachrichtenschnittstelle verwendet.

Es werden zwei konkrete Simulationssysteme erzeugt, wobei die gewählten Simulatoren für beide gleich sind. Der einzige Unterschied besteht darin, dass für das erste System die Standardkonfiguration für das CC2420-Modul verwendet wird, während im zweiten die Sendesignalstärke des ersten Senders reduziert wird.

Simulationsergebnisse

Da im ersten Simulationssystem die Standardkonfiguration verwendet wird, überträgt der Transceiver mit voller Leistung. Zwischen 2 s und 3 s Simulationszeit ist nur ein Sender aktiv, so dass alle Signale korrekt empfangen werden können. Danach kollidiert jedes zweite Signal des ersten Senders mit einem Signal des zweiten Senders, so dass nur die Hälfte der Signale des ersten Senders und keins der Signale des zweiten Senders empfangen werden können. Dieses Verhalten wird in Listing 7.6 gezeigt.

```
1  ...
2  3,0000: Sender 2: Sent CC2420Send with value 0x20 0x00 0x00
3  3,0001: Sender 1: Sent CC2420Send with value 0x10 0x00 0x0a
4  3,0002: Sender 2: Received CC2420Sending with value true
5  3,0003: Sender 1: Received CC2420Sending with value true
6  3,0004: Receiver: Received CC2420Cca with value false
7  3,0007: Sender 2: Received CC2420SendFinished
8  3,0008: Sender 1: Received CC2420SendFinished
9  3,0008: Receiver: Received CC2420Cca with value true
10 3,0010: Sender 2: Received CC2420Cca with value true
11 3,0011: Sender 1: Received CC2420Cca with value true
12 ...
13 3,1001: Sender 1: Sent CC2420Send with value 0x10 0x00 0x0b
14 3,1003: Sender 1: Received CC2420Sending with value true
15 3,1005: Receiver: Received CC2420Cca with value false
16 3,1006: Sender 2: Received CC2420Cca with value false
17 3,1008: Sender 1: Received CC2420SendFinished
18 3,1008: Sender 2: Received CC2420Cca with value true
19 3,1008: Receiver: Received CC2420Recv with value 0x10 0x00 0x0b, CRC
       true, RSSI -67
20 3,1008: Receiver: Received CC2420Cca with value true
21 3,1011: Sender 1: Received CC2420Cca with value true
22 ...
```

Listing 7.6.: Ausgabe der ersten SDL-Simulation.

Zunächst übertragen beide Sender nahezu gleichzeitig (Zeilen 2 und 3). Obwohl CCA verwendet wird, finden beide Übertragungen statt, da die Kalibrierungszeit des ersten Senders noch nicht abgelaufen ist, wenn der zweite Sender seine Übertragung beginnt. Aus diesem Grund kollidieren die Rahmen. Der Empfänger stellt fest, dass das Medium belegt ist (Zeile 6), kann jedoch keinen gültigen Rahmen empfangen. Da die Mediumbelegung nur von solchen Knoten festgestellt werden kann, die sich nicht im Sendemodus befinden, wird diese hier von den Sendern nicht erkannt. Da der zweite Sender nur alle 200 ms überträgt, wird der nächste Rahmen des ersten Senders (Zeile 13) erfolgreich empfangen (Zeile 19).

Im zweiten Simulationssystem wird eine *CC2420Setup*-Nachricht verwendet, um die Sendesignalstärke des ersten Senders zu reduzieren (die des zweiten Senders bleibt unverändert). Da die Entfernung zum Empfänger gleich ist, ist das Signal des zweiten Senders stärker als das des ersten, wenn es beim Empfänger eintrifft. Auf diese Art und Weise kann ein Capturing-Effekt beobachtet werden, d. h. der Empfang des Signals vom zweiten Sender wird durch das interferierende Signal des ersten Senders nicht gestört. Dieses Verhalten wird in Listing 7.7 gezeigt.

Wie in der ersten Simulation übertragen beide Sender nahezu gleichzeitig (Zeilen 2 und 3). Da der Rahmen des zweiten Senders stärker ist, kann er erfolgreich empfangen werden (Zei-

```
1  ...
2  3,0000: Sender 2: Sent CC2420Send with value 0x20 0x00 0x00
3  3,0001: Sender 1: Sent CC2420Send with value 0x10 0x00 0x0a
4  3,0002: Sender 2: Received CC2420Sending with value true
5  3,0003: Sender 1: Received CC2420Sending with value true
6  3,0004: Receiver: Received CC2420Cca with value false
7  3,0007: Sender 2: Received CC2420SendFinished
8  3,0007: Receiver: Received CC2420Recv with value 0x20 0x00 0x00, CRC
       true, RSSI -67
9  3,0008: Sender 1: Received CC2420SendFinished
10 3,0008: Receiver: Received CC2420Cca with value true
11 3,0010: Sender 2: Received CC2420Cca with value true
12 3,0011: Sender 1: Received CC2420Cca with value true
13 ...
14 3,1001: Sender 1: Sent CC2420Send with value 0x10 0x00 0x0b
15 3,1003: Sender 1: Received CC2420Sending with value true
16 3,1005: Receiver: Received CC2420Cca with value false
17 3,1006: Sender 2: Received CC2420Cca with value false
18 3,1008: Sender 1: Received CC2420SendFinished
19 3,1008: Sender 2: Received CC2420Cca with value true
20 3,1008: Receiver: Received CC2420Recv with value 0x10 0x00 0x0b, CRC
       true, RSSI -72
21 3,1008: Receiver: Received CC2420Cca with value true
22 3,1011: Sender 1: Received CC2420Cca with value true
23 ...
```

Listing 7.7.: Ausgabe der zweiten SDL-Simulation.

le 8). Dies ist nur möglich, da der Empfang des Rahmens des zweiten Senders vor dem des ersten Senders beginnt, da der Transceiver nicht von einem gerade empfangenen Rahmen auf einen mit größerer Signalstärke umschalten kann. Der nächste Rahmen des ersten Senders (Zeile 14) kann erfolgreich empfangen werden (Zeile 20), da der zweite Sender nur alle 200 ms überträgt, d. h. es gibt keinen kollidierenden Rahmen.

Die durchgeführten Simulationen zeigen, dass sich das CC2420-Modul sowohl für eigenständige ns-3-Simulationen als auch für Simulationen mit dem FERAL-Framework (unter Verwendung der Simulationskomponente ns-3) eignet. Durch die Nutzung vorhandener Simulationskomponenten von FERAL ergeben sich zusätzliche Möglichkeiten, um das Verhalten von Knoten zu spezifizieren (hier wurde dazu beispielsweise SDL verwendet).

7.7.2. Simulation des Adaptive Cruise Control-Szenarios

Während im vorangegangenen Abschnitt die Funktionalität des CC2420-Moduls gezeigt wurde, wird nun die in dieser Arbeit entwickelte Simulationskomponente ns-3 bei der Simulation eines komplexeren Systems verwendet. Dabei handelt es sich um ein Adaptive Cruise Control (ACC)-Szenario, welches einen realistischen Anwendungsfall aus der Automobildomäne darstellt. Das System umfasst verschiedene Komponenten, die mit unterschiedlichen Simulatoren simuliert wurden. Anstelle des CC2420-Moduls kommen hier das von ns-3 bereitgestellte WLAN-Medium sowie das vereinfachte geswitchte Ethernet-Medium (s. Abschnitt 7.5.1) zum Einsatz. Die vorgestellte Simulation ist auch in [BCG+13, BCG+14] zu finden.

ACC ist eine erweiterte Geschwindigkeitsregelung, welche darauf abzielt, eine Zielgeschwindigkeit eines Versuchsfahrzeugs beizubehalten, wobei Störgrößen (beispielsweise der aktuelle Gradient oder der Luftwiderstand) die Geschwindigkeit beeinflussen. Zusätzlich zu einer ein-

fachen Geschwindigkeitsregelung wird ein Radarsensor verwendet, um Hindernisse vor dem Fahrzeug zu erkennen. Abhängig von Distanz zu und Geschwindigkeit von Hindernissen wird die Geschwindigkeit des Fahrzeugs angepasst, um einen minimalen Sicherheitsabstand einzuhalten. Falls erforderlich, wird eine Notbremsung durchgeführt.

In den verwendeten Simulationssystemen wird ein Versuchsfahrzeug simuliert, welches eine hügelige Straße entlangfährt (s. Abbildung 7.20).

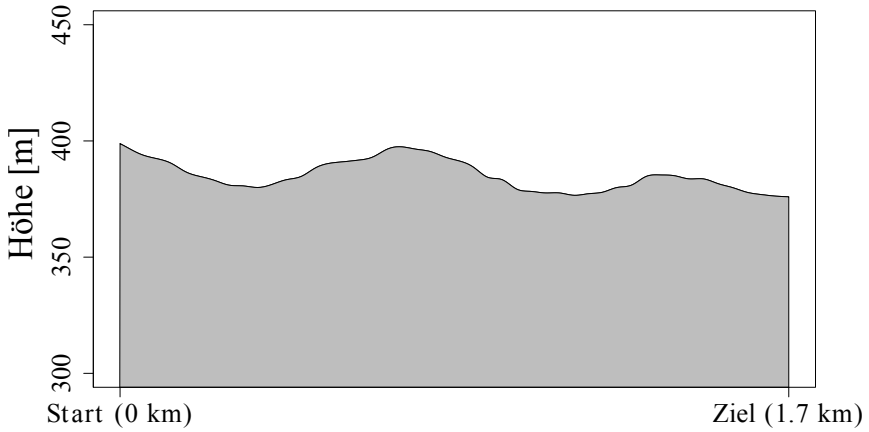

Abbildung 7.20.: Höhenprofil des Szenarios [BCG+14].

Der resultierende Gradient fungiert als Störgröße für die Regelstrecke und muss von ACC berücksichtigt werden. Neben dem Versuchsfahrzeug befinden sich weitere Fahrzeuge auf der Straße. Ein Auto mit geringerer Geschwindigkeit fährt voraus, was das ACC-System des Versuchsfahrzeugs dazu zwingt, dessen Geschwindigkeit automatisch anzupassen. Bei etwa 1.7 km stoppt das vorausfahrende Auto, was beim Versuchsfahrzeug eine Bremsung veranlasst, um einen Unfall zu vermeiden.

Im Folgenden wird zunächst das abstrakte Simulationssystem für FERAL definiert. Um die Austauschbarkeit der CSCs zu gewährleisten und somit verschiedene Kommunikationstechnologien evaluieren zu können, werden Bridges und Gateways verwendet. Das abstrakte System wird dann mit konkreten Simulatoren und Modellen instanziiert, um so eine Menge konkreter Simulationssysteme zu erhalten. Anschließend werden die Simulationsergebnisse vorgestellt.

Abstraktes Simulationssystem

Abbildung 7.21 zeigt das abstrakte Simulationssystem des ACC-Szenarios. Der Algorithmus zur Kontrolle der Geschwindigkeit des Versuchsfahrzeugs ist ein Proportional-Integral-Derivative (PID)-Regler, welcher Bestandteil des *ACControllers* ist. Die Geschwindigkeit vorausfahrender Fahrzeuge sowie die Distanz zu diesen oder zu anderen Hindernissen wird vom *RadarSensor* gemessen. *CarModel* repräsentiert das physikalische Modell des Versuchsfahrzeugs (Regelstrecke) mitsamt Motor und Motorsteuergerät. Das Modell berücksichtigt den Einfluss der Fahrzeugmasse, den Gradienten sowie ein vereinfachtes lineares Modell für die Widerstandskräfte (z. B. Roll- und Windwiderstand), welche sich auf die Geschwindigkeit auswirken. Eingaben für das Modell sind der aktuelle Gradient und prozentuale Stellwerte für die Aktuatoren (Motor und Bremsen), welche vom *ACController* geliefert werden. Ausgegeben wird die derzeitige Geschwindigkeit des Versuchsfahrzeugs. *SpeedInput* stellt die Zielgeschwindigkeit über den *EngineControlBus* (*ECB)* zur Verfügung, und *GradientInput* liefert den aktuellen Gradienten als Störgröße für die Regelstrecke. Für die von *RadarSensor*, *SpeedInput* und *GradientInput* gesendeten Nachrichten können beispielsweise aufgezeichne-

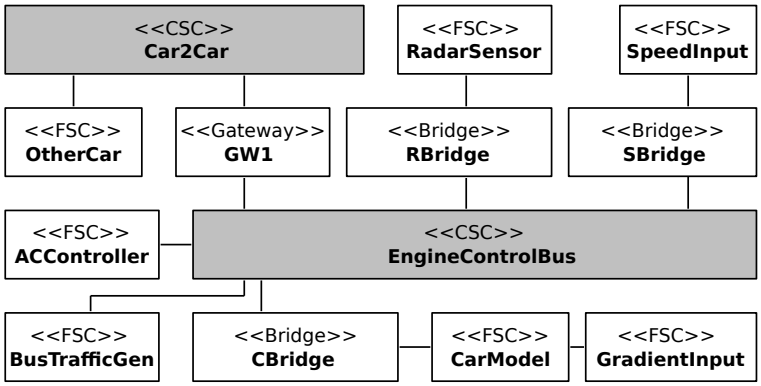

Abbildung 7.21.: Abstraktes Simulationssystem des ACC-Szenarios [BCG+13, BCG+14].

te Nachrichten verwendet werden, um spezifische Szenarien zu wiederholen oder Feldtests nachzubilden. Zusätzlich gibt es einen konfigurierbaren Traffic Generator *BusTrafficGen*, welcher die Kommunikationscharakteristika weiterer Komponenten simuliert. Das System wird um ein weiteres Medium (*Car2Car*) erweitert. Dieses simuliert die Kommunikation zwischen Fahrzeugen, indem Verkehrswarnungen von Fahrzeug zu Fahrzeug propagiert werden. Diese Warnungen werden vom Gateway *GW1* an *ECB* weitergeleitet.

Verkehrswarnungen werden als sporadische Nachrichten modelliert, während aktuelle Geschwindigkeit, Motorsteuerwerte und Zielgeschwindigkeit periodische Nachrichten mit einem Intervall von 20 ms (aktuelle Geschwindigkeit und Motorsteuerung) bzw. 100 ms (Zielgeschwindigkeit) darstellen. Distanz- und Geschwindigkeitswerte eines Hindernisses werden vom Radarsensor mit einem Intervall von 100 ms gesendet. Befindet sich ein Hindernis innerhalb des minimalen Sichrheitsabstands, so wird dieses Intervall auf 20 ms reduziert. Das Bremssignal ist ein sporadisches Signal, welches vom *ACController* erzeugt wird, wenn der minimale Sicherheitsabstand unterschritten wird.

Konkrete Simulationssysteme

Durch die Auswahl von spezifischen Modellen, Simulatoren und Kommunikationstechnologien wird aus dem abstrakten Simulationssystem eine Menge von konkreten Simulationssystemen erzeugt. Die konkreten Simulatoren sind in Tabelle 7.4 zusammengefasst.

Komponente	**Typ**	**Simulator**
ECB	CSC	CAN, FlexRay, ns-3 (Switched Ethernet)
Car2Car	CSC	ns-3 (WLAN)
CarModel	FSC	Simulink
ACController	FSC	SDL
Andere	FSC	Native Java-Implementierung

Tabelle 7.4.: Simulatoren in den konkreten Simulationssystemen [BCG+14].

Abgesehen vom *ECB* werden alle FSCs und CSCs mit einem einzelnen Simulator und Verhaltensmodell simuliert. Um den Einfluss verschiedener Kommunikationstechnologien auf die Übertragungsverzögerungen zu untersuchen, wird der *ECB* mit CAN, FlexRay und dem von

ns-3 bereitgestellten vereinfachten geswitchten Ethernet-Medium (s. Abschnitt 7.5.1) instanziiert (letzteres wird im Folgenden einfach als Ethernet bezeichnet). Um den Austausch der vom *ECB* verwendeten Kommunikationstechnologie zu vereinfachen, sind die meisten der FSCs durch Bridges von den CSCs entkoppelt. Dadurch ist keine Anpassung der FSCs erforderlich. Für den *ACController* ist eine solche Entkopplung nicht möglich, da dessen SDL-Modell gleichzeitig den Hardwaretreiber für den Zugriff auf die entsprechende Kommunikationstechnologie simuliert. Deshalb wurden für den *ACController* drei unterschiedliche, aber sehr ähnliche Verhaltensmodelle verwendet. Die Simulationskomponente ns-3 findet zusätzlich zur Simulation des *ECB* auch bei der Simulation des *Car2Car*-Mediums Verwendung, bei dem WLAN als Kommunikationstechnologie eingesetzt wird.

Der CAN-Bus läuft mit 1 MBit/s, FlexRay mit 10 MBit/s mit exklusiven statischen Slotreservierungen, und Ethernet läuft ebenfalls mit 10 MBit/s. Für FlexRay werden zwei unterschiedliche Konfigurationen verwendet, die auf demselben TDMA-Schema basieren: In der ersten Konfiguration erzeugen die Applikationen ihre Nachrichten unabhängig vom Kommunikationszyklus, während in der zweiten Konfiguration die Erzeugung von Nachrichten für tatsächliche Geschwindigkeit und Zielgeschwindigkeit sowie die Radarnachrichten mit dem FlexRay-Kommunikationszyklus und den Slotreservierungen innerhalb des Zyklus synchronisiert sind. Bei der Verwendung von Ethernet werden die Nachrichten mittels UDP-Broadcasts übertragen und an Ports zugestellt, die für jeden Nachrichtentyp eindeutig sind. CAN-IDs, FlexRay-Slots und Ethernet-Ports der einzelnen Nachrichtentypen sind in Tabelle 7.5 zusammengefasst.

Nachrichtentyp	**CAN-ID**	**FlexRay-Slot**	**Ethernet-Port**
Radarsignal	20	3; 88	200
Tatsächliche Geschwindigkeit	70	1; 86	700
Zielgeschwindigkeit	45	2	450
Motorsteuerung	110	5; 90	1100
Bremssignal	10	6; 91	100
Verkehrswarnung	188	67	1880

Tabelle 7.5.: Nachrichtentypen im ACC-System [BCG+13, BCG+14].

Es ist zu beachten, dass FlexRay ein TDMA-Schema verwendet, bei dem für einige sicherheitskritische Nachrichten (Nachrichten für Radarsignal, tatsächliche Geschwindigkeit, Motorsteuerung und Bremssignal) zwei Übertragungsslots pro Zyklus verwendet werden, um Jitter und Reaktionsverzögerung zu reduzieren.

Simulationsergebnisse

Die Ergebnisse eines Simulationslaufs mit Ethernet als *ECB* sind in Abbildung 7.22 dargestellt. Die gepunktete Linie zeigt die vom Fahrer vorgegebene Zielgeschwindigkeit und die durchgezogene Linie die tatsächliche Geschwindigkeit, die vom *ACController* geregelt wird. Die gestrichelte Linie stellt den Abstand zum nächsten vorausfahrenden Fahrzeug dar.

Das Versuchsfahrzeug startet mit einer Zielgeschwindigkeit von 50 km/h, die nach 3 s Simulationszeit auf 140 km/h erhöht wird. Das vorausfahrende Fahrzeug fährt mit einer konstanten Geschwindigkeit von 100 km/h und beginnt nach 54 s zu bremsen. Nach 20 s unterschreitet die Distanz zu dem vorausfahrenden Fahrzeug den doppelten minimalen Sicherheitsabstand,

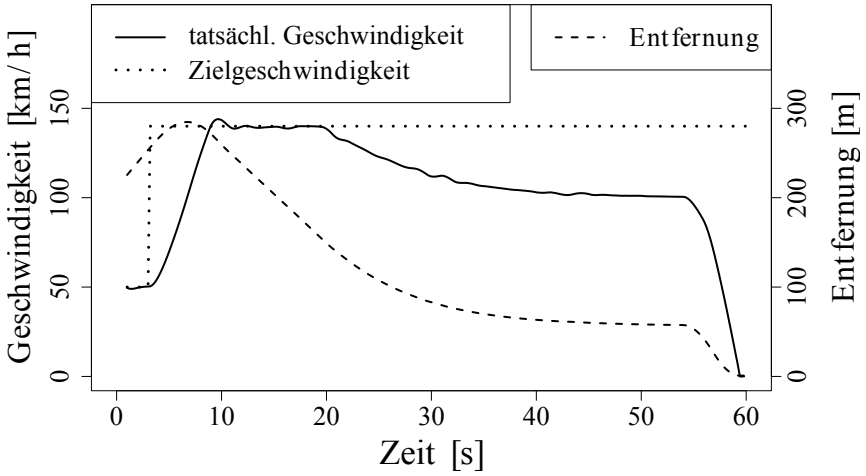

Abbildung 7.22.: Automatische Geschwindigkeitsanpassung durch den *ACController* zur Einhaltung eines minimalen Sicherheitsabstands (vgl. [BCG+14]).

woraufhin das Versuchsfahrzeug automatisch seine Geschwindigkeit reduziert und die vorgegebene Zielgeschwindigkeit ignoriert. Bei Erreichen des minimalen Sicherheitsabstands (ca. 50 m bei 100 km/h) hat sich das Versuchsfahrzeug an die Geschwindigkeit des vorausfahrenden angepasst. Sobald das vorausfahrende Fahrzeug bremst, erzwingt der *ACController* automatisch einen Bremsvorgang, um einen Unfall zu verhindern.

Abbildung 7.23 zeigt die beobachteten durchschnittlichen, maximalen und minimalen Verzögerungen für den *ECB* bei Verwendung von CAN, Ethernet und den beiden FlexRay-Konfigurationen für die ACC-relevanten Nachrichten.

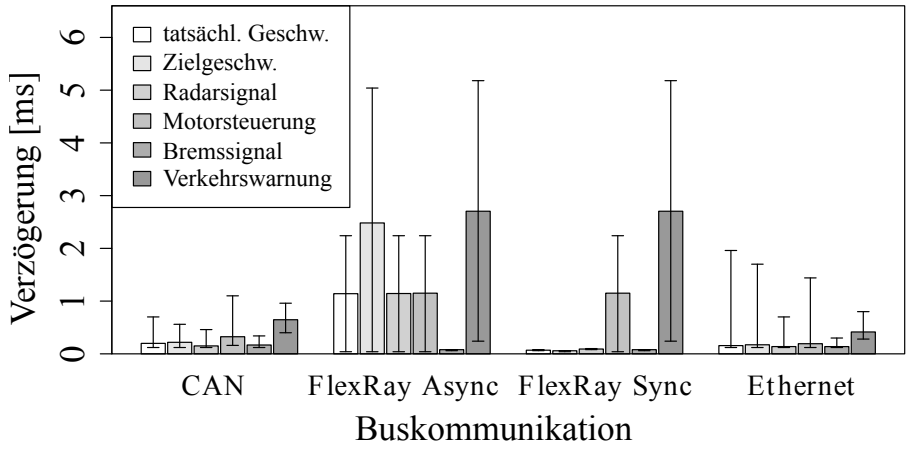

Abbildung 7.23.: Durchschnittliche, maximale und minimale Verzögerung der einzelnen Nachrichtentypen für unterschiedliche Kommunikationstechnologien [BCG+14].

FlexRay Sync bezieht sich auf die Lösung, bei der eine Synchronisation zwischen dem FlexRay-Kommunikationszyklus und allen periodisch Daten versendenden Applikationen außer der Motorsteuerung vorgenommen wurde, wohingegen FlexRay Async keine derartige Synchronisation verwendet (die Motorsteuerung ist nicht mit dem Kommunikationszyklus

synchronisiert, da dies eine Anpassung des Regelungsalgorithmus erfordern würde). Abgesehen davon verwenden beide Varianten denselben Kommunikationsschedule und dieselbe Slotkonfiguration.

Der Traffic Generator erhöht durch zusätzliche Übertragungen die Auslastung des Mediums auf 39.7% (CAN), 9.8% (FlexRay Async und Sync) bzw. 23.9% (Ethernet). Da bei CAN und Ethernet die Rahmen direkt übertragen werden können, ist die durchschnittliche Übertragungsverzögerung kleiner als bei FlexRay Async. Im Gegensatz zu FlexRay garantieren diese Protokolle jedoch keine maximale Übertragungsverzögerung. Beispielsweise führt eine andere Konfiguration mit 79% Busauslastung für CAN dazu, dass Bremsnachrichten eine maximale Verzögerung von 5.5 ms haben (in Abbildung 7.23 nicht gezeigt). Der durchschnittliche Performanzgewinn von Ethernet gegenüber CAN ist nicht sehr hoch, obwohl die Datenrate um Faktor 10 höher ist. Die maximale Verzögerung von Ethernet ist größer als die von CAN, da das Ethernet-Protokoll im Vergleich zu der relativ kleinen Payload von 2-8 Bytes einen verhältnismäßig großen Overhead hinzufügt. Ein Vorteil von FlexRay besteht darin, dass die maximale Verzögerung eine garantierte obere Schranke darstellt und unabhängig von der Auslastung des Mediums ist. Eine Synchronisation zwischen den Applikationen und dem FlexRay-Kommunikationszyklus minimiert Verzögerung und Jitter (s. FlexRay Sync in Abbildung 7.23) verglichen mit einem unsynchronisierten oder ereignisgetriggerten Ansatz. Ein Nachteil davon ist jedoch die enge Kopplung; außerdem ist eine solche Synchronisation nur für periodische Nachrichten möglich.

Die Verzögerungen von Verkehrswarnungen sind im Allgemeinen höher, denn sie bestehen aus der Verzögerung zur Übertragung der Nachricht von Fahrzeug zu Fahrzeug (über WLAN), einer Verarbeitungszeit des Gateways und der anschließenden Übertragung über den *ECB*.

Die durchgeführten Simulationen des ACC-Szenarios zeigen, dass FERAL ein nützliches Werkzeug zur Simulation umfangreicher Systeme darstellt und die Integration von ns-3 dabei einen signifikanten Beitrag leistet. In diesem Fall wurde durch die Simulationskomponente ns-3 die Evaluation verschiedener Entwurfsalternativen (Verwendung der Kommunikationstechnologien CAN, FlexRay Async und Sync sowie Ethernet für den *ECB*) unterstützt und die Spezifikation eines drahtlosen Mediums für die Kommunikation zwischen Fahrzeugen ermöglicht.

7.8. Vergleich mit existierenden Simulationsansätzen

Es existieren zahlreiche Simulationsansätze, die darauf abzielen, spezielle Simulatoren entweder zu koppeln oder ineinander zu integrieren. Im Folgenden wird ein Überblick über vorhandene Ansätze gegeben und diese werden mit dem Simulationsframework FERAL verglichen. Auch der Simulator ns-3 wurde bereits um zahlreiche Module erweitert. Zur Simulation des CC2420-Transceivers existieren ebenfalls einige Ansätze, jedoch gab es bisher kein entsprechendes ns-3-Modul.

