# Entwicklung und Analyse eines fehlertoleranten Atomic Broadcast Protokolls

DISSERTATION

zur Erlangung des Doktorgrades
Dr. rer. nat.

der Fakultät für Wirtschaftswissenschaften
der Universität Duisburg-Essen

vorgelegt
von

Valentin Fitz

aus
Ekpendy, Kasachstan

Betreuer:
Prof. Dr. Klaus Echtle
Dependability of Computing Systems
Universität Duisburg-Essen

Essen, Dezember 2021

**Gutachter**:

Prof. Dr. Klaus Echtle
Prof. Dr. Pedro José Marrón

**Tag der mündlichen Prüfung**:

16.08.2022

*Für Alina und Elena*

**DOI:** 10.17185/duepublico/76974
**URN:** urn:nbn:de:hbz:465-20221124-121845-2

## Kurzfassung

Seit Jahrzehnten gewinnt der Einsatz von sicherheitskritischen verteilten Systemen mit harten Echtzeitanforderungen in vielen Bereichen wie dem Automobilsektor, Industrie- und Flugkontrollsystemen, der Medizin oder im Nuklearbereich, stetig an Wichtigkeit. Erste Lösungen zur Erzielung von Übereinstimmung unter den Komponenten eines verteilen Systems in Gegenwart von Fehlern existieren bereits seit dem Beginn des ersten Einsatzes eines verteilten Echtzeitsystems und werden seitdem für die unterschiedlichsten Einsatzgebiete um neue Ideen und Lösungen erweitert. Diese Dissertation befasst sich mit dem neuen Protokoll *Fault Tolerant Atomic Broadcast and Agreement in Bridge Connected Networks (FABAN)*, welches für Bridge-verbundene Netzwerke konzipiert ist und Übereinstimmung durch unteilbare Rundsprüche realisiert, indem Nachrichten auf $f + 1$ redundanten Pfaden verteilt werden.

Das ursprünglich informell beschriebene Protokoll, insbesondere die Anforderungen an das redundante Routing, wird für die Fehlerannahme $f = 1$ mathematisch formalisiert und bewiesen. Anschließend wird eine algorithmische Lösung zur effizienten Generierung von Routings vorgestellt, formal bewiesen und experimentell verifiziert. Probleme des Protokolls bei der Toleranz von mehr als einem Fehler werden untersucht und als Folge dessen eine abwärtskompatible Erweiterung von FABAN, genannt ExFABAN, für die Toleranz von $f \geq 1$ Fehlern entwickelt. Die Erweiterung wird ebenfalls mathematisch analysiert und hinsichtlich der formulierten Anforderungen formal bewiesen. Sowohl im Bezug auf die Anwendung von FABAN als auch auf ExFABAN wird eine ausführliche Evaluation von geeigneten Netzwerktopologien durchgeführt. Anschließend werden die mathematischen theoretischen Ergebnisse mit Hilfe von Simulationen im Simulationsframework OMNeT++ auf reale, nicht vorhergesehene Probleme untersucht. Dabei werden wenige, das Konzept der Implementierung betreffende Probleme gefunden und Lösungen ausgearbeitet. Zum Abschluss wird eine Demonstration einer Implementierung von FABAN auf realer Hardware, einem Netzwerk aus Ethernet-verbundenen BananaPi R1 Mikrocontrollern, vorgestellt.

Insgesamt wurden mit *Fault Tolerant Atomic Broadcast and Agreement in Bridge Connected Networks (FABAN)* sowie *Extended Fault Tolerant Atomic Broadcast and Agreement in Bridge Connected Networks for Multiple Faults (ExFABAN)* praxistaugliche Atomic Broadcast Protokolle für Bridge-verbundene Netzwerke entwickelt, die als Ethernet-Spezialisierung zunehmend im Automatisierungsbereich eingesetzt werden. Die Protokolle lösen dabei das Problem der fehlertoleranten Übereinstimmung von dezentralen Knoten, die gemeinsam eine Automatisierung durchführen, in effizienter Weise hinsichtlich Topologie und Nachrichtenaufwand.

## Abstract

For decades, the use of safety-critical distributed systems with hard real-time requirements has been gaining in importance in many areas such as the automotive sector, industrial and flight control systems, medicine and the nuclear sector. Since the beginning of the usage of safety-critical real-time distributed systems, solutions for achieving agreement among the components of a distributed system in the presence of faults have been developed and the need for new and more efficient solutions targeting a wide variety of areas of application is still increasing. This thesis deals with the new agreement protocol *Fault Tolerant Atomic Broadcast and Agreement in Bridge Connected Networks (FABAN)*, which is designed for bridge-connected networks and aims to achieve agreement with atomic broadcasts by distributing messages along $f + 1$ redundant paths to all receivers, while up to $f$ components maybe faulty at the same time.

The protocol was originally described informally, especially the requirements concerning the message routing, and is reformulated and proven mathematically for the case $f = 1$. In addition, an algorithm for generating message routings is developed, proven and experimentally analysed. FABAN faces some problems tolerating more than one single fault and as a consequence of the studies of the limitations of the original idea behind FABAN, a fully backward compatible extension of the protocol called ExFABAN is developed and mathematically formulated and proven for the general case of tolerating an arbitrary number $f \geq 1$ of faults. For FABAN as well as for ExFABAN a detailed evaluation of suitable network topologies has been carried out. Afterwards all theoretical results are verified in simulations implemented in the simulation framework OMNeT++. Theoretically not considered problems have been detected and solutions developed. Finally, a demonstration of an implementation of FABAN on real hardware, a network of Ethernet-connected BananaPi R1 microcontroller, is presented as final contribution of this thesis.

Overall, practical protocols have been developed with *Fault Tolerant Atomic Broadcast and Agreement in Bridge Connected Networks (FABAN)* and *Extended Fault Tolerant Atomic Broadcast and Agreement in Bridge Connected Networks for Multiple Faults (ExFABAN)*, which are used as an Ethernet specialization in the automation sector. The protocols solve the problem of the fault-tolerant consensus of decentralized nodes, which perform automation tasks together, in an efficient manner with regard to topology and message overhead.

## Danksagung

Ich möchte mich an dieser Stelle bei allen herzlich bedanken, die mich während der Erstellung dieser Arbeit unterstützt haben.

Mein besonderer Dank gilt Prof. Dr. Klaus Echtle, der meine Arbeit sowohl zu meiner Zeit als wissenschaftlicher Mitarbeiter an seinem Lehrstuhl *Verlässlichkeit von Rechensystemen*, als auch in der darauffolgenden Zeit nach seiner Pensionierung großartig betreut hat. Inhaltliche Diskussionen, hilfreiche Kommentare sowie Korrekturen hatten einen großen positiven Einfluss auf diese Arbeit.

Des Weiteren danke ich meiner gesamten Familie für den gebotenen Rückhalt und die unermüdliche Unterstützung. Insbesondere gaben mir meine Kinder eine spezielle Motivation.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| **BPi R1** | BananaPi R1 |
| **BCN** | Bridge-Connected-Network |
| **CAN** | Controller Area Network |
| **CB** | Checking Bridge |
| **DB** | Distributing Bridge |
| **DoS** | Denial of Service |
| **ExFABAN** | Extended Fault Tolerant Atomic Broadcast and Agreement in Bridge Connected Networks for Multiple Faults |
| **FABAN** | Fault-Tolerant Atomic Broadcast and Agreement in Bridge-Connected Networks |
| **FB** | Forwarding Bridge |
| **FPGA** | Field Programmable Gate Array |
| **LAN** | Local-Area-Network |
| **OMNeT++** | Objective Modular Network Testbed in C++ |
| **MTBF** | Mean Time between Failures |
| **MTTF** | Mean Operating Time to Failure |
| **MTTR** | Mean Time To Repair |
| **OM** | Oral Messages |
| **PB** | Prechecking Bridge |
| **PTP** | Precision Time Protocol |
| **SM** | Signed Messages |
| **TSN** | Time-Sensitive-Networking |
| **WAN** | Wide-Area-Network |

# Inhaltsverzeichnis

Kapitel 1

# Einleitung

## 1.1. Motivation

Mit der Erfindung des modernen Computers durch Conrad Zuse in der Mitte des zwanzigsten Jahrhunderts wurde die Grundlage für das einige Jahrzehnte später beginnende Informationszeitalter geschaffen. Während im persönlichen Bereich ein hohes Entwicklungstempo der technologischen Erfindungen, beginnend bei ersten Personal Computern über Notebooks bis hin zu Smartphones sowie Computerelementen in Autos, Haushaltsgeräten wie Fernsehern zu beobachten war, hat sich selbstverständlich auch die Anforderung und das Ausmaß der Kommunikation entsprechend schnell entwickelt. Das Internet ist seit dem Ende des zwanzigsten Jahrhunderts im stetigen Wachstum und die Vernetzung von Rechnern nimmt in allen Bereichen, nicht nur im persönlichen Bereich, immer weiter zu. Solche Verbunde von Rechnern, welche sich dem Benutzer als ein einziges System präsentieren, werden in der Informatik als *verteilte Systeme* bezeichnet [TV07].

Besonderes Augenmerk gilt sicherheitskritischen verteilten Systemen, engl. *safety critical Systems*. Einzelne Fehlfunktionen in solchen Systemen können gravierende Folgen nach sich ziehen. Diese können in den schlimmsten Fällen Leben kosten oder verheerende Schäden am System selbst oder in der Umgebung anrichten [Kni02]. Oftmals müssen sicherheitskritische Systeme harte Echtzeiteigenschaften erfüllen, das heißt, über eine harte Zeitschranke hinaus verzögerte Ergebnisse müssen als fehlerhaft gewertet werden. Typische Anwendungsgebiete mit harten Echtzeitanforderungen in sicherheitskritischen verteilten Systemen findet man im Automobilbereich in der Industrie, Flugkontrollsystemen [Dav+07], im medizinischen Bereich sowie in Nuklearanlagen [Kop+89].

Komponenten in verteilten Systemen müssen im Betrieb häufig Einigkeit bzw. Übereinstimmung untereinander erzielen. Sicherheitskritische Systeme dürfen nicht bzw. mit

nur sehr geringer Wahrscheinlichkeit ausfallen, so dass Fehlfunktionen in einzelnen oder mehreren Komponenten im System derart behandelt werden müssen, dass fehlertolerantes Verhalten erzielt wird. Dieses Problem wurde ursprünglich von Lamport et al. im Jahr 1982 als das *Problem der Byzantinischen Generäle* formuliert. In der nachfolgenden Zeit wurden eine Vielzahl von Übereinstimmungsprotokollen für diverse Szenarien entwickelt und publiziert [TS92; AH90; Bra87; Bou15b]. Um Übereinstimmung - auch in der Präsenz von Fehlern - zu erzielen, können Rundsprüche, engl. *Broadcasts*, verwendet werden. Hierbei kann jeder Sender seine Daten per Rundspruch an alle Empfänger verteilen, woraufhin diese anhand der empfangenen Nachrichten eine übereinstimmende Sicht gewinnen und eine individuelle, aber in allen fehlerfreien Empfängern konsistente oder gar gleiche Entscheidung treffen. Während viele Lösungen einen Nachrichtenaustausch zu bestimmten, periodischen Zeiten realisieren, sind Ansätze mit flexiblem Sendeverhalten schwerer zu umzusetzen. Senden Komponenten völlig unabhängig Rundsprüche, muss sichergestellt werden, dass eine globale Totalordnung der Verarbeitung von Nachrichten in allen Empfänger eingehalten wird. Rundsprüche mit dieser Eigenschaft werden in der Literatur als unteilbare Rundsprüche, engl. *Atomic Broadcast*, bezeichnet.

Die vorliegende Arbeit behandelt das Protokoll FABAN, welches Übereinstimmung durch unteilbare Rundsprüche löst. Im Gegensatz zu vielen Übereinstimmungsprotokollen für sicherheitskritische Systeme mit harten Echtzeitanforderungen, welche teure Feldbussysteme nutzen, wie zum Beispiel Controller Area Network (CAN) oder FlexRay, ist FABAN konzipiert für günstigere Bridge-verbundene Netzwerke, bei welchen Netzwerkknoten mit Bridges und Bridges wiederum untereinander verbunden sind. Die Entwicklung des Protokolls wurde von Prof. Dr. Klaus Echtle an der *Universität Duisburg-Essen* am Lehrstuhl *Verlässlichkeit von Rechensystemen* begonnen und anschließend um Forschung rund um das Protokoll ergänzt, woraus sich die folgenden zwei Publikationen

- Klaus Echtle und Valentin Fitz, **FABAN - Fault-Tolerant Atomic Broadcast and Agreement in Bridge-Connected Networks**, *ARCS 2016; 29th International Conference on Architecture of Computing Systems*, 2016
- Valentin Fitz und Klaus Echtle, **General Extension of FABAN and a Topology-Based Approach to Tolerate Any Number of Faults**, *ARCS 2018; 31th International Conference on Architecture of Computing Systems*, 2018

sowie die vorliegende Dissertation ergeben haben.

Die Verteilung von Nachrichten durch das FABAN-Protokoll erfolgt über Weiterleitungen zwischen Sender und jedem Empfänger über Verbindungen zwischen einzelnen Bridges. Hierbei ist das Ziel, die Funktionsweise von Bridges nur minimal verändern zu müssen. Die Redundanz zur Toleranz von Fehlern wird dabei möglichst gering gehalten und die Art der tolerierten Fehler schließt nur sehr wenige Spezialfälle aus. Byzantinische Fehler werden ebenfalls toleriert. FABAN stellt hierbei die erste Lösung - mit geringen

Einschränkungen - für die folgende Kombination von Anforderungen dar [EF16]:

- Übereinstimmung durch unteilbare Rundsprüche
- Flexibles und unabhängiges Sendeverhalten von Sendern
- Byzantinische Fehler
- Flexible Netzwerktopologien mit $f + 1$ redundanten Pfaden zwischen jedem Sender und jedem Empfänger.

## 1.2. Zielsetzung und Aufgabenstellung

Die ursprüngliche Spezifikation von FABAN lag nur informell und unbewiesen vor. Des Weiteren wurden bei der daran anschließenden Forschung einige Probleme im Zusammenhang mit der Toleranz von mehr als einem Fehler erkannt, für welche im Anschluss Lösungen gesucht wurden. Aus diesem Grund wird FABAN im Folgenden stets nur im Zusammenhang mit der Toleranz von exakt einem Fehler gesehen.