7.8.1. Kopplung und Erweiterung von Simulatoren

Es existieren verschiedene Ansätze für die Kopplung einzelner Simulatoren bzw. für die Erweiterung existierender Simulatoren um neue Funktionalität. In [SSK09] wenden die Autoren die Kopplung von Simulatoren auf den Automobilbereich an, um die Auswirkungen von Car-to-X-Systemen auf das Verhalten der Fahrzeuge zu simulieren. Dies erfordert die Kopplung eines Netzwerksimulators, eines Straßenverkehrssimulators und eines Funkkanalsimulators. Die in [SKG07] vorgestellte Arbeit zeigt die Anwendung des Simulatorkopplungsansatzes auf

die Netzwerkdomäne, indem Simulatoren für Protokolle der Sicherungsschicht und solche der Netzwerkschicht gekoppelt werden.

In [NL02] wird ein modellbasierter Ansatz für die Entwicklung von X-by-Wire-Applikationen vorgestellt, welcher Matlab Simulink verwendet. In [LWL08] dient Simulink zur Entwicklung eines Modells für den CAN-Bus, welches Performanz-Simulationen hinsichtlich der Kommunikation erlaubt.

In [KGGR05] wird ns-2 um ein Modul zur Simulation von SDL-Systemen erweitert. Ein ähnlicher, jedoch aktuellerer Ansatz, welcher SDL an ns-3 koppelt, wird in [BF10] beschrieben. Dieser Ansatz wird in [BF12] erweitert, wobei dort der Fokus auf der Modellierung von Konfigurationen liegt. Die in [KGGR05] beschriebene Simulatorkopplung wurde zu einem generischen Framework weiterentwickelt (s. z. B. [KB06, KG08, Kuh09]), welches den Vorläufer von FERAL darstellt und neben ns-2 und einem Modul zur Simulation von SDL-Systemen beispielsweise Avrora [TLP05] integriert. Avrora ist ein taktgenauer Befehlssatzsimulator für Atmel AVR-Mikrocontroller. Auch das Ptolemy-Framework [EJL+03], von dem einige Konzepte für FERAL übernommen wurden, ermöglicht die generische Kopplung von Simulatoren.

FERAL ist weder auf die Kopplung bestimmter Simulatoren – beispielsweise SDL und ns-2 [KGGR05] bzw. ns-3 [BF10, BF12] – noch auf eine spezielle Strategie wie die vertikale Kopplung zweier Simulatoren in der Netzwerkdomäne [SKG07] beschränkt. Auch ist FERAL nicht auf eine bestimmte Anwendungsdomäne limitiert (beispielsweise die Simulation von Car-to-X-Systemen [SSK09]), und Simulationssysteme können sich aus Komponenten zusammensetzen, die mit unterschiedlichen Beschreibungstechniken modelliert sind. Im Gegensatz dazu erzwingen beispielsweise die in [NL02] und [LWL08] beschriebenen Ansätze die Verwendung von Matlab Simulink. Darüber hinaus wird die Simulation von Komponenten mit unterschiedlichen Abstraktionsebenen in einem einzigen Szenario von diesen nicht unterstützt.

7.8.2. Integration von Modulen in ns-3

Aufgrund seiner modularen Struktur ist ns-3 gut erweiterbar, weshalb zahlreiche Simulationsmodule von Dritten entwickelt wurden. Unter diesen befinden sich Module für Routingprotokolle, beispielsweise für das Ad-hoc On-demand Distance Vector Routing (AODV) [PR99], dessen Implementierung für ns-3 in [PKG+10] beschrieben wurde. Ein Modul für das Destination-Sequenced Distance Vector (DSDV)-Routingprotokoll [PB94] wurde in [NCÇ+11] vorgestellt, während [VMM08] einen IPv6-Stack für ns-3 präsentiert.

Darüber hinaus wurde in [BM09] ein ns-3-Framework für spektrumspezifische Simulationen vorgestellt. Die Autoren implementieren spektrumspezifische Kanal- und Physical-Layer-Modelle. Dadurch kann analysiert werden, wie die Performanz von Protokollen auf höheren Protokollschichten durch frequenzabhängige Aspekte der Kommunikation auf der physikalischen Schicht beeinflusst wird.

7.8.3. Ansätze zur Simulation des CC2420-Transceivers

In der Literatur werden verschiedene Simulationsansätze für den CC2420-Transceiver beschrieben. Die Autoren von [SJK07] erweitern den TinyOS SIMulator (TOSSIM) [LLWC03] um ein verbessertes drahtloses Propagierungsmodell und einen physikalischen Funkstack, der auf dem CC2420-Transceiver basiert. TinyOS [Uni15] ist ein weit verbreitetes Betriebssystem für drahtlose Sensornetze. Sein Quellcode kann direkt für TOSSIM verwendet werden, welcher auch Betriebssystemoverhead berücksichtigt. Das in [SJK07] vorgestellte CC2420-Modell verwendet u. a. CCA, misst die Empfangssignalstärke und erlaubt die Konfiguration von Sende-

signalstärke und Kanal. Die Resultate, die man mit diesem Simulator erhält, sind unabhängig von einem konkreten Prozessor; jedoch sind sie spezifisch für das TinyOS-Betriebssystem. Deshalb kann dieser nicht verwendet werden, um Protokolle unabhängig von einem konkreten Betriebssystem zu evaluieren.

In [PP08] wird AvroraZ vorgestellt, welcher den Avrora-Simulator [TLP05] um ein Modul zur Simulation des CC2420-Transceivers erweitert. Das Simulationsmodul beinhaltet Adresserkennung, Acknowledgements für die Rahmen, CCA und einige andere Eigenschaften. Eine ähnliche Lösung wird in [JLW+09] präsentiert. Dort wird ein Befehlssatzsimulator für Sensornetze vorgestellt, welcher ein detailliertes CC2420-Simulationsmodul verwendet. Dieser Simulator stellt eine taktgenaue Prozessoremulation des Atmel ATmega128 [Atm11] bereit, welche unabhängig vom verwendeten Betriebssystem ist, und modelliert auch die interne Struktur des CC2420-Transceivers (z. B. Register und Hauptspeicher). Das Paper liefert jedoch nicht viele Details zur CC2420-Implementierung. Sowohl [PP08] als auch [JLW+09] beschreiben Befehlssatzsimulatoren. Diese sind für große Netze nicht geeignet, da Simulationen damit sehr zeitaufwendig sind. Darüber hinaus sind diese Simulatoren auf spezielle Prozessoren beschränkt und deshalb für eine generische Evaluierung von Protokollen auf höheren Schichten nicht geeignet.

8 Kapitel 8. Fazit und Ausblick

8.1. Ergebnisse der Arbeit

Einen wesentlichen Bestandteil der vorliegenden Arbeit bildet das formale Modell, welches dazu dient, Netzwerke zu beschreiben und die Bedingungen zu identifizieren, unter denen Übertragungen interferieren. Das Modell beinhaltet die globale Reservierungsbedingung, die festlegt, ob ein Slot s für einen Sendevorgang eines Knotens a an einen Knoten b verwendet werden kann. Aus der globalen Reservierungsbedingung wurden mehrere lokale Reservierungsbedingungen abgeleitet. Diese aggregieren die Informationen der Nachbarknoten, um die Entscheidung, ob s zum Senden von a an b reserviert werden kann, lokal im Knoten a treffen zu können. Da eine direkte Kommunikation mit Interferenznachbarn mittels regulärer Übertragungen nicht möglich ist, wurde die 2-Hop-Interferenzannahme verwendet. Dadurch können die Informationen von Interferenznachbarn der Knoten a und b auf Informationen von deren Kommunikationsnachbarn abgebildet werden. Weiterhin wurde eine Form der Reservierungsbedingung definiert, die sich zur Propagierung von Reservierungsinformationen bei der Ermittlung einer Route vom Ziel zur Quelle eignet.

Anschließend wurden existierende QoS-Routing- und Reservierungsprotokolle analysiert, wobei der Schwerpunkt auf Protokollen lag, die deterministische Reservierungen in Form von Zeitslots vornehmen. Dabei wurden Probleme identifiziert (beispielsweise fehlerhafte Slotreservierungen). Diese Probleme sind u. a. auf parallele Routensuchen und auf Kollisionen bzw. indeterministische Verzögerungen von Kontrollpaketen zurückzuführen.

Einen weiteren zentralen Bestandteil der Arbeit bildet das Protokoll RBBQR. Dieses ermittelt QoS-Routen auf deterministische Weise, ohne dass Kollisionen bei Kontrollpaketen auftreten. Für die ermittelten Routen werden Zeitslots reserviert. Parallele Routensuchen werden netzweit serialisiert, so dass keine gegenseitigen Blockierungen oder Fehlreservierungen aufgrund von noch nicht propagierten Reservierungsinformationen vorkommen. Fehlreservierungen aufgrund von Selbstinterferenz werden erkannt und behandelt. Ferner wird berücksichtigt, dass die Interferenzreichweite i. Allg. größer ist als die Kommunikationsreichweite.

Zusätzlich wurde das Protokoll QMRP vorgestellt. Dieses behandelt die Ermittlung von Multicast-Routen in teilstationären Netzen und die Reservierung von Slots für die entsprechenden Routen. Die Topologie der stationären Knoten wird ermittelt. Diese Information wird genutzt, um Routing und Reservierungen zentralisiert durchzuführen. Mobile Knoten werden durch geeignete Verfahren unterstützt, so dass Routenbrüche proaktiv vermieden werden.

Den letzten Kernbestandteil der Arbeit bilden die Simulationskomponente ns-3 sowie das CC2420-Modul. Durch die Simulationskomponente ns-3 wird die Verwendung einiger Komponenten des Netzwerksimulators ns-3 aus dem FERAL-Framework heraus ermöglicht. Dies kann beispielsweise genutzt werden, um ein Protokoll mit SDL formal zu spezifizieren und diese Spezifikation mittels ns-3 zu simulieren. Dadurch können zur Evaluation von Protokollen realistische Medien (mit entsprechenden Verzögerungs- und Verlustmodellen) eingesetzt

werden, ohne dass die Spezifikation zusätzlich für ns-3 implementiert werden muss (dies würde neben dem höheren Aufwand auch ein höheres Fehlerpotenzial beinhalten). Auch die Verwendung von komplexen Mobilitätsmodellen ist auf diese Weise möglich, um die Bewegungen von Knoten abzubilden. Das CC2420-Modul ermöglicht die Verwendung des CC2420-Transceivers sowohl für eigenständige ns-3-Simulationen als auch für Simulationen mit FERAL.

8.2. Mögliche Weiterentwicklungen

Für das Protokoll RBBQR sind noch einige Weiterentwicklungen denkbar. Beispiele hierfür sind die Unterstützung zusätzlicher QoS-Metriken (z. B. Übertragungsverzögerung) sowie eine Erweiterung auf Multicast-Übertragungen. Zudem wurden einige Aspekte von Ad-Hoc-Netzen noch nicht berücksichtigt, beispielsweise das nachträgliche Hinzufügen neuer Knoten. Auch die Mobilität vorhandener Knoten wurde bisher nur insofern betrachtet, als dass diese sich auseinander bewegen können. Bewegen Knoten sich jedoch aufeinander zu, so können neue Interferenzlinks entstehen, womit bestehende Reservierungen möglicherweise nicht mehr exklusiv sind. Weiterhin wäre es sinnvoll, eine Fehlerbehandlung zu integrieren, um mit unzuverlässigen Übertragungen umgehen zu können. Zusätzlich könnte man das Protokoll ATDP [Kra13, KCG15] verwenden, um die Topologie des Netzes zu ermitteln, wodurch die Einschränkung der Black-Burst-Reichweite und die Ermittlung der Distanz zur Quelle in Phase 1 obsolet wären. Auch die in Phase 2 verwendeten Heuristiken für die Slotauswahl könnten noch ergänzt werden, beispielsweise durch die Integration von MRFP [SCCC06].

RBBQR wurde formal in SDL spezifiziert, jedoch bisher nur funktional simuliert (dazu wurde der interne Simulator der Rational SDL Suite [IBM15] verwendet). Für eine umfassende Evaluation wäre es sinnvoll, das Verhalten unter Verwendung real existierender Hardware (WLAN, CC2420, ...) sowie realistischer Verzögerungs- und Verlustmodelle zu untersuchen. Dafür eignet sich der Simulator ns-3. Um die erstellte SDL-Spezifikation von RBBQR mit ns-3 zu simulieren, kann das Framework FERAL und die in dieser Arbeit entwickelte Simulationskomponente ns-3 verwendet werden. Dies erfordert allerdings eine Anpassung der aktuellen Version der SDL-Spezifikation, beispielsweise die Verwendung eines an der Schnittstelle zu FERAL unterstützten Rahmenformats.

Für das Protokoll QMRP ist ebenfalls eine Unterstützung weiterer QoS-Metriken denkbar. Auch hier wurden bisher einige Aspekte von Ad-Hoc-Netzen vernachlässigt, beispielsweise der Ausfall von Knoten sowie das nachträgliche Hinzufügen neuer Knoten, und es wurde noch keine Fehlerbehandlung integriert. Zudem sind einige Aspekte des Protokolls noch nicht vollständig ausgearbeitet, beispielsweise die Anfragen an den Masterknoten zum Abonnieren eines Dienstes (dies führt zur Erzeugung einer entsprechenden Route) sowie die Benachrichtigung der Knoten auf der gewählten Route. Auch für QMRP wäre eine Simulation mit realistischen Medien wünschenswert; jedoch wurde das Protokoll nicht in SDL spezifiziert, sondern die funktionale Simulation wurde in C++ implementiert. Somit müsste entweder eine SDL-Spezifikation oder ein QMRP-Modul für ns-3 erstellt werden.

Für die Simulationskomponente ns-3 wäre es lohnend, weitere Kommunikationstechnologien von ns-3 aus FERAL heraus verfügbar zu machen und mehr Möglichkeiten für den Aufbau des Protokollstapels auf ns-3-Seite vorzusehen (z. B. die Verwendung von Routing-Protokollen, die im Simulator ns-3 bereits als Module vorhanden sind).

A Anhang A.

Paketformate von RBBQR

Im Folgenden werden die von RBBQR verwendeten Paketformate beschrieben. Dabei werden sowohl die intern verwendeten Paketformate als auch die Schnittstellen von RBBQR zur Anwendung und zur MAC-Schicht dargestellt. Die internen Paketformate werden lediglich aufgelistet, da sie in Kapitel 5 ausführlich behandelt werden.

A.1. Intern verwendete Paketformate

Die internen Paketformate unterteilen sich in solche zum Etablieren einer Route (Phasen 1 bis 3 sowie zur Übermittlung des Routenstatus), zur Datenübertragung und zur Verwaltung.

Phase 1

- Routenanforderung (Route Request Arbitration und Route Request Data)

  RREQA(QuellID : NodeId)

  Übertragung per Arbitrating Transfer mit $n_{hops} = n_{maxHops}$

  RREQD(ZielID : NodeId, QHopCount : HopCnt,
  QoSMetrik : QosMetric, QoSWert : QosVal)

  Übertragung per Cooperative Transfer mit $n_{hops} = 1$; $n_{maxHops}$ Mal wiederholt; Black-Burst-Reichweite wird auf die Kommunikationsreichweite eingeschränkt

Phase 2

- Antwort auf Routenanforderung (Preliminary Arbitration sowie Route Reply Arbitration und Route Reply Data)

  PreArb(RouteFound : Bit) *(vorgeschaltete Arbitrierung)*

  Übertragung per Arbitrating Transfer mit $n_{hops} = n_{maxHops}$

  RREPA(QHopCountInv : HopCnt, SenderID : NodeId) *(nur zur Arbitrierung)*

  Übertragung per Arbitrating Transfer mit $n_{hops} = 2$

  RREPD(SenderID : NodeId, SenderRouteID : RouteId,
  RouteLenZiel : RouteLen, MaxRouteLength : RouteLen,
  NumberOfSlots(TX2HopNachfolger) : SlotCnt, TXSlots(2HopNachfolger) : SlotSet,
  NumberOfSlots(TX1HopNachfolger) : SlotCnt, TXSlots(1HopNachfolger) : SlotSet,
  NumberOfSlots(TXSender) : SlotCnt, TXSlots(Sender) : SlotSet,
  NumberOfSlots(RXSender) : SlotCnt, RXSlots(Sender) : SlotSet,
  LowestUsedSlot : SlotNr)

  Übertragung als reguläres Paket

Phase 3

- Festlegung einer Route (Construction Request Information und Construction Request Data)

  CREQI(SenderID : NodeId, EmpfängerID : NodeId,
  TXSlots(Vorgänger) : SlotSet, TXSlots(Sender) : SlotSet)

  Übertragung per Bit-based Transfer

  CREQD(SenderRouteID : RouteId, EmpfängerRouteID : RouteId,
  RouteLenQuelle : RouteLen, TXSlots(1HopNachfolger) : SlotSet,
  TXSlots(2HopNachfolger) : SlotSet, TXSlots(3HopNachfolger) : SlotSet)

  Übertragung als reguläres Paket

- Fehlreservierung bei den Empfangsslots aufgetreten (CREQ Collision)

  CREQC(EmpfängerID : NodeId,
  NumberOfSlots(CollSender) : SlotCnt, CollSlots(Sender) : SlotSet,
  NumberOfSlots(RXSender) : SlotCnt, RXSlots(Sender) : SlotSet)

  Übertragung als reguläres Paket

- Update der gewählten Sendeslots aufgrund von Fehlreservierung (CREQ Update)

  CREQU(SenderID : NodeId,
  NumberOfSlots(NewTXSender) : SlotCnt, NewTXSlots(Sender) : SlotSet)

  Übertragung per Bit-based Transfer

- Routensuche gescheitert (CREQ Failure) (Fehlreservierung aufgetreten und zu wenige alternative Slots verfügbar)

  CREQF(EmpfängerID : NodeId)

  Übertragung als reguläres Paket

Status der Routensuche

- Routenstatus (Route Status): Route gefunden (Route Found), Routensuche erfolgreich beendet (Finish Success), Routensuche erfolglos beendet (Finish Failure)

  RSTAT(Type : RType) mit RType = {RFND, FIN_SUCC, FIN_FAIL}

  Übertragung per Cooperative Transfer mit $n_{hops} = n_{maxHops}$

Datenübertragung

- Senden bzw. Empfangen von Daten

  DATA(EmpfängerID : NodeId, EmpfängerRouteID : RouteId, DataValue : DataVal)

  Übertragung als reguläres Paket

Verwaltung

- Propagierung der Sende- und Empfangsreservierungen eines Knotens (Slot Count sowie Slot Propagation Arbitration und Slot Propagation Data)

  SCOUNT(MaxNumberOfReservedSlots : SlotCnt)

  Übertragung per Arbitrating Transfer mit $n_{hops} = n_{maxHops}$

SPROPA(SenderID : NodeId) *(nur zur Arbitrierung)*

Übertragung per Arbitrating Transfer mit $n_{hops} = 2$

SPROPD(SenderID : NodeId,
NumberOfSlots(TXSender) : SlotCnt, TXSlots(Sender) : SlotSet,
NumberOfSlots(RXSender) : SlotCnt, RXSlots(Sender) : SlotSet)

Übertragung per Bit-based Transfer

- Propagierung eines Routenbruchs (Route Break Arbitration und Route Break Data)

  RBRKA(SenderID : NodeId) *(nur zur Arbitrierung)*

  Übertragung per Arbitrating Transfer mit $n_{hops} = 2$

  RBRKD(VorgängerID : NodeId, VorgängerRouteID : RouteId)

  Übertragung als reguläres Paket

- Freigabe einer Route (Route Delete)

  RDEL(EmpfängerID : NodeId, EmpfängerRouteID : RouteId)

  Übertragung als reguläres Paket (innerhalb eines Datenslots)

A.2. Schnittstelle zur Anwendung

Die Anwendung kann zwar zu einem gewissen Grad von RBBQR und der darunterliegenden MAC-Schicht entkoppelt werden; eine vollständige Entkopplung ist jedoch nicht möglich. Um eine ausreichende Anzahl an Slots für eine Route reservieren zu können, muss die Anwendung die Dauer eines Superslots sowie die Größe des Typs DataVal kennen. Die Größe des Typs DataVal muss auch deshalb bekannt sein, damit die Anwendung die Daten entsprechend fragmentieren kann. Im Normalfall wird jede Anwendung pro Ziel nur eine Route mit einer ausreichenden Slotanzahl reservieren. Werden jedoch zu einem späteren Zeitpunkt mehr Slots benötigt (oder kommt eine Route mit größerer Slotanzahl nicht zustande), so können weitere Routen zum Senden an denselben Zielknoten angefordert werden. Um diese Routen zu verwalten, werden die RouteIDs von RBBQR an die Anwendung übergeben. Soll ein Datenpaket versendet werden, so übergibt die Anwendung dieses zusammen mit der entsprechenden RouteID an RBBQR. Das Ziel ist durch die RouteID eindeutig festgelegt. Das Datenpaket wird in dem zu der entsprechenden Route gehörenden Puffer abgelegt (für jede Route existiert ein solcher Puffer, aus dem die Pakete in FIFO-Reihenfolge versendet werden).

A.2.1. Schnittstelle von der Anwendung zu RBBQR

- Routenanforderung der Anwendung

  APP_RREQ(ZielID : NodeId, QoSMetrik : QosMetric, QoSWert : QosVal)

- Explizites Freigeben einer Route

  APP_FreeRoute(RouteID : RouteId)

- Versenden von Daten über eine existierende Route

  APP_SendData(RouteID : RouteId, DataValue : DataVal)

A.2.2. Schnittstelle von RBBQR zur Anwendung

- Route wurde etabliert *(Antwort auf APP_RREQ)*

  APP_RouteEstablished(RouteID : RouteId, ZielID : NodeId,
  QoSMetrik : QosMetric, QoSWert : QosVal)

  Die Anwendung legt eine Tabelle an, um jede RouteID dem entsprechenden Ziel sowie der QoS-Anforderung (QoS-Metrik und QoS-Wert) zuordnen zu können.

- Route konnte nicht etabliert werden *(Antwort auf APP_RREQ)*

  APP_RouteFailure(ZielID : NodeId, QoSMetrik : QosMetric, QoSWert : QosVal,
  Fehler : RError)
  mit RError = {NO_ROUTE_FOUND, ROUTE_NOT_ESTABLISHED}

  Der Fehler NO_ROUTE_FOUND tritt auf, wenn in Phase 2 keine Route zwischen Quelle und Ziel ermittelt werden konnte, welche die QoS-Anforderung erfüllt.

  ROUTE_NOT_ESTABLISHED zeigt an, dass zwar in Phase 2 eine Route gefunden wurde, diese aber in Phase 3 nicht etabliert werden konnte (weil eine Fehlreservierung aufgrund von Selbstinterferenz aufgetreten ist und als Ersatz für die fehlreservierten Slots nicht mehr genügend freie Slots zur Verfügung standen).

- Route wurde gelöscht

  APP_RouteDeleted(RouteID : RouteId)

  Eine Route kann gelöscht werden, weil zu lange keine Daten darüber versendet wurden oder weil sie gebrochen ist (durch Bruch eines Links oder Ausfall eines Knotens).

  Nach Erhalt von APP_RouteEstablished kennt die Anwendung die Zuordnung der RouteID zu ZielID, QoS-Metrik und QoS-Wert und kann den entsprechenden Tabelleneintrag entfernen.

- Fehler beim Puffern von Daten *(Antwort auf APP_SendData)*

  APP_DataFailure(Fehler : DError)
  mit DError = {BUFFER_OVERFLOW, ROUTE_INVALID}

  BUFFER_OVERFLOW bezeichnet einen Pufferüberlauf beim Zwischenspeichern eines zu versendenden Datums.

  ROUTE_INVALID zeigt an, dass die entsprechende Route nicht existiert.

- Empfang von Daten

  APP_RcvData(QuellID : NodeId, DataValue : DataVal)

A.3. Schnittstelle zur MAC-Schicht

Die Schnittstelle zur MAC-Schicht besteht aus zwei Arten von Signalen. Dabei handelt es sich einerseits um Signale zum Versenden von RBBQR-Paketen über das Medium bzw. zum Empfangen solcher Pakete, andererseits um Signale zum Austausch von Informationen zwischen RBBQR und der MAC-Schicht.

A.3.1. Schnittstelle von RBBQR zur MAC-Schicht

Die Signale zum Paketversand beinhalten Übertragungen per Arbitrating Transfer, Cooperative Transfer, Bit-based Transfer sowie reguläre Übertragungen. Es wird keine Adressierung auf MAC-Ebene verwendet, da die Adressierung ausschließlich von RBBQR übernommen wird.

Die Signale zur Übergabe von Informationen an die MAC-Schicht werden zur korrekten Erzeugung der virtuellen Slots benötigt. Sie beinhalten ausschließlich Informationen, die netzweit zur Verfügung stehen. Damit ist gewährleistet, dass die Berechnung der virtuellen Slots (Typ und Dauer) in allen Knoten übereinstimmt.

Signale zum Paketversand

- Senden per Arbitrating Transfer

  SendArb(ArbSeq : Integer, ArbReichw : HopCnt,
  RbbqrPcktLen : Length, RbbqrPckt : RbbqrEnc, Reichweite : BBRange)

  ArbSeq gibt die Arbitrierungssequenz an. Diese ist zwar identisch zu RbbqrPckt, wird aber aus Gründen der einfacheren Implementierbarkeit hier nochmals als Integer übergeben.

  Weitere Parameter sind die Anzahl Hops, über die arbitriert werden soll, das zu übertragende Paket mitsamt seiner Länge sowie die Black-Burst-Reichweite. Letztere kann entweder auf die Kommunikationsreichweite eingeschränkt werden, oder sie entspricht der Interferenzreichweite. Die Paketlänge wird hier nur aus Gründen der einfacheren Implementierbarkeit übergeben, da sie beim Arbitrating Transfer im Voraus bekannt ist und somit vorab in der MAC-Schicht konfiguriert werden kann.

- Senden per Cooperative Transfer

  SendCoop(CoopSeq : CoopString, CoopReichw : HopCnt,
  RbbqrPcktLen : Length, RbbqrPckt : RbbqrEnc, Reichweite : BBRange)

  Das Paketformat ist analog zu SendArb. Auch hier werden die zu übertragende Sequenz (CoopSeq), welche identisch zu RbbqrPckt ist, sowie die Paketlänge nur aus Gründen der einfacheren Implementierbarkeit übergeben.

- Senden per Bit-based Transfer

  SendBitb(RbbqrPcktLen : Length, RbbqrPckt : RbbqrEnc)

  Im Gegensatz zum Arbitrating Transfer und Cooperative Transfer wird beim Bit-based Transfer die Paketlänge im MAC-Rahmen benötigt, da sie im Voraus nicht bekannt ist (vgl. Abschnitt 3.3.2.4).

- Senden mittels regulärer Übertragung

  SendReg(RbbqrPcktLen : Length, RbbqrPckt : RbbqrEnc)

  Bei regulären Übertragungen wird die Paketlänge ebenfalls im MAC-Rahmen benötigt (analog zum Bit-based Transfer).

Signale zur Übergabe von Informationen an die MAC-Schicht

- Anzahl benötigter Slots an MAC-Schicht mitteilen

  neededSlots(NumberOfSlots : SlotCnt)

Dieses Signal dient dazu, der MAC-Schicht in Phase 1 (nach netzweiter Auswahl einer Routenanforderung) die Anzahl der angeforderten Slots mitzuteilen. Die MAC-Schicht berechnet daraus die Dauer der virtuellen Slots für Phase 3.

- Anzahl maximal zum Senden bzw. zum Empfangen reservierter Slots an MAC-Schicht mitteilen (dabei wird implizit mitgeteilt, dass ein SCOUNT-Paket empfangen wurde)

  scountSlots(NumberOfSlots : SlotCnt)

  Dieses Signal wird nach Empfang von SCOUNT an die MAC-Schicht gesendet, damit diese die Dauer des SPROPD-Slots berechnen kann. Wenn kein SCOUNT empfangen wurde, entfallen die virtuellen Slots für SPROPA und SPROPD. Dies ist erforderlich, da die Dauer des SPROPD-Slots ansonsten unbekannt wäre.

- Beginn von Phase 3 bzw. Phase 1 der nächsten Routensuche an MAC-Schicht mitteilen

  startPh3, **startPh1**

  Da Phase 2 keine konstante Dauer hat, muss der Beginn von Phase 3 mittels eines Signals mitgeteilt werden. Die MAC-Schicht generiert dann entsprechende virtuelle Slots für Phase 3. Ebenso wird nach Abschluss einer Routensuche der Beginn der nächsten Phase 1 signalisiert. Da Phase 1 für ein gegebenes Netz eine konstante Dauer hat, wird für Phase 2 kein solches Signal benötigt.

- MAC-Schicht mitteilen, dass ein RREQA empfangen wurde

  rreqaRec

  Solange kein RREQA empfangen wurde, erzeugt die MAC-Schicht in Phase 1 stets RREQA-Slots. Sobald ein RREQA empfangen wurde, wird dies der MAC-Schicht mitgeteilt. Der nächste virtuelle Slot ist dann ein RREQD-Slot.

- MAC-Schicht mitteilen, dass im nächsten Slot ein RSTAT gesendet wird

  reqRstatSlot

  Solange die Routensuche noch nicht beendet ist, erzeugt die MAC-Schicht in Phase 2 abwechselnd PreArb-, RREPA- und RREPD-Slots. Wird anhand der vorgeschalteten Arbitrierung (mittels PreArb) erkannt, dass als Nächstes ein RSTAT gesendet werden soll, so wird der MAC-Schicht mitgeteilt, dass ein virtueller Slot für RSTAT generiert werden soll.

A.3.2. Schnittstelle von der MAC-Schicht zu RBBQR

Die Signale zum Paketempfang werden von der MAC-Schicht verwendet, um Pakete, die per Arbitrating Transfer, Cooperative Transfer, Bit-based Transfer oder mittels regulärer Übertragungen empfangen wurden, an RBBQR weiterzuleiten.

Die Signale zur Übergabe von Informationen an RBBQR werden benötigt, um RBBQR den Beginn eines Superslots bzw. der entsprechenden virtuellen Slots mitzuteilen.

Signale zum Paketempfang

- Empfang per Arbitrating Transfer

  RcvArb(RbbqrPckt : RbbqrEnc)

- Empfang per Cooperative Transfer

  RcvCoop(RbbqrPckt : RbbqrEnc)

- Empfang per Bit-based Transfer

  RcvBitb(RbbqrPckt : RbbqrEnc, Reichweite : BBRange)

  Die MAC-Schicht ermittelt, ob sich der Sender in Kommunikationsreichweite befindet, und gibt dies im Parameter Reichweite an. In einer realen Implementierung kann dies anhand der Empfangssignalstärke der einzelnen Black Bursts erfolgen. Wenn sich die Werte für die einzelnen Bits eines Pakets unterscheiden, muss davon ausgegangen werden, dass der Sender sich nicht in Kommunikationsreichweite befindet. Da sich beim Bit-based Transfer jeder Empfänger nur in Reichweite eines Senders befinden darf, kann es nicht vorkommen, dass der Wert durch den gleichzeitigen Empfang des Signals mehrerer Sender verfälscht wird.

- Empfang mittels regulärer Übertragung

  RcvReg(RbbqrPckt : RbbqrEnc)

Signale zur Übergabe von Informationen an RBBQR

- Beginn eines Superslots an RBBQR mitteilen

  beginOfSuperslot

  RBBQR benötigt den Zeitpunkt, zu dem ein Superslot beginnt, um die Timer zum Versenden des nächsten SPROP-Pakets sowie für die Einträge der Routen- und Slottabelle relativ dazu stellen zu können. Diese laufen immer zum Ende eines Superslots aus (vgl. Abschnitt 5.4.2 sowie Anhänge B.3 und B.4).

- Beginn eines virtuellen Slots (bzw. einer virtuellen Slotregion im Fall eines Datenslots) an RBBQR mitteilen

  rreqaSlot, **rreqdSlot**, **rrepaPreSlot**, **rrepaSlot**, **rstatSlot**, **rrepdSlot**, **creqiSlot**, **creqdSlot**, **scountSlot**, **spropaSlot**, **spropdSlot**, **rbrkaSlot**, **rbrkdSlot**, **dataSlot**(Slot : SlotNr)

  Die MAC-Schicht erzeugt virtuelle Slots für die von RBBQR versendeten Kontrollpakete, deren Beginn jeweils mit einem eigenen Signal mitgeteilt wird. Bei den Datenslots handelt es sich um virtuelle Slotregionen mit konstanter Dauer, deren Beginn ebenfalls signalisiert wird.

B Anhang B.

Tabellenformate von RBBQR

RBBQR legt Verwaltungsinformationen für Routen und Slotreservierungen in verschiedenen Tabellen ab, welche jeder Knoten separat vorhält und verwaltet. Diese werden in den folgenden Abschnitten beschrieben.

Daneben muss jeder Knoten während der Routensuche seine Hopdistanz zur Quelle (QHopCount) aufzeichnen. Diese wird in Phase 1 ermittelt und in Phase 2 zur Arbitrierung mittels RREPA benötigt. Zudem zeichnet jeder Knoten in Phase 2 seine aktuelle ($\hat{=}$ minimale) Routenlänge zum Ziel (CurRouteLen) auf, um nachfolgende RREPDs mit einer größeren Routenlänge verwerfen zu können. Dies dient zur Vermeidung von Routingschleifen. Die minimale Routenlänge wird anhand des ersten erhaltenen RREPDs ermittelt und kann größer als die Hopdistanz des Knotens zum Ziel sein. Weitere RREPDs können eine größere Routenlänge haben, jedoch keine kleinere (vgl. Abschnitt 5.2.2.4). Sowohl QHopCount als auch CurRouteLen werden nach Abschluss der Routensuche nicht mehr benötigt, deshalb müssen die Werte nicht in die permanente Routentabelle eingetragen werden.

B.1. Temporäre Zieltabelle

Die temporäre Zieltabelle wird benötigt, um Routenanforderungen der Anwendung aufzuzeichnen, bis diese bearbeitet sind (d. h. bis entweder eine Route gefunden wurde oder die Routensuche gescheitert ist). Sie enthält folgende Spalten:

- ZielID : NodeId
- QoSMetrik : QosMetric
- QoSWert : QosVal

Beim Start von Phase 1 prüft jeder Knoten, ob seine temporäre Zieltabelle mindestens einen Eintrag enthält. Ist dies der Fall, so nimmt der Knoten mit einem RREPA-Paket an der Arbitrierung teil. Wurde die Arbitrierung gewonnen, so wird eine Routenanforderung aus der Tabelle ausgewählt und mittels RREPD-Paketen im gesamten Netz propagiert.

Der zugehörige Eintrag wird aus der temporären Zieltabelle entfernt, sobald die Routensuche abgeschlossen wurde (erfolgreich oder erfolglos). Der Empfang eines RSTAT(FIN_SUCC)-Pakets kennzeichnet ein erfolgreiches Ende der Routensuche. In diesem Fall wird die RouteID der gefundenen Route der Anwendung mitgeteilt (in einem APP_RouteEstablished-Paket; vgl. Anhang A.2.2), zusammen mit den Parametern aus dem gerade bearbeiteten Eintrag der temporären Zieltabelle. Für die Route wurde dann bereits ein Eintrag in der permanenten Routentabelle generiert. Für ein erfolgloses Ende gibt es verschiedene Möglichkeiten. Kann die QoS-Anforderung bereits im Ziel nicht erfüllt werden, so sendet dieses ein RSTAT(FIN_FAIL)-Paket, welches von der Quelle in Phase 2 empfangen wird. Wenn das Ziel

zwar die QoS-Anforderung erfüllen kann, es aber keine entsprechende Route zwischen Quelle und Ziel gibt, so läuft die Quelle in Phase 2 in einen Timeout und sendet anschließend ein RSTAT(FIN_FAIL)-Paket. Wird eine Route gefunden, welche dann aufgrund einer nicht korrigierbaren Fehlreservierung durch Selbstinterferenz nicht etabliert werden kann, so empfängt die Quelle in Phase 3 ein CREQF-Paket und sendet ebenfalls ein RSTAT(FIN_FAIL). Bei einem erfolglosen Ende der Routensuche wird eine Fehlermeldung an die Anwendung geschickt (APP_RouteFailure-Paket; vgl. Anhang A.2.2).

B.2. Temporäre Routentabelle

Die temporäre Routentabelle wird in Phase 2 erstellt und zeichnet die gefundenen (partiellen) Routen zum Zielknoten auf, bis von der Quelle eine Route ausgewählt und diese im Netz etabliert wurde. Sie enthält die folgenden Spalten:

- RouteID : RouteId
- NachfolgerRouteID : RouteId
- NachfolgerID : NodeId
- Slots : SlotSetList
- F_{RX}Nachfolger : SlotSet

In der Tabelle wird die eigene RouteID zusammen mit der RouteID sowie der ID des Nachfolgers abgespeichert (da die Route vom Ziel zur Quelle ermittelt und von der Quelle zum Ziel etabliert wird).

Der Slots-Eintrag enthält eine Liste von Slotmengen. Handelt es sich um einen Zwischenknoten, so enthält diese Liste i. d. R. genau eine Slotmenge, nämlich die Sendeslots für den 3-Hop-Nachfolger (dies ist für alle Zwischenknoten mit Ausnahme der 1-, 2- und 3-Hop-Vorgänger des Ziels der Fall; für diese ist die Liste leer). Handelt es sich um die Quelle, so sind zusätzlich noch die Sendeslots für den 2-Hop-Nachfolger, den 1-Hop-Nachfolger und die eigenen Sendeslots enthalten.

Der F_{RX}Nachfolger-Eintrag enthält die Menge der beim Nachfolger zum Empfangen freien Slots. Im Normalfall wird diese Information (die im RREPD-Paket propagiert wird) nur einmalig benötigt, um in Phase 2 die freien Sendeslots für den Link zum Nachfolger zu berechnen. Tritt jedoch in Phase 3 eine Fehlreservierung durch Selbstinterferenz bei den gewählten Sendeslots auf, so müssen die freien Sendeslots für den Link erneut berechnet werden, wozu die freien Empfangsslots des Nachfolgers bekannt sein müssen.

Nach Festlegung der Route (Quelle) bzw. nach Erhalt eines CREQD-Pakets (Zwischen- und Zielknoten) wird der Eintrag für die ausgewählte Route in die permanente Routentabelle übernommen. Bei der Quelle enthält die temporäre Routentabelle nur einen Eintrag, da immer die erste gefundene Route gewählt wird. Beim Ziel ist ebenfalls nur ein Eintrag enthalten, da dieses nur ein RREPD versendet. Die temporäre Routentabelle wird nach Abschluss der Routensuche in allen Knoten gelöscht.

Beispiele

Die folgenden Beispiele zeigen exemplarische Routensuchen sowie die zugehörigen Einträge der temporären Routentabellen der an der Route beteiligten Knoten.

Die normalen Pfeile bezeichnen die gefundene(n) Route(n) und die unterbrochenen bezeichnen die inverse(n) Route(n), auf der (bzw. auf denen) die RREPD-Pakete propagiert werden. Die RouteIDs wurden zufällig vergeben.

Beispiel 1

Das erste Beispiel ist in Abbildung B.1 dargestellt. Bei diesem leitet jeder Knoten nur ein RREPD weiter und hat somit auch nur einen Eintrag in der temporären Routentabelle.

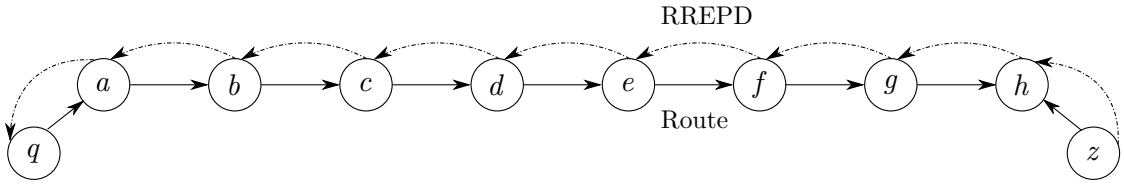

Abbildung B.1.: Erstes Beispiel zur Routensuche.

Aus Gründen der Übersichtlichkeit sind alle Einträge der temporären Routentabellen in Tabelle B.1 zusammengefasst. Die zusätzliche erste Spalte bezeichnet die ID des jeweiligen Knotens.

ID	**RouteID**	**Nachfolger**		**Slots**	$\boldsymbol{F_{RX}}$**Nachfolger**
		RouteID	**ID**		
z	5	–	–	–	–
h	10	5	z	–	$F_{RX}(z)$
g	7	10	h	–	$F_{RX}(h)$
f	4	7	g	–	$F_{RX}(g)$
e	1	4	f	$[LK(h,z)]$	$F_{RX}(f)$
d	120	1	e	$[LK(g,h)]$	$F_{RX}(e)$
c	3	120	d	$[LK(f,g)]$	$F_{RX}(d)$
b	8	3	c	$[LK(e,f)]$	$F_{RX}(c)$
a	1	8	b	$[LK(d,e)]$	$F_{RX}(b)$
q	7	1	a	$[LK(c,d), LK(b,c), LK(a,b), LK(q,a)]$	$F_{RX}(a)$

Tabelle B.1.: Einträge der temporären Routentabellen für erstes Beispiel.

Beispiel 2

Das zweite Beispiel ist in Abbildung B.2 dargestellt. Bei diesem gibt es zwei hop-minimale Routen zum Ziel. Da die Routensuche jedoch beendet ist, sobald die Quelle das erste RREPD erhalten hat, wird die zweite dieser Routen nicht bis zur Quelle propagiert.

Für die IDs gelte $q > a > b > c > d > e > f > g > z$. Zudem gelte $\forall\, a \in V : SN_1(a) = IN_1(a) = CN_1(a) \cup CN_2(a)$, d. h. die Arbitrierung erfolgt über vier Kommunikationshops

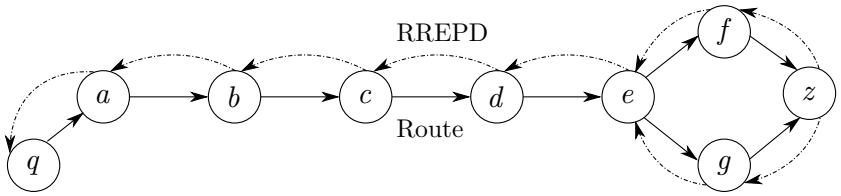

Abbildung B.2.: Zweites Beispiel zur Routensuche.

(in Abbildung B.2 sind nur die Kommunikationslinks eingezeichnet). Für die Arbitrierung mittels RREPA wird QHopCount(Inv) sowie die SenderID verwendet (vgl. Abschnitt 5.2.2.3). Zunächst sendet z sein RREPD. Im nächsten Schritt findet eine Arbitrierung zwischen f und g statt, die f aufgrund der größeren ID gewinnt (der QHopCount ist gleich). Da e einen geringeren QHopCount hat als g, gewinnt e die anschließende Arbitrierung. Analog gewinnen d, c und b gegen g. Danach können a und g parallel senden. Da e zu diesem Zeitpunkt nicht weiß, ob die erste gefundene Route bis zur Quelle propagiert werden kann, wird die neue Route ebenfalls aufgezeichnet, so dass die temporäre Routentabelle von e nun zwei Einträge enthält. Wenn q das RREPD-Paket von a empfängt, wird diese Route gewählt und es wird per RSTAT(RFND) bekannt gegeben, dass eine Route gefunden wurde. Daher senden e, d, c, b und a kein weiteres RREPD.

Aus Gründen der Übersichtlichkeit sind alle Einträge der temporären Routentabellen in Tabelle B.2 zusammengefasst. Die zusätzliche erste Spalte bezeichnet die ID des jeweiligen Knotens.

ID	**RouteID**	**Nachfolger**		**Slots**	$\boldsymbol{F_{RX}}$**Nachfolger**
		RouteID	**ID**		
z	4	–	–	–	–
f	11	4	z	–	$F_{RX}(z)$
g	37	4	z	–	$F_{RX}(z)$
e	8	11	f	–	$F_{RX}(f)$
	9	37	g	–	$F_{RX}(g)$
d	1	8	e	–	$F_{RX}(e)$
c	5	1	d	$[LK(f,z)]$	$F_{RX}(d)$
b	3	5	c	$[LK(e,f)]$	$F_{RX}(c)$
a	12	3	b	$[LK(d,e)]$	$F_{RX}(b)$
q	89	12	a	$[LK(c,d), LK(b,c),$ $LK(a,b), LK(q,a)]$	$F_{RX}(a)$

Tabelle B.2.: Einträge der temporären Routentabellen für zweites Beispiel.

B.3. Permanente Routentabelle

Die permanente Routentabelle eines Knotens zeichnet die etablierten Routen auf, die durch diesen führen (unabhängig davon, ob der Knoten Quelle, Zwischenknoten oder Ziel einer Route ist). Sie enthält die folgenden Spalten:

- RouteID : RouteId
- QuellID : NodeId
- RouteLenQuelle : RouteLen
- NachfolgerRouteID : RouteId
- NachfolgerID : NodeId
- VorgängerRouteID : RouteId
- VorgängerID : NodeId
- Sendeslots : SlotSet
- Empfangsslots : SlotSet
- Datenpuffer : DataBuffer

Die ID der Quelle wird benötigt, damit der Zielknoten diese zusammen mit den empfangenen Daten an die Anwendung weiterleiten kann.

RouteLenQuelle (in Phase 3 ermittelt) bezeichnet die Länge der Route vom aktuellen Knoten bis zur Quelle und wird für den zur Route gehörenden Timer benötigt (s. u.). RouteLenQuelle kann länger als die (in Phase 1 ermittelte) Hopdistanz des Knotens zur Quelle sein (dies ist der Fall, wenn keine hop-minimale Route zustande kommt).

Während ID und RouteID des Nachfolgers zum Weiterleiten von Daten entlang der Route dienen, werden ID und RouteID des Vorgängers benötigt, um Fehlermeldungen (z. B. bei einem Routenbruch) zur Quelle weiterleiten zu können. Es ist jedoch zu beachten, dass dazu nicht die zum Senden an den Routennachfolger reservierten Slots genutzt werden können, da der Vorgänger in diesen möglicherweise nicht empfangen kann.

Die Sende- und Empfangsslots sind zwar bereits in der Slottabelle (s. Anhang B.4) enthalten, jedoch gibt es dort keine Korrelation zu der zugehörigen RouteID. Diese wird einerseits für das Versenden von Daten benötigt, andererseits auch deshalb, da beim Entfernen einer Route die zugehörigen Sende- und Empfangsreservierungen freigegeben werden müssen.

Der Datenpuffer hat eine global festgelegte Maximalgröße. Seine Einträge enthalten jeweils ein Datum vom Typ DataVal, welches innerhalb eines Slots versendet werden kann. Will die auf dem Quellknoten laufende Anwendung Daten versenden, so werden diese in den Puffer für die entsprechende Route eingetragen. Ist der Puffer voll, so erfolgt eine Fehlermeldung. In den Zwischenknoten kann der Puffer verwendet werden, um empfangene Daten abzulegen, bis sie an den nächsten Knoten der Route gesendet werden können.

Zum Versenden von Daten prüft jeder Knoten in jedem Datenslot, ob es sich um einen Sendeslot handelt. Ist dies der Fall, so wird der zugehörige Eintrag der permanenten Routentabelle ermittelt, das erste Datum aus dem zugehörigen Puffer genommen und (zusammen mit ID und RouteID des Nachfolgers) als DATA-Paket versendet.

Entfernung der Einträge

Die Entfernung eines Eintrags kann einerseits explizit erfolgen, indem eine Route von der Anwendung (Quelle) freigegeben wird. Es wird dann ein RDEL-Paket an den nächsten Knoten gesendet, welches bis zum Ziel propagiert wird. Dabei ist darauf zu achten, dass der Eintrag erst entfernt werden darf, nachdem das RDEL-Paket gesendet (Quelle) bzw. weitergeleitet wurde (Zwischenknoten).

Darüber hinaus kann eine Route auch implizit durch einen Timeout entfernt werden. Um Timeouts zu realisieren, assoziiert jeder Knoten einen RouteTimer mit jeder vergebenen RouteID. Dessen Ablaufzeitpunkt berechnet sich wie folgt:

$$\text{RouteTimer}(\text{RouteID}) =_{Df} t_{superslotBegin} + (1 + n_{waitSuperslots} + \text{RouteLenQuelle}) \cdot d_{superslot}\,.$$

Dabei bezeichnet $t_{superslotBegin}$ den Anfang des aktuellen Superslots und $d_{superslot}$ die Dauer eines Superslots. $n_{waitSuperslots} \geq 1$ ist die Anzahl der bis zur Freigabe einer Route abzuwartenden Superslots (zusätzlich zu dem aktuellen Superslot). RouteTimer laufen immer zum Ende eines Superslots ab, so dass die freigegebenen Datenslots der Route im nächsten Superslot wieder für eine neue Routensuche verwendet werden können.

Ein RouteTimer wird gestartet, sobald die zugehörige Route etabliert ist. Für die Quelle wird er neu gestartet, sobald die Anwendung Daten über diese Route versendet. Für die Zwischenknoten und das Ziel erfolgt der Neustart, sobald ein DATA-Paket über die Route empfangen wird (erkennbar an der darin enthaltenen RouteID).

Der Term $\text{RouteLenQuelle} \cdot d_{superslot}$ wird benötigt, da anderenfalls eine Route in einem Knoten in einen Timeout laufen könnte, während sie im Vorgänger noch gültig ist, da dort gerade noch rechtzeitig ein Datenpaket ankommt. Das folgende Beispiel verdeutlicht dies.

Es wird eine Route betrachtet, die neben Quelle und Ziel nur einen Zwischenknoten enthält. Lässt man $\text{RouteLenQuelle} \cdot d_{superslot}$ weg und wählt $n_{waitSuperslots} = 1$, so haben die RouteTimer aller Knoten den Wert $t_{superslotBegin} + 2 \cdot d_{superslot}$. Es werden die in Tabelle B.3 aufgelisteten Sendereservierungen (S) und Empfangsreservierungen (E) verwendet.

Slot	1	2	3	4
Quelle	S			S
Zwischenknoten	E	S	S	E
Ziel		E	E	

Tabelle B.3.: Beispiel für Sende- und Empfangsreservierungen.

Sendet die Quelle nun im ersten Superslot nur in Slot 1 Daten, so können diese ebenfalls im ersten Superslot vom Zwischenknoten weitergeleitet werden. Anschließend sind alle RouteTimer auf das Ende des zweiten Superslots gesetzt. Sendet die Quelle im zweiten Superslot erst in Slot 4 Daten, so werden die RouteTimer für Quelle und Zwischenknoten auf das Ende des dritten Superslots gesetzt. Der Zwischenknoten kann jedoch im zweiten Superslot seine Daten nicht mehr weiterleiten, so dass der RouteTimer im Zielknoten zum Ende des zweiten Superslots abläuft.

B.4. Slottabelle

Die Slottabelle eines Knotens a zeichnet Informationen über den Reservierungs- und Blockierungszustand der einzelnen Slots auf. Sie enthält die folgenden Spalten:

- Interferenznachbar : NodeId
- IsInCommRange : Boolean
- TX : SlotSet
- RX : SlotSet
- $B_{TX,I}$: SlotSet
- $B_{RX,I}$: SlotSet

In der Tabelle wird für jeden Interferenznachbarn b eingetragen, ob sich dieser in Kommunikationsreichweite zu a befindet (IsInCommRange) und in welchen Slots a durch b zum Senden ($B_{TX,I}$) bzw. zum Empfangen ($B_{RX,I}$) blockiert ist. Ist b ein Kommunikationsnachbar, so wird zusätzlich eingetragen, in welchen Slots an diesen gesendet (TX) bzw. von diesem empfangen wird (RX). Im Gegensatz zu den Sende- und Empfangsreservierungen kann es bei den Blockierungen für einen Slot mehrere verursachenden Knoten gleichzeitig geben. Es ist auch beispielsweise möglich, dass ein Slot gleichzeitig zum Senden reserviert und zum Empfangen blockiert ist.

Anmerkung: Da ein Knoten nur in seinen Sende- und Empfangsslots aktiv sein muss, könnte die Kenntnis über diese zusätzlich für Duty Cycling genutzt werden. Dabei handelt es sich um einen Mechanismus zur Energieeinsparung, bei dem der Transceiver ausgeschaltet wird, wenn er gerade nicht benötigt wird. Hier wird jedoch kein Duty Cycling umgesetzt.

Anmerkung: Die Information über die zum Senden und zum Empfangen reservierten Slots ist bereits in der permanenten Routentabelle enthalten (vgl. Anhang B.3), jedoch wurde sie mit in die Slottabelle aufgenommen, um die gesamte Slotinformation an einer Stelle zur Verfügung zu haben. Dies ist beispielsweise zum Versenden der reservierten Slots mittels SPROPD sinnvoll.

Entfernung der Einträge

Sobald eine Route in einem Knoten freigegeben wird (s. Anhang B.3), werden die zugehörigen Sende- und Empfangsslots aus der TX-Slotmenge für den Nachfolger bzw. aus der RX-Slotmenge für den Vorgänger entfernt. Ein durch einen Interferenznachbarn b blockierter Slot s wird aus der entsprechenden $B_{TX,I}$- bzw. $B_{RX,I}$-Menge entfernt, sobald ein SPROPD-Paket von b empfangen wird, in dem s nicht mehr als Empfangs- bzw. als Sendeslot aufgelistet ist (vgl. Abschnitt 5.4.2).