Mit diesem Ausgangspunkt sind folgende Ziele für diese Arbeit entstanden:

#### Ziel 1: Formale Analyse und formaler Beweis von FABAN

Das erste wichtige Ziel dieser Arbeit ist die Formalisierung der informell gegebenen Spezifikation von FABAN durch mathematische Mittel und einem formalen Beweis der behaupteten Eigenschaften im Bezug auf die Fehlertoleranzeigenschaften.

#### Ziel 2: Algorithmische Generierung von redundanten Routings

Für überschaubare Netzwerktopologien lässt sich ein redundantes FABAN-Routing relativ leicht intuitiv konstruieren. Eine algorithmische, allgemeingültige Methode für die Feststellung der Eignung einer Topologie für FABAN sowie der darauf folgenden Generierung eines gültigen Routings stellt das zweite Ziel dieser Arbeit dar.

#### Ziel 3: Weiterentwicklung des Protokolls für die Mehrfehlerannahme

Mehrfachfehler werden in der ursprünglichen Idee von FABAN nicht in dem Maße toleriert, wie angestrebt. Sowohl das Herausstellen der Ursachen der Probleme als auch das Finden einer Lösung des Problems und der damit zusammenhängenden Weiterentwicklung des Protokolls wird als drittes Ziel festgelegt.

#### Ziel 4: Untersuchung der Eignung von Netzwerktopologien

Aufgrund der Anforderung der redundanten Routings zwischen Sender und jedem Empfänger - Details sind zunächst nicht relevant - sind nicht alle Netzwerktopologien geeignet, um darauf FABAN sowie die Weiterentwicklung auszuführen. Eine kritische, qualitative und systematische Auseinandersetzung der Möglichkeiten der Eignungen von unterschiedlichen Netzwerktopologien und Klassen von Topologien stellt das letzte Ziel der theoretischen Auseinandersetzungen mit dem Protokoll und der neuen Erweiterung dar.

#### Ziel 5: Validierung/Prüfung durch Simulation

Im ersten Teil der abschließenden Untersuchung auf mögliche Praxisprobleme, soll eine detaillierte Simulation des Protokolls helfen Erkenntnisse zu liefern. Hierbei soll besonderes Augenmerk auf konzeptionelle Fehler, aber auch auf die Praxistauglichkeit der Umsetzung gelegt werden.

#### Ziel 6: Demonstration auf realer Hardware

Im zweiten und letzten Teil der abschließenden Untersuchung auf mögliche Praxisprobleme soll zu demonstrativen Zwecken eine Implementierung auf realer Hardware umgesetzt werden. Hierbei soll die tatsächliche Realisierbarkeit überprüft werden und es sollen gegebenenfalls weitere Probleme oder Schwierigkeiten lokalisiert werden. Die Performanz soll ausdrücklich nicht untersucht werden, da grundlegende Performanz-Eigenschaften bereits durch die Protokollspezifikation festgelegt werden. Weitere Einflüsse auf die Performanz sind stark vom verwendeten Netzwerk sowie den verwendeten Komponenten abhängig. Des Weiteren bietet sich die verwendete Hardware sehr gut für die Entwicklung eines Prototyps an, ist aber im Hinblick auf Performanz weniger geeignet, wie in Kapitel 8 ausgeführt wird.

## 1.3. Aufbau der Arbeit

Die vorliegende Dissertation beginnt mit einem Überblick grundlegender Konzepte und Begriffe, auf welchen die Arbeit aufbaut (Kapitel 2) bevor das Protokoll FABAN in der ursprünglichen Version der Spezifikation im Detail beschrieben wird (Kapitel 3). Mit einer Übersicht über den Stand der Technik (Kapitel 4) werden die Grundlagen für die nachfolgenden Inhalte abgeschlossen.

Die nachfolgenden Kapitel behandeln die im vorangehenden Unterkapitel formulierten Ziele. Kapitel 5 beschäftigt sich mit der formalen Verifikation von FABAN, der algorithmischen Erzeugung von Routings sowie der Untersuchung der Eignung von Topologien für

die Einfehlerannahme. Kapitel 6 beschreibt eine Erweiterung von FABAN, welche eine beliebige Anzahl von Fehlern tolerieren kann, und untersucht Klassen von Topologien bezüglich der Eignung für die Mehrfehlerannahme. Kapitel 7 stellt das Implementierungskonzept einer Simulation von FABAN sowie der Erweiterung vor und evaluiert umfangreiche durchgeführte Simulationen. Anschließend wird in Kapitel 8 eine Implementierung auf realer Hardware vorgestellt und auf mögliche, real auftretende Probleme untersucht.

In Kapitel 9 wird eine Zusammenfassung und ein Ausblick auf mögliche anschließende Arbeit gegeben und damit die Arbeit abgeschlossen.

Kapitel 2

# Grundlagen

Dieses Kapitel stellt eine Einführung in grundlegende, aus der Literatur bekannte und im Rahmen dieser Ausarbeitung essentielle Bereiche der Informatik dar. Diese beginnt mit einem Einstieg in das Gebiet der Fehlertoleranz in Form einer Übersicht über die jeweiligen Teilgebiete und deren Verhältnissen zueinander. Mit der Definition und Vorstellung von wichtigen Eigenschaften von verteilten Systemen werden weitere Grundlagen eingeführt, welche im Folgenden stets sowohl für alle vorgestellten Problemszenarien als auch für alle ausgearbeitete Lösungen als betrachtete Systeme zugrunde gelegt werden. Kommunikationsprotokolle werden vorgestellt und sowohl das Problem der Übereinstimmung in verteilten Systemen als auch zugehörige wichtige und bereits bekannte Resultate werden diskutiert. Des Weiteren wird mit der Gruppe der Rundspruchprotokolle, insbesondere den unteilbaren Rundsprüchen, eine Klasse von Kommunikationsparadigmen vorgestellt, welche in gewisser Hinsicht Ähnlichkeiten zu Übereinstimmungsprotokollen aufzeigen. Abgeschlossen wird dieses Kapitel mit einer auf diese Arbeit ausgerichteten, detaillierten und vollständigen Fehlerklassifizierung.

## 2.1. Fehlertoleranz

Analog zum wahren Leben, in dem nahezu in jedem Lebensbereich und in jedem Tätigkeitsfeld Fehler Einfluss auf bestimmte Aspekte haben, betrifft dies das Gebiet der Informatik in gleichem Maße. Aus diesem Grund beschränken sich die Gebiete der *Fehlertoleranz* und der *Zuverlässigkeit* auch nicht nur auf die Kommunikation in verteilten Systemen, den Fokus dieser Arbeit, sondern besteht aus diversen Teilgebieten, welche hier geringere Relevanz haben, in anderen Gebieten der Informatik wiederum größere Bedeutung erfahren. Nichtsdestotrotz ist es wichtig, einen möglichst vollständigen Überblick

der Fehlertoleranz zu geben, so dass auch Zusammenhänge zu verwandten Problemen thematisiert bzw. gefolgert werden können.

Während die *Zuverlässigkeit* sich damit auseinandersetzt, Vorhersagen für das Auftreten von Fehlern bzw. Fehlverhalten in Abhängigkeit unterschiedlicher Aspekte wie z.B. Fehlerraten von Hardwarekomponenten, äußerlichen physikalischen Einflüssen oder menschlichen Fehlern zu treffen und diese zu reduzieren, beschäftigt sich die Fehlertoleranz damit, einzelne, wohl spezifizierte Fehler zu behandeln und trotz auftretender Fehler ein fehlerfreies Verhalten von Hard- und Softwaresystemen zu garantieren oder diese zu beseitigen.

Um *Fehler* genau zu definieren, benötigt man zunächst die Definition der *inneren* bzw. *äußeren Spezifikation* von Systemen, wobei hier ein beliebiges und allgemeines System betrachtet werden kann, welches vordefinierte Aufgaben erfüllen muss [Ech90]. Auf eine exakte Definition des Begriffs *System* wird hier bewusst verzichtet.

**Definition 2.1** (Innere- und Äußere Spezifikation)
Die **Innere Spezifikation** eines Systems beschreibt die interne Struktur des Systems und das Verhalten der internen Komponenten. Die **Äußere Spezifikation** beschreibt das Ein-/Ausgabe-Verhalten eines Systems, das heißt die von externen Komponenten/Systemen/Benutzern benötigten, als auch die zur Verfügung gestellten Funktionen.

Damit lassen sich nun nach [Ech90] Fehler detailliert und semantisch differenziert einführen:

**Definition 2.2** (Fehler)
Ein **Fehler** eines Systems ist definiert als eine Kombination aus.

- dem **Fehlzustand**, welcher die Verletzung der *inneren Spezifikation* bezeichnet und
- dem **Funktionsausfall**, welcher die Verletzung der *äußeren Spezifikation* bezeichnet.

Der *Fehlzustand* wird in der englischsprachigen Literatur mit dem Begriff *Error* bezeichnet und beschreibt, dass ein System oder eine Komponente nicht korrekt arbeitet, unabhängig davon, ob sich das Fehlverhalten bemerkbar macht oder nicht. Das Eintreten des Fehlzustandes ist allerdings die notwendige Voraussetzung für das Eintreten eines Funktionsausfalls, welcher wiederum stets ein *Fehlverhalten* zeigt sowie auf eine *Fehlerursache* zurückzuführen ist [Avi76; Ech90; Nør00]:

- Das *Fehlverhalten*, im Englischen *Failure*, bezeichnet die Art und Weise, wie sich ein Fehler bemerkbar macht.

- Die *Fehlerursache*, im Englischen *Fault*, bezeichnet einen oder mehrere Funktionsausfälle von Subsystemen tieferer Schichten und bezeichnen bzw. beschreiben damit im weiteren Sinne die Ursache eines Fehlerverhaltens. Darunter zählen beispielsweise Entwurfsfehler, Herstellungsfehler oder Betriebsfehler wie Hardware-, Bedienungs-, und Wartungsfehler. Diese Beispiele sind im Rahmen dieser Arbeit ausreichend, so dass auf eine vollständige Auflistung verzichtet und auf [Ech90] verwiesen wird.

Fehlzustände können eintreten, ohne dass Funktionsausfälle folgen. Der Fehlzustand kann durch eine Fehlerursache, beispielsweise einem physikalischen Defekt oder einer fehlerhaften Berechnung, erklärt werden. Ein Fehlverhalten wiederum ist keine zwingende Folge, so dass auch kein Funktionsausfall eintreten muss [Avi76]. Im Allgemeinen und insbesondere in dieser Arbeit sind nur solche Fehler interessant, welche neben einem Fehlzustand auch einen Funktionsausfall implizieren. Aus diesem Grund wird im Folgenden lediglich von **Fehlern** gesprochen, welche sich durch einen *Funktionsausfall* in Folge eines *Fehlzustands* des Systems oder einer Komponente bemerkbar machen.

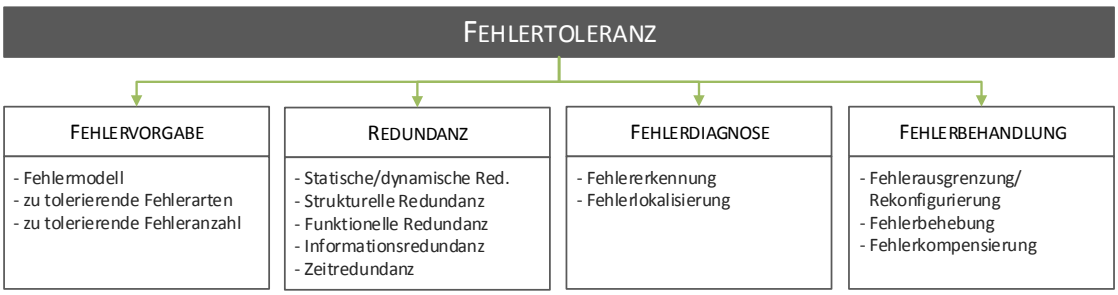



Abbildung 2.1.: Schematische Darstellung der Fehlertoleranz

Mit *Fehlertoleranz* wird in der Informatik der Umgang mit Fehlern in weiterem Sinne bezeichnet. Grob strukturiert ergibt sich eine Differenzierung der Fehlertoleranz in die folgenden vier Teilgebiete, welche in Abbildung 2.1 visualisiert sind und eine Übersicht über deren Kernaspekte bieten:

- Die **Fehlervorgabe** spezifiziert das konkret zugrunde gelegte Fehlermodell für ein gegebenes System. Dies umfasst den Umfang der Restriktionen der zu tolerierenden Fehler, d.h. sie spezifiziert sowohl die Art als auch die Anzahl der zu tolerierenden Fehler. Fehler, die in der Fehlervorgabe nicht enthalten sind, müssen nicht toleriert werden. [Ech90; Hol91; Cou+12]
- Die **Redundanz** beschreibt die Mittel, die genutzt werden können, um Fehlertoleranzeigenschaften einem System zu verleihen. Zusätzliche Mittel sind dabei eine zwingend erforderliche Voraussetzung um Fehler tolerieren zu können. Diese lassen sich im Detail wie folgt unterteilen: [Ech90]

  - **Strukturelle Redundanz** bezeichnet das Vorhandensein von zusätzlichen, für die Funktion des Systems entbehrliche Zusatzkomponenten. Ein prominentes Beispiel sind RAID-Systeme, die im einfachsten Fall eine Spiegelung einer Festplatte realisieren. [CC08]
  - **Funktionelle Redundanz** bezeichnet das Bereitstellen von zusätzlichen, aber für den Normalbetrieb entbehrlichen, Funktionen. Dies umfasst insbesondere Diversität, also die Umsetzung von Funktionen auf unterschiedliche Weisen. [LPS01; NZB13; Lar+14]
  - **Informationsredundanz** bezeichnet das Bereitstellen von zusätzlicher Information, wie z.B. das Anhängen von Signaturen oder Prüfsummen an Nachrichten. [Ech99; Bou15b]
  - **Zeitredundanz** bezeichnet das Gewähren von zusätzlichen Zeiten, die über dem Zeitbedarf der normalen Ausführungen hinausgehen, d.h. das längere Warten auf Nachrichten oder andere Ereignisse. [MRB90; SH94; TS94]
- **Statische/Dynamische Redundanz** unterscheidet zwischen dem dauerhaften Bereitstellen (statisch) und dem an bestimmte Ausnahmefälle gebundene Bereitstellen (dynamisch) der oben genannten redundanten Mittel. Die *Rekonfiguration*, also das im Fehlerfall stattfindende, automatische Ausgliedern von fehlerhaften Komponenten mit anschließender Eingliederung von bisher passiven Komponenten, stellt hierbei das prominenteste Beispiel der dynamischen Redundanz dar. [CS02]
- Die **Fehlerdiagnose** behandelt die Erkennung und die Lokalisierung von Fehlern im System. Hierbei kann es reichen, Fehler anhand von Auswirkungen im System zu erkennen, woraufhin andere Maßnahmen erfolgen können wie z.B. die Behandlung der Fehler bzw. Fehlerauswirkungen (siehe nachfolgenden Stichpunkt). [BS85; Ise93; SF91]
- Die **Fehlerbehandlung** umfasst den Umgang mit Fehlern. Hierbei können erkannte Fehler unter anderem durch Rekonfiguration ausgegrenzt, durch Reparatur behoben, durch Fehlermaskierung kompensiert oder schlicht durch Ausnahmebehandlungen anwendungsspezifisch behandelt werden.

Für die Fehlervorgabe ist neben der Spezifikation der Fehlerarten die Spezifikation von Fehlerorten, d.h. den Komponenten eines Systems, die von Fehlern betroffen sein dürfen, essentiell. Komponenten von Systemen, deren Fehler nicht toleriert werden müssen, beispielsweise solche, die nur für Testzwecke existieren, werden als *Perfektionskern* bezeichnet. Restliche Komponenten werden in sogenannte *Einzelfehlerbereiche* unterteilt, welche wiederum, abhängig von der Spezifikation, von Fehlern in gewissem Maße betroffen sein dürfen [Ech90]:

**Definition 2.3** (Perfektionskern, Einzelfehlerbereich)
Ein System $S$ kann stets disjunkt zerlegt werden in einen *Perfektionskern* $K$ und $n \in \mathbb{N}$ *Einzelfehlerbereiche* $E_i,\ 1 \leq i \leq n$, d.h.

$$S = K \dot{\cup} \left( \dot{\bigcup_{i=1}^{n}} E_i \right). \tag{2.1}$$

Vereinigungen von beliebigen Einzelfehlerbereichen aus $\{E_1, \ldots, E_n\}$ werden als *Fehlerbereiche* bezeichnet. Dabei müssen Fehlerbereiche nicht zwingend paarweise disjunkt definiert werden.

Hiermit lässt sich nun formal die Fehlertoleranz wie folgt definieren. Dabei ist dies eine leichte Abwandlung der Definition aus [Ech90].

**Definition 2.4** ($f$-Fehlertoleranz)
Ein System $S$ mit einer gegebenen Fehlervorgabe heißt *$f$-fehlertolerant*, falls dieses seine spezifizierte Funktion erfüllt, während zu jedem beliebigen Zeitpunkt höchstens $f$ Fehlerbereiche fehlerhaft sind.

Der *Grad der Fehlertoleranz* bzw. der *Fehlertoleranzgrad* eines $f$-fehlertoleranten Systems bezeichnet die Anzahl der zur gleichen Zeit tolerierten Fehler und ist gleich $f$.

Eine Klassifizierung der Fehlerarten ist in den meisten Fällen abhängig vom zugrunde liegenden System. In der Softwareentwicklung würde man logischerweise auf vollkommen andere Fehlerarten achten als in der Entwicklung von Hardware. Auch andere Faktoren wie die Einsatzumgebung haben großen Einfluss darauf, welche Fehlerarten betrachten werden müssen. Einflüsse durch beispielsweise Radioaktivität auf Hardware ist in den allermeisten Anwendungsgebieten völlig irrelevant, in speziellen Umgebungen wie beispielsweise dem Weltall allerdings von größter Wichtigkeit, um den ordnungsgemäßen Betrieb von Hardware gewährleisten zu können. [Fuc+14; Lan+15] Aus diesem Grund, wird die Fehlerklassifizierung dieser Arbeit im Anschluss an einige nachfolgende Grundlagen Unterkapitel nachgereicht, um besser auf die hier im Fokus stehenden Bereiche eingehen zu können.

## 2.2. Verteilte Systeme und Protokolle

Der Begriff des „Verteilten Systems“ wurde in der Literatur auf unterschiedliche Weisen definiert. Eine grobe Beschreibung des Begriffs gibt [TV07] wie folgt an:

> A distributed system is a collection of independent computers that appears to its users as a single coherent system. Quelle:Andrew S. Tanenbaum [TV07]

Eine alternative, aber ebenfalls sowohl sinnvolle als auch simple Charakterisierung von verteilten Systemen stammt von Leslie Lamport [Gho14]:

> A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable. Quelle:Leslie Lamport [Gho14]

Obwohl beide Beschreibungen zunächst sehr unterschiedlich erscheinen, sind diese keineswegs widersprüchlich, sondern konsistent zueinander. In Abbildung 2.2 stellen die Rechnerkomponenten $S_1, S_2$ und $S_3$, welche mittels unabhängiger Operationen und einer Kommunikation untereinander nebenläufig eine gemeinsame Aufgabe erfüllen, ein verteiltes System dar. Für die Benutzer $U_1, U_2$ und $U_3$ erscheint das System als ein einzelnes, zusammenhängendes System, wobei ein Fehlverhalten der Komponenten $S_1, S_2$ bzw. $S_3$ entweder unbemerkt bliebe oder auf den fehlerhaften Betrieb des gesamten Systems zurückgeführt werden würde.

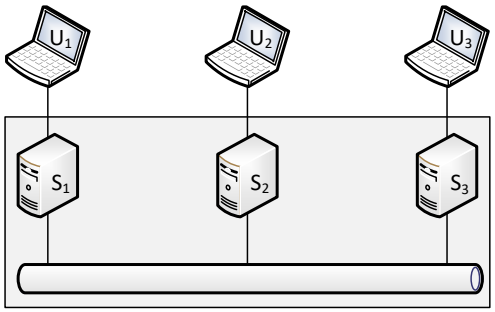



Abbildung 2.2.: Einfaches Beispiel für ein verteiltes System

Insbesondere liefern beide Darstellungen Implikationen von Eigenschaften und Kennzeichen, welche Systeme als verteilte Systeme charakterisieren. Daraus lässt sich folgende formale und hinreichend exakte Definition folgern [Cou+12]:

**Definition 2.5** (Verteiltes System)
Ein **verteiltes System** ist ein *System* von Hard- und Softwarekomponenten in einem Rechnersystem, welche mittels Kommunikation über Nachrichten miteinander kooperieren, um globale Aufgaben zu bewältigen. Dies lässt sich durch folgende drei Eigenschaften beschreiben:

1. *(Nebenläufigkeit)* Komponenten des Rechnersystems sind unabhängig voneinander und arbeiten nebenläufig.
2. *(Lokale Uhren)* Komponenten des Rechnersystems sind unabhängig und haben folglich nur ein lokales Verständnis von Zeit.
3. *(Unabhängige Fehler)* Rechnerkomponenten sind unabhängig voneinander anfällig für ursächliche Fehler unterschiedlichen Ursprungs.

Die uneingeschränkte Unabhängigkeit der Komponenten des Rechnersystems führen zu den drei Eigenschaften *Nebenläufigkeit*, *Lokalität der Uhren* und *Unabhängigkeit der Fehler* aus Definition 2.5. Verteilte Systeme erfüllen gemeinsame Aufgaben durch Koordination über einen Austausch von Nachrichten. Dies erfordert im Allgemeinen ein gemeinsames Verständnis von Zeit. Die Unabhängigkeit der lokalen Uhren einzelner Komponenten impliziert durch äußere Einflüsse und herstellungsbedingte Ursachen zwangsläufig Abweichungen zwischen den jeweiligen Uhren, welche in der Regel im Laufe des Betriebs zunehmen. Um dies zu begrenzen, werden in verteilten System Uhrensynchronisationsprotokolle periodisch eingesetzt, um Abweichungen in regelmäßigen Abständen zu korrigieren. Fehlerhaftes Verhalten einzelner Komponenten kann die Funktionsfähigkeit anderer Komponenten nicht beeinflussen. Lediglich Folgeberechnungen auf Basis erhaltener fehlerhafter Nachrichten anderer Komponenten können selbstverständlich zu Folgefehlern führen.

Eine wichtige Klassifizierung von verteilten Systemen erfolgt häufig in **synchrone** und **asynchrone Systeme**. Diese implizieren für die Betrachtung verteilter Systeme der einen oder der anderen Kategorie, zusätzliche bzw. fehlende Eigenschaften des Systems, die in folgender Definition zusammengefasst werden [HT94]:

**Definition 2.6** (Synchrone- und asynchrone Systeme)
Ein verteiltes System ist ein **synchrones System**, falls die drei folgenden Eigenschaften erfüllt sind:

1. Die Verzögerung eines Nachrichtenaustauschs zwischen zwei Komponenten des Systems ist nach oben beschränkt.

2. Die Drift der lokalen Uhren zweier beliebiger Komponenten des Systems ist nach oben beschränkt.
3. Die benötigte Zeit für einen einzelnen Ausführungsschritt ist nach oben beschränkt.

Andernfalls heißt ein System **asynchrones System**.

Die theoretische Unterteilung in synchrone und asynchrone Systeme mittels strenger Eigenschaften ermöglicht es, Aussagen über das System zu folgern und damit die Koordination innerhalb des Systems besser zu planen. Während im Allgemeinen die drei Synchronitätseigenschaften den Umfang und die Komplexität von Problemen, die in verteilten Systemen aufkommen, reduzieren, sind diese in der Realität nur auf Echtzeit-Betriebssystemen oder nur sehr ineffizient umsetzbar [Bar97; SR04]. Problemlösungen für synchrone Systeme können allerdings helfen, Lösungen für realistischere Systeme zu finden. Realistische verteilte Systeme müssen nicht zwangsläufig vollkommen asynchron sein, sondern können Synchronitätseigenschaften auch nur teilweise erfüllen. Solche Systeme werden im Folgenden als **teilsynchrone** verteilte Systeme bezeichnet, ohne nähere Anforderungen zu spezifizieren. Ist folglich einem verteilten System weder eindeutig Synchronität noch ausgeprägte Asynchronität, d.h. Verletzung aller drei Synchronitätseigenschaften, zuzuordnen, so wird dieses als teilsynchrones System betrachtet. [CT91]

Gemeinsame Aufgaben erledigen verteilte Systeme über den Austausch von Nachrichten durch Ausführung eines oder mehrerer *Kommunikationsprotokolle*, im Folgenden einfach als *Protokolle* bezeichnet, welche eine Reihe von Regeln festlegen, gemäß derer die Datenübertragungen zwischen einzelnen Kommunikationskomponenten ablaufen müssen. Unabhängig von den eingesetzten Protokollen ist jede Kommunikation schlussendlich auf die Übertragung von *Bits* auf dem zugrunde liegenden Übertragungsmedium zurückzuführen. Protokolle stellen jedoch eine zwingend erforderliche Grundlage des gemeinsamen Verständnisses von *Bitvektoren* zur Verfügung. Nach Gerard J. Holzmann [Hol91] besteht jede Spezifikation eines Kommunikationsprotokolls aus fünf Aspekten, welche in folgender Definition zusammengefasst sind:

**Definition 2.7** (Kommunikationsprotokoll)
Ein **Kommunikationsprotokoll** ist ein Satz von Regeln, welche die Kommunikation zwischen (vom Protokoll spezifizierten) Kommunikationspartnern definieren. Jede Spezifikation eines Protokolls besteht aus folgenden fünf Aspekten:

1. Der *Dienst* bzw. die *Leistung*, die vom Protokoll erbracht wird.
2. Die *Annahmen* an die Einsatzumgebung des Protokolls.
3. Die *Nachrichtenmenge*, die vom Protokoll verwendet wird.

4. Das *Nachrichtenformat* jeder Nachricht der Nachrichtenmenge.
5. Die *Ablaufregeln* des Nachrichtenaustauschs (Algorithmus).

Hierbei kann jeder der fünf Aspekte sehr vielschichtig sein. Beispielsweise kann ein Protokoll eine Vielzahl von unterschiedlichen Nachrichtenformaten verwenden, so dass automatisch sowohl die Nachrichtenmenge in Ihrer Größe zunimmt als auch die Ablaufregel an Komplexität gewinnen.

Eine allgemein betrachtete Kommunikation zwischen zwei Kommunikationsteilnehmern kann in unterschiedliche Ebenen oder Schichten unterteilt werden. Eine klar getrennte Betrachtung und Realisierung von Funktionalitäten, liefert neben Vorteilen im Bereich *Design* und *Entwicklung* in Form einer klaren und übersichtlichen Struktur eines Systems, auch Vorteile bezüglich der Aspekte *Wartbarkeit* und *Erweiterbarkeit*. Gängige Ansätze für Kommunikationsprotokolle basieren auf einer hierarchischen Gliederung in Abstraktionsschichten [Dij68; Par72; Neu86], so genannten **Schichtenmodellen**, in welchen Schichten funktional von niedrigeren Schichten abhängig sind. Abbildung 2.3 skizziert die Idee hierarchischer Schichten.

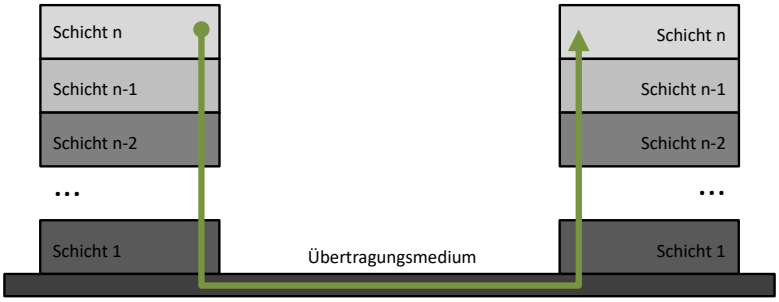



Abbildung 2.3.: Struktur eines allgemeinen hierarchischen Schichtenmodells

Die hierarchische Struktur in Abbildung 2.3 besteht aus $n \in \mathbb{N}$ Schichten. Jede Schicht $m \in \{2, ..., n\}$ ist hierbei zwingend von den Funktionalitäten der niederen Schichten $i \in \{1, ..., m-1\}$ abhängig, während eine umgekehrte Abhängigkeit in der Regel nicht vorliegt. Schicht 1 repräsentiert die niedrigste Schicht, welche zwingend die Funktion der Übertragung von Bitvektoren erfüllen muss.