Um einen Eintrag der Slottabelle entfernen zu können, wenn der zugehörige Interferenznachbar die Interferenzreichweite verlassen hat, assoziiert jeder Knoten einen SlotTabTimer mit jedem Interferenznachbarn b, für den ein Eintrag in der Slottabelle existiert. Dessen Ablaufzeitpunkt berechnet sich wie folgt:

$$\text{SlotTabTimer}(b) =_{Df} t_{superslotBegin} + (1 + n_{maxCompNodes}) \cdot d_{superslot}\,.$$

Dabei bezeichnet $n_{maxCompNodes}$ die Anzahl an Superslots, innerhalb derer jeder Knoten einmal sein SPROPD-Paket versendet (vgl. Abschnitt 5.4.2). Analog zu den RouteTimern laufen auch SlotTabTimer immer zum Ende eines Superslots ab. Läuft SlotTabTimer(b) ab, so müssen alle Routen, die b als Nachfolger haben, freigegeben werden (s. Abschnitt 5.4.3).

Ein SlotTabTimer wird gestartet, sobald für den zugehörigen Interferenznachbarn ein Eintrag in der Slottabelle erzeugt wurde. Das Erzeugen eines solchen Eintrags erfolgt durch das erste empfangene SPROPD- oder CREQI-Paket von diesem Knoten. Durch den Empfang eines SPROPD-Pakets wird der SlotTabTimer für den Sender neu gestartet.

Anhang C.

C SDL-Spezifikation von RBBQR

Das Protokoll RBBQR wurde mittels SDL (s. Abschnitt 3.2) formal spezifiziert und mit dem internen Simulator der Rational SDL Suite [IBM15] simuliert. Der Aufbau eines entsprechenden Simulationssystems ist in Abschnitt 5.7.1 beschrieben. In diesem Anhang werden einige der Services, in die der RBBQR-Prozess unterteilt ist, detaillierter betrachtet.

C.1. Service *TimerMgmt*

Der Service *TimerMgmt* ist für das (Zurück)Setzen von Timern sowie die Behandlung abgelaufener Timer zuständig. Abbildung C.1 zeigt die Behandlung eines abgelaufenen SlotTab-Timers.

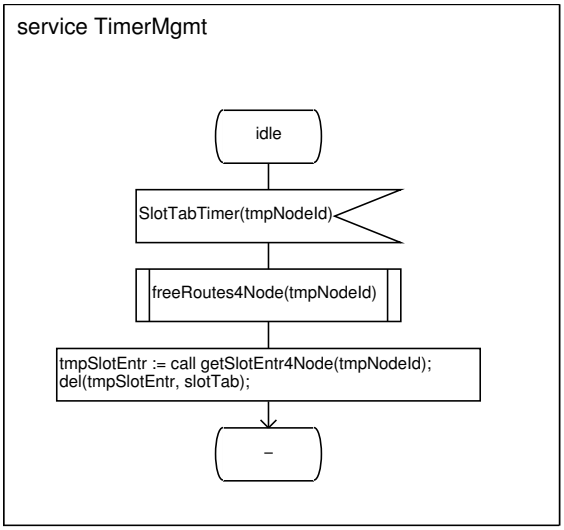

Abbildung C.1.: Behandlung eines abgelaufenen SlotTabTimers.

Beim Ablaufen eines solchen Timers werden alle Routen, die den entsprechenden Knoten als Nachfolger haben, freigegeben (Prozedur *freeRoutes4Node*). *freeRoutes4Node* entfernt für jede zu löschende Route den Eintrag aus der permanenten Routentabelle. Ist der Knoten die Quelle, so wird die Anwendung mittels eines APP_RouteDeleted-Pakets informiert. Anderenfalls wird der Vorgänger auf der Route mittels eines RBRKD-Pakets benachrichtigt. Anschließend wird die zu dem nicht mehr erreichbaren Knoten gehörende Zeile der Slottabelle gelöscht. Routen, die diesen Knoten als Vorgänger haben, brauchen nicht explizit freigegeben zu werden. Da keine Daten mehr empfangen werden, laufen die RouteTimer solcher Routen ab, was zu deren Freigabe führt (s. u.).

Anmerkung: Der Aufruf von Prozeduren ohne Rückgabewert erfolgt in SDL durch die Verwendung eines entsprechenden Symbols (s. *freeRoutes4Node*); der Aufruf von Prozeduren mit Rückgabewert erfolgt durch das Schlüsselwort *call* (s. *getSlotEntr4Node*).

Abbildung C.2 zeigt die Behandlung eines abgelaufenen RouteTimers.

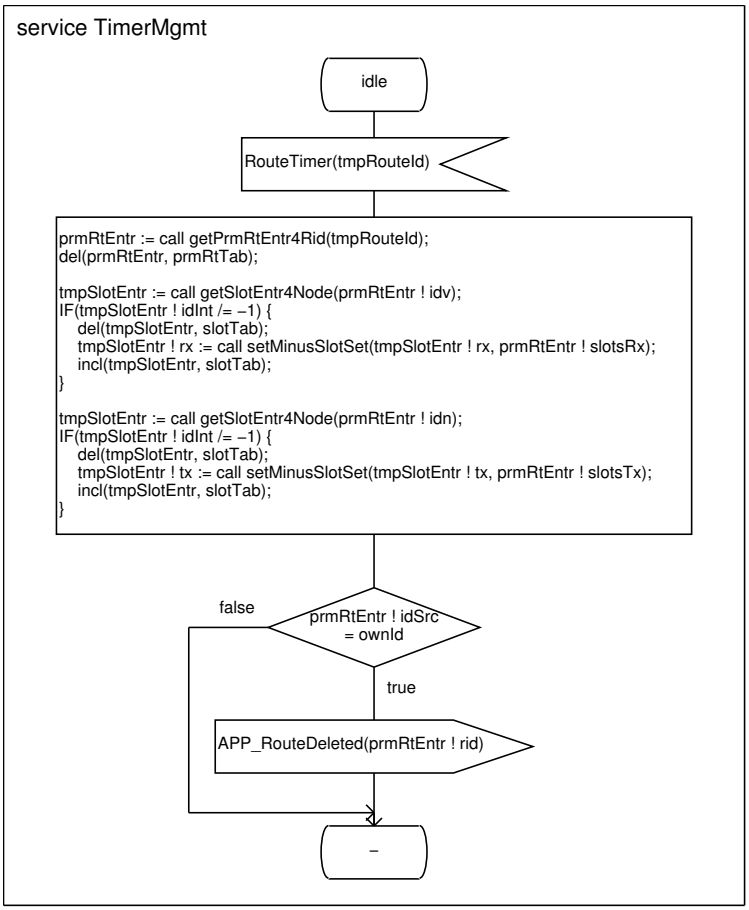

Abbildung C.2.: Behandlung eines abgelaufenen RouteTimers.

Beim Ablaufen eines RouteTimers werden der zugehörige Eintrag aus der permanenten Routentabelle sowie die Empfangs- und Sendereservierungen aus der Slottabelle entfernt. Ist der Knoten die Quelle, so wird die Anwendung mittels APP_RouteDeleted benachrichtigt. Ein wesentlicher Unterschied zum Ablaufen eines SlotTabTimers besteht darin, dass beim RouteTimer kein RBRKD an den Vorgänger gesendet werden muss, da der Timeout einer Route in jedem Knoten lokal erfolgt.

Anmerkung: In SDL ist die direkte Manipulation von Datenstrukturen nur eingeschränkt möglich. Deshalb muss *tmpSlotEntr* zunächst aus der Slottabelle entfernt und nach der Modifikation wieder hinzugefügt werden.

C.2. Service *Data*

Der *Data*-Service dient zum Versenden und Empfangen von DATA- und RDEL-Paketen (beides erfolgt in reservierten Datenslots). Abbildung C.3 zeigt das Versenden dieser Pakete.

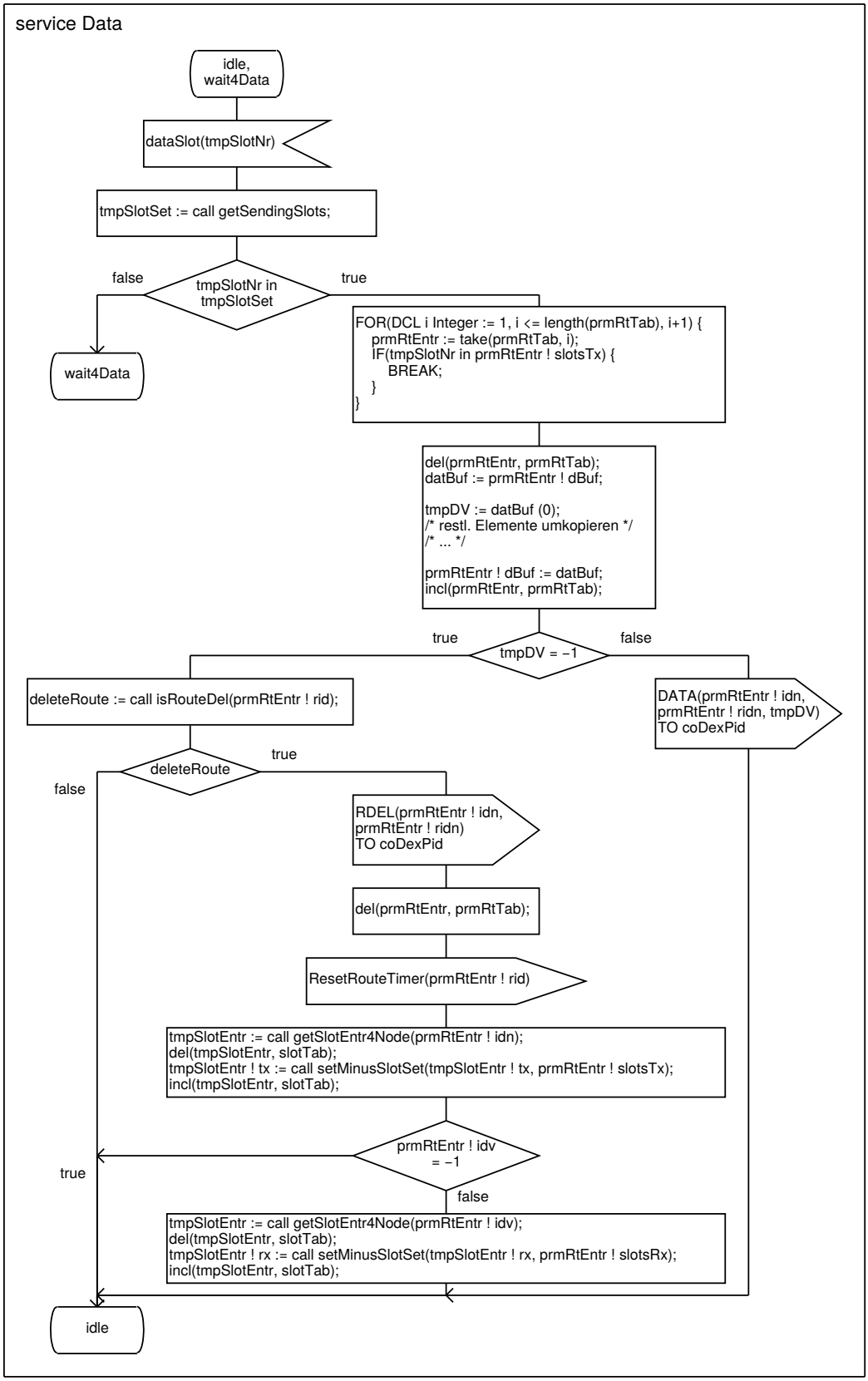

Abbildung C.3.: Versenden von DATA und RDEL.

Ein Knoten befindet sich im Zustand *idle*, wenn im vorherigen Datenslot gesendet oder empfangen wurde. Ist dies nicht der Fall, so bleibt der Knoten im Zustand *wait4Data*. Wenn der aktuelle Slot beginnt, so wird zunächst geprüft, ob für diesen eine Sendereservierung existiert. In diesem Fall wird der passende Eintrag in der permanenten Routentabelle gesucht. Anschließend wird der erste Wert aus dem zugehörigen Datenpuffer genommen und in einem Datenpaket gesendet, sofern der Puffer nicht leer ist (Datenpuffer und Eintrag in der Routentabelle müssen explizit neu gesetzt werden, da ein direktes Modifizieren nicht möglich ist). Wurde das Ende des Puffers erreicht, so wird geprüft, ob eine Anforderung zum Löschen der Route vorliegt (dies kann nur der Fall sein, wenn der Knoten nicht das Ziel ist). Liegt eine solche Anforderung vor, so wird ein RDEL gesendet, der entsprechende Eintrag der permanenten Routentabelle gelöscht und der Timer für die Route zurückgesetzt. Zusätzlich werden die Sende- und Empfangsreservierungen aus der Slottabelle gelöscht, wobei letztere nur existieren, wenn der Knoten nicht die Quelle ist.

C.3. Service *SpropRbrk*

Der Service *SpropRbrk* dient zum Versenden und Empfangen von SCOUNT-, SPROPA-, SPROPD-, RBRKA- und RBRKD-Paketen. Abbildung C.4 zeigt exemplarisch das Versenden und Empfangen von SPROPA- und SPROPD-Paketen.

Ein Knoten a nimmt nur an der Arbitrierung mittels SPROPA teil, wenn die Variable *contending* gesetzt ist. Es wird dann ein SPROPA gesendet und überprüft, ob das empfangene SPROPA mit dem gesendeten übereinstimmt. Ist dies der Fall, so überträgt a seine Sende- und Empfangsreservierungen mittels SPROPD (nachdem im Zustand *wait4SpropdSlot* auf den Beginn des entsprechenden Slots gewartet wurde). Anschließend wird der SpropTimer gesetzt, um das Versenden des nächsten SPROPD auszulösen (dieser Timer wird lokal verwaltet, da er nur vom *SpropRbrk*-Service verwendet wird). Die Variable *contendig* wird auf *false* gesetzt, da a bis zum Ablaufen des Timers nicht an der Arbitrierung teilnehmen soll.

Hat der Knoten a nicht an der Arbitrierung teilgenommen oder diese verloren, so wartet er auf ein empfangenes SPROPD-Paket (Zustand *wait4Spropd*). Bei Empfang eines solchen werden die Empfangs- und Sendereservierungen des Senders b als Sende- bzw. Empfangsblockierungen in die Slottabelle eingetragen (Slots, die zum Senden an b oder zum Empfangen von b reserviert sind, werden nicht als zum Senden bzw. Empfangen blockiert eingetragen). Sofern es für b noch keinen Eintrag in der Slottabelle gibt, wird ein solcher angelegt. Zudem muss der SlotTabTimer für den (existierenden oder neu erzeugten) Eintrag gesetzt werden. Weiterhin wird geprüft, ob b vorher ein Kommunikationsnachbar von a war und jetzt keiner mehr ist. Dazu dient die Variable *senderInCommRange*, deren Wert anhand von Informationen der MAC-Schicht bei Empfang des (mittels Bit-based Transfer übertragenen) SPROPD-Pakets gesetzt wird (in Abbildung C.4 nicht dargestellt). In diesem Fall müssen alle Routen, die b als Nachfolger haben, freigegeben werden (analog zum Ablaufen eines SlotTabTimers).

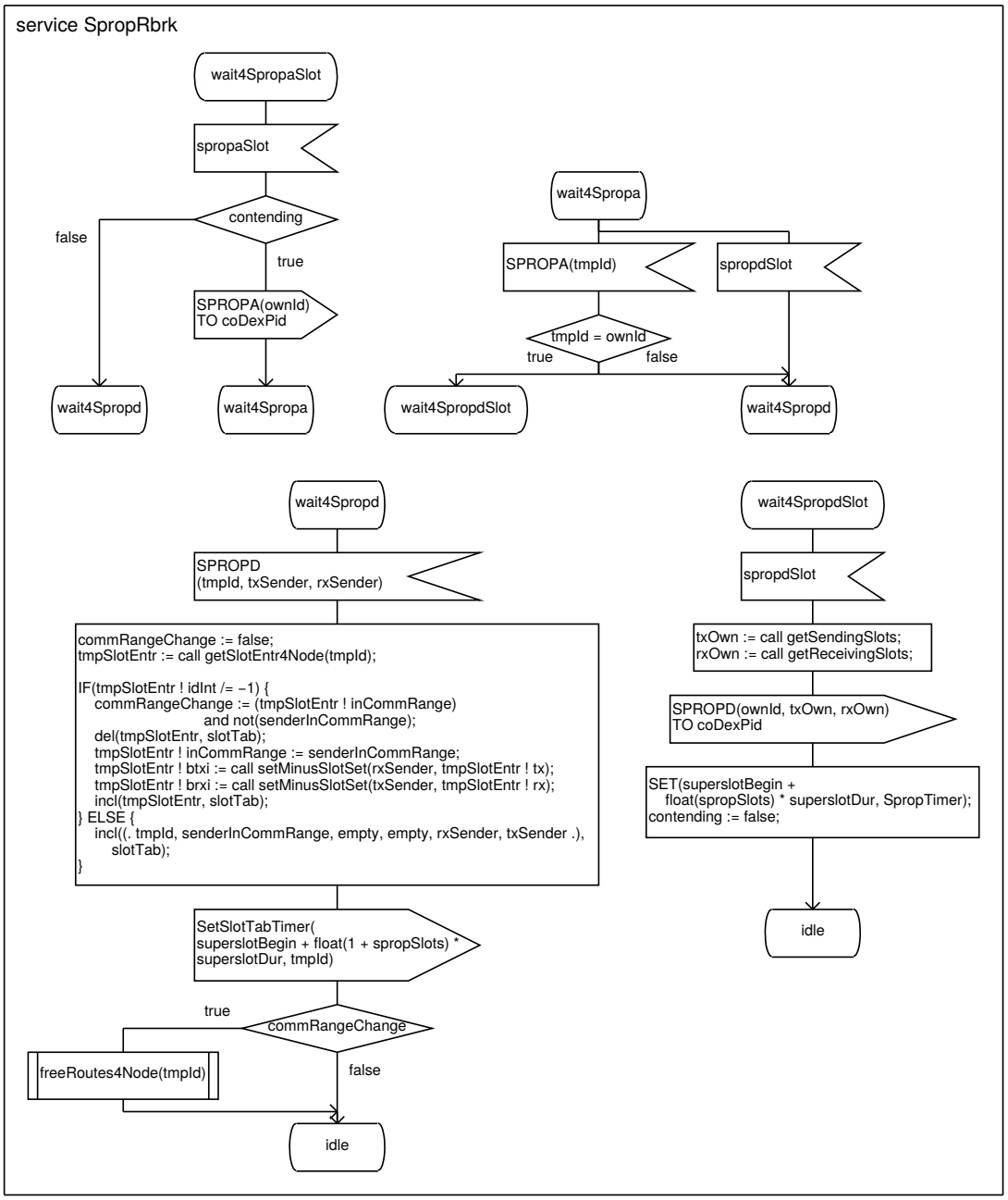

Abbildung C.4.: Versenden und Empfangen von SPROPA und SPROPD.

C.4. Service *RouteSearch*

Der Service *RouteSearch* ist für die Durchführung von Routensuchen zuständig. Im Folgenden werden einige Aspekte der Phasen 1 – 3 einer Routensuche exemplarisch beschrieben.

C.4.1. Phase 1

In Phase 1 prüft zunächst jeder Knoten, ob ein Eintrag in der temporären Zieltabelle vorliegt. Ist dies der Fall, so nimmt der Knoten mittels RREQA an der Arbitrierung teil und

wartet anschließend, ob das empfangene RREQA mit dem gesendeten übereinstimmt. Nimmt der Knoten nicht an der Arbitrierung teil, so wird auf das RREQA eines anderen Knotens gewartet. Das Warten auf ein empfangenes RREQA wird in beiden Fällen durch den Zustand *wait4Rreqa* abgebildet. Die Verarbeitung von RREQA sowie das Versenden und Empfangen von RREQD wird in Abbildung C.5 gezeigt.

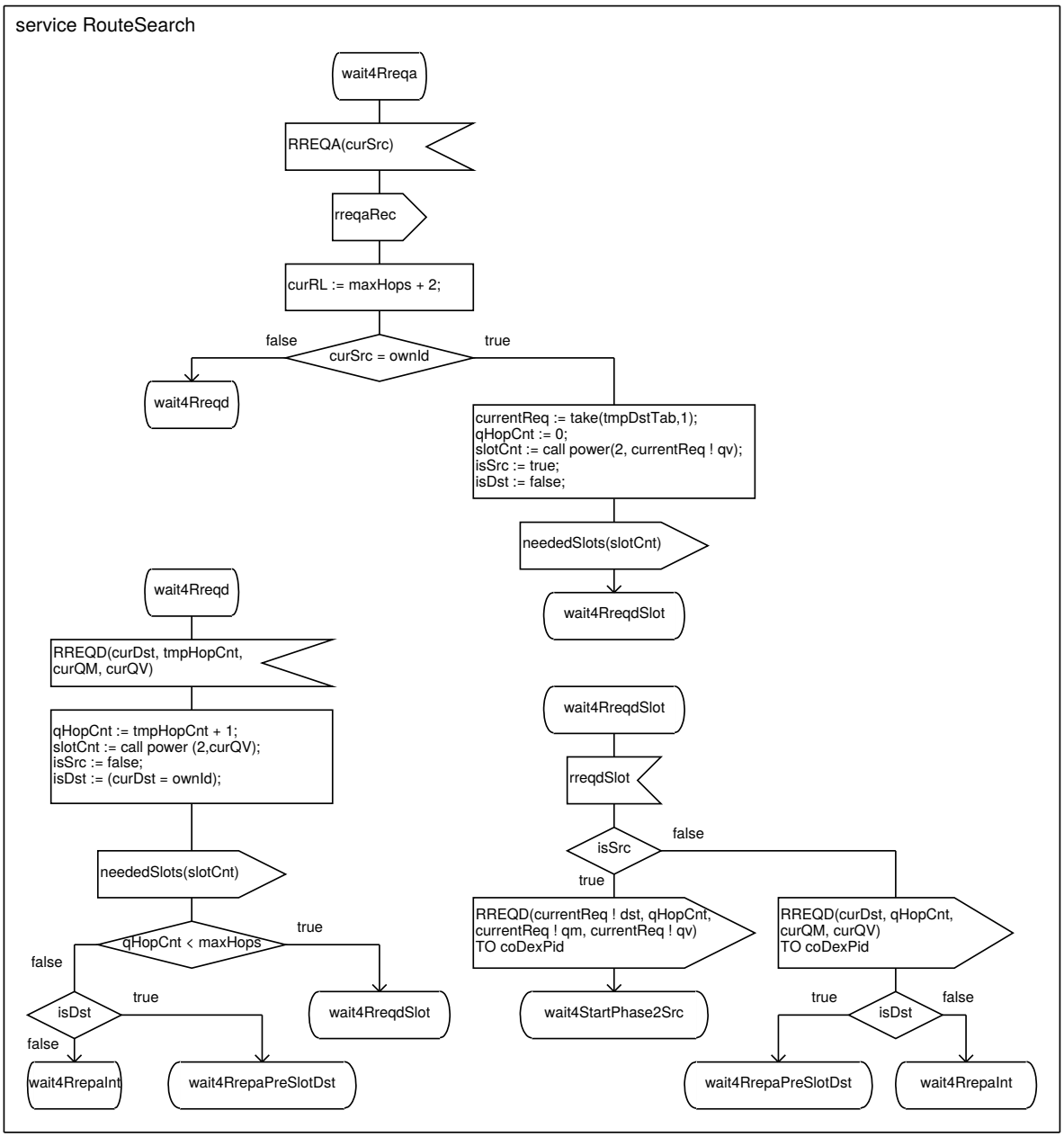

Abbildung C.5.: Verarbeitung von RREQA; Senden und Empfangen von RREQD.

Sobald ein RREQA empfangen wurde, wird dies der MAC-Schicht gemeldet (Signal *rreqaRec*). Diese generiert dann virtuelle Slots für RREQD-Pakete. Die aktuelle Routenlänge wird mit dem maximal möglichen Wert initialisiert (hier $n_{maxHops} + 2$, da MaxRouteLength $=_{Df}$ QHopCount + 2 verwendet wird), weil das erste weiterzuleitende Paket dann in jedem Fall

keine größere Routenlänge hat. Verliert der Knoten die Arbitrierung (oder hat er nicht daran teilgenommen), so wartet er im Zustand *wait4Rreqd* auf das RREQD des Gewinners. Bei Empfang eines RREQD wird der MAC-Schicht die Anzahl benötigter Slots mitgeteilt (Signal *neededSlots*; dieses wird zur Berechnung der Dauer der CREQI- und CREQD-Slots benötigt). Das RREQD-Paket wird weitergeleitet, sofern es noch nicht im gesamten Netz propagiert wurde. Nach dessen Weiterleitung (oder wenn das Paket nicht mehr weiterzuleiten ist) hängt der Folgezustand davon ab, ob es sich um den Ziel- oder einen Zwischenknoten handelt, denn der Zielknoten muss Phase 2 initiieren. Deshalb wartet dieser im Zustand *wait4RrepaPreSlotDst* auf den ersten virtuellen Slot für Phase 2, während die übrigen Knoten im Zustand *wait4RrepaInt* auf empfangene Pakete warten.

Der Knoten, der die Arbitrierung gewinnt, wird die Quelle und bearbeitet die erste Routenanforderung seiner temporären Zieltabelle. Auch in diesem Fall wird der MAC-Schicht die Anzahl der benötigten Slots mitgeteilt. Nach dem Senden des RREQD wird im Zustand *wait4StartPhase2Src* auf den Beginn von Phase 2 gewartet, da die Quelle die von der MAC-Schicht generierten virtuellen Slots für RREPD zählen muss, um einen eventuellen Timeout zu erkennen (vgl. Abschnitt 5.5.2).

C.4.2. Phase 2

Abbildung C.6 zeigt exemplarisch den Empfang und die Verarbeitung der RREPA- und RREPD-Pakete in Zwischenknoten.

Nach Empfang eines RREPA wird geprüft, ob die Arbitrierung gewonnen wurde. Ist dies der Fall, so wartet der Knoten im Zustand *wait4RrepdSlotInt* auf den nächsten virtuellen RREPD-Slot. Anderenfalls (d. h. der Knoten hat nicht an der Arbitrierung teilgenommen oder diese verloren) wartet er auf das RREPD des Gewinners (Zustand *wait4NextRrepdInt*).