Die Differenzierung von Protokollen in unterschiedliche Schichten hat im Sinn der Modularisierung den Vorteil, dass strikt voneinander separierbare Aufgaben in Form unterschiedlicher Protokolle realisiert werden können. Die daraus entstehende Aufgabenteilung wird folglich nicht nur gemäß einem einzigen Protokoll, sondern durch eine Reihe von Protokollen, auch *Protokollstapel* genannt, realisiert [TV07], so dass einzelne Protokolle mit leichtem Aufwand durch gleichwertige ausgetauscht werden können. Hierbei

ist die Anzahl der Protokolle pro Schicht nicht zwingend auf ein Protokoll beschränkt, sondern kann durchaus ebenfalls aus einer Reihe von Protokollen bestehen.

Jedes Protokoll des Protokollstapels fügt an eine zu sendende Nachricht eigene Informationen hinzu, so dass eine rekursive Zusammensetzung der Nachricht um Metainformationen entsteht, die vom Empfänger wiederum in umgekehrter Reihenfolge verarbeitet werden. Daraus entsteht ein Nachrichtenformat mit strukturiert ineinander geschachtelten, protokollspezifischen *Metainformationen*. Abbildung 2.4 visualisiert den Aufbau einer Nachricht eines allgemeinen Protokollstapels.

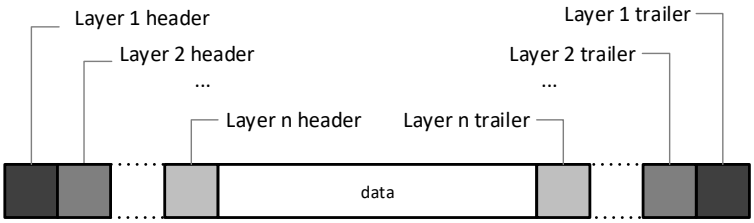



Abbildung 2.4.: Rekursiver Nachrichtenaufbau gemäß eines allgemeinen Protokollstapels

Es ist anzumerken, dass die Metadaten jeder Schicht sich wiederum ebenfalls aus einer Reihe von protokollspezifischen Metadaten der entsprechenden Schicht zusammensetzen können. Des Weiteren wird nicht zwingend vorausgesetzt, dass in jeder Schicht sowohl ein *Header* als auch ein *Trailer* angehängt wird.

Im Detail wird während eines Sendevorgangs von *Nutzdaten* $d \in \{0,1\}^l$, $l \in \mathbb{N}$, von einem Sender $S \in \mathcal{N}$ zu einem Empfänger $E \in \mathcal{N}$, wobei $\mathcal{N}$ die Menge aller Komponenten eines Systems bezeichnet, an den Nutzinhalt $d$ sukzessive protokollspezifische Information angehängt. Im vereinfachten Fall, dass Nachrichten zwischen den Schichten weder aufgeteilt noch zusammengefasst werden, lässt sich dieser Vorgang bei einem Protokollstapel eines Systems, abhängig vom Zustand des Senders $\mathcal{Z}_S(t)$ zur Zeit $t \geq 0$, in einem Schichtenmodell mit $n \in \mathbb{N}$ Schichten mittels $n$ Funktionen

$$h_k^{\mathcal{Z}_S(t)} : \{0,1\}^{l_{k+1}} \to \{0,1\}^{l_k}, \quad k \in \{1,...,n\} \tag{2.2}$$

ausdrücken, wobei $l_{n+1} := l$ und $l_{n+1} < \cdots < l_1$ gilt und die maximale Größe von Nutzdaten, die in einem einzelnen *Nachrichtenpaket* auf der Bitübertragungsebene übertragen werden können, sich durch die Spezifikation der einzelnen Protokolle des Protokollstapels ergibt. Diese Funktionen $h_k^{\mathcal{Z}_S(t)}$ bilden Bitvektoren der nächsthöheren Schicht $k$, das heißt sowohl die ursprünglichen Nutzdaten als auch jegliche Metainformationen höherer Schichten, auf breitere Bitvektoren in Abhängigkeit des aktuellen Zustands des Systems ab. Der Zustand schließt sowohl lokale Variablen als auch das Zeitverständnis der jeweiligen Systemkomponenten ein. Für den Sendevorgang bedeutet dies, dass eine Nachricht $m$

sukzessive wie folgt erzeugt wird:

$$m = h_1^{\mathcal{Z}_S(t)}\left(h_2^{\mathcal{Z}_S(t)}\left(...h_n^{\mathcal{Z}_S(t)}\left(d\right)...\right)\right). \tag{2.3}$$

Auf Empfängerseite wird die erhaltene Nachricht in Abhängigkeit des Zustandes des Empfängers $\mathcal{Z}_E(t)$ in umgekehrter Reihenfolge mittels passend gewählter Funktionen

$$\bar{h}_k^{\mathcal{Z}_E(t)} : \{0,1\}^{l_k} \rightarrow \{0,1\}^{l_{k+1}},\ 1 \leq k \leq n \tag{2.4}$$

mit der Eigenschaft

$$\bar{h}_k^{\mathcal{Z}_E(t)}\left(h_k^{\mathcal{Z}_S(t)}(x)\right) = x \text{ für } x \in \{0,1\}^{l_{k+1}} \tag{2.5}$$

entpackt und verarbeitet. Aufgrund unterschiedlicher Dimensionen von Definitions- und Zielmengen der Funktionen $h_k^{\mathcal{Z}_S(t)}$, sind diese zwar nicht bijektiv und folglich auch nicht umkehrbar, jedoch kann mittels Einschränkung der Zielmenge auf den Wertebereich stets eine Umkehrfunktion gefunden werden. Anzumerken ist, dass innerhalb von Protokollstapeln ein oder mehrere *kryptographische Protokolle* zum Einsatz kommen können, so dass die ursprünglich zu sendenden Nutzdaten nicht ohne speziell durch die Protokollspezifikation bereitgestellten und zusätzlichen Metainformationen mit entsprechenden Dechiffrierungsalgorithmen aus der Nachricht $m$ gelesen werden können. Des Weiteren kann innerhalb eines Protokollstapels eine *Datensegmentierung* stattfinden, so dass unter anderem die Beschränkung der Nutzdatengröße in Schichten oberhalb der Segmentierung aufgehoben werden kann. Dies und weiterer Nutzen der Datensegmentierung spielt jedoch im Rahmen dieser Arbeit keine Rolle und wird nicht weiter behandelt.

Neben einer Vielzahl bekannter Schichtenmodelle für Protokolle, sind insbesondere die zwei Schichtenmodelle *Department of Defense* Modell (DoD-Modell) [CC83], auch als *TCP/IP-Modell* bekannt, und *Open Systems Interconnection* Modell (OSI-Modell) [ITU94] als Beispiele hervorzuheben, welche die Kommunikation zwischen Kommunikationsteilnehmern in vier bzw. sieben Schichten unterteilen. Hierbei ist das OSI-Modell als eine Weiterentwicklung des DoD-Modells entstanden, indem einige Schichten feiner unterteilt wurden. Tabelle 2.1 gibt eine Übersicht über die einzelnen Schichten beider Modelle und verdeutlicht ihren Zusammenhang.

| | OSI-Modell | DoD-Modell | |
|---|---|---|---|
| 7 | Anwendungsschicht [Application layer] | Anwendungsschicht | 4 |
| 6 | Darstellungsschicht [Presentation layer] | | |
| 5 | Sitzungsschicht [Session layer] | | |
| 4 | Transportschicht [Transport layer] | Transportschicht | 3 |
| 3 | Vermittlungsschicht [Network layer] | Internetschicht | 2 |
| 2 | Sicherungsschicht [Data link layer] | Netzzugangsschicht | 1 |
| 1 | Bitübertragungsschicht [Physical layer] | | |

Tabelle 2.1.: OSI-Modell und DoD-Modell

Beide Schichtenmodelle liefern eine Reihe von Beispielen von Protokollen und Protokollarten unterschiedlichen Verwendungszwecks, von *High-Level*-Schichten wie der *Anwendungsschicht* bis zu *Low-Level*-Schichten wie der *Bitübertragungsschicht* bzw. der *Netzzugangsschicht*. Tabelle 2.2 gibt eine Übersicht über die Einordnung von einigen Protokollen in das OSI-Modell.

| Schicht | Protokolle |
|---|---|
| Anwendungsschicht | HTTP, FTP, SMTP, CORBA, IIOP |
| Darstellungsschicht | TLS security, CORBA data representation |
| Sitzungsschicht | SIP |
| Transportschicht | TCP, UDP |
| Vermittlungsschicht | IP, ATM virtual circuits |
| Sicherungsschicht | Ethernet MAX, ATM cell transfer, PPP |
| Bitübertragungsschicht | Ethernet baseband signalling, ISDN |

Tabelle 2.2.: Einordnung bekannter Protokolle im OSI-Modell

Hierbei sei zu erwähnen, dass nicht jedes System zwangsläufig alle Schichten realisieren muss. Viele kleine Echtzeit-Automatisierungssysteme, welche mit Feldbussystemen wie *CAN* oder *FlexRay* arbeiten, decken beispielsweise die transportorientierten Schichten 1 bis 4 ab [Rob91; SJ08].

## 2.3. Fehlertolerante Übereinstimmungsprotokolle

Eine essentielle Anforderungen in verteilten Systemen ist das Vermögen, *Übereinstimmung* erzielen zu können. Viele Anwendungen erfordern in der Praxis einen Austausch von Informationen zwischen Prozessen bzw. Rechnern eines verteilten Systems, um ein gemeinsames und einheitliches Verständnis ausgetauschter Informationen oder sogar eine einheitliche Sicht auf Systemzustände zu haben und darauf aufbauend konsistente Entscheidungen treffen zu können. Die Ursachen für Inkonsistenz zwischen Prozessen bzw. Rechnern reichen von grundlegenden Eigenschaften der Nebenläufigkeit über fehlerhaftes Verhalten der Prozesse bzw. Rechner oder fehlerhafter Übertragung von Informationen aufgrund von Herstellungsfehlern einzelner Komponenten, Implementierungsfehlern der Software oder äußeren zufälligen Einflüssen der Umwelt [Had87; Ech90; Cri91; TV07] bis hin zu bösartigen externen Eingriffen in das System durch Ausnutzung von Sicherheitslücken vorhandener, aber notwendiger externer Schnittstellen. [She92; Zla15]

Grundlegende und wichtige Anwendungsbeispiele für notwendige Übereinstimmung unter einer Teilmenge von Prozessen oder Rechnern eines verteilten Systems sind [TV07; KS08]:

- Wahl eines Koordinators für die Verwaltung von Aufgaben in verteilten Systemen
- Entscheidung über Rollback oder Commit von Transaktionen in Datenbanken
- Zuweisung von Aufgaben an einzelne Komponenten verteilter Systeme
- Uhrensynchronisation
- Prozesssynchronisation
- Unteilbare Rundsprüche

Insbesondere wird im Laufe dieser Arbeit auf den letzten Punkt, die Unteilbaren Rundsprüche bzw. Atomic Broadcast Protokolle, näher eingegangen und der Zusammenhang und die Wichtigkeit der Übereinstimmung verdeutlicht. Bestehen in einem System Fehlertoleranz-Anforderungen, so nimmt die Komplexität von Protokollen zur Lösung der Übereinstimmugsprobleme mit dem Grad der Fehlertoleranz-Anforderungen zu.

Die Gruppe der Kommunikationsprotokolle zur Erzielung von Übereinstimmung bzw. zur Lösung von *Übereinstimmungsproblemen* zwischen Prozessen bzw. Rechnern wird als Gruppe der *Übereinstimmungsprotokolle* bezeichnet. Eine in der Literatur häufig synonym verwendete Bezeichnung für das Übereinstimmungsproblem ist das Problem der *Byzantinischen Übereinstimmung*. Dieses Problem geht nach Leslie Lamport [LSP82] auf die Legende der Byzantinischen Generäle aus dem Jahr 1453 n. Chr. zurück, der zufolge eine Reihe von Byzantinischen Generälen bei der Belagerung von Konstantinopel, Übereinstimmung mittels der Kommunikation über Boten über den Angriffsplan

treffen mussten. Hierbei intrigierten einige Generäle gegen andere, so dass Inkonsistenz im Informationsaustausch folgte. Leslie Lamport hat das Problem auf drei Generäle reduziert und bewiesen, dass mindestens ein weiterer General notwendig ist, damit treue Generäle fehlerhafte Information als solche identifizieren können und eine fehlerfreie übereinstimmende Entscheidung treffen können.

Als Ableitungen dessen haben sich unterschiedliche (englische) Bezeichnungen und Definitionen für im Detail unterschiedliche Übereinstimmungsprobleme etabliert. Im Folgenden werden diese Begriffe definiert. Den Anfang macht das Problem der *Byzantinischen Übereinstimmung*, welche im Englischen als *Byzantine Agreement* bezeichnet wird:

**Definition 2.8** (Byzantinische Übereinstimmung)
Sei $\mathcal{P}$ eine endliche Menge von Prozessen, welche untereinander kommunizieren können. Das Problem der **Byzantinischen Übereinstimmung** beschreibt das Problem der Übereinstimmung zwischen allen Prozessen $\mathcal{P}$ eines (teilweise) fehlerhaften verteilten Systems bezüglich eines von einem Sender $s \in \mathcal{P}$ an die Prozesse $\mathcal{P} \backslash \{s\}$ (indirekt) übermittelten Wertes. Ein Kommunikationsprotokoll erzielt Byzantinische Übereinstimmung, falls dieses folgende Eigenschaften impliziert [KS08]:

1. *(Agreement)* Alle fehlerfreien Prozesse entscheiden sich für den selben Wert.
2. *(Integrity/Validity)* Falls der Sender fehlerfrei ist, entscheiden sich alle fehlerfreien Prozesse für den vom Sender gesendeten Wert.
3. *(Termination)* Jeder fehlerfreie Prozess entscheidet sich in endlicher Zeit für einen Wert.

Bei dieser Definition wird nicht zwingend vorausgesetzt, dass eine Entscheidung auf einen tatsächlich vesendeten Wert geschieht, sondern es ist auch definitionskonform eine Entscheidung auf einen (zuvor festgelegten) Defaultwert zu treffen. Dies ist beispielsweise dann von Vorteil, wenn inkonsistente Daten verschickt werden und somit keine Entscheidung, basierend auf den empfangenen Werten, erzielt werden kann.