Nach Empfang eines RREPD wird zunächst geprüft, ob das Paket verworfen werden soll. Dies ist der Fall, wenn bereits ein Paket mit kürzerer Routenlänge weitergeleitet wurde oder die maximale Anzahl an Weiterleitungen bzw. die maximale Routenlänge erreicht ist. Bei letzterem kann das Paket bereits hier verworfen werden, da die Routenlänge bei der nächsten Weiterleitung überschritten würde, und da das Paket noch nicht bei der Quelle angekommen ist, müsste es noch mindestens einmal weitergeleitet werden. Wird das Paket nicht verworfen, so werden zunächst die freien Sende- und Empfangsslots berechnet. Beträgt die Länge der bisherigen Route höchstens drei Hops (d. h. zwei Hops beim Sender), so wird geprüft, ob auf dieser die QoS-Anforderung potenziell erfüllt werden kann (vgl. Abschnitt 2.7; hier kann jedoch für den nächsten Link jeweils nur F_{RX} verwendet werden, da F_{TX} noch unbekannt ist). Kann die QoS-Anforderung nicht erfüllt werden, so wird das RREPD verworfen. Indem die notwendigen Bedingungen vom Ziel ausgehend geprüft werden, ist sichergestellt, dass für den letzten Link der Route die QoS-Anforderung erfüllt werden kann. Beträgt die Routenlänge mindestens vier Hops, so werden (mit der Prozedur *calcChosenSlots*) die Sendeslots für den 3-Hop-Nachfolger gewählt und aus den Mengen der freien Sendeslots für den aktuellen Knoten sowie den 1- und 2-Hop-Nachfolger entfernt. Anschließend wird überprüft, ob die notwendigen Bedingungen für die sich vor dem 3-Hop-Nachfolger befindenden Links erfüllt sind (wiederum unter Verwendung von F_{RX} für den nächsten Link). Sind diese nicht erfüllt, so wird das RREPD verworfen. Anderenfalls ist zwar nicht garantiert, dass die Route erfolgreich bis zur Quelle fortgesetzt werden kann, aber es ist sichergestellt, dass für den 2-Hop-Nachfolger noch genügend freie Sendeslots zur Verfügung stehen. Es wird dann ein Eintrag in der temporären Routentabelle sowie in der Liste der zu versendenden RREPDs erzeugt, und der Knoten wartet im Zustand *wait4RrepaPreSlotInt* auf den nächsten Slot für die (vorgeschaltete) Arbitrierung.

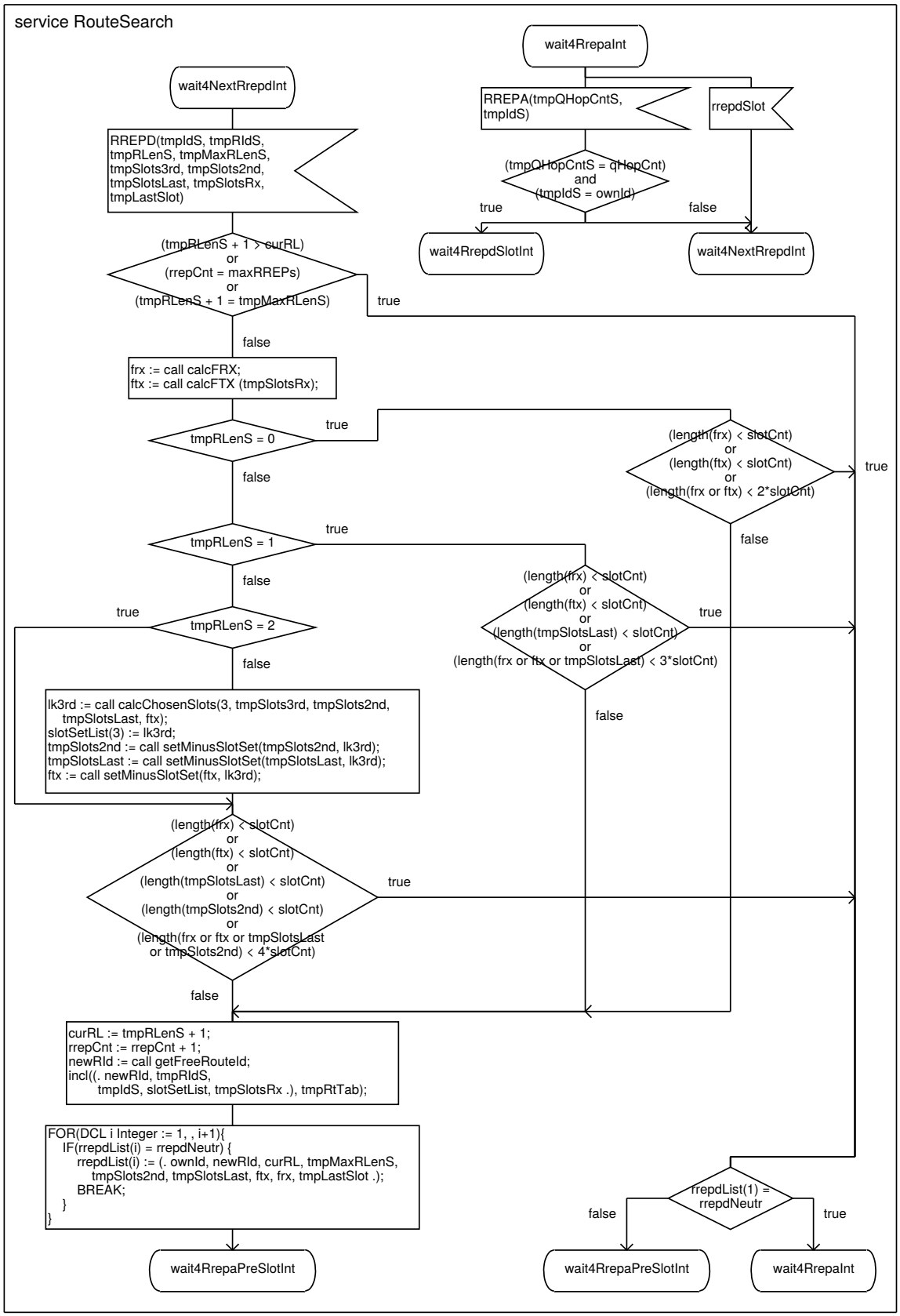

Abbildung C.6.: Verarbeitung von RREPA und RREPD in Zwischenknoten.

Wird das empfangene RREPD verworfen, so wird geprüft, ob der Knoten noch zu versendende RREPDs hat, welche aus anderen empfangenen RREPDs resultieren. In diesem Fall nimmt der Knoten ebenfalls im nächsten Slot an der Arbitrierung teil; anderenfalls wartet er auf weitere RREPAs (Zustand *wait4RrepaInt*).

C.4.3. Phase 3

Abbildung C.7 zeigt exemplarisch den Empfang und die Verarbeitung eines CREQI-Pakets in einem Zwischenknoten.

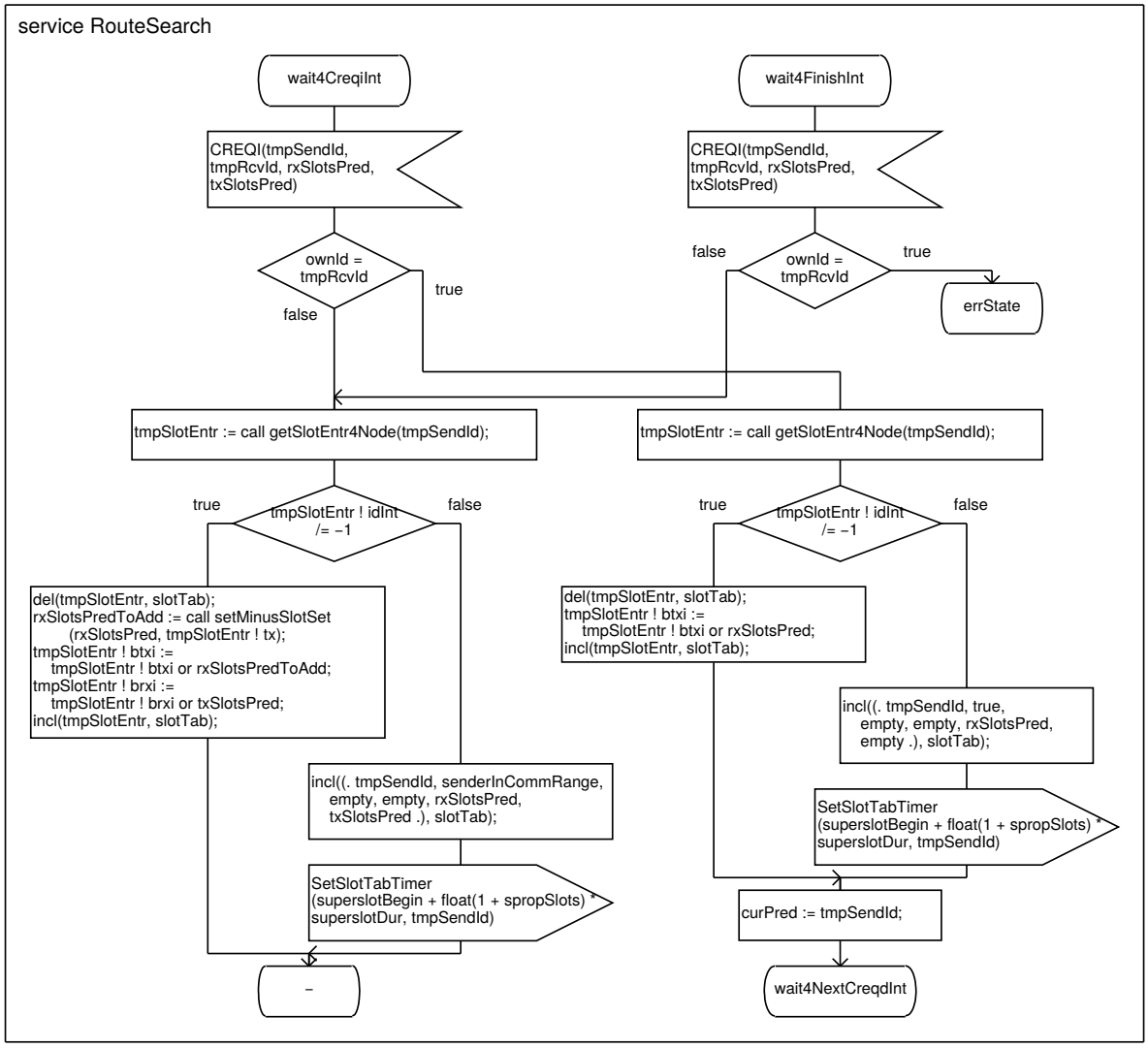

Abbildung C.7.: Verarbeitung von CREQI in Zwischenknoten.

Ein CREQI kann einerseits empfangen werden, wenn der Knoten noch nicht Bestandteil der Route ist (Zustand *wait4CreqiInt*), andererseits aber auch, wenn dies bereits der Fall ist (Zustand *wait4FinishInt*).

Ist der Knoten bereits Bestandteil der Route, so kann ein empfangenes CREQI nicht an diesen adressiert sein. Es werden lediglich die Empfangs- und Sendeslots des Senders als zum Senden bzw. zum Empfangen blockiert in die Slottabelle eingetragen (da der Empfänger dann

nicht der Routennachfolger, sondern lediglich ein Interferenznachbar des Senders ist). Dabei ist darauf zu achten, dass die Empfangsslots des Senders nur dann als zum Senden blockiert eingetragen werden, wenn sie nicht zum Senden reserviert sind (dies ist der Fall, wenn der Knoten das CREQI-Paket seines Nachfolgers auf der Route empfängt).

Ist der Knoten noch nicht Bestandteil der Route und ein empfangenes CREQI ist nicht an diesen adressiert, so ist der Ablauf der gleiche wie wenn dieser bereits Bestandteil der Route ist. Es kann dann jedoch der Fall eintreten, dass für den Sender noch kein Eintrag in der Slottabelle existiert. In diesem Fall wird ein neuer angelegt und ein zugehöriger SlotTabTimer gestartet. Der Wert der Variablen *senderInCommRange* wird anhand von Informationen der MAC-Schicht bei Empfang des CREQI-Pakets gesetzt (analog zum Empfang von SPROPD; vgl. Anhang C.3).

Ist das CREQI an den empfangenden Knoten adressiert, so ist dieser der Routennachfolger und trägt die Empfangsslots des Vorgängers als zum Senden blockiert in seine Slottabelle ein (auch hier muss ggf. ein neuer Eintrag erzeugt werden; jedoch befindet sich der Sender auf jeden Fall in Kommunikationsreichweite, da er der Vorgänger auf der Route ist). Die eigenen Empfangsreservierungen (d. h. die Sendeslots der Vorgängers) können noch nicht eingetragen werden, da noch nicht feststeht, ob bei diesen Fehlreservierungen auftreten (vgl. Abschnitt 5.2.3.2). Zudem wird der aktuelle Vorgänger gespeichert (Variable *curPred*), um nach dem Versenden eines CREQC (im Falle einer Fehlreservierung bei den Empfangsslots) ein empfangenes CREQU zuordnen zu können. Anschließend wird auf das zu dem empfangenen CREQI gehörende CREQD gewartet (Zustand *wait4NextCreqdInt*).

Abbildung C.8 zeigt den Empfang und die Verarbeitung des zu einem CREQI-Paket gehörenden CREQD in einem Zwischenknoten. Durch die Verwendung des Zustands *wait4NextCreqdInt* wird das CREQD nur vom Routennachfolger empfangen. Zunächst wird geprüft, ob von den zum Senden vorgesehenen Slots mindestens einer bereits zum Senden blockiert ist. Ist dies der Fall, so werden aus den noch freien Sendeslots des Knotens diejenigen ermittelt, die weder für diesen noch für die nächsten drei Knoten der Route zum Senden vorgesehen sind. Stehen nicht genug alternative Sendeslots zur Verfügung, so ist die Routensuche gescheitert (im Zustand *wait4NextSlotFailInt* wird auf den nächsten Slot gewartet, um ein CREQF zu senden). Anderenfalls werden die blockierten Sendeslots durch freie ersetzt. Anschließend wird geprüft, ob von den zum Empfangen vorgesehenen Slots mindestens einer bereits zum Empfangen blockiert ist. Ist dies der Fall, so wird (im nächsten Slot) ein CREQC gesendet, um vom Vorgänger alternative Slots anzufordern. Zuvor werden die nicht zu Problemen führenden Empfangsreservierungen bereits in die Slottabelle eingetragen, um fehlerhafte Blockierungen zu vermeiden (s. u.). Anschließend wird (im Zustand *wait4CrequInt*) auf das CREQU des Vorgängers gewartet. Bei Empfang eines CREQU werden die Empfangsslots entsprechend angepasst, und der reguläre Ablauf wird fortgesetzt. Liegen keine Fehlreservierungen bei den Empfangsslots vor (oder wurden diese behoben), so werden die Empfangsreservierungen in die Slottabelle eingetragen (die Sendereservierungen werden zur Vermeidung fehlerhafter Blockierungen erst bei der Weiterleitung von CREQD eingetragen; s. u.). Zusätzlich wird ein Eintrag in der permanenten Routentabelle erzeugt. Der zugehörige Timer wird jedoch erst gestellt, wenn die Routensuche erfolgreich beendet wurde. Danach wird (im Zustand *wait4CreqiSlotRegInt*) auf den nächsten CREQI-Slot gewartet.

Vermeidung fehlerhafter Blockierungen

Bei der Spezifikation von RBBQR mittels SDL wurde ein – selten auftretendes – Problem identifiziert. Trägt ein Knoten a seine Sendereservierungen bereits in die Slottabelle ein, oh-

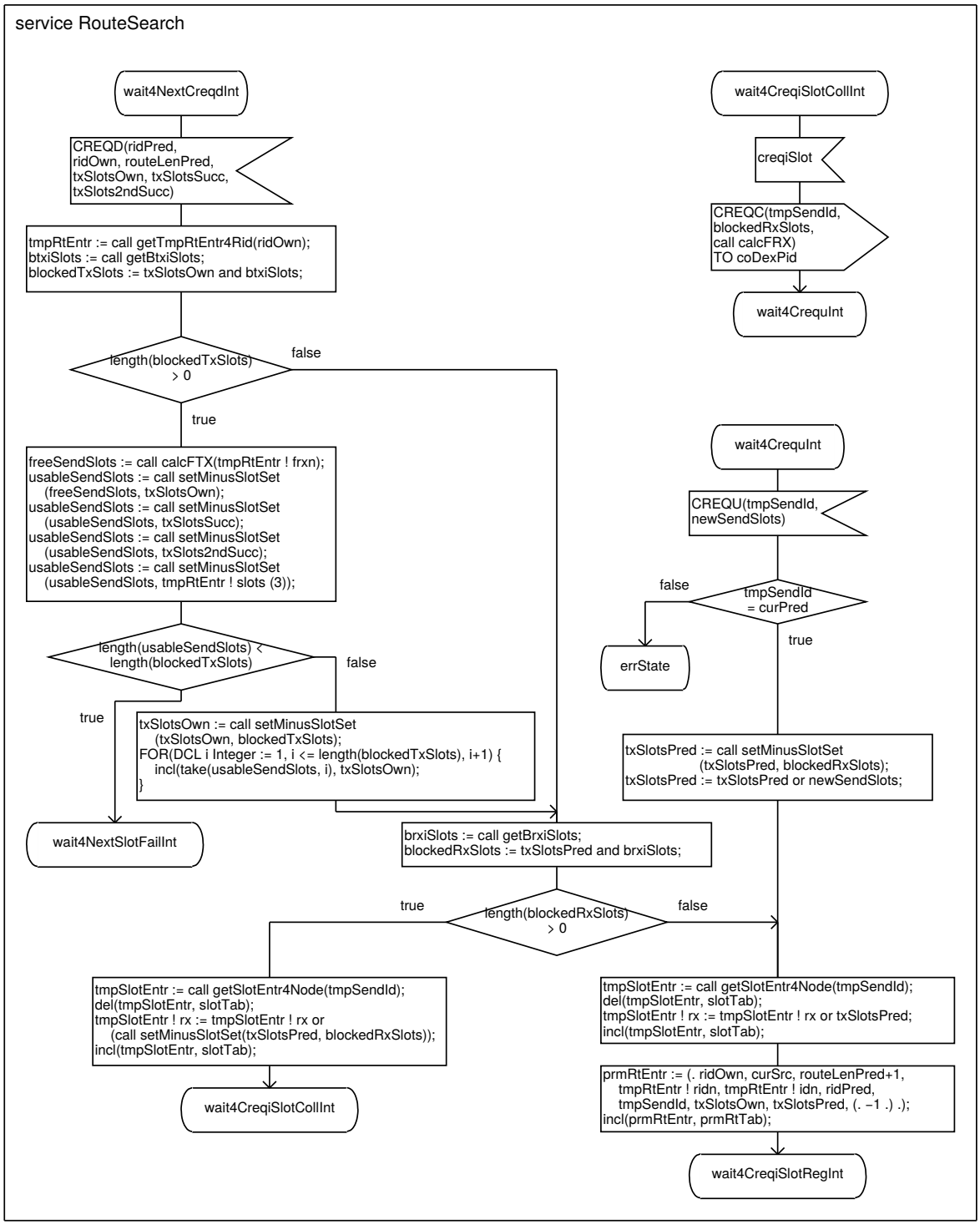

Abbildung C.8.: Verarbeitung von CREQD in Zwischenknoten.

ne dass die zugehörigen Empfangsreservierungen beim Empfänger b gleichzeitig eingetragen werden, kann es zu Inkonsistenzen kommen, wenn zwischen diesen beiden Vorgängen ein Superslotwechsel stattfindet. Zu Beginn eines Superslots werden zunächst virtuelle Slots für SCOUNT, SPROPA und SPROPD erzeugt. Sendet nun Knoten a ein SPROPD, so erhält

b die – bereits eingetragenen – Sendereservierungen von a und markiert die Slots als zum Empfangen blockiert, da keine zugehörigen Empfangsreservierungen vorliegen. Wenn dann die Empfangsreservierungen eingetragen werden sollen, sind die entsprechenden Slots bereits zum Empfangen blockiert. Somit ist es erforderlich, dass Sende- und Empfangsreservierungen immer zum selben Zeitpunkt eingetragen werden. Aus diesem Grund wurde festgelegt, dass jeder Knoten seine Sendereservierungen beim Versenden und seine Empfangsreservierungen bei der Verarbeitung von CREQD einträgt.

D Anhang D.
Funktionale Simulation von RBBQR

Im Folgenden werden drei funktionale Simulationen von RBBQR vorgestellt, welche zum Nachweis der grundsätzlichen Funktionalität des Protokolls dienen. Diese wurden mit dem internen Simulator der Rational SDL Suite [IBM15] unter Verwendung der in Abschnitt 5.7.1 und Anhang C beschriebenen SDL-Spezifikation ausgeführt. Die zugrunde gelegten Hardwareparameter und szenariospezifischen Parameter entsprechen den in Abschnitt 5.6 verwendeten (insbesondere gilt damit $n_{maxSlot} = 99$). Für die Dauer der Superslots wird $d_{superslot} = 1\,\mathrm{s}$ verwendet.

D.1. Erste Simulation

Die erste Simulation wurde in Abschnitt 5.7.2 bereits teilweise vorgestellt. Deshalb wird hier nur auf Aspekte eingegangen, die dort noch nicht detailliert erläutert wurden. Aus Gründen der Übersicht wird das verwendete Netzwerk in Abbildung D.1 nochmals dargestellt.

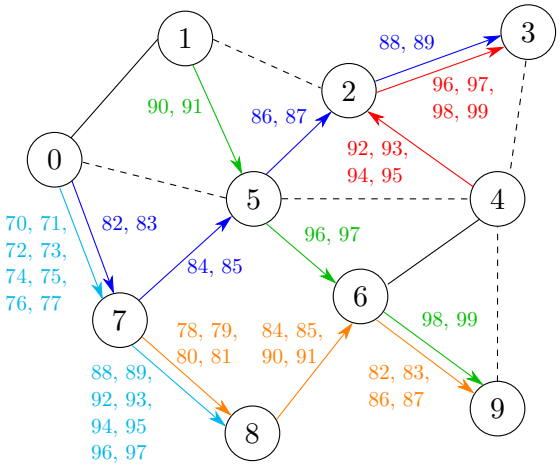

Abbildung D.1.: Netzwerk für erste funktionale Simulation (Wdh. von Abbildung 5.29).

In diesem Netz werden fünf Routen etabliert, was bereits in Abschnitt 5.7.2 detailliert betrachtet wurde. Im Folgenden wird zunächst auf die Propagierung von Netzzustandsinformationen eingegangen. Anschließend wird die Freigabe der etablierten Routen in chronologischer Reihenfolge beschrieben. Dabei werden spezielle Aspekte jeweils zusätzlich anhand von Listings verdeutlicht.

Propagierung von Netzzustandsinformationen

Die Propagierung der Netzzustandsinformationen mit SPROPD-Paketen erfolgt jeweils zu Beginn eines Superslots (vgl. Abschnitt 5.5). Listing D.1 zeigt einen entsprechenden Ausschnitt.

```
*** TRANSITION START
*       PId     : <<Block Nodes:9/Block rbbqr>> MainProc:9
*       Service: SpropRbrk
*       Now     : 0.0466
*   OUTPUT of SPROPD to <<Block Nodes:9/Block rbbqr>> CoDex:9
*       Parameter(s) : 9, [  ], [  ]
*   SET on timer SpropTimer at 10.0000
...
*** TRANSITION START
*       PId     : <<Block Nodes:7/Block rbbqr>> MainProc:7
*       Service: SpropRbrk
*       Now     : 0.0466
*   OUTPUT of SPROPD to <<Block Nodes:7/Block rbbqr>> CoDex:7
*       Parameter(s) : 7, [  ], [  ]
*   SET on timer SpropTimer at 10.0000
...
*** TRANSITION START
*       PId     : <<Block Nodes:8/Block rbbqr>> MainProc:8
*       Service: SpropRbrk
*       Now     : 1.0466
*   OUTPUT of SPROPD to <<Block Nodes:8/Block rbbqr>> CoDex:8
*       Parameter(s) : 8, [  ], [  ]
*   SET on timer SpropTimer at 11.0000
...
*** TRANSITION START
*       PId     : <<Block Nodes:3/Block rbbqr>> MainProc:3
*       Service: SpropRbrk
*       Now     : 1.0466
*   OUTPUT of SPROPD to <<Block Nodes:3/Block rbbqr>> CoDex:3
*       Parameter(s) : 3, [  ], [ 96, 97, 98, 99 ]
*   SET on timer SpropTimer at 11.0000
...
*** TRANSITION START
*       PId     : <<Block Nodes:6/Block rbbqr>> MainProc:6
*       Service: SpropRbrk
*       Now     : 2.0466
*   OUTPUT of SPROPD to <<Block Nodes:6/Block rbbqr>> CoDex:6
*       Parameter(s) : 6, [  ], [ 96, 97 ]
*   SET on timer SpropTimer at 12.0000
...
```

Listing D.1.: Etablierung des Netzzustands durch SPROPD-Pakete.

Anmerkung: Analog zu Listing 5.1 (Seite 107) wurden in allen Listings dieses Anhangs die vom SDL-Laufzeitsystem vergebenen PIds an die IDs der jeweiligen Knoten angepasst.

Die SPROPD-Pakete enthalten jeweils die ID des Senders sowie dessen zum Senden und zum Empfangen reservierte Slots. Im ersten Superslot können die Knoten 9 und 7 gleichzeitig ein SPROPD versenden, denn bei der vorherigen Arbitrierung mittels SPROPA gewinnt der Knoten mit der höchsten ID (9) und zusätzlich Knoten 7 (Knoten mit nächstkleinerer ID, der kein 1- oder 2-Hop-Interferenznachbar von 9 ist). Alle anderen Knoten sind 1- oder 2-Hop-Interferenznachbarn von 7 oder 9. Analog senden im zweiten Superslot 8 und 3 parallel. Im dritten Superslot sendet lediglich Knoten 6; in den folgenden Superslots senden die Knoten 5, 4, 2, 1 und 0. Da $n_{maxNodes} = 10$ als obere Schranke für die Anzahl der bei der Arbitrierung konkurrierenden Knoten ($n_{maxCompNodes}$) verwendet wird, gilt für jeden Knoten

SpropTimer = $t_{superslotBegin} + 10 \cdot 1\,\mathrm{s}$ (vgl. Abschnitt 5.4.2). Somit sendet im neunten und zehnten Superslot kein Knoten. Im elften senden dann wieder 9 und 7 parallel (im Unterschied zum ersten Superslot nehmen jetzt jedoch keine anderen Knoten mehr an der Arbitrierung teil). Anschließend wird der restliche Ablauf analog fortgesetzt.

Anmerkung: Zusätzlich zu den SPROPD-Paketen werden auch durch Routensuchen Statusinformationen propagiert. Beispielsweise sendet Knoten 4 im ersten Superslot ein CREQI-Paket (vgl. Listing 5.1 in Abschnitt 5.7.2). Da der Knoten zu diesem Zeitpunkt noch kein SPROPD-Paket versendet hat, kennen die Interferenznachbarn ihn noch nicht.