Eine Variante des Problems der Byzantinischen Übereinstimmung wird im Englischen als *Consensus Problem* bezeichnet. Ins Deutsche lässt sich diese Bezeichnung sowohl als *Übereinstimmungsproblem* als auch als *Konsensproblem* übersetzen. Um die Bezeichnungen in dieser Arbeit konsistent zu halten, wird im Folgenden der Begriff *Übereinstimmungsproblem* als allgemeiner Sammelbegriff für alle Probleme verwendet, welche eine beliebige Form der Übereinstimmung erzielen wollen, während der Begriff *Konsensproblem* für das im folgende definierte Problem verwendet wird [KS08]:

**Definition 2.9** (Konsensproblem)
Sei $\mathcal{P}$ eine endliche Menge von Prozessen, welche untereinander kommunizieren können. Das **Konsensproblem** beschreibt das Problem der Einigung auf einen gemeinsamen Wert zwischen allen Prozessen $\mathcal{P}$ eines (teilweise) fehlerhaften verteilten Systems in Folge eines Austausches von Initialwerten der Prozesse aus $\mathcal{P}$ untereinander. Ein Kommunikationsprotokoll löst das Konsensproblem, falls dieses folgende Eigenschaften impliziert:

1. *(Agreement)* Alle fehlerfreien Prozesse entscheiden sich für den selben Wert.
2. *(Integrity/Validity)* Haben alle fehlerfreien Prozesse den selben Initialwert, so entscheiden sich alle fehlerfreien Prozesse auch für diesen Initialwert.
3. *(Termination)* Jeder fehlerfreie Prozess entscheidet sich in endlicher Zeit für einen Wert.

Eine dritte und letzte erwähnenswerte Variante des Übereinstimmungsproblems, ist die so genannte *Interaktive Übereinstimmung*. Diese wird in der englischsprachigen Literatur als *Interactive Consistency* bezeichnet [KS08]:

**Definition 2.10** (Interaktive Übereinstimmung)
Sei $\mathcal{P}$ eine endliche Menge von Prozessen, welche untereinander kommunizieren können. Das Problem der **interaktiven Übereinstimmung** beschreibt das Problem der Einigung auf eine Menge von Werten $W = \{w(p) \mid p \in \mathcal{P}\}$ zwischen allen Prozessen $\mathcal{P}$ eines (teilweise) fehlerhaften verteilten Systems, in Folge eines Austausches von Initialwerten der Prozesse aus $\mathcal{P}$ untereinander. Ein Kommunikationsprotokoll erzielt interaktive Übereinstimmung, falls dieses folgende Eigenschaften impliziert:

1. *(Agreement)* Alle fehlerfreien Prozesse entscheiden sich für die selbe Menge von Werten $W = \{w(p) \mid p \in \mathcal{P}\}$. Wenn die Werte in $W$ gemäß einer globalen Reihenfolge geordnet werden, so bilden diese den so genannten *Konsistenzvektor*.
2. *(Integrity/Validity)* Für jeden fehlerhaften Prozess $p \in \mathcal{P}$ mit dem Initialwert $v(p)$ entscheidet sich jeder fehlerfreie Prozess auch für den Initialwert $w(p) = v(p)$.
3. *(Termination)* Jeder fehlerfreie Prozess entscheidet sich in endlicher Zeit für eine Menge von Werten.

Anzumerken ist, dass in allen drei vorangehenden Definitionen die Eigenschaft *Termination* keine zum Sendezeitpunkt einer Nachricht bekannte Zeitschranke für die Zustellung der Nachricht voraussetzt. Es lassen sich allerdings alle Definitionen in einer verschärften Variante der Eigenschaft *Termination* mit der Voraussetzung einer solchen Zeitschranke formulieren.

Trotz der unterschiedlichen Bezeichnungen für die Probleme der Übereinstimmung in unterschiedlichen Anwendungsfällen, sind diese drei zuvor vorgestellten Übereinstimmungsprobleme nicht unabhängig voneinander. Es wird für nachfolgende Beschreibungen folgende Notation eingeführt:

- Byzantinische Übereinstimmung ($\mathcal{BG}$)
- Konsensproblem ($\mathcal{C}$)
- Interaktive Übereinstimmung ($\mathcal{IC}$)

Tatsächlich ist es so, dass eine Lösung eines der drei Probleme dazu führt, dass daraus die anderen Probleme ebenfalls gelöst werden können. Dies wird im folgenden Satz formuliert und bewiesen [Fis83]:

**Satz 2.11**
Die drei Übereinstimmungsprobleme $\mathcal{C}$, $\mathcal{BG}$ und $\mathcal{IC}$ sind äquivalent zueinander, so dass eine Lösung für eines dieser Probleme, Lösungen für die anderen beiden Probleme impliziert, das heisst

$$\mathcal{BG} \Leftrightarrow \mathcal{IC} \Leftrightarrow \mathcal{C}. \tag{2.6}$$

*Beweis.*
Gegeben seien ein verteiltes System, bestehend aus einer endlichen Menge von Prozessen, mit $|\mathcal{P}| = n$ Komponenten sowie Protokolle $BG$, $C$ und $IC$, welche das Problem der Byzantinischen Übereinstimmung, das Konsensproblem bzw. das Interaktive Übereinstimmungsproblem lösen. Die Äquivalenz der drei Probleme wird durch drei Implikationen gezeigt:

1. Implikation: $\mathcal{BG} \Rightarrow \mathcal{IC}$
   Die $n$-fache Ausführung des Protokolls $BG$, wobei jeder Prozess in genau einer Ausführung die Rolle des Senders übernimmt, führt dazu, dass alle fehlerfreien Prozesse die gleiche Menge $W = \{w(p) \mid p \in P\}$ von Werten erhalten, so dass Interaktive Übereinstimmung resultiert.
2. Implikation: $\mathcal{IC} \Rightarrow \mathcal{C}$
   Die Ausführung des Protokolls $IC$ führt dazu, dass jeder fehlerfreie Prozessor eine Menge $W = \{w(p) \mid p \in P\}$ von Werten erhält. Eine nach einem einheitlichen Algorithmus jeweils lokal getroffene Entscheidung für einen Wert aus $W$ impliziert eine Lösung des Konsensproblems.
3. Implikation: $\mathcal{C} \Rightarrow \mathcal{BG}$
   Um Byzantinische Übereinstimmung mit einem Konsensprotokoll zu erreichen, sendet der Sender zunächst seinen Wert an alle Prozesse. Anschließend führen alle

Prozesse das Konsensprotokoll mit dem erhaltenen Wert (der Sender mit seinem eigenen Wert) aus. Unabhängig davon, ob der Sender fehlerfrei ist oder nicht (und damit seinen Wert evtl. inkonsistent verteilt hat), wird in jedem Fall Konsistenz erzielt, indem entweder

- die Einigung auf den Wert des Senders (falls der Sender an alle Prozesse den gleichen Wert gesendet hat) oder
- auf einen Default-Wert (falls der Sender fehlerhaft ist und verschiedene Werte gesendet hat) entsteht.

Diese Implikationen führen zur behaupteten Äquivalenz. □

Von nun an wird zwischen den drei konkreten Problemen der Übereinstimmung nicht mehr unterschieden und stellvertretend für die drei äquivalenten Begriffe die einfache Bezeichnung *Übereinstimmungsproblem* verwendet.

Die Möglichkeiten, Übereinstimmung in verteilten Systemen zu erzielen wurde im Laufe der Zeit bereits ausführlich untersucht [Fis83; FLP85; DLS88; TS92]. In Abhängigkeit vom betrachteten Fehlermodell und des Grades der Synchronität des zugrunde liegenden verteilten Systems, variieren diese Möglichkeiten, auch im Bezug auf zusätzliche Systemanforderungen. Tabelle 2.3 stellt eine Übersicht dieser Resultate dar. Diese gehen im Allgemeinen auf die Arbeit von M. J. Fischer, N. A. Lynch und M. S.Paterson [FLP85] zurück, welche gezeigt haben, dass fehlertolerante Übereinstimmung in asynchronen Systemen nicht möglich ist.

| Fehlertyp | Synchrones System | Asynchrones System |
|---|---|---|
| Fehlerfrei | ✓ | ✓ |
| Ausfallfehler | ✓ | ✗ |
| Byzantinischer Fehler | ✓ | ✗ |

TABELLE 2.3.: Übersicht über die Möglichkeiten Übereinstimmung in synchronen bzw. asynchronen Systemen zu erzielen in Abhängigkeit unterschiedlicher Fehlermodelle, wobei der Ausfallfehler einen Ausfall von Funktionen von Komponenten und der Byzantinische Fehler beliebige Fehler bezeichnet. Eine genauere Fehlerklassifizierung folgt in Kapitel 2.5

Während fehlertolerante Übereinstimmung in asynchronen Systemen, welche reale verteilte Systeme am besten abbilden, nicht möglich ist, sind die Anforderungen für synchrone Systeme zu stark für eine Vielzahl von verteilten Systemen, insbesondere solchen ohne Echtzeiteigenschaften wie sie zum Beispiel in Echtzeit-Betriebssystemen zu finden sind. Um diesem Problem zu entgehen, gibt es neben den Ansätzen der *Fehlermaskierung*, *Übereinstimmung durch Fehlerdetektoren* [CT91] und der *Übereinstimmung durch*

*Randomisierung* [CD89; Cou+12], welche in dieser Arbeit keine weitere Rolle spielen, die Möglichkeit *teilsynchrone verteilte Systeme* zu betrachten [DLS88]. Teilsynchrone verteilte Systeme können je nach Grad der geforderten Synchronität noch als Modelle für reale Systeme anwendbar sein und gleichzeitig Übereinstimmung ermöglichen. Das im Laufe dieser Arbeit im Fokus stehende Übereinstimmungsprotokoll wird in Kapitel 8 auf einem realen System mit nur teilsynchronen Eigenschaften behandelt.

## 2.4. Unteilbarer Rundspruch

Neben dem Problem der Übereinstimmung in verteilten Systemen, spielen *Rundsprüche*, im Englischen *Broadcasts*, eine wichtige Rolle. Während das Ziel der *Übereinstimmung* die Einigung zwischen allen bzw. einer Teilmenge von Komponenten eines verteilten Systems beschreibt, ist es das Ziel eines Rundspruchs, lokale Informationen einer Komponente allen anderen Komponenten zukommen zu lassen. Rundsprüche können im Allgemeinen der Vermittlungsschicht des OSI-Referenzmodells zugeordnet werden.

Im fehlerfreien Fall ist es offensichtlich, dass man einerseits bereits mit einem einfachen (geordneten) Rundspruch Übereinstimmung erzielen kann, andererseits ein Übereinstimmungsprotokoll auch einen Rundspruch abbilden kann. Komplexer wird der Zusammenhang, wenn man Systeme betrachtet, die von Fehlern betroffen sein können.

In diesem Zusammenhang hat sich in den letzten 40 Jahren eine Reihe von unterschiedlichen Rundsprucharten mit zunehmenden Anforderungen ergeben. Hierbei ist es insbesondere in dieser Arbeit wichtig, dass nicht nur zwischen den Anforderungen einzelner Rundsprucharten differenziert wird, sondern auch abhängig vom zugrunde liegenden System. Hier ist die Differenzierung zwischen asynchronen und synchronen Systemen von größerer Bedeutung.

Die nachfolgenden Definitionen haben Ihre Gültigkeit sowohl in asynchronen als auch in synchronen verteilten Systemen [BJ87; CT91; HT94; KS08]. Dabei wird stets eine Menge von Komponenten bzw. Prozessen eines verteilten Systems $\mathcal{P}$ und ein Sender $s \in \mathcal{P}$ betrachtet. Nachrichten werden mit $m \in \mathcal{M}$ bezeichnet, wobei $\mathcal{M}$ eine allgemeine Menge möglicher Nachrichten darstellt. Für einen Prozess $p \in \mathcal{P}$ wird die lokale Zeit mit $t^p > 0$ bezeichnet. Weiterhin wird mit

$$t^s_{bc}(m) \in \mathbb{R}^+ \tag{2.7}$$

der Zeitpunkt des Sendens der Rundspruchnachricht $m \in \mathcal{M}$ von Sender $s \in \mathcal{P}$ und mit

$$t^p_{del}(m) \in \mathbb{R}^+ \tag{2.8}$$

der Zeitpunkt der Verarbeitung der empfangenen Nachricht von Prozess $p$ bezeichnet. Hierbei stehen die Abkürzungen in den Indizes *bc* bzw. *del* für *broadcast* bzw. *delivery*.

Zuletzt bezeichnet $m_{|m'}$ die Abhängigkeit der Nachricht $m \in \mathcal{M}$ von einer anderen Nachricht $m' \in \mathcal{M}$. Diese Abhängigkeit kann beliebiger Form sein, so dass diese Bezeichnung lediglich symbolisiert, dass zwischen beiden Nachricht eine Verbindung besteht.

**Definition 2.12** (Reliable Broadcast)
Ein Rundspruch der Nachricht $m \in \mathcal{M}$ des Senders $s \in \mathcal{P}$ heißt **zuverlässiger Rundspruch** bzw. **Reliable Broadcast**, falls dieser die folgenden Eigenschaften erfüllt:

1. *(Agreement)* Verarbeitet ein fehlerfreier Prozess $q$ einen empfangenen Rundspruch $m$, dann verarbeiten alle fehlerfreien Prozesse die Nachricht ebenfalls.
2. *(Integrity)* Ein fehlerfreier Prozess verarbeitet eine Nachricht $m$ höchstens einmal, wobei die Verarbeitung genau dann stattfindet, wenn $m$ nicht verfälscht wurde.
3. *(Validity)* Initiiert ein fehlerfreier Sender einen Rundspruch $m$, so wird dieser die Nachricht auch sicher verarbeiten.

Die Eigenschaft *Integrity* in vorangehender Definition fordert in anderen Worten, dass nur eine Nachricht von einem fehlerfreien Sender oder aber eine fehlerfreie Nachricht eines fehlerhaften Senders - das heißt es ist keine Abweichung der Nachricht vom Protokoll zu erkennen - von fehlerfreien Empfängern akzeptiert werden darf.

Durch zusätzliche Anforderung bezüglich der Sende- und Verarbeitungsreihenfolge, haben sich in der Literatur weitere Abwandlungen des zuverlässigen Rundspruchs ergeben:

**Definition 2.13** (Special Reliable Broadcasts)
Ein Reliable Broadcast einer Nachricht $m$ von Sender $s \in \mathcal{P}$ heißt

- *FIFO Broadcast*, falls für beliebige zwei Nachrichten $m_1, m_2$ von $s$ folgende Implikation stets erfüllt ist:

$$t^s_{bc}(m_1) < t^s_{bc}(m_2) \Rightarrow t^p_{del}(m_1) < t^p_{del}(m_2) \quad \forall p \in \mathcal{P} \tag{2.9}$$

- *Causal Broadcast*, falls folgende Bedingung erfüllt ist:

$$m_{|m'} \Rightarrow t^p_{del}(m') < t^p_{del}(m) \quad \forall p \in \mathcal{P} \tag{2.10}$$

- *Atomic Broadcast*, falls folgende Bedingung erfüllt ist:

$$t^p_{del}(m') < t^p_{del}(m) \Rightarrow t^q_{del}(m') < t^q_{del}(m) \quad \forall p, q \in \mathcal{P} \tag{2.11}$$

- *FIFO Atomic Broadcast*, falls (2.9) und (2.11) erfüllt ist.
- *Causal Atomic Broadcast*, falls (2.10) und (2.11) erfüllt ist.

Aus den Definitionen geht unmittelbar hervor, dass der FIFO Broadcast eine Verallgemeinerung des Causal Broadcast darstellt und der Atomic Broadcast eine zusätzliche Voraussetzung einführt, so dass sich Abwandlungen mit der Eigenschaft (2.11), auch *Totalordnung* genannt, aller drei Broadcast-Arten ergeben. Abbildung 2.5 visualisiert diese Zusammenhänge.

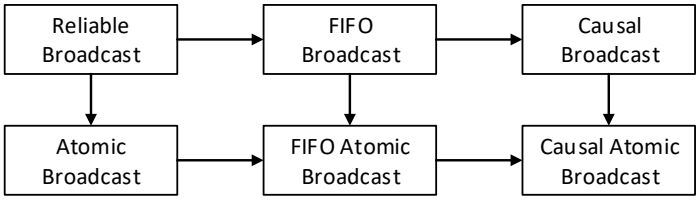



Abbildung 2.5.: Übersicht der Variationen von Reliable Broadcast Protokollen

Eine spezielle Definition von Atomic Broadcast existiert in der Literatur und einigen wissenschaftlichen Veröffentlichungen für synchrone verteilte Systeme. Unter der Prämisse zusätzlicher Einschränkungen bezüglich dem Austausch von Nachrichten, kann die Eigenschaft der Totalordnung verschärft werden, so dass nach Flaviu Cristian, losgelöst von der Definition des Reliable Broadcast, sich folgende Definition eines Atomic Broadcasts für synchrone Systeme ergibt [BD85; Cri90; Cri+95]:

**Definition 2.14** (Atomic Broadcast für Synchrone Systeme)
Ein Rundspruch in einem synchronen verteilten System heißt **unteilbarer Rundspruch** bzw. **Atomic Broadcast**, falls dieser die folgenden drei Eigenschaften erfüllt:

1. *(Atomicity)* Verarbeitet ein Prozessor $p \in \mathcal{P}$ eine Nachricht $m \in \mathcal{M}$ zum (lokalen) Zeitpunkt $t^p_{del}(m) > 0$, dann
   - wurde $m$ von einem fehlerfreien Prozessor bzw. fehlerfrei von einem fehlerhaften Prozessor gesendet und
   - alle anderen fehlerfreien Prozessoren verarbeiten die Nachricht zum gleichen Zeitpunkt, gemessen jeweils am eigenen lokalen Zeitverständnis jedes Empfängers, d.h.
   $$t^q_{del}(m) = t^p_{del}(m) \quad \forall q \in \mathcal{P} \tag{2.12}$$

2. *(Order)* Nachrichten werden bei allen fehlerfreien Empfängern in der gleichen Reihenfolge verarbeitet, d.h.

$$t^p_{del}(m') < t^p_{del}(m) \Rightarrow t^q_{del}(m') < t^q_{del}(m) \quad \forall m, m' \in \mathcal{M}\ \forall p, q \in \mathcal{P} \tag{2.13}$$

3. *(Termination)* Jeder fehlerfreie Empfänger verarbeitet eine Rundspruchnachricht eines Senders $s \in \mathcal{P}$ zu einer bestimmten Zeit, d.h.

$$\forall m \in \mathcal{M}\ \exists \Delta > 0 : t^p_{del}(m) = t^s_{bc}(m) + \Delta \quad \forall p \in \mathcal{P} \tag{2.14}$$

Anzumerken ist, dass für zwei Nachrichten, welche zu verschiedenen Zeitpunkten verarbeitet werden, die Eigenschaft (2.13) aus (2.12) folgt. Für Nachrichten, die zur gleichen Zeit verarbeitet werden, stellt (2.13) eine zusätzliche Anforderung dar.

Der Unterschied zwischen der Definition des Atomic Broadcasts für synchrone Systeme und der Variante für asynchrone Systeme beschränkt sich darauf, dass bei ersterer die Anforderung an die Zustellung bzw. die Verarbeitung der Nachricht deutlich strenger formuliert wird. Folglich kann die Definition 2.14 als ein Spezialfall der allgemeineren Variante in Definition 2.13 betrachtet werden.

Die zuvor vorgestellten Broadcast Protokoll-Klassen haben Gemeinsamkeiten mit dem Übereinstimmungsproblem. Von besonderem Interesse ist in dieser Arbeit der Zusammenhang zwischen dem Atomic Broadcast und den Übereinstimmungsproblemen. Aufgrund der Äquivalenz der in Abschnitt 2.3 vorgestellten Übereinstimmungsprobleme genügt die Untersuchung des Zusammenhangs zum *Byzantinischen Übereinstimmungsproblem*.

Ausgehend von einem gegebenem Atomic Broadcast Protokoll, lässt sich byzantinische Übereinstimmung erzielen, indem der Sender der Daten, über die Übereinstimmung im verteilen System erzielt werden soll, die zu verteilenden Daten als Atomic Broadcast initiiert. Die Atomic Broadcast Eigenschaften garantieren, dass

- alle fehlerfreien Komponenten des Systems die gesendete Nachricht fehlerfrei empfangen und folglich sich auf diesen Wert einigen können,
- die Fehlerfreiheit des Senders impliziert, dass alle fehlerfreien Empfänger die Nachricht erhalten und verarbeiten und
- fehlerfreie Komponenten in endlicher Zeit die Nachricht zustellen.

Folglich sind die Eigenschaften des Problems der *Byzantinischen Übereinstimmung* erfüllt. Diese Lösung ist an keinerlei Bedingungen geknüpft, d.h. sie funktioniert sowohl auf synchronen und asynchronen System als auch für eine beliebige Anzahl und eine beliebige Art von Fehlern. Dies hat allerdings auch zur Folge, dass für ein gegebenes System mit festgelegter Fehlerspezifikation es im Allgemeinen schwieriger ist, ein Atomic Broadcast Protokoll zu konstruieren, als das Übereinstimmungsproblem zu lösen [HT94].

Umgekehrt ist es nicht so simpel, einen Atomic Broadcast mit einem Übereinstimmungsprotokoll zu realisieren. Dies wurde zunächst für Ausfallfehler untersucht und eine Lösung veröffentlicht [HT94]. Über ein Jahrzehnt später folgte die Untersuchung der Möglichkeit Atomic Broadcasts mit einem Übereinstimmungsprotokoll mit beliebigen byzantinischen Fehlern zu realisieren. Dabei wurde gezeigt, dass unter etwas stärkeren Eigenschaften des Übereinstimmungsprotokolls (verschärfte Variante der *Validity*-Eigenschaft), dies ohne weitere Einschränkungen möglich sei, während dies unter normalen Bedingungen, das heißt, dass das Übereinstimmungsproblem genau die Eigenschaften aus Definition 2.9 erfüllt, nicht möglich ist [MHS11].

## 2.5. Fehlerklassifizierung

Der Umfang beliebiger Fehlerarten kann im Allgemeinen nicht angegeben werden und hängt in erster Linie von dem betrachteten Szenario ab. Folglich ist für die Untersuchung der Fehlertoleranz von Systemen eine detaillierte Fehlerklassifizierung essentiell. In dem hier betrachteten Szenario von verteilten Systemen, dessen Komponenten mittels (physikalischer) Verbindungen miteinander kommunizieren, müssen jedwede Fehler in Komponenten und Verbindungen in Betracht gezogen werden, um eine vollständige Fehlerklassifizierung zu erhalten.

Hierbei können die Ursachen von Fehlern beliebig sein, d.h. es werden solche Fehler in Betracht gezogen, die als Folge von fehlerhaften Implementierungen, physikalischen Defekten oder äußerlichen Einflüssen wie hohen Temperaturschwankungen oder Radioaktivität auftreten. Unabhängig von der tatsächlichen, nicht vollständig quantifizierbaren, Ursache eines Fehlers, kann das resultierende Fehlverhalten differenziert betrachtet werden. Abbildung 2.6 skizziert eine vollständige Fehlerklassifizierung, von Fehlern, die als Effekt einer beliebigen Ursache eintreten können und im nachfolgenden erläutert werden.

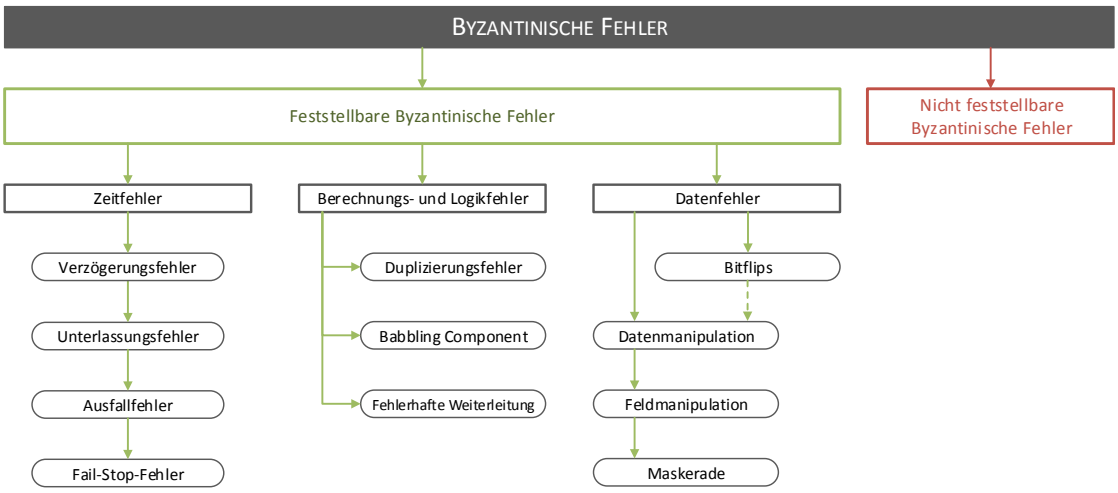



Abbildung 2.6.: Detaillierte Fehlerklassifizierung

Die Obermenge aller möglichen Fehler wird als *Byzantinischer Fehler* bezeichnet. Diese Bezeichnung geht auf das Problem der Byzantinischen Generäle von Lamport zurück [LSP82], welches bereits in Kapitel 2.3 kurz thematisiert wurde. Die feingranulare Unterteilung aus Abbildung 2.6 wird nachfolgend im Detail erläutert:

- Nicht feststellbare Byzantinische Fehler umfassen solche Fehler, die mit Hilfe der durch das System gegebenen Mechanismen bzw. aufgrund der Designs der Protokolle nicht als solche erkannt werden können. Dazu zählen beispielsweise signatur- oder prüfsummenrobuste Nachrichtenveränderungen, welche nicht durch (genügend viele) nacheinander und/oder auf verschiedenen Wegen übertragenen Nachrichtenkopien erkannt werden können. Der Umfang der Fehler, die in dieser Kategorie fallen, ist system- bzw. protokollabhängig. Es ist zwingend erforderlich, für ein gegebenes System sowohl den Umfang dieser Fehlerkategorie als auch die Rate bzw. Wahrscheinlichkeit des Auftretens solcher Fehler zu quantifizieren, da das Auftreten dieser Fehler unerkannt bleiben würde [Cri+95; Lyn96; KS08].
- Feststellbare Byzantinische Fehler umfassen solche Fehler, die mit Hilfe der durch das System gegebenen Mechanismen bzw. aufgrund der Designs der Protokolle als solche tatsächlich erkannt werden können.
  - Zeitfehler umfassen alle Fehler, welche den in der Spezifikation erlaubten zeitlichen Verlauf von Prozessen oder Nachrichten beeinflussen.
    - Verzögerungsfehler beschreiben Fehler, welche Prozesse oder Nachrichten außerhalb der in der Spezifikation erlaubten Verarbeitungs- bzw. Transferzeit um eine zusätzliche Zeit $t > 0$ verzögern. [KS08]

- Unterlassungsfehler beschreiben Fehler, in Folge derer das Empfangen oder das Senden einzelner Nachrichten gemäß Spezifikation nicht stattfindet. Unterlassungsfehler können als Untermenge der Verzögerungsfehler gesehen werden mit $t = \infty$. [Cri+95; KS08]
- Ausfallfehler beschreiben das vollständige Ausfallen von Komponenten, so dass ab dem Zeitpunkt des Auftretens des Fehlers, alle nachfolgenden Empfangs- und Sendevorgänge der betroffenen Komponenten nicht stattfinden. Folglich kann auch der Ausfallfehler als Sonderfall des Unterlassungsfehlers gesehen werden. [Cri+95; KS08]
- Fail-Stop-Fehler stellen einen speziellen Fall des Ausfallfehlers dar. Während der Ausfall von Komponenten gemäß des Ausfallfehlers nicht zwangsläufig zur Folge hat, dass betroffene fehlerfreie Komponenten davon Kenntnis erlangen, ist gerade dies beim Fail-Stop-Fehler der Fall. Hierfür muss ein redundanter Mechanismus einer beliebigen Form existieren, der genau dies sicherstellt [KS08]. Ist dies nicht möglich, kann diese Fehlermenge als nicht existent angesehen werden.