Freigabe von Routen

Zum Zeitpunkt 5 s wird die Route $1 \rightarrow 5 \rightarrow 6 \rightarrow 9$ durch die Anwendung explizit freigegeben. Listing D.2 zeigt dies.

```
*** TRANSITION START
*       PId     : <<Block Nodes:1/Block app>> AppProc:1
*       Now     : 5.0000
*   OUTPUT of APP_FreeRoute to <<Block Nodes:1/Block rbbqr>> MainProc:1
*       Parameter(s) : 0
...
*** TRANSITION START
*       PId     : <<Block Nodes:1/Block rbbqr>> MainProc:1
*       Service: Data
*       Input   : dataSlot
*       Now     : 5.9775
*       Parameter(s) : 90
*   ASSIGN  prmRtEntr := (. 0, 1, 0, 0, 5, -1, -1, [ 90, 91 ], [  ], (:
    (others:-1) :) .)
*   OUTPUT of RDEL to <<Block Nodes:1/Block rbbqr>> CoDex:1
*       Parameter(s) : 5, 0
*   CALL OPERATION del
*   ASSIGN  tmpSlotEntr := (. 5, true, [ 90, 91 ], [  ], [ 84, 85 ], [
    86, 87, 96, 97 ] .)
*   ASSIGN  tmpSlotEntr ! tx := [  ]
...
*** TRANSITION START
*       PId     : <<Block Nodes:5/Block rbbqr>> MainProc:5
*       Service: Data
*       Input   : dataSlot
*       Now     : 5.9825
*       Parameter(s) : 96
*   ASSIGN  prmRtEntr := (. 0, 1, 1, 0, 6, 0, 1, [ 96, 97 ], [ 90, 91 ],
    (: (others:-1) :) .)
*   OUTPUT of RDEL to <<Block Nodes:5/Block rbbqr>> CoDex:5
*       Parameter(s) : 6, 0
*   CALL OPERATION del
*   ASSIGN  tmpSlotEntr := (. 6, true, [ 96, 97 ], [  ], [ 84, 85, 90, 91
    ], [ 82, 83, 86, 87, 98, 99 ] .)
*   ASSIGN  tmpSlotEntr ! tx := [  ]
*   ASSIGN  tmpSlotEntr := (. 1, true, [  ], [ 90, 91 ], [  ], [  ] .)
*   ASSIGN  tmpSlotEntr ! rx := [  ]
...
```

Listing D.2.: Explizite Freigabe der Route $1 \rightarrow 5 \rightarrow 6 \rightarrow 9$.

Das APP_FreeRoute-Paket enthält die RouteID der freizugebenden Route. Die Freigabe

wird mit Hilfe von RDEL-Paketen entlang der Route propagiert (diese enthalten jeweils ID und RouteID des Nachfolgers). Dazu wird stets der nächste für die Route reservierte Datenslot verwendet (in diesem Fall Slot 90 für Knoten 1, Slot 96 für Knoten 5, usw.). Gleichzeitig entfernt jeder Knoten den zugehörigen Eintrag aus seiner permanenten Routentabelle sowie die Sende- und Empfangsreservierungen aus seiner Slottabelle. Ein Eintrag der permanenten Routentabelle enthält neben der RouteID die ID der Quelle, die Routenlänge zu dieser, RouteID und ID von Nachfolger und Vorgänger, die Sendeslots, die Empfangsslots und einen Puffer für noch nicht gesendete Daten. Ein Eintrag in der Slottabelle enthält neben der ID des Interferenznachbarn und der Information, ob dieser sich in Kommunikationsreichweite befindet, die zum Senden an und zum Empfangen von diesem reservierten Slots sowie die durch diesen zum Senden und zum Empfangen blockierten Slots.

Zum Zeitpunkt 6.9999 s erfolgt ein Bruch des Kommunikationslinks zwischen den Knoten 2 und 4 (der Interferenzlink bleibt bestehen). Damit ist die Route $4 \rightarrow 2 \rightarrow 3$ nicht mehr verwendbar. Listing D.3 zeigt die implizite Freigabe dieser Route.

In den Knoten 2 und 3 läuft der RouteTimer für die entsprechende RouteID ab (zum Zeitpunkt 10 s bzw. 11 s), da keine Daten mehr über die Route empfangen werden. Nach dem Ablaufen des RouteTimers entfernen die Knoten jeweils den zugehörigen Eintrag aus ihrer permanenten Routentabelle sowie die Sende- bzw. Empfangsreservierungen aus ihrer Slottabelle. Knoten 4 gibt die Route und die zugehörigen Sendereservierungen frei, nachdem beim Empfang des nächsten von 2 gesendeten SPROPD-Pakets festgestellt wird, dass 2 kein Kommunikationsnachbar von 4 mehr ist. Zudem wird die Anwendung mit einem APP_-RouteDeleted-Paket (dieses enthält die entsprechende RouteID) über die Freigabe der Route informiert.

Über die Route $0 \rightarrow 7 \rightarrow 8$ werden ab dem Zeitpunkt 9.8 s keine Daten mehr versendet. Alle beteiligten Knoten geben die Route nach einem entsprechenden Timeout frei. Dies ist analog zur Freigabe der Route $4 \rightarrow 2 \rightarrow 3$ in den Knoten 2 und 3.

Zum Zeitpunkt 14.9999 s erfolgt ein Linkbruch zwischen den Knoten 5 und 7 (Kommunikations- und Interferenzlink), was einen Bruch der Route $0 \rightarrow 7 \rightarrow 5 \rightarrow 2 \rightarrow 3$ zur Folge hat. Die Freigabe in den Knoten 5, 2 und 3 erfolgt analog zu den Knoten 2 und 3 bei der Route $4 \rightarrow 2 \rightarrow 3$. Listing D.4 zeigt die Freigabe der restlichen Route.

Knoten 7 erhält nun kein SPROPD-Paket mehr von 5, weshalb der entsprechende SlotTab-Timer zum Zeitpunkt 24 s abläuft. 7 muss dann alle Routen, bei denen 5 der Nachfolger ist, freigeben (in diesem Fall nur eine). Zusätzlich müssen die zugehörigen Sende- und Empfangsreservierungen sowie die gesamte zu 5 gehörende Zeile der Slottabelle entfernt werden. Für die Empfangsreservierungen kann die Freigabe jedoch erst erfolgen, wenn dem Vorgänger 0 mitgeteilt wird, dass die Route nicht mehr intakt ist (mit einem RBRKD-Paket, welches ID und RouteID des Vorgängers enthält), denn anderenfalls könnte 0 die Slots noch zum Senden nutzen. Dies könnte zu Kollisionen führen, wenn zwischenzeitlich eine neue Routensuche stattfinden und Knoten 7 die Slots beispielsweise zum Empfangen von einem anderen Knoten reservieren würde. Beim Versand von RBRKD können die Empfangsreservierungen gelöscht werden, denn dieses Paket wird wettbewerbsfrei gesendet. Unter Annahme der Single Network Property ist damit garantiert, dass es korrekt empfangen wird. Die Sendereservierungen des Vorgängers werden bei Erhalt von RBRKD gelöscht.

Anmerkung: Das Problem, dass die Freigabe von Reservierungen verzögert werden muss, tritt nur dann auf, wenn Teile von Routen mittels RBRKD in umgekehrter Richtung freigegeben werden. Bei einer Freigabe mittels RDEL oder durch Ablaufen der RouteTimer ist stets

```
*** TRANSITION START
*       PId     : <<Block Nodes:2/Block rbbqr>> MainProc:2
*       Service: TimerMgmt
*       Input   : RouteTimer
*       Now     : 10.0000
*       Parameter(s) : 0
*   ASSIGN  prmRtEntr := (. 0, 4, 1, 0, 3, 0, 4, [ 96, 97, 98, 99 ], [
    92, 93, 94, 95 ], (: (others:-1) :) .)
*   CALL OPERATION del
*   ASSIGN  tmpSlotEntr := (. 4, true, [  ], [ 92, 93, 94, 95 ], ... .)
*   ASSIGN  tmpSlotEntr ! rx := [  ]
*   ASSIGN  tmpSlotEntr := (. 3, true, [ 88, 89, 96, 97, 98, 99 ], [  ],
    ... .)
*   ASSIGN  tmpSlotEntr ! tx := [ 88, 89 ]
...
*** TRANSITION START
*       PId     : <<Block Nodes:3/Block rbbqr>> MainProc:3
*       Service: TimerMgmt
*       Input   : RouteTimer
*       Now     : 11.0000
*       Parameter(s) : 0
...
*** TRANSITION START
*       PId     : <<Block Nodes:4/Block rbbqr>> MainProc:4
*       Service: SpropRbrk
*       Input   : SPROPD
*       Now     : 15.0652
*       Parameter(s) : 2, [ 88, 89 ], [ 86, 87 ]
*   ASSIGN  tmpSlotEntr := (. 2, true, [ 92, 93, 94, 95 ], [  ], [ 86, 87
    ], [ 88, 89, 96, 97, 98, 99 ] .)
*   ASSIGN  tmpSlotEntr ! inCommRange := false
*   ASSIGN  tmpSlotEntr ! brxi := [ 88, 89 ]
*   PROCEDURE START freeRoutes4Node
*       Parameter(s) : 2
*   PROCEDURE START freeRoute4Rid
*       Parameter(s) : 0
*   ASSIGN  prt := (. 0, 4, 0, 0, 2, -1, -1, [ 92, 93, 94, 95 ], [  ], ...
*   CALL OPERATION del
*   ASSIGN  tmpSE := (. 2, false, [ 92, 93, 94, 95 ], [  ], [ 86, 87 ], [
    88, 89 ] .)
*   ASSIGN  tmpSE ! tx := [  ]
*   OUTPUT of APP_RouteDeleted to <<Block Nodes:4/Block app>> AppProc:4
*       Parameter(s) : 0
...
```

Listing D.3.: Implizite Freigabe der Route $4 \rightarrow 2 \rightarrow 3$.

gewährleistet, dass die Route in einem Knoten erst dann freigegeben wird, wenn die Freigabe im Vorgänger bereits erfolgt ist. Auf diese Weise können lediglich Empfangsreservierungen existieren, deren zugehörige Sendereservierungen bereits entfernt wurden, jedoch nicht umgekehrt.

Zum Zeitpunkt 19.9999 s bricht der Interferenzlink zwischen den Knoten 4 und 9. Dies führt dazu, dass die Knoten keine SPROPD-Pakete mehr vom jeweils anderen erhalten, hat jedoch keine weiteren Auswirkungen.

```
*** TRANSITION START
*       PId     : <<Block Nodes:7/Block rbbqr>> MainProc:7
*       Service: TimerMgmt
*       Input   : SlotTabTimer
*       Now     : 24.0000
*       Parameter(s) : 5
*   PROCEDURE START freeRoutes4Node
*       Parameter(s) : 5
*   PROCEDURE START freeRoute4Rid
*       Parameter(s) : 0
*   ASSIGN  prt := (. 0, 0, 1, 1, 5, 0, 0, [ 84, 85 ], [ 82, 83 ], (:
    (others:-1) :) .)
*   CALL OPERATION del
*   ASSIGN  tmpSE := (. 5, true, [ 84, 85 ], [  ], [  ], [ 86, 87 ] .)
*   ASSIGN  tmpSE ! tx := [  ]
*   ASSIGN  tmpSlotEntr := (. 5, true, [  ], [  ], [  ], [ 86, 87 ] .)
*   CALL OPERATION del
...
*** TRANSITION START
*       PId     : <<Block Nodes:7/Block rbbqr>> MainProc:7
*       Service: SpropRbrk
*       Now     : 24.0639
*   ASSIGN  tmpSE := (. 0, true, [  ], [ 82, 83 ], [  ], [  ] .)
*   ASSIGN  tmpSE ! rx := [  ]
*   OUTPUT of RBRKD to <<Block Nodes:7/Block rbbqr>> CoDex:7
*       Parameter(s) : 0, 0
...
*** TRANSITION START
*       PId     : <<Block Nodes:0/Block rbbqr>> MainProc:0
*       Service: SpropRbrk
*       Input   : RBRKD
*       Now     : 24.0644
*       Parameter(s) : 0, 0
*   ASSIGN  prt := (. 0, 0, 0, 0, 7, -1, -1, [ 82, 83 ], [  ], (:
    (others:-1), :) .)
*   CALL OPERATION del
*   ASSIGN  tmpSE := (. 7, true, [ 82, 83 ], [  ], [  ], [ 78, 79, 80,
    81, 84, 85 ] .)
*   ASSIGN  tmpSE ! tx := [  ]
*   OUTPUT of APP_RouteDeleted to <<Block Nodes:0/Block app>> AppProc:0
*       Parameter(s) : 0
...
```

Listing D.4.: Verwendung von RBRKD zur Freigabe der Route $0 \rightarrow 7 \rightarrow 5 \rightarrow 2 \rightarrow 3$.

Zum Zeitpunkt 24.9999 s fällt schließlich Knoten 8 aus. Dies ist äquivalent zu einem Bruch aller Kommunikations- und Interferenzlinks dieses Knotens und führt somit zum Bruch der (letzten noch bestehenden) Route $7 \rightarrow 8 \rightarrow 6 \rightarrow 9$. Die Freigabe erfolgt analog zu Route $0 \rightarrow 7 \rightarrow 5 \rightarrow 2 \rightarrow 3$.

D.2. Zweite Simulation

Die zweite Simulation verwendet das Netzwerk aus Abbildung D.2, für das $n_{maxNodes} = 10$ und $n_{maxHops} = 9$ gilt. Die 2-Hop-Interferenzannahme ist nicht erfüllt, da sich die Knoten 5

und 8 in Interferenzreichweite befinden, obwohl sie 3-Hop-Kommunikationsnachbarn sind. In diesem Netz werden zwei Routen etabliert.

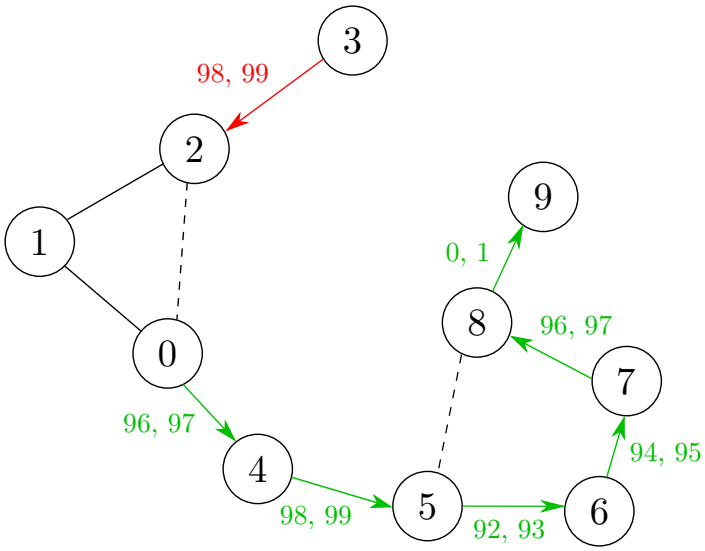

Abbildung D.2.: Netzwerk für zweite funktionale Simulation.

Zum Zeitpunkt 0 s fordern die Knoten 0 und 3 gleichzeitig eine Route an. Knoten 3 gewinnt aufgrund der höheren ID die Arbitrierung und sendet eine Routenanforderung mit Knoten 2 als Ziel und zwei Slots, wodurch die Route $3 \rightarrow 2$ etabliert wird. Mittels SDFP werden die Slots 98 und 99 für den einzigen Link der Route ausgewählt.

Nach der Etablierung dieser Route nimmt nur noch Knoten 0 an der Arbitrierung teil und fordert anschließend eine Route zu Knoten 9 mit zwei Slots an. Dadurch wird die einzige hop-minimale Route $0 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9$ etabliert.

Da diese Route länger als vier Hops ist, legt hier – im Gegensatz zu den vorherigen Simulationen – nicht die Quelle die Slots für die gesamte Route fest. Knoten 5 legt die Slots für den Link $(8, 9)$ fest. Da für $(5, 6)$, $(6, 7)$, $(7, 8)$ und $(8, 9)$ jeweils alle Slots verwendbar sind, hat LCFP keinen Einfluss. SDFP wählt zunächst die Slots 98 und 99, wie Listing D.5 zeigt. Die RREPD-Pakete enthalten jeweils ID und RouteID des Senders, die aktuelle Routenlänge zum Ziel, die maximale Routenlänge, die freien Sendeslots des 2-Hop-Nachfolgers, des 1-Hop-Nachfolgers und des Senders (zum Senden an den jeweiligen Routennachfolger), die freien Empfangsslots des Senders (zum Empfangen von einem beliebigen Kommunikationsnachbarn) sowie den kleinsten vergebenen Sendeslot für den 3-Hop-Nachfolger.

Analog wählt Knoten 4 mittels SDFP die Slots 96 und 97 für $(7, 8)$. Anschließend muss Knoten 0 die Slots für $(6, 7)$ festlegen. Auf $(0, 4)$ sind die Slots 98 und 99 nicht nutzbar. Diese sind jedoch auch für $(6, 7)$ nicht nutzbar, da sie bereits für $(8, 9)$ verwendet wurden. Somit hat LCFP keinen Einfluss, und SDFP wählt die Slots 94 und 95. Anschließend legt 0 die Slots für $(5, 6)$ fest. Die Slots 98 und 99 werden auch hier nicht berücksichtigt (aufgrund der 2-Hop-Interferenzannahme). Somit wählt SDFP die Slots 92 und 93. Für $(4, 5)$ sind 98 und 99 wieder verwendbar und werden durch LCFP ausgewählt. Für $(0, 4)$ schließlich hat LCFP keinen Einfluss, und SDFP wählt die Slots 96 und 97.

In Phase 3 stellt Knoten 8 jedoch fest, dass die zum Senden vorgesehenen Slots 98 und 99 bereits durch den Interferenznachbarn 5 zum Senden blockiert sind. Es werden deshalb die ersten verfügbaren Slots 0 und 1 als alternative Sendeslots verwendet.

```
*** TRANSITION START
*       PId     : <<Block Nodes:5/Block rbbqr>> MainProc:5
*       Service: RouteSearch
*       Input   : RREPD
*       Now     : 0.5743
*       Parameter(s) : 6, 0, 3, 8, [ 0, 1, ..., 98, 99 ], [ 0, 1, ..., 98,
    99 ], [ 0, 1, ..., 98, 99 ], [ 0, 1, ..., 98, 99 ], -1
*   PROCEDURE RETURN calcChosenSlots : [ 98, 99 ]
...
*** TRANSITION START
*       PId     : <<Block Nodes:5/Block rbbqr>> MainProc:5
*       Service: RouteSearch
*       Now     : 0.6056
*   OUTPUT of RREPD to <<Block Nodes:5/Block rbbqr>> CoDex:5
*       Parameter(s) : 5, 0, 4, 8, [ 0, 1, ..., 96, 97 ], [ 0, 1, ..., 96,
    97 ], [ 0, 1, ..., 96, 97 ], [ 0, 1, ..., 98, 99 ], 98
...
```

Listing D.5.: Initiale Auswahl der Slots für Link $(8, 9)$ durch Knoten 5.

D.3. Dritte Simulation

Die dritte Simulation verwendet das Netzwerk aus Abbildung D.3, welches ähnlich zum zweiten Netzwerk ist. Es gilt ebenfalls $n_{maxNodes} = 10$ und $n_{maxHops} = 9$. Die 2-Hop-Interferenzannahme ist auch hier nicht erfüllt, da sich die Knoten 4 und 9 in Interferenzreichweite befinden, obwohl sie 5-Hop-Kommunikationsnachbarn sind.

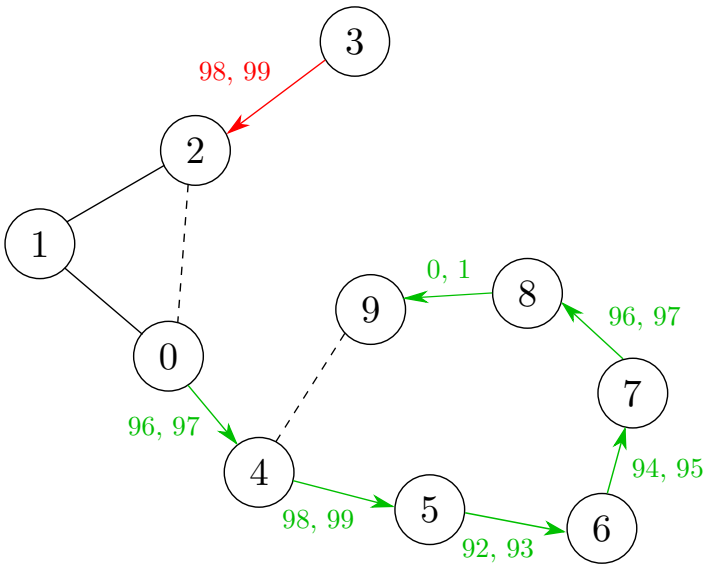

Abbildung D.3.: Netzwerk für dritte funktionale Simulation.

Analog zur zweiten Simulation werden die beiden Routen $3 \rightarrow 2$ sowie $0 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9$ etabliert. Die initiale Verteilung der Slots entspricht ebenfalls der zweiten Simulation. Bei der Etablierung der zweiten Route stellt Knoten 9 jedoch in Phase 3 nach Erhalt des CREQD-Pakets fest, dass die zum Empfangen vorgesehenen Slots 98 und 99 bereits durch den Interferenznachbarn 4 zum Empfangen blockiert sind. Knoten 9 sendet daraufhin

ein CREQC an 8, welcher alternative Sendeslots auswählt. Es werden die ersten verfügbaren Slots 0 und 1 gewählt und mit einem CREQU propagiert. Listing D.6 zeigt den Ablauf.

```
*** TRANSITION START
*       PId     : <<Block Nodes:9/Block rbbqr>> MainProc:9
*       Service: RouteSearch
*       Input   : CREQD
*       Now     : 0.8636
*       Parameter(s) : 0, 0, 5, [  ], [  ], [  ]
*   PROCEDURE RETURN getBrxiSlots : [ 98, 99 ]
...
*** TRANSITION START
*       PId     : <<Block Nodes:9/Block rbbqr>> MainProc:9
*       Service: RouteSearch
*       Now     : 0.8640
*   OUTPUT of CREQC to <<Block Nodes:9/Block rbbqr>> CoDex:9
*       Parameter(s) : 8, [ 98, 99 ], [ 0, 1, ..., 96, 97 ]
...
*** TRANSITION START
*       PId     : <<Block Nodes:8/Block rbbqr>> MainProc:8
*       Service: RouteSearch
*       Now     : 0.8901
*   OUTPUT of CREQU to <<Block Nodes:8/Block rbbqr>> CoDex:8
*       Parameter(s) : 8, [ 0, 1 ]
...
```

Listing D.6.: Auswahl und Propagierung alternativer Slots für Link $(8, 9)$ durch Knoten 8.

Das CREQD-Paket enthält die RouteIDs von Sender und Empfänger, die Routenlänge zur Quelle sowie die ausgewählten Sendeslots für den 1-, 2- und 3-Hop-Nachfolger. CREQC enthält die ID des Empfängers, die fehlreservierten Slots sowie die freien Empfangsslots des Senders. CREQU enthält die ID des Senders sowie dessen neu zum Senden reservierte Slots. Somit können die Interferenznachbarn des Senders von CREQU (außer dem Routennachfolger) eine Empfangsblockierung für diesen und die entsprechenden Slots in ihre Slottabelle eintragen. Hier wird das CREQU-Paket jedoch nur vom Routennachfolger (Knoten 9) empfangen, welcher entsprechende Empfangsreservierungen vornimmt.

E Anhang E.

Verwendung der Simulationskomponente ns-3

Listing E.1 zeigt die generelle Verwendung der Simulationskomponente ns-3 im Kontext von FERAL-Simulationssystemen.

```
1  SimulationComponent valueGenerator = ...;
2  SimulationComponent valueConsumer = ...;
3
4  NS3TimeTriggeredComponent ns3Component = new NS3TimeTriggeredComponent();
5  NS3CommunicationMedium cc2420 = new NS3CC2420Medium();
6  cc2420.setCommType(NS3CommunicationType.mediumSpecificCommunication);
7
8  List<Port> genPorts = new ArrayList<Port>();
9  genPorts.add(valueGenerator.getPort("output"));
10 genPorts.add(valueGenerator.getPort("input"));
11 new NS3CommunicationEndpoint("GeneratorComponent", cc2420,
       valueGenerator, genPorts);
12
13 List<Port> consPorts = new ArrayList<Port>();
14 consPorts.add(valueConsumer.getPort("input"));
15 new NS3CommunicationEndpoint("ConsumerComponent", cc2420, valueConsumer,
       consPorts);
16
17 ns3Component.addMedium(cc2420);
```

Listing E.1.: Verwendung der Simulationskomponente ns-3.

Die Komponenten *valueGenerator* und *valueConsumer* kommunizieren über ein mittels ns-3 simuliertes Medium. Dabei erzeugt *valueGenerator* Werte und verschickt diese, während *valueConsumer* die empfangenen Werte verarbeitet.

Zunächst muss eine *NS3TimeTriggeredComponent* erstellt werden (Zeile 4), welche die Ansteuerung von ns-3 übernimmt, die Nachrichten überträgt sowie die simulierten Medien verwaltet. Anschließend wird ein entsprechendes Medium angelegt (Zeile 5; hier ein CC2420-Medium). Als Kommunikationsmodus wird *mediumSpecificCommunication* verwendet (Zeile 6), d. h. es werden CC2420-spezifische Nachrichten ausgetauscht. Danach muss für jede Simulationskomponente, die man mit diesem Medium verbinden will, ein *NS3CommunicationEndpoint* angelegt werden (Zeilen 11 und 15). Beim Erzeugen des Endpunkts kann man entweder automatisch einen Namen vergeben lassen oder selbst einen angeben, der dann allerdings eindeutig sein muss. Der Name dient zur Identifikation der zugehörigen *PcuApplication* auf ns-3-Seite. Weitere Parameter sind das Medium, die Simulationskomponente, zu der der Endpunkt gehört, sowie die zu verschaltenden Ports der Simulationskomponente. Diese werden zu einer Liste hinzugefügt und an den Konstruktor übergeben (Zeilen 8 – 11 und 13 – 15).

Es wird dann automatisch ermittelt, um welchen Porttyp es sich handelt. Da jeder Endpunkt genau einen *InputPort* und genau einen *OutputPort* hat, werden *OutputPorts* der Simulationskomponente mit dem *InputPort* des Endpunkts verbunden (und umgekehrt), und es wird jeweils ein entsprechender Link angelegt. Außerdem wird verifiziert, dass die angegebenen Ports alle zur übergebenen Simulationskomponente gehören.

Prinzipiell ist es ausreichend, einen *InputPort* und / oder *OutputPort* der Simulationskomponente mit dem Endpunkt zu verbinden. Die Möglichkeit, mehrere Ports eines Typs angeben zu können, wurde für existierende Simulationskomponenten vorgesehen, die bereits mehrere Ports eines Typs haben, auf die die Inputs / Outputs verteilt werden. Diese können dann im Endpunkt zusammengeführt werden.

Der erzeugte Endpunkt wird automatisch zu dem entsprechenden Medium hinzugefügt. Zum Schluss wird das Medium zur Simulationskomponente ns-3 hinzugefügt (Zeile 17).

Anmerkung: Zur Übertragung der generierten Werte müsste für *valueGenerator* nur der *OutputPort* verschaltet werden. Der *InputPort* wird jedoch verwendet, um beispielsweise *CC2420Cca-*, *CC2420Sending-* und *CC2420SendFinished*-Nachrichten verarbeiten zu können.

Bei der soeben beschriebenen Variante müssen alle zu verschaltenden Ports zu einer Liste hinzugefügt und an den Konstruktor von *NS3CommunicationEndpoint* übergeben werden. Oft tritt jedoch der Fall ein, dass genau ein *InputPort* und genau ein *OutputPort* mit dem Endpunkt verschaltet werden sollen, womit diese Schreibweise eher umständlich ist. Weiterhin ist es möglich, dass für den Endpunkt ein *InputPort* und ein *OutputPort* verwendet werden sollen, die zu *verschiedenen* Simulationskomponenten gehören. Dies ist beispielsweise der Fall, wenn Bridges (s. Abschnitt 7.3 bzw. [BCG+13, BCG+14]) verwendet werden. In diesem Fall gibt es für beide Kommunikationsrichtungen jeweils eine separate Bridge, was es erforderlich macht, dass die übergebenen Ports zu unterschiedlichen Simulationskomponenten gehören können. Deshalb gibt es zusätzliche Konstruktoren für *NS3CommunicationEndpoint*, denen jeweils genau ein *InputPort* und genau ein *OutputPort* (die zu verschiedenen Komponenten gehören können) übergeben wird. Es wird in diesem Fall keine Simulationskomponente übergeben, da nicht geprüft werden muss, ob die Ports alle zur selben Komponente gehören. Listing E.2 zeigt ein Beispiel.

```
new NS3CommunicationEndpoint("GeneratorComponent", cc2420,
    valueGenerator.getInputPort("input"),
    valueGenerator.getOutputPort("output"));
```

Listing E.2.: Verwendung eines *InputPorts* und eines *OutputPorts*.

Drahtlose Medien haben zusätzlich eine *MobilityConfig*, welche das verwendete Mobilitätsmodell repräsentiert. Diese wird in Form von String-Werten an ns-3 übergeben. Es können hier alle Werte angegeben werden, die ns-3 verarbeiten kann. Die *MobilityConfig* weist den Knoten eine initiale Position zu. Zudem wird ein Bewegungsmodell definiert, welches die Bewegung der Knoten während der Simulation festlegt. Wenn der Benutzer keine *MobilityConfig* angibt, so werden die Knoten initial auf einem zweidimensionalen Gitter angeordnet, und es wird ein *ConstantPositionMobilityModel* verwendet, d. h. die Knoten bewegen sich nicht. Listing E.3 zeigt die standardmäßig verwendete *MobilityConfig*.

Die *positionAllocAttributes* wurden aus einem von ns-3 bereitgestellten Beispiel entnommen. Die Knoten werden auf einem Gitter angeordnet, welches durch ein kartesisches Koordinatensystem beschrieben wird. Dazu wird ein *GridPositionAllocator* verwendet. Die Attribute *MinX* und *MinY* geben an, an welcher X- bzw. Y-Koordinate das Gitter beginnt, während

```
String positionAllocator = "ns3::GridPositionAllocator";
Map<String,String> positionAllocAttributes = new HashMap<String,String>();
positionAllocAttributes.put("MinX", "0.0");
positionAllocAttributes.put("MinY", "0.0");
positionAllocAttributes.put("DeltaX", "5.0");
positionAllocAttributes.put("DeltaY", "10.0");
positionAllocAttributes.put("GridWidth", "3");
positionAllocAttributes.put("LayoutType", "RowFirst");
String mobilityModel = "ns3::ConstantPositionMobilityModel";
```

Listing E.3.: Standardmäßig verwendete *MobilityConfig*.

DeltaX und *DeltaY* beschreiben, wie viel Abstand jeweils zwischen zwei Knoten gelassen wird. *GridWidth* gibt an, wie viele Knoten jeweils in einer Reihe oder Spalte angeordnet werden, und *LayoutType* beschreibt, ob Reihen oder Spalten verwendet werden (*RowFirst* oder *ColumnFirst*).

Will man nicht die Standardkonfiguration verwenden, so kann man eine benutzerdefinierte *MobilityConfig* erstellen. Listing E.4 zeigt ein Beispiel.

```
String positionAllocator = "ns3::GridPositionAllocator";
Map<String,String> positionAllocAttributes = new HashMap<String,String>();
positionAllocAttributes.put(...); // s. Listing E.3
String mobilityModel = "ns3::RandomWalk2dMobilityModel";
Map<String,String> mobilityModelAttributes = new HashMap<String,String>();
mobilityModelAttributes.put("Bounds", "-50|50|-50|50");

NS3MobilityConfig mob = new NS3MobilityConfig(positionAllocator,
    positionAllocAttributes, mobilityModel, mobilityModelAttributes);
NS3CommunicationMedium cc2420 = new NS3CC2420Medium(mob);
```

Listing E.4.: Verwendung einer benutzerdefinierten *MobilityConfig*.

Hier werden den Knoten dieselben initialen Positionen zugewiesen wie in Listing E.3. Statt des *ConstantPositionMobilityModels* wird jedoch ein *RandomWalk2dMobilityModel* verwendet. Bei diesem bewegen sich die Knoten mit einer zufälligen Geschwindigkeit und Richtung innerhalb des spezifizierten Rechtecks.

Da die Konfiguration sich hier immer auf ein Medium als Ganzes bezieht, ist es nicht möglich, einzelnen Knoten explizite Koordinaten zuzuweisen. Die Verwendung der *MobilityConfig* in FERAL ist jedoch ähnlich zu der in ns-3, da dort i. d. R. ein Mobilitätsmodell auf allen Knoten eines Containers installiert wird.

F

Anhang F.

Zustandsautomaten des CC2420-Transceivers und -Simulationsmoduls

Abbildung F.1 stellt den vollständigen Zustandsautomaten des CC2420-Transceivers dar, soweit dieser im Datenblatt zu finden ist. Der Teil des Automaten, der im Simulationsmodul umgesetzt wurde, ist blau markiert.

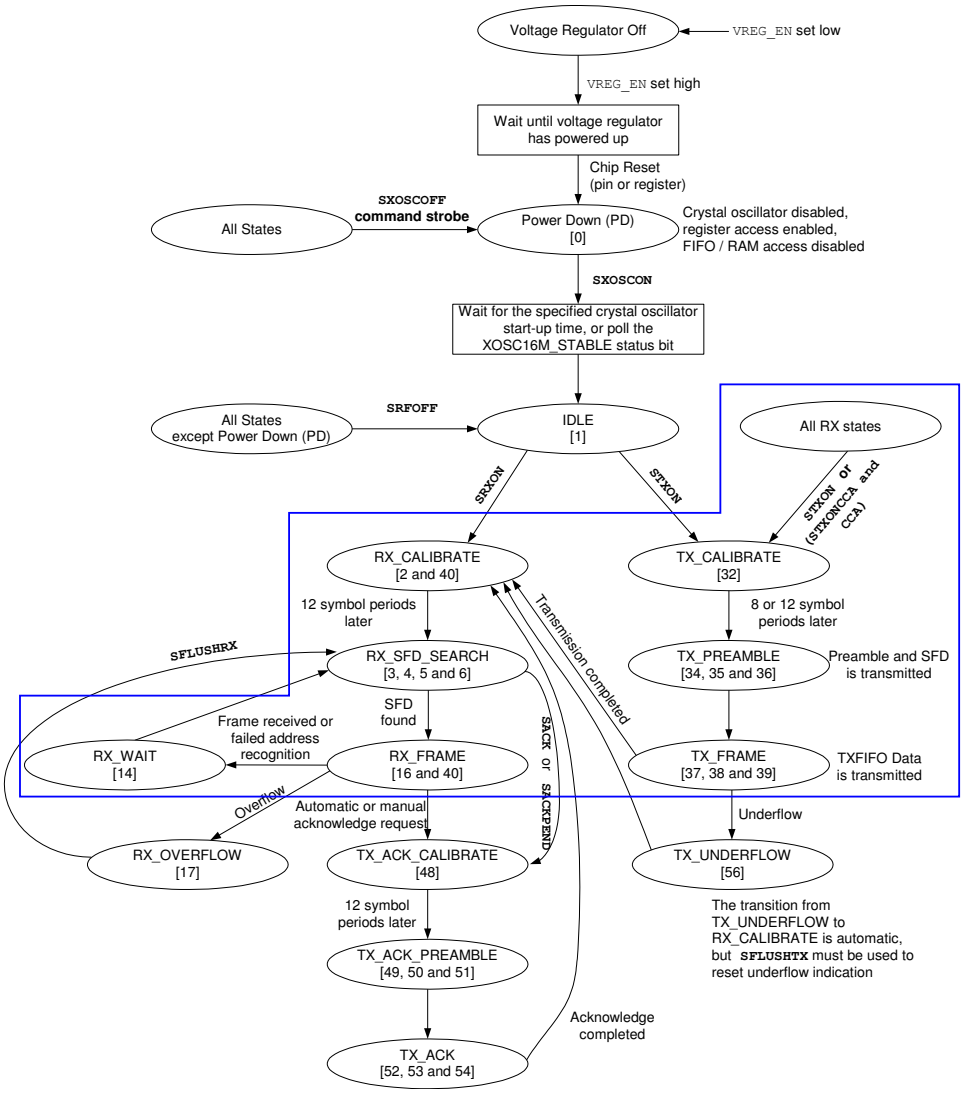

Abbildung F.1.: Zustandsautomat des CC2420-Transceivers [Tex13].

Der zugehörige Automat des Simulationsmoduls kombiniert den Zustand des Transceivers mit dem des Mediums. Er ist in Abbildung F.2 dargestellt und stammt im Wesentlichen von der für die Vorgänger von FERAL entwickelten CC2420-Simulationskomponente (vgl. [Kuh09]). Die Farben der Kanten kennzeichnen die unterschiedlichen Auslöser für Zustandsübergänge.

Tabelle F.1 gibt einen Überblick über die verwendeten Zustände. Die mit RX beginnenden Zustände beschreiben die Empfangszustände des Simulationsmoduls, die mit TX beginnenden die Sendezustände.

Zustand	**Bedeutung**
RXCAL	Receiver kalibriert, Medium ist frei
RXCALNOISE	Receiver kalibriert, Medium ist belegt
RX	CCA ist ungültig (erste acht Symbole im RX-Modus), Medium ist frei
RXCCA	CCA ist gültig, Medium ist frei
RXHDR	Empfang eines Paketheaders (Präambel, SFD)
RXHDRCAP	Empfang eines Paketheaders, zweites Paket ist zu schwach für Kollision, beeinflusst aber CCA
RXDATA	Empfang von Paketdaten (Paketheader wurde vollständig empfangen)
RXDATACAP	Empfang von Paketdaten, zweites Paket ist zu schwach für Kollision, beeinflusst aber CCA
RXCOLL	Header vollständig empfangen, Daten durch Kollision zerstört
RXNOISE	Empfang von Rauschen (zerstörter Header), CCA ist ungültig
RXNOISECCA	Empfang von Rauschen, CCA ist gültig
TXCAL	Transmitter kalibriert, Medium ist frei
TXCALNOISE	Transmitter kalibriert, Medium ist belegt
TX	Sendevorgang läuft, kein kollidierendes Paket auf dem Medium
TXNOISE	Sendevorgang läuft, kollidierendes Paket auf dem Medium

Tabelle F.1.: Zustände des CC2420-Simulationsmoduls.

Die Zustände des Simulationsmoduls entsprechen nicht genau denen des Transceivers, da in der Simulation zusätzlich der Zustand des Mediums berücksichtigt werden muss. Im Wesentlichen wird ein Zustand des Transceivers auf zwei Zustände des Simulationsmoduls abgebildet; jedoch werden manche Zustände des Transceivers im Simulationsmodul zusammengefasst und andere auf mehrere aufgeteilt. Konkret wird folgende Abbildung der Zustände verwendet:

- RX_CALIBRATE wird auf RXCAL und RXCALNOISE abgebildet (Unterscheidung zwischen freiem und belegtem Medium).
- RX_SFD_SEARCH wird auf RX, RXNOISE, RXCCA, RXNOISECCA, RXHDR und RXHDRCAP abgebildet (Unterscheidung zwischen freiem und belegtem Medium, ungültigem und gültigem CCA sowie danach, ob gerade ein Header empfangen wird und ob es ein weiteres Paket gibt, welches keine Kollision verursacht).

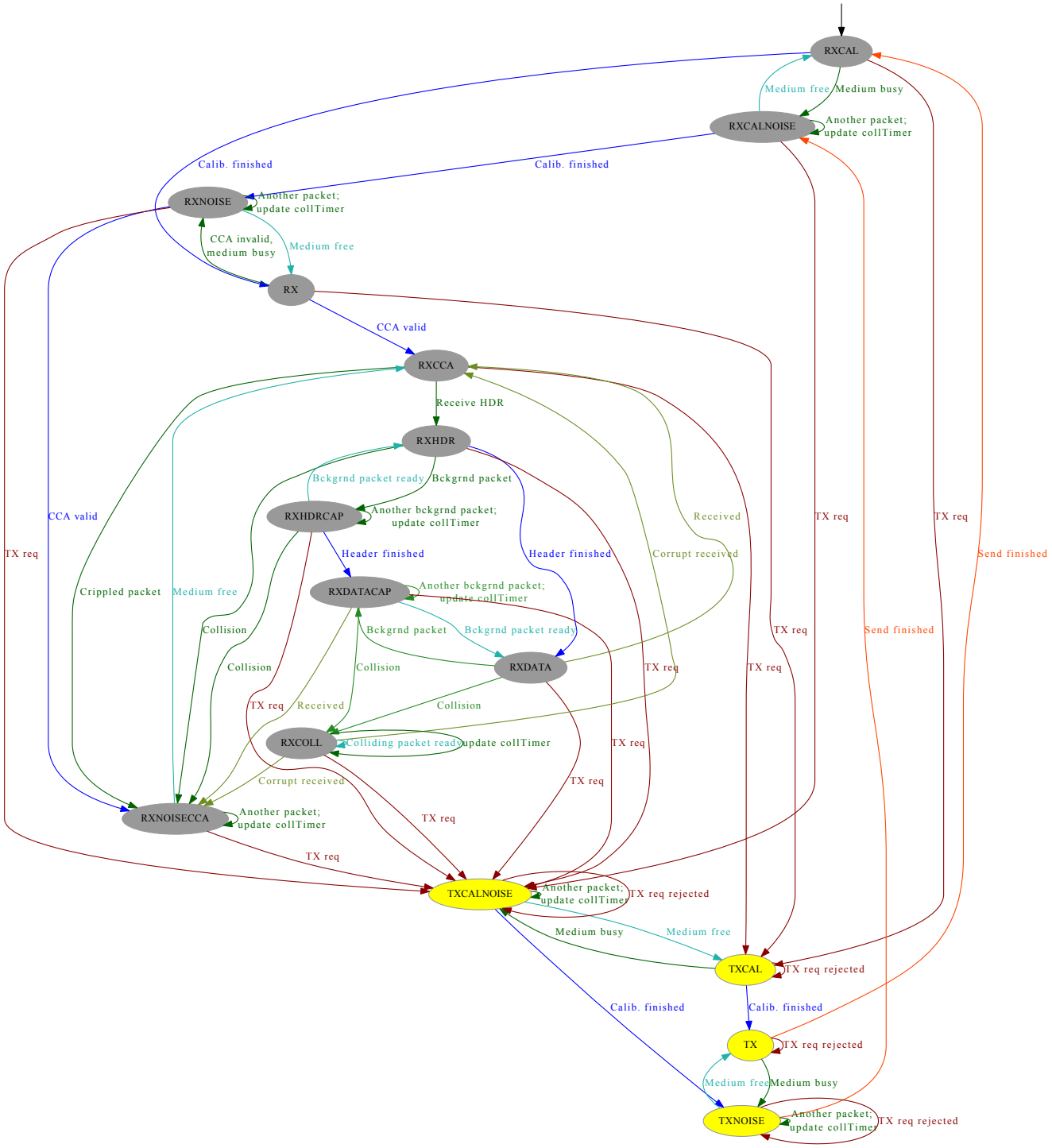

Abbildung F.2.: Zustandsautomat des CC2420-Simulationsmoduls.
Verwendete Farben: rxFromMedium, rxTimerExpired, collTimerExpired, txRequest, txTimerExpired, stateTimerExpired.

- RX_FRAME wird auf RXDATA und RXDATACAP abgebildet (abhängig davon, ob es noch ein weiteres Paket gibt, das keine Kollision verursacht).
- Den Zustand RX_WAIT gibt es in der Form in der Simulation nicht; er ist implizit in RXCCA, RXNOISECCA und RXCOLL enthalten. Ferner ist im Simulationsmodul keine Adresserkennung vorhanden (einer der Übergänge von RX_FRAME nach RX_-WAIT).
- TX_CALIBRATE wird auf TXCAL und TXCALNOISE abgebildet.
- TX_PREAMBLE und TX_FRAME werden zusammengefasst und auf TX und TX-NOISE abgebildet (beim Senden unterscheidet das Simulationsmodul nicht zwischen Header und Daten).
- Der Übergang aus allen RX-Zuständen nach TX_CALIBRATE entspricht dem Übergang aus den RX-Zuständen des Simulationsmoduls in die Sendezustände TXCAL und TXCALNOISE (je nach Zustand des Mediums).

Publikationsliste

[1] Braun, T. ; Christmann, D. ; Gotzhein, R. ; Igel, A.: Model-driven Engineering of Networked Ambient Systems with SDL-MDD. In: *Proceedings of the 3rd International Conference on Ambient Systems, Networks and Technologies (ANT 2012), the 9th International Conference on Mobile Web Information Systems (MobiWIS-2012), Niagara Falls, Ontario, Canada, August 27-29, 2012*, Bd. 10, Elsevier, 2012 (Procedia Computer Science), S. 490–498. http://dx.doi.org/10.1016/j.procs.2012.06.063

[2] Braun, T. ; Christmann, D. ; Gotzhein, R. ; Igel, A. ; Forster, T. ; Kuhn, T.: Virtual Prototyping with Feral – Adaptation and Application of a Simulator Framework. In: *The 24th IASTED International Conference on Modelling and Simulation*, 2013, S. 270–279

[3] Braun, T. ; Christmann, D. ; Gotzhein, R. ; Igel, A. ; Kuhn, T.: Virtual Prototyping of Distributed Embedded Systems with FERAL. In: *International Journal of Modelling and Simulation* 2 (2014), Nr. 34. http://dx.doi.org/10.2316/Journal.205.2014.2.205-5968

[4] Gebhardt, J. ; Gotzhein, R. ; Igel, A. ; Kramer, C.: QoS Multicast Routing in Partially Mobile Wireless TDMA Networks. In: *IEEE Global Communications Conference (GLOBECOM 2015), San Diego, USA, December 06-10, 2015*, IEEE, 2015

[5] Igel, A. ; Gotzhein, R.: QoS-Routing und Reservierungen in mobilen Ad-Hoc-Netzwerken. In: *Fachgespräch Sensornetze, 15.-16. September 2011, Paderborn*, 2011

[6] Igel, A. ; Gotzhein, R.: An Analysis of the Interference Problem in Wireless TDMA Networks. In: *The Eighth International Conference on Wireless and Mobile Communications (ICWMC 2012)*, IARIA, 2012, S. 187–194

[7] Igel, A. ; Gotzhein, R.: A CC2420 Transceiver Simulation Module for ns-3 and its Integration into the FERAL Simulator Framework. In: *The Fifth International Conference on Advances in System Simulation (SIMUL 13)*, IARIA, 2013, S. 156–164

Literaturverzeichnis

[Atm09] Atmel: *Datasheet AT86RF230*. http://www.atmel.com/dyn/resources/prod_documents/doc5131.pdf, 2009

[Atm11] Atmel: *Datasheet ATmega128*. http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf, 2011. – Revision 2467X-AVR-06/11

[BBCG11] Becker, Philipp ; Birtel, Martin ; Christmann, Dennis ; Gotzhein, Reinhard: Black-Burst-Based Quality-of-Service Routing (BBQR) for Wireless Ad-Hoc Networks. In: *11th International Conference on New Technologies in Distributed Systems (NOTERE'2011), Paris, France*, IEEE, 2011, S. 1–8. http://dx.doi.org/10.1109/NOTERE.2011.5957973

[BCG+13] Braun, Tobias ; Christmann, Dennis ; Gotzhein, Reinhard ; Igel, Anuschka ; Forster, Thomas ; Kuhn, Thomas: Virtual Prototyping with Feral – Adaptation and Application of a Simulator Framework. In: *The 24th IASTED International Conference on Modelling and Simulation*, 2013, S. 270–279

[BCG+14] Braun, Tobias ; Christmann, Dennis ; Gotzhein, Reinhard ; Igel, Anuschka ; Kuhn, Thomas: Virtual Prototyping of Distributed Embedded Systems with FERAL. In: *International Journal of Modelling and Simulation* 2 (2014), Nr. 34. http://dx.doi.org/10.2316/Journal.205.2014.2.205-5968

[BCGI12] Braun, Tobias ; Christmann, Dennis ; Gotzhein, Reinhard ; Igel, Anuschka: Model-driven Engineering of Networked Ambient Systems with SDL-MDD. In: *Proceedings of the 3rd International Conference on Ambient Systems, Networks and Technologies (ANT 2012), the 9th International Conference on Mobile Web Information Systems (MobiWIS-2012), Niagara Falls, Ontario, Canada, August 27-29, 2012*, Bd. 10, Elsevier, 2012 (Procedia Computer Science), S. 490–498. http://dx.doi.org/10.1016/j.procs.2012.06.063

[Bec07] Becker, Philipp: QoS Routing Protocols for Mobile Ad-hoc Networks – A Survey / Technische Universität Kaiserslautern, Fachbereich Informatik. 2007 (368/08). – Forschungsbericht. – https://kluedo.ub.uni-kl.de/files/1999/BelAmI_D2.5.01_-_QoS_Routing_Survey_-_Technical_Report.pdf

[BF10] Brumbulli, Mihal ; Fischer, Joachim: SDL Code Generation for Network Simulators. In: *System Analysis and Modeling: About Models - 6th International Workshop, SAM 2010, Oslo, Norway, October 4-5, 2010, Revised Selected Papers*, Bd. 6598, Springer, 2010 (Lecture Notes in Computer Science), S. 144–155. http://dx.doi.org/10.1007/978-3-642-21652-7_9

[BF12] Brumbulli, Mihal ; Fischer, Joachim: Simulation Configuration Modeling of Distributed Communication Systems. In: *System Analysis and Modeling:*

Theory and Practice - 7th International Workshop, SAM 2012, Innsbruck, Austria, October 1-2, 2012. Revised Selected Papers, Bd. 7744, Springer, 2012 (Lecture Notes in Computer Science), S. 198–211. http://dx.doi.org/10.1007/978-3-642-36757-1_12

[Bir09] Birtel, Martin: *Formal Specification and Evaluation of Black Burst Quality-of-Service Routing*, Technische Universität Kaiserslautern, Fachbereich Informatik, Diplomarbeit, 2009

[BM09] Baldo, Nicola ; Miozzo, Marco: Spectrum-aware Channel and PHY layer modeling for ns3. In: *4th International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS '09, Pisa, Italy, October 20-22, 2009*, ICST/ACM, 2009, S. 2:1–2:8. http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7647

[CAH96] Campbell, Andrew ; Aurrecoechea, Cristina ; Hauw, Linda: A Review of QoS Architectures. In: *Multimedia Systems* 6 (1996), S. 138–151

[Car10] Cardieri, Paulo: Modeling Interference in Wireless Ad Hoc Networks. In: *IEEE Communications Surveys and Tutorials* 12 (2010), Nr. 4, S. 551–572. http://dx.doi.org/10.1109/SURV.2010.032710.00096

[CF94] Chlamtac, Imrich ; Faragó, András: Making Transmission Schedules Immune to Topology Changes in Multi-Hop Packet Radio Networks. In: *IEEE/ACM Trans. Netw.* 2 (1994), Nr. 1, S. 23–29. http://dx.doi.org/10.1109/90.282605

[CF08] Christophides, Fanos ; Friderikos, Vasilis: Iterative hybrid graph and interference aware scheduling algorithm for STDMA networks. In: *Electronics Letters* 44 (2008), Nr. 8, S. 558–559. http://dx.doi.org/10.1049/el:20083260

[CGK09] Christmann, Dennis ; Gotzhein, Reinhard ; Kuhn, Thomas: Multi-hop Clock Synchronization in Wireless Ad-Hoc Networks. In: *ECEASST* 17 (2009). http://journal.ub.tu-berlin.de/eceasst/article/viewFile/219/201

[CGR12] Christmann, Dennis ; Gotzhein, Reinhard ; Rohr, Stephan: The Arbitrating Value Transfer Protocol (AVTP) – Deterministic Binary Countdown in Wireless Multi-Hop Networks. In: *21st International Conference on Computer Communications and Networks, ICCCN 2012, Munich, Germany, July 30 - August 2, 2012*, IEEE, 2012, S. 1–9. http://dx.doi.org/10.1109/ICCCN.2012.6289227

[Chr10] Christmann, Dennis: On the Behavior of Black Bursts in Tick-Synchronized Networks / Technische Universität Kaiserslautern, Fachbereich Informatik. 2010 (377/10). – Forschungsbericht. – http://vs.cs.uni-kl.de/publications/2010/Ch10/

[Chr15] Christmann, Dennis: *Distributed Real-time Systems – Deterministic Protocols for Wireless Networks and Model-Driven Development with SDL*, Technische Universität Kaiserslautern, Fachbereich Informatik, Dissertation, 2015. – https://kluedo.ub.uni-kl.de/files/4112/Dissertation_Christmann.pdf

[CKL02] Chen, Yuh-Shyan ; Ko, Yun-Wen ; Lin, Ting-Lung: A Lantern-Tree-Based QoS Multicast Protocol for Wireless Ad-Hoc Networks. In: *Proceedings of the 11th*

International Conference on Computer Communications and Networks, ICCCN 2002, 14-16 October, 2002, Hyatt Regency Miami, Miami, Florida, USA, IEEE, 2002, S. 242–247. http://dx.doi.org/10.1109/ICCCN.2002.1043073

[CLL07] Chen, Yuh-Shyan ; Lin, Tsung-Hung ; Lin, Yun-Wei: A hexagonal-tree TDMA-based QoS multicasting protocol for wireless mobile ad hoc networks. In: *Telecommunication Systems* 35 (2007), Nr. 1-2, S. 1–20. http://dx.doi.org/10.1007/s11235-007-9037-1