- Berechnungs- und Logikfehler sind solche, welche auf fehlerhafte Berechnungen bzw. fehlerhafte Prozessoperationen zurückzuführen sind, soweit es sich nicht um Datenfehler handelt.
  - Duplikation bezeichnet das Senden einer fehlerfreien Nachricht in mehrfacher Ausführung.
  - Babbling Component bezeichnet eine Komponente eines verteilten Systems, welche außerhalb ihrer Spezifikation willkürliche Nachrichten versendet.
  - Fehlerhafte Weiterleitung bezeichnet das Senden einer Nachricht an eine falschen Komponente.
- Datenfehler bezeichnet im Allgemeinen Fehler, in Folge derer Nachrichteninhalte verfälscht werden.
  - Bitflips können aufgrund von Hardwaredefekten oder äußerlichen physikalischen Einflüssen (Stromschwankungen, Radioaktivität, Beschädigung) auftreten.
  - Datenmanipulation können aufgrund von fehlerhaften Berechnungen auftreten und einzelne bzw. Gruppen von Bits verfälschen. Datenmanipulation kann als Spezialfall von Bitflips gesehen werden.
  - Feldmanipulation beschreibt einen Spezialfall der Bitflips mit Rechenfehlern als Ursache. In dieser Variante ist die Nachrichtenverfälschung auf

einzelne oder mehrere Felder einer Nachricht beschränkt, so dass Effekte mit semantischen Folgen eintreten können. Feldmanipulation kann als Spezialfall von Datenmanipulation angesehen werden.

- Maskerade ist eine von einer Menge von Varianten der Feldmanipulation in welcher das Sender-Identifikationsfeld (Sender MAC, Sender ID, ...) der Nachricht manipuliert wird. Diese Variante ist bemerkenswert, weil Verfälschungen dieses Felds zur Folge haben, dass sich Netzwerkknoten als andere ausgeben können.

In der Literatur wird häufig eine deutlich gröbere Fehlerklassifizierung, angepasst auf das jeweilige Gebiet angeführt. Dieses beschränkt sich in dem Fall von Systemen mit Nachrichtenaustausch auf den Unterlassungsausfall, Zeitfehler und Byzantinische Fehler. Abbildung 2.7 skizziert diese grobe Unterteilung inklusive der detaillierten Klassifizierung eingebettet nach [Cri+95; KS08]. Gröbere Unterteilungen sind insbesondere dann sinnvoll, wenn man verschiedene Fehlerarten mit den selben redundanten Mitteln tolerieren kann. Die Wichtigkeit der Unterschiede der Granularität der Fehlerklassifizierungen, werden im Laufe der Arbeit anhand von Beschreibungen unterschiedlicher Protokolle verdeutlicht.


Abbildung 2.7.: Grobe Fehlerklassifizierung

Fehler müssen selbstverständlich nicht zwingend einer speziellen Fehlerart angehören, sondern dürfen, wie in der Realität üblich, eine Kombination aus verschiedenen Fehlerarten sein. Betrachtet man als Beispiel eine Komponente $p \in \mathcal{P}$, welche eine Nachricht $m \in \mathcal{M}$ empfängt und an die Komponenten $q_1, q_2, q_3 \in \mathcal{P}$ weiterleiten soll, dann kann es durchaus passieren, dass $p$

- die Nachricht $m$ an $q_1$ unverfälscht aber verzögert sendet,
- eine Verfälschung der Nachricht $m'$ an $q_2$ sendet und
- den Sendevorgang an $q_3$ unterlässt.

Dieses Beispiel würde folglich eine Kombination aus einem Verzögerungsfehler, einem Byzantinischen Fehler, einem Unterlassungsfehler und einem Bitflip darstellen.

Die Toleranz der zuvor beschriebenen Fehlerarten kann im Allgemeinen auf unterschiedliche Weisen mit unterschiedlichen redundanten Mitteln erzielt werden. Genaueres ist abhängig von der Spezifikation der jeweiligen fehlertoleranten Protokolle und wird in diesem Zusammenhang in den nachfolgenden Kapitel beschrieben.

Kapitel 3

# Fehlertoleranter unteilbarer Rundspruch und Übereinstimmung in Bridge-verbundenen Netzwerken

Dieses Kapitel der Arbeit stellt das neue Kommunikationsprotokoll *Fault-Tolerant Atomic Broadcast and Agreement in Bridge-Connected Networks* vor. Dabei handelt es sich um ein fehlertolerantes Protokoll zur Erzielung von Übereinstimmung unter einer Menge von Rechenkomponenten, welche über Bridges in einem Netzwerk über Punkt-zu-Punkt-Verbindungen untereinander kommunizieren. Hierbei wird Übereinstimmung mittels dem Konzept des unteilbaren Rundspruchs (Atomic Broadcast) und die geforderte Fehlertoleranz durch unterschiedliche Arten der Redundanz erzielt.

Der Beginn der Entwicklung des Protokolls wurde an der *Universität Duisburg-Essen*, am Lehrstuhl *Verlässlichkeit von Rechensystemen* von Prof. Dr. Klaus Echtle initiiert, woraufhin diesbezügliche Forschung seit 2014 durchgeführt wurde. In Folge dessen entstand die erste Publikation über FABAN [EF16], welche zunächst das Protokoll für die Toleranz von *einem* Fehler thematisierte.

Der Zweck dieses Kapitels ist es, den Stand des noch sehr jungen Protokolls zu Beginn der Forschung für diese Arbeit darzulegen, um eine klare Grenze zu der selbst geleisteten Arbeit zu ziehen. Die nachfolgenden Abschnitte beschreiben zunächst die Protokollidee für die Toleranz von $f = 1$ Fehlern, skizzieren anschließend die Ziele und die Idee für die Toleranz von $f > 1$ Fehlern und schließen das Kapitel mit zu dem damaligen Zeitpunkt noch offenen Fragen und Problemen ab.

## 3.1. Protokollbeschreibung

Das Kommunikationsprotokoll FABAN ist ein Protokoll zur Erzielung von Übereinstimmung, indem dieses eine neue Lösung für das Problem des Atomic Broadcasts in synchronen Systemen darstellt. Hierbei verfolgt FABAN das Ziel, möglichst wenige Einschränkungen bezüglich der Art der zu tolerierenden Fehlern zu machen und gleichzeitig den Mehraufwand bei Kommunikation und Daten gering zu halten.

Eine grundlegende Schwierigkeit des Übereinstimmungsproblems liegt darin, dass Fehler in einzelnen Komponenten eines Systems Einfluss auf andere fehlerfreie Komponenten des Systems nehmen können, indem verfälschte oder verzögerte Nachrichten propagiert oder gar unterdrückt werden. Zudem muss in Betracht gezogen werden, dass protokollspezifische Schlüsselentscheidungen, aufgeteilt auf unterschiedliche Komponenten, ebenfalls von Fehlern betroffen sein können, so dass eine hinreichend genügend große Redundanz vorhanden sein muss, um diese Fehler zu erkennen bzw. zu tolerieren. Ein Beispiel für die Separierung der Rollen findet sich in der Protokollfamilie *Paxos* mit den Rollen *Proposer*, *Coordinator*, *Acceptor* und *Learning Node* wieder. [LM04; Lam06; Mar+10]

Die Grundidee von FABAN basiert darauf, einerseits Schlüsselentscheidungen nicht an einer einzelnen Stelle zu bündeln, sondern bestimmte Schlüsselentscheidungen auf genügend viele unterschiedliche Orte zu verteilen, um dadurch das Risiko von Fehlfunktionen zu reduzieren und andererseits die zu verteilende Nachricht auf unterschiedlichen Wegen an alle Empfänger zuzustellen. Abschließend ergänzt werden diese beiden Kernmechanismen durch gängige Hilfsmittel wie z.B. Zähler und Signaturen, um eine möglichst große Breite der Fehlertoleranz zu erzielen. Eine Realisierung von FABAN kann am sinnvollsten auf der *Vermittlungsschicht* des OSI-Referenzmodells erfolgen.

Die nachfolgenden Unterabschnitte beschreiben vollständig die Funktionsweise des Protokolls für die Toleranz von $f = 1$ Fehler.

### 3.1.1. Systemannahmen

FABAN ist speziell für verteilte Systeme entwickelt worden, welche über *physikalische Verbindungen*, auch *Links* genannt, miteinander kommunizieren. Komponenten des Systems werden neben den bereits erwähnten *Links* in folgende Arten unterteilt:

- Netzwerkknoten, oder auch *Sender- und Empfängerknoten*, stellen Rechenkomponenten dar, welche autonom ihre Arbeit ausführen und Informationen mit anderen Netzwerkknoten des verteilten Systems austauschen. Ein Netzwerkknoten darf hierbei maximal mit einer Bridge über einen Link verbunden sein, während direkte Verbindungen zwischen Netzwerkknoten nicht erlaubt sind.

- Bridges stellen zusätzliche Komponenten des verteilten Systems dar, deren Aufgabe es ist, eine indirekte Kommunikation zwischen den Netzwerkknoten über die Bridges zu realisieren. Verbindungen dürfen sowohl zu Netzwerkknoten als auch zu anderen Bridges existieren. Die maximale Anzahl der erlaubten Verbindungen ist hierbei von der verwendeten Hardware abhängig und wird vom Protokoll nicht beschränkt.

Netzwerke dieser Art werden als *Bridge-Connected-Networks (BCNs)* oder auch *Bridge-verbundene-Netzwerke* bezeichnet. Sowohl die Begrifflichkeiten der *Netzwerkknoten* und *Bridges*, als auch BCNs, entstammen dem Ethernet-Bereich, welcher im Rahmen des Projekts *IEEE 802* behandelt und standardisiert wird. Während allgemeine Ethernet-Netze keine Echtzeiteigenschaften bieten, arbeitet die *Time-Sensitive Networking Task Group (IEEE 802.1)* an Mechanismen zur Erweiterung von Ethernet-basierter Kommunikation um Echtzeiteigenschaften. Dazu zählen insbesondere die *Zeitsynchronisation* sowie das *Scheduling und Traffic-Shaping*. [SS16; IEE21]

Abbildung 3.1 zeigt drei Beispiele von BCNs.

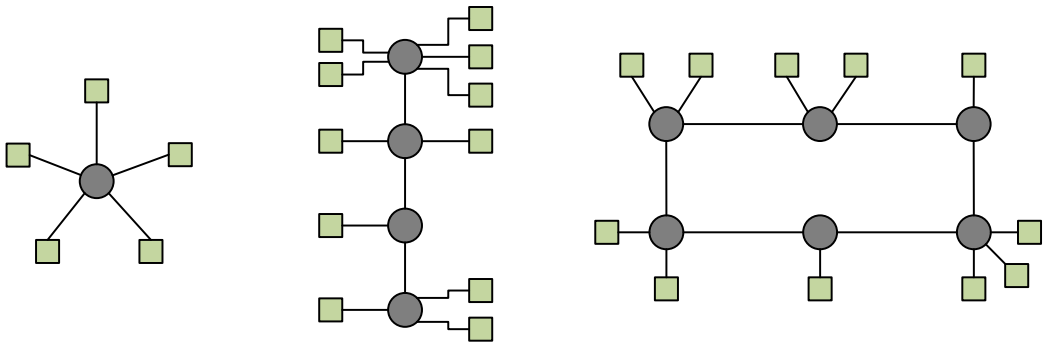


Abbildung 3.1.: Einfache Beispiele für Bridge-Connected-Networks, bestehend aus Netzwerkknoten (grün) und Bridges (grau)

FABAN kann nicht auf allen Netzwerken dieser Art angewendet werden, sondern nur auf solchen, auf denen genügend viele redundante Pfade zwischen der Bridge, mit welcher der Sender verbunden ist, und jeder Bridge, mit welcher ein Empfänger verbunden ist, existieren. Dies wird in Kapitel 5 genauer untersucht.

Neben der allgemeinen Art und dem allgemeinen Aufbau der zugrunde gelegten Netzwerke werden folgende Annahmen an BCNs festgelegt, welche zwingende Voraussetzungen für die erfolgreiche Anwendung von FABAN darstellen:

- Synchrone Uhren
  Die lokalen Uhren aller Komponenten, d.h. aller Netzwerkknoten und aller Bridges, müssen synchronisiert sein. Kleinere Abweichungen der lokalen Uhren, geschuldet

Hardwaretoleranzen durch unterschiedliche Drift oder hinnehmbaren Differenzen durch das Uhrensynchronisationsprotokoll, sind hierbei erlaubt, werden in der FABAN-Spezifikation beachtet, und spiegeln realitätsnahes Verhalten wieder. Um dies zu gewährleisten sollte in einer realen Umsetzung ein nebenläufig bzw. periodisch ausgeführtes Uhrensynchronisationsprotokoll benutzt werden und es muss die mögliche Abweichung zwischen beliebigen fehlerfreien Uhren zwingend in Betracht gezogen werden. Des Weiteren muss die Netzwerktopologie stets genügend Redundanz aufweisen, damit diese dem gewählten Uhrensynchronisationsprotokoll genügt. [KO87; ST87; Moz19]

- Beschränkte pro Hop Latenz
  Die maximale Dauer der Übertragung einer Nachricht zwischen zwei direkt physikalisch verbundenen Komponenten des Systems muss nach oben beschränkt sein. Mit *Ethernet* hat der Großteil der Menschheit mit der gängigsten Variante eines BCN täglich im häuslichen Gebrauch zu tun. Diese Netzwerkart kann beschränkte Pro-Hop-Latenzen zwar nicht garantieren, kann allerdings mit einem probabilistischen Ansatz mit einer genügend großen Schranke für die Pro-Hop-Latenz dennoch in Betracht gezogen werden. Schranken kann man einerseits analytisch herleiten, andererseits aber auch auf experimentelle Art und Weise mit zusätzlichen Toleranzen approximieren. Dieser Ansatz wird im Laufe der Arbeit in Kapitel 8 näher analysiert. Des Weiteren werden obere Schranken für die Übertragung über einen Hop bei Ethernet mit Time-Sensitive-Networking (TSN) garantiert, so dass insbesondere diese Ethernet-Variante optimale Voraussetzungen für FABAN bietet.