[CN98] Chen, Shigang ; Nahrstedt, Klara: An Overview of Quality of Service Routing for Next-Generation High-Speed Networks: Problems and Solutions. In: *Network, IEEE* 12 (1998), Nr. 6, S. 64–79. http://dx.doi.org/10.1109/65.752646

[CN99] Chen, Shigang ; Nahrstedt, Klara: Distributed Quality-of-Service Routing in Ad-Hoc Networks. In: *IEEE Journal on Selected Areas in Communications* 17 (1999), Nr. 8, S. 1488–1505. http://dx.doi.org/10.1109/49.780354

[CTSK04] Chen, Yuh-Shyan ; Tseng, Yu-Chee ; Sheu, Jang-Ping ; Kuo, Po-Hsuen: An On-Demand, Link-State, Multi-Path QoS Routing in a Wireless Mobile Ad-Hoc Network. In: *Computer Communications* 27 (2004), Nr. 1, S. 27–40. http://dx.doi.org/10.1016/S0140-3664(03)00176-2

[DBT05] Dousse, Olivier ; Baccelli, François ; Thiran, Patrick: Impact of interferences on connectivity in ad hoc networks. In: *IEEE/ACM Trans. Netw.* 13 (2005), Nr. 2, S. 425–436. http://dx.doi.org/10.1109/TNET.2005.845546

[DDWQ02] De, Swades ; Das, Sajal K. ; Wu, Hongyi ; Qiao, Chunming: Trigger-Based Distributed QoS Routing in Mobile Ad Hoc Networks. In: *Mobile Computing and Communications Review* 6 (2002), Nr. 3, S. 22–35. http://dx.doi.org/10.1145/581291.581298

[EHS97] Ellsberger, Jan ; Hogrefe, Dieter ; Sarma, Amardeo: *SDL – Formal Object-oriented Language for Communication Systems.* Prentice Hall, 1997

[EJL+03] Eker, Johan ; Janneck, Jörn W. ; Lee, Edward A. ; Liu, Jie ; Liu, Xiaojun ; Ludvig, Jozsef ; Neuendorffer, Stephen ; Sachs, Sonia R. ; Xiong, Yuhong: Taming Heterogeneity - The Ptolemy Approach. In: *Proceedings of the IEEE* 91 (2003), Nr. 1, S. 127–144. http://dx.doi.org/10.1109/JPROC.2002.805829

[Ets01] Etschberger, Konrad: *Controller Area Network: Basics, Protocols, Chips and Applications.* IXXAT Press, 2001. – ISBN 9783000073762

[FGJ+05] Fliege, Ingmar ; Geraldy, Alexander ; Jung, Simon ; Kuhn, Thomas ; Webel, Christian ; Weber, Christian: Konzept und Struktur des SDL Environment Framework (SEnF) / Technische Universität Kaiserslautern, Fachbereich Informatik. 2005 (341/05). – Forschungsbericht. – http://vs.cs.uni-kl.de/publications/2005/FlGe+05/

[FGW06] Fliege, Ingmar ; Grammes, Rüdiger ; Weber, Christian: ConTraST – A Configurable SDL Transpiler and Runtime Environment. In: *System Analysis and Modeling: Language Profiles, 5th International Workshop, SAM 2006, Kaiserslautern, Germany, May 31 - June 2, 2006, Revised Selected Papers*,

Bd. 4320, Springer, 2006 (Lecture Notes in Computer Science), S. 216–228. http://dx.doi.org/10.1007/11951148_14

[FID+10] Font, Juan L. ; Iñigo, Pablo ; Domínguez, Manuel ; Sevillano, José Luis ; Amaya, Claudio: Architecture, design and source code comparison of ns-2 and ns-3 network simulators. In: *Proceedings of the 2010 Spring Simulation Multiconference, SpringSim 2010, Orlando, Florida, USA, April 11-15, 2010*, SCS/ACM, 2010, S. 109:1–109:8. http://dx.doi.org/10.1145/1878537.1878651

[Fle05] FlexRay Consortium: *FlexRay Requirements Specification Version 2.1.* 2005

[Fle10] FlexRay Consortium: *FlexRay Communication System Protocol Specification Version 3.0.1.* 2010

[Fli09] Fliege, Ingmar: *Component-based Development of Communication Systems*, Technische Universität Kaiserslautern, Fachbereich Informatik, Dissertation, 2009

[For11] Forster, Otto: *Analysis 2: Differentialrechnung im $\mathbb{R}^n$, gewöhnliche Differentialgleichungen.* 9. Auflage. Vieweg+Teubner Verlag, 2011 (Grundkurs Mathematik). – ISBN 9783834805751

[Geb15] Gebhardt, Johann: *QoS Multicast Routing in Partially Mobile, TDMA-based Networks*, Technische Universität Kaiserslautern, Fachbereich Informatik, Masterarbeit, 2015

[GGIK15] Gebhardt, Johann ; Gotzhein, Reinhard ; Igel, Anuschka ; Kramer, Christopher: QoS Multicast Routing in Partially Mobile Wireless TDMA Networks. In: *IEEE Global Communications Conference (GLOBECOM 2015), San Diego, USA, December 06-10, 2015*, IEEE, 2015

[GJTW05] Gupta, Rajarshi ; Jia, Zhanfeng ; Tung, Teresa ; Walrand, Jean C.: Interference-aware QoS Routing (IQRouting) for Ad-Hoc Networks. In: *Proceedings of the Global Telecommunications Conference, 2005. GLOBECOM '05, St. Louis, Missouri, USA, 28 November - 2 December 2005*, IEEE, 2005, S. 2599–2604. http://dx.doi.org/10.1109/GLOCOM.2005.1578231

[GK00] Gupta, Piyush ; Kumar, Panganamala R.: The Capacity of Wireless Networks. In: *IEEE Transactions on Information Theory* 46 (2000), Nr. 2, S. 388–404. http://dx.doi.org/10.1109/18.825799

[GK11] Gotzhein, Reinhard ; Kuhn, Thomas: Black Burst Synchronization (BBS) – A Protocol for Deterministic Tick and Time Synchronization in Wireless Networks. In: *Computer Networks* 55 (2011), Nr. 13, S. 3015–3031. http://dx.doi.org/10.1016/j.comnet.2011.05.014

[Gro12] Groh, Robert: *Beiträge zur Integration von CC2420 und SDL in ns-3*, Technische Universität Kaiserslautern, Fachbereich Informatik, Bachelorarbeit, 2012

[HST06] Hsu, Chih-Shun ; Sheu, Jang-Ping ; Tung, Shen-Chien: An On-Demand Bandwidth Reservation QoS Routing Protocol for Mobile Ad Hoc Networks. In: *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC 2006), 5-7 June 2006, Taichung, Taiwan*, IEEE, 2006, S. 198–207. http://dx.doi.org/10.1109/SUTC.2006.39

[Hu93] Hu, Limin: Distributed Code Assignments for CDMA Packet Radio Networks. In: *Networking, IEEE/ACM Transactions on* 1 (1993), Nr. 6, S. 668–677. http://dx.doi.org/10.1109/90.266055

[IBM15] IBM Corporation: *Rational SDL Suite*. http://www-01.ibm.com/software/awdtools/sdlsuite/, 2015

[IG11] Igel, Anuschka ; Gotzhein, Reinhard: QoS-Routing und Reservierungen in mobilen Ad-Hoc-Netzwerken. In: *Fachgespräch Sensornetze, 15.-16. September 2011, Paderborn*, 2011

[IG12] Igel, Anuschka ; Gotzhein, Reinhard: An Analysis of the Interference Problem in Wireless TDMA Networks. In: *The Eighth International Conference on Wireless and Mobile Communications (ICWMC 2012)*, IARIA, 2012, S. 187–194

[IG13] Igel, Anuschka ; Gotzhein, Reinhard: A CC2420 Transceiver Simulation Module for ns-3 and its Integration into the FERAL Simulator Framework. In: *The Fifth International Conference on Advances in System Simulation (SIMUL 13)*, IARIA, 2013, S. 156–164

[Ins11] Institute of Electrical and Electronics Engineers: *IEEE Standard 802 Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. New York, NY, USA : IEEE Computer Society, 2011. – http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf

[Ins12a] Institute of Electrical and Electronics Engineers: *IEEE Standard 802 Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. New York, NY, USA : IEEE Computer Society, 2012. – http://standards.ieee.org/getieee802/download/802.11-2012.pdf

[Ins12b] Institute of Electrical and Electronics Engineers: *IEEE Standard for Ethernet*. New York, NY, USA : IEEE Computer Society, 2012. – http://standards.ieee.org/getieee802/download/802.3-2012_section1.pdf

[Int12a] International Telecommunication Union (ITU): *ITU-T Recommendation Z.100 (12/11) - Specification and Description Language - Overview of SDL-2010*. http://www.itu.int/rec/T-REC-Z.100/en, 2012

[Int12b] International Telecommunication Union (ITU): *ITU-T Recommendation Z.106 (12/11) - Specification and Description Language - Common Interchange Format for SDL-2010*. http://www.itu.int/rec/T-REC-Z.106-201112-I, 2012

[JGWV05] Jia, Zhanfeng ; Gupta, Rajarshi ; Walrand, Jean C. ; Varaiya, Pravin: Bandwidth Guaranteed Routing for Ad Hoc Networks with Interference Consideration. In: *Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC 2005), 27-30 June 2005, Murcia, Cartagena, Spain*, IEEE, 2005, S. 3–9. http://dx.doi.org/10.1109/ISCC.2005.37

[JLW+09] Joe, Hyunwoo ; Lee, Jonghyuk ; Woo, Duk-Kyun ; Mah, Pyeongsoo ; Kim, Hyungshin: Demo abstract: A High-Fidelity Sensor Network Simulator Using Accurate CC2420 Model. In: *Proceedings of the 8th International Conference on Information Processing in Sensor Networks, IPSN 2009, April 13-16, 2009, San Francisco, California, USA*, ACM, 2009, S. 429–430

[JPPQ05] Jain, Kamal ; Padhye, Jitendra ; Padmanabhan, Venkata N. ; Qiu, Lili: Impact of Interference on Multi-Hop Wireless Network Performance. In: *Wireless Networks* 11 (2005), Nr. 4, S. 471–487. http://dx.doi.org/10.1007/s11276-005-1769-9

[JW04] Jawhar, Imad ; Wu, Jie: A Race-Free Bandwidth Reservation Protocol for QoS Routing in Mobile Ad Hoc Networks. In: *37th Hawaii International Conference on System Sciences (HICSS-37 2004), CD-ROM / Abstracts Proceedings, 5-8 January 2004, Big Island, HI, USA*, IEEE, 2004. http://dx.doi.org/10.1109/HICSS.2004.1265693

[KB06] Kuhn, Thomas ; Becker, Philipp: A Simulator Interconnection Framework for the Accurate Performance Simulation of SDL Models. In: *System Analysis and Modeling: Language Profiles, 5th International Workshop, SAM 2006, Kaiserslautern, Germany, May 31 - June 2, 2006, Revised Selected Papers*, Bd. 4320, Springer, 2006 (Lecture Notes in Computer Science), S. 133–147. http://dx.doi.org/10.1007/11951148_9

[KCG15] Kramer, Christopher ; Christmann, Dennis ; Gotzhein, Reinhard: Automatic Topology Discovery in TDMA-based Ad Hoc Networks. In: *International Wireless Communications and Mobile Computing Conference, IWCMC 2015, Dubrovnik, Croatia, August 24-28, 2015*, IEEE, 2015, S. 634–639. http://dx.doi.org/10.1109/IWCMC.2015.7289157

[KFBG13] Kuhn, Thomas ; Forster, Thomas ; Braun, Tobias ; Gotzhein, Reinhard: FERAL – Framework for Simulator Coupling on Requirements and Architecture Level. In: *11th ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMCODE 2013, Portland, OR, USA, October 18-20, 2013*, IEEE, 2013, S. 11–22

[KG08] Kuhn, Thomas ; Gotzhein, Reinhard: Model-Driven Platform-Specific Testing through Configurable Simulations. In: *Model Driven Architecture - Foundations and Applications, 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings*, Bd. 5095, Springer, 2008 (Lecture Notes in Computer Science), S. 278–293. http://dx.doi.org/10.1007/978-3-540-69100-6_19

[KGGR05] Kuhn, Thomas ; Geraldy, Alexander ; Gotzhein, Reinhard ; Rothländer, Florian: ns+SDL - The Network Simulator for SDL Systems. In: *SDL 2005: Model Driven, 12th International SDL Forum, Grimstad, Norway, June 20-23, 2005, Proceedings*, Bd. 3530, Springer, 2005 (Lecture Notes in Computer Science), S. 103–116. http://dx.doi.org/10.1007/11506843_7

[Kra13] Kramer, Christopher: *Ermittlung des Netzzustands von Funk-Netzwerken*, Technische Universität Kaiserslautern, Fachbereich Informatik, Projektarbeit, 2013. – http://vs.cs.uni-kl.de/publications/2013/Kr13/

[Kuh09] Kuhn, Thomas: *Model Driven Development of MacZ – A QoS Medium Access Control Layer for Ambient Intelligence Systems*, Technische Universität Kaiserslautern, Fachbereich Informatik, Dissertation, 2009

[Lin01] Lin, Chunhung R.: On-Demand QoS Routing in Multihop Mobile Networks. In: *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, Bd. 3, IEEE, 2001, S. 1735–1744. http://dx.doi.org/10.1109/INFCOM.2001.916671

[LL99] Lin, Chunhung R. ; Liu, Jain-Shing: QoS Routing in Ad Hoc Wireless Networks. In: *IEEE Journal on Selected Areas in Communications* 17 (1999), Nr. 8, S. 1426–1438. http://dx.doi.org/10.1109/49.779924

[LLWC03] Levis, Philip ; Lee, Nelson ; Welsh, Matt ; Culler, David E.: TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In: *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys 2003, Los Angeles, California, USA, November 5-7, 2003*, ACM, 2003, S. 126–137. http://dx.doi.org/10.1145/958491.958506

[LRSG12] Labbé, Isabelle ; Roy, Jean-François ; St-Onge, Francis ; Gagnon, Benoit: Spatial Reuse and Interference-Aware Slot Scheduling in a Distributed TDMA MANET System. In: *The Eighth International Conference on Wireless and Mobile Communications (ICWMC 2012)*, IARIA, 2012, S. 294–303

[LTS02] Liao, Wen-Hwa ; Tseng, Yu-Chee ; Shih, Kuei-Ping: A TDMA-based Bandwidth Reservation Protocol for QoS Routing in a Wireless Mobile Ad Hoc Network. In: *IEEE International Conference on Communications, ICC 2002, April 28 - May 2, 2002, New York City, NY, USA*, IEEE, 2002, S. 3186–3190. http://dx.doi.org/10.1109/ICC.2002.997423

[LTWS01] Liao, Wen-Hwa ; Tseng, Yu-Chee ; Wang, Shu-Ling ; Sheu, Jang-Ping: A Multi-path QoS Routing Protocol in a Wireless Mobile ad Hoc Network. In: *Networking - ICN 2001, First International Conference, Colmar, France, July 9-13, 2001 Proceedings, Part 2*, Bd. 2094, Springer, 2001 (Lecture Notes in Computer Science), S. 158–167. http://dx.doi.org/10.1007/3-540-47734-9_16

[LWL08] Li, Fang ; Wang, Lifang ; Liao, Chenglin: CAN (Controller Area Network) Bus Communication System Based on Matlab/Simulink. In: *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, IEEE, 2008, S. 1–4. http://dx.doi.org/10.1109/WiCom.2008.1004

[Mat15] MathWorks: *Simulink - Simulation and Model-Based-Design.* http://www.mathworks.com/products/simulink/, 2015

[NCÇ+11] Narra, Hemanth ; Cheng, Yufei ; Çetinkaya, Egemen K. ; Rohrer, Justin P. ; Sterbenz, James P. G.: Destination-Sequenced Distance Vector (DSDV) Routing Protocol Implementation in ns-3. In: *4th International ICST Conference on Simulation Tools and Techniques, SIMUTools '11, Barcelona, Spain, March 22 - 24, 2011*, ICST/ACM, 2011, S. 439–446. http://dx.doi.org/10.4108/icst.simutools.2011.245588

[NG09] Nissler, Mattias ; Gotzhein, Reinhard: Performance evaluation of multi-path routing in reservation-based wireless networks. In: *Proceedings of the 12th International Symposium on Modeling Analysis and Simulation of Wireless and Mobile*

Systems, MSWiM 2009, Tenerife, Canary Islands, Spain, October 26-19, 2009, ACM, 2009, S. 268–273. http://dx.doi.org/10.1145/1641804.1641849

[Nis08] Nissler, Mattias: *Modeling and Analysis of Optimal Slot Allocations in Wireless Ad-Hoc Networks*, Technische Universität Kaiserslautern, Fachbereich Informatik, Diplomarbeit, 2008

[NL02] Nossal, Roman ; Lang, Roland: Model-Based System Development: An Approach to Building X-by-Wire Applications. In: *IEEE Micro* 22 (2002), Nr. 4, S. 56–63. http://dx.doi.org/10.1109/MM.2002.1028476

[NLT04] Ng, Jim M. ; Low, Chor P. ; Teo, Hui S.: On-demand QoS Multicast Routing and Reservation Protocol for MANETs. In: *Proceedings of the IEEE 15th International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2004, 5-8 September 2004, Barcelona, Spain*, IEEE, 2004, S. 2504–2508. http://dx.doi.org/10.1109/PIMRC.2004.1368771

[NS-15] NS-3 Consortium: *The ns-3 Network Simulator*. http://www.nsnam.org, 2015. – Project Homepage

[PB94] Perkins, Charles E. ; Bhagwat, Pravin: Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In: *Proceedings of the ACM SIGCOMM '94*, ACM, 1994, S. 234–244. http://dx.doi.org/10.1145/190314.190336

[PKG+10] Paul, Amrita B. ; Konwar, Shantanu ; Gogoi, Upola ; Chakraborty, Angshuman ; Yeshmin, Nilufar ; Nandi, Sukumar: Implementation and Performance Evaluation of AODV in Wireless Mesh Networks using NS-3. In: *Education Technology and Computer (ICETC), 2010 2nd International Conference on*, Bd. 5, IEEE, 2010, S. V5–298 – V5–303. http://dx.doi.org/10.1109/ICETC.2010.5530060

[PP08] Paz Alberola, Rodolfo de ; Pesch, Dirk: AvroraZ: Extending Avrora with an IEEE 802.15.4 Compliant Radio Chip Model. In: *Proceedings of the 3rd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks, PM2HW2N 2008, Vancouver, British Columbia, Canada, October 31, 2008*, ACM, 2008, S. 43–50. http://dx.doi.org/10.1145/1454630.1454637

[PR99] Perkins, Charles E. ; Royer, Elizabeth M.: Ad-hoc On-Demand Distance Vector Routing. In: *2nd Workshop on Mobile Computing Systems and Applications (WMCSA '99), February 25-26, 1999, New Orleans, LA, USA*, IEEE, 1999, S. 90–100. http://dx.doi.org/10.1109/MCSA.1999.749281

[SCCC06] Shih, Kuei-Ping ; Chang, Chih-Yung ; Chen, Yen-Da ; Chuang, Tsung-Han: Dynamic bandwidth allocation for QoS routing on TDMA-based mobile ad hoc networks. In: *Computer Communications* 29 (2006), Nr. 9, S. 1316–1329. http://dx.doi.org/10.1016/j.comcom.2005.10.009

[Sch03] Schiller, Jochen: *Mobile Communications*. 2. Auflage. Boston : Addison-Wesley, 2003

[SJK07] Suh, Changsu ; Joung, Jung-Eun ; Ko, Young-Bae: New RF Models of the TinyOS Simulator for IEEE 802.15.4 Standard. In: *IEEE Wireless Communications and Networking Conference, WCNC 2007, Hong Kong, China, 11-15 March, 2007*, IEEE, 2007, S. 2236–2240. http://dx.doi.org/10.1109/WCNC.2007.418

[SK99] Sobrinho, João L. ; Krishnakumar, Anjur S.: Quality-of-Service in Ad Hoc Carrier Sense Multiple Access Wireless Networks. In: *IEEE Journal on Selected Areas in Communications* 17 (1999), Nr. 8, S. 1353–1368. http://dx.doi.org/10.1109/49.779919

[SKG07] Siddique, Mohammad M. ; Könsgen, Andreas J. ; Görg, Carmelita: Vertical Coupling between Network Simulator and IEEE 802.11 Based Simulator. In: *Information and Communication Technology, 2007. ICICT '07. International Conference on*, IEEE, 2007, S. 127–130. http://dx.doi.org/10.1109/ICICT.2007.375357

[SLC05] Shao, Wenjian ; Li, Victor O. K. ; Chan, King S.: A Distributed Bandwidth Reservation Algorithm for QoS Routing in TDMA-based Mobile Ad Hoc Networks. In: *High Performance Switching and Routing, 2005. HPSR. 2005 Workshop on*, IEEE, 2005, S. 317–321. http://dx.doi.org/10.1109/HPSR.2005.1503246

[SN02] Shah, Samarth H. ; Nahrstedt, Klara: Predictive Location-Based QoS Routing in Mobile Ad Hoc Networks. In: *IEEE International Conference on Communications, ICC 2002, April 28 - May 2, 2002, New York City, NY, USA*, IEEE, 2002, S. 1022–1027. http://dx.doi.org/10.1109/ICC.2002.997009

[SSK09] Schumacher, Henrik ; Schack, Moritz ; Kürner, Thomas: Coupling of Simulators for the Investigation of Car-to-X Communication Aspects. In: *4th IEEE Asia-Pacific Services Computing Conference, IEEE APSCC 2009, Singapore, December 7-11 2009, Proceedings*, IEEE, 2009, S. 58–63. http://dx.doi.org/10.1109/APSCC.2009.5394139

[SVV15] Sgora, Aggeliki ; Vergados, Dimitrios J. ; Vergados, Dimitrios D.: A Survey of TDMA Scheduling Schemes in Wireless Multihop Networks. In: *ACM Comput. Surv.* 47 (2015), Nr. 3, S. 53:1–53:39. http://dx.doi.org/10.1145/2677955

[SWB05] Saghir, Mohammed ; Wan, Tat C. ; Budiarto, Rahmat: Load Balancing QoS Multicast Routing Protocol in Mobile Ad Hoc Networks. In: *Technologies for Advanced Heterogeneous Networks, First Asian Internet Engineering Conference, AINTEC 2005, Bangkok, Thailand, December 13-15, 2005, Proceedings*, Bd. 3837, Springer, 2005 (Lecture Notes in Computer Science), S. 83–97. http://dx.doi.org/10.1007/11599593_7

[SWW10] Su, Yi-Sheng ; Wu, Tsung-Cheng ; Wang, Chung-Hsuan: Distributed Quality of Service Routing in Multihop TDMA Ad Hoc Networks. In: *Computer Symposium (ICS), 2010 International*, IEEE, 2010, S. 734–739. http://dx.doi.org/10.1109/COMPSYM.2010.5685417

[Tex13] Texas Instruments: *CC2420 Datasheet*. http://www.ti.com/lit/ds/symlink/cc2420.pdf, 2013. – Revision SWRS041c

[TLP05] Titzer, Ben ; Lee, Daniel K. ; Palsberg, Jens: Avrora: Scalable Sensor Network Simulation with Precise Timing. In: *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005, April 25-27, 2005, UCLA, Los Angeles, California, USA*, IEEE, 2005, S. 477–482. http://dx.doi.org/10.1109/IPSN.2005.1440978

[Uni15] University of California, Berkeley: *TinyOS Homepage*. http://www.tinyos.net, 2015

[USC15] USC Information Sciences Institute: *The Network Simulator – ns-2*. http://www.isi.edu/nsnam/ns/, 2015

[VMM08] Vincent, Sébastien ; Montavont, Julien ; Montavont, Nicolas: Implementation of an IPv6 stack for NS-3. In: *3rd International ICST Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS 2008, Athens, Greece, October 20-24, 2008*, ICST/ACM, 2008, S. 75:1–75:9. http://dx.doi.org/10.4108/ICST.VALUETOOLS2008.4374

[WJ07] Wu, Huayi ; Jia, Xiaohua: QoS multicast routing by using multiple paths/trees in wireless ad hoc networks. In: *Ad Hoc Networks* 5 (2007), Nr. 5, S. 600–612. http://dx.doi.org/10.1016/j.adhoc.2006.04.001

[WL12] Wang, Neng-Chung ; Lee, Chao-Yang: A multi-path QoS multicast routing protocol with slot assignment for mobile ad hoc networks. In: *Inf. Sci.* 208 (2012), S. 1–13. http://dx.doi.org/10.1016/j.ins.2012.04.040

[YK05] Yang, Yaling ; Kravets, Robin: Contention-Aware Admission Control for Ad Hoc Networks. In: *IEEE Trans. Mob. Comput.* 4 (2005), Nr. 4, S. 363–377. http://dx.doi.org/10.1109/TMC.2005.52

[ZC02] Zhu, Chenxi ; Corson, M. Scott: QoS routing for mobile ad hoc networks. In: *Proceedings IEEE INFOCOM 2002, The 21st Annual Joint Conference of the IEEE Computer and Communications Societies, New York, USA, June 23-27, 2002*, Bd. 2, IEEE, 2002, S. 958–967. http://dx.doi.org/10.1109/INFCOM.2002.1019343

[Zhu02] Zhu, Chenxi: *Medium Access Control and Quality-of-Service Routing for Mobile Ad Hoc Networks*, University of Maryland, Department of Electrical and Computer Engineering, Dissertation, 2002

[ZL13] Zhen, Xu ; Long, Zhou: Bandwidth Constrained Multicast Routing for TDMA-Based Mobile Ad Hoc Networks. In: *JCM* 8 (2013), Nr. 3, S. 161–167. http://dx.doi.org/10.12720/jcm.8.3.161-167

Anuschka Igel

Lebenslauf

Schulbildung

08/1990 – 12/1991 **Grundschule**, *Grundschule Schillingen*.

12/1991 – 06/1994 **Grundschule**, *Grundschule Thaleischweiler-Fröschen*.

08/1994 – 06/1999 **Gymnasium**, *Leibniz-Gymnasium Pirmasens*.

08/1999 – 03/2003 **Gymnasium**, *Gymnasium am Römerkastell Bad Kreuznach*, Abschluss: Abitur.

Studium

04/2003 – 03/2009 **Studium der Informatik**, *Technische Universität Kaiserslautern*, Abschluss: Diplom.
Projektarbeit: „Entwicklung einer GUI für MaLiAn“
Diplomarbeit: „Maximum-Likelihood-Analyse markierter Strukturen“

Berufserfahrung

05/2009 **Wissenschaftliche Hilfskraft**, *Technische Universität Kaiserslautern, Arbeitsgruppe „Algorithmen und Komplexität“*.

06/2009 – 05/2015 **Wissenschaftliche Mitarbeiterin**, *Technische Universität Kaiserslautern, Arbeitsgruppe „Vernetzte Systeme“*.