### 3.1.2. Protokollablauf

Möchte ein Sender eine FABAN-Nachricht $m \in \mathcal{M}$ an alle anderen Netzwerkknoten des verteilten Systems senden, so geschieht dies in dem Fall $f = 1$, wobei $f$ die zu tolerierende Anzahl von Fehlern bezeichnet, auf zwei redundanten Wegen, welche im Zusammenhang mit FABAN als *Wellen* bezeichnet werden. Insbesondere im Fall $f = 1$ wurden dabei in bisherigen Publikationen die Begriffe *Welle* und *Gegenwelle* benutzt, welche in darauffolgenden Forschungsarbeiten und Publikationen, mit zunehmender Anzahl an Wellen für die Toleranz von mehr Fehlern, als auch in dieser Dissertation nur noch vereinfacht als nummerierte Wellen bezeichnet werden. [EF16; FE18]

Von FABAN-Nachrichten durchlaufene Bridges handeln abhängig von den bisher durchlaufenen Bridges auf unterschiedliche Art und Weise. Die erste durchlaufene Bridge agiert als *Distributing Bridge (DB)*, die zweite als *Checking Bridge (CB)* und alle darauffolgenden Bridges als *Forwarding Bridges (FBs)*. Wichtig ist zu bemerken, dass eine Bridge in speziellen Fällen wiederholt passiert werden kann. Ist dies der Fall, so verhält diese sich entsprechend der Anzahl der bereits durchlaufenen Hops als DB, CB

oder FB. In diesem Zusammenhang wird ein *Hop* gleichgesetzt mit der Übertragung einer Nachricht von einer Bridge zu einer benachbarten Bridge bzw. zwischen einem Netzwerkknoten und einer Bridge (vgl. Abbildung 3.2).

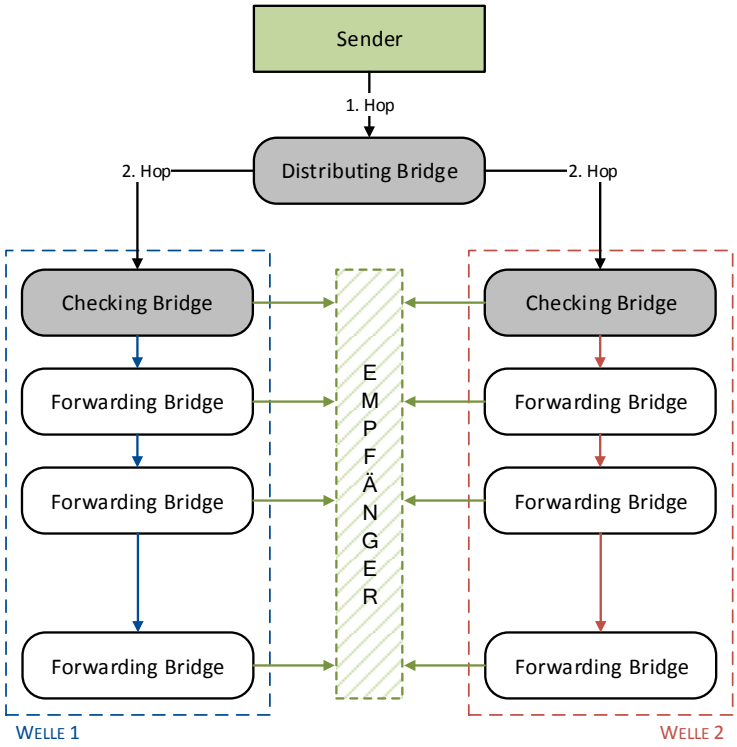



Abbildung 3.2.: Grundlegender Kommunikationsablauf von FABAN. Die Mengen der Forwarding Bridges im Kommunikationsablauf beider Wellen (rot und blau gestrichelt umrahmte Bereiche) sind hierbei stets identisch, beide Wellen durchlaufen die Forwarding Bridges allerdings in unterschiedlicher Reihenfolge. Der zentral grün gestrichelt umrandete und schraffierte Bereich stellt hierbei auch die Menge aller Empfänger da, wobei jeder Empfänger mit exakt einer Bridge verbunden ist.

FBs stellen die Nachrichten schlussendlich an alle Empfänger zu. Dieses Kommunikationsschema wird in Abbildung 3.2 schematisch skizziert. Die detaillierte Verhaltensweise wird in den nachfolgenden Punkten genauer beschrieben.

**Nachrichtenaufbau** Eine FABAN Nachricht, siehe Abbildung 3.3, benötigt neben den eigentlich zu sendenden Daten zusätzliche Informationen, um die erwünschte Fehlertoleranz zu ermöglichen. Neben essentiellen Informationen wie der *Sender ID* $i_S$, einer im System einheitlichen Identifikationsnummer des Senders, und der *Sequenznummer* $n_S$,

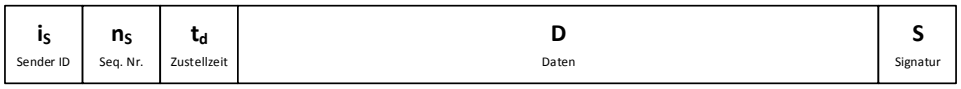



Abbildung 3.3.: Nachrichtenaufbau einer FABAN Nachricht

einer fortlaufenden, senderspezifischen Zahl aus $\mathbb{N}_0$, berechnet der Sender der Nachricht zusätzlich die *Zustellzeit* $t_d$, welche festlegt, wann die Nachricht bei den Empfängern gemäß der lokalen Uhrzeit der Empfänger verarbeitet werden muss. Ohne eine solche Zeitangabe wäre eine globale Totalordnung des Nachrichtenempfangs ausgeschlossen. [Cri+95] Die Zustellzeit lässt sich aufgrund der Annahme der beschränkten pro-Hop-Latenzen in Abhängigkeit von

- der maximalen Anzahl der Hops $N_{hops}$ zwischen Bridges entlang beider Wellen,
- der maximalen Dauer der Übertragung einer Nachricht zwischen zwei Bridges bzw. einer Bridge und einem Netzwerkknoten $\delta_{hop}$ und
- der maximalen Verweildauer einer Nachricht in einer Bridge bzw. einem Netzwerkknoten $\delta_B$

wie folgt berechnen:

$$t_d = (N_{hops} + 2) \cdot \delta_{hops} + (N_{hops} + 1) \cdot \delta_B. \tag{3.1}$$

Hierbei ist hervorzuheben, dass $N_{hops}$ nur die Hops zwischen Bridges bezeichnet und den ersten sowie den letzten Hop vom Sender zu der Distributing Bridge sowie von der Forwarding Bridge zum jeweiligen Empfänger, aus Gründen der Konsistenz weiterer Rechnungen im Laufe dieser Arbeit, ausschließt.

Zuletzt wird über diese drei Felder zusammen mit den eigentlichen Nutzdaten eine Signatur gebildet und an die Nachricht angehängt. Mit Ausnahme des Feldes der Sender ID, welches mindestens $\lceil \ln(|\mathcal{N}|) \rceil$ Bits groß sein muss, damit alle möglichen Sender aus $\mathcal{N}$ unterschieden werden können, unterliegt die Größe der anderen Felder keinen weiteren Anforderungen und kann in einer Implementierung genügend groß gewählt werden. Eine FABAN-Nachricht $m$ mit der angehängten Signatur $S$ wird im folgenden mit $m : S$ bezeichnet.

**Signatur** Als Signatur darf jedes beliebige, dem Anwendungsfall genügende Verfahren benutzt werden. Weitere Anforderungen werden seitens der Protokollbeschreibung nicht gemacht. Im Allgemeinen sind auch kryptographisch schwache Verfahren, die allerdings gut technische Fehler erkennen können, vollkommen ausreichend um den angestrebten

Anwendungsgebieten zu genügen. Besonders eignen sich Signaturverfahren, die ausschließlich der Fehlererkennung (und nicht der Aufdeckung von menschlichen Angriffen) dienen. Als Beispiel sind hier die CRC-basierten Signaturverfahren mit ungerader Multiplikation zu nennen. [Ech18]

Während der Weiterleitung einer FABAN-Nachricht von einer Bridge zur nächsten wird diese nicht wiederholt signiert, allerdings beim Durchlaufen der Distributing Bridge als auch der Checking Bridge jeweils modifiziert um zu gewährleisten, dass sowohl die Distributing Bridge als auch die Checking Bridge exakt einmal durchlaufen werden. Das Durchlaufen der Forwarding Bridges entsprechend der Wellen wird nicht durch Signaturen sichergestellt, weil für den Fall von fehlerhaftem Verhalten Redundanz durch die andere Welle gegeben ist. Die Signaturmodifikation basiert auf einfachen Operationen, die abhängig sind von drei globalen konstanten Masken, die als $d_{mask}$, $c_{mask}$ und $r_{mask}$ bezeichnet werden. Die ersten beiden Masken haben die gleiche Länge wie die Signatur des gewählten Verfahrens, dürfen sonst frei gewählt werden und finden in Bridges, die als DB oder CB agieren, Anwendung. Die letzte Maske wird in den Empfängerknoten verwendet und wird gemäß folgender Formel berechnet

$$r_{mask} \coloneqq \mathrm{ROR}(d_{mask}, 1) \veebar \mathrm{ROR}(c_{mask}, 2), \tag{3.2}$$

wobei $\mathrm{ROR}(v, n)$ die Rotation des Wertes $v$ um $n$ Bits nach rechts bezeichnet. Die genaue Signaturmodifikation in den Bridges als auch in den Empfängern wird in den Beschreibungen der jeweiligen Komponenten folgen. Die Grundidee ist die Separierung der Signaturmodifikation innerhalb der entsprechenden Bridges in unterschiedliche und unabhängige Operationen, welche an unabhängigen Stellen durchgeführt werden. Konkret gewählt wurden hierfür die unabhängigen und gleichzeitig primitiven Funktionen der Rotation innerhalb des eingehenden Ports sowie der XOR-Verknüpfung innerhalb des ausgehenden Ports in Distributing Bridge und Checking Bridge. In Kombination mit der Umkehrung all dieser Signaturmodifikationen der Distributing Bridge sowie Checking Bridge bei den Empfängern, wird eine Schutzwirkung gegeben, welche die Feststellung garantiert, ob eine Nachricht DB und CB tatsächlich in dieser Reihenfolge durchlaufen hat. Die Separierung der primitiven Operationen in eingehende und ausgehende Ports reduzieren das Nicht-Erkennen eines Fehlerverhaltens einer Bridge, welche sich aufgrund von technischen Fehlern fälschlicherweise als DB oder CB ausgibt. Die in den nachfolgenden Abschnitten beschriebenen Signaturmodifikationen sind in Abbildung 3.4 dargestellt.

**Beispiel:** Für frei gewählte Masken $d_{mask} = 421B78C8_{:16}$ und $c_{mask} = EF869AE3_{:16}$ erhält man für die Empfängermaske

$$\Rightarrow r_{mask} = 210DBC64_{:16} \veebar FBE1A6B8_{:16} = DAEC1ADC_{:16} \tag{3.3}$$

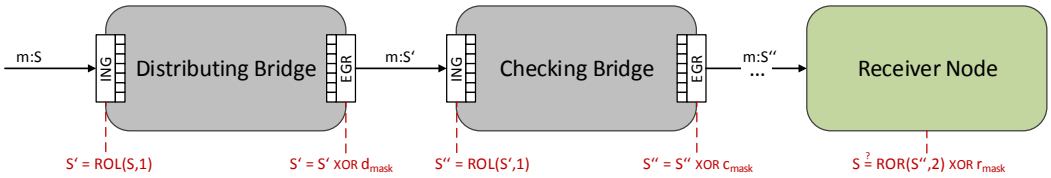



Abbildung 3.4.: Übersicht der FABAN-Signaturmodifikation

**Distributing Bridge** Sobald eine FABAN-Nachricht $m : S$ direkt von dem Sender eintrifft, hat die Bridge die Aufgabe als Distributing Bridge die Nachricht zu duplizieren und anschließend, jeweils für eine Welle, an zwei benachbarte Bridges, welche als Checking Bridges agieren werden, zu versenden. Beim Eintreffen der Nachricht wird im eingehenden Port die Signatur der Nachricht um einen Bit nach links rotiert. Nach dem Duplizieren der Nachricht wird die Signatur in den ausgehenden Ports XOR-verknüpft mit der globalen $d_{mask}$, d.h. es wird $m : S'$ an die CBs gesendet mit (vgl. Abbildung 3.4)

$$S' := \mathrm{ROL}(S, 1) \underline{\vee} d_{mask} \tag{3.4}$$

**Checking Bridge** Erhält eine Bridge eine Nachricht $m : S'$ von einer Distributing Bridge, so hat sie die Aufgabe als Checking Bridge zu überprüfen, ob diese Nachricht noch rechtzeitig bei allen Empfängern vor der errechneten Zustellzeit $t_d$ ankommen kann, vorausgesetzt, dass alle nachfolgenden Bridges fehlerfrei arbeiten. Ist dies nicht der Fall, dann ist davon auszugehen, dass die Nachricht in der Distributing Bridge unerlaubt verzögert wurde. In diesem Fall wird die Nachricht unterdrückt und folglich keine Ausbreitung entlang der betreffenden Welle initiiert. Ist die Überprüfung positiv, dann wird die Nachricht zu den Bridges weitergeleitet, welche gemäß des Routings der Welle dieser CB, Ziele der Weiterleitung sind. Beim Eintreffen der Nachricht wird im eingehenden Port die Signatur um ein Bit nach links rotiert und vor dem Weiterleiten an benachbarte Bridges in den ausgehenden Ports XOR-verknüpft mit der globalen $c_{mask}$, d.h. es wird $m : S''$ an die FBs gesendet mit (vgl. Abbildung 3.4)

$$S'' := \mathrm{ROL}(S', 1) \underline{\vee} c_{mask} \tag{3.5}$$

**Forwarding Bridge** Hat eine Nachricht eine Distributing Bridge und eine Checking Bridge durchlaufen, so verhält sich jede weitere durchlaufene Bridge als FB. Eine Forwarding Bridge leitet lediglich die empfangene Nachricht gemäß den Wellenbeschreibungen weiter und hat keine sonstigen Aufgaben.

**Zustellung** Empfängt ein Netzwerkknoten eine FABAN-Nachricht, so prüft dieser zunächst, ob die Nachricht unerlaubt verfälscht wurde. Dafür werden mit (vgl. Abbildung

3.4)

$$S := \mathrm{ROR}(S'', 2) \veebar r_{mask} \tag{3.6}$$

die Signaturmodifikationen der DB und der CB rückgängig gemacht und anschließend gemäß des verwendeten Signaturverfahrens die Nachricht auf Verfälschungen überprüft. Sollte die Nachricht nicht verfälscht worden sein, aber aufgrund eines anderen Fehlers nicht exakt einmal sowohl die DB als auch die CB, in dieser Reihenfolge, durchlaufen haben, würde das durch die Überprüfung der Signatur in Kombination mit der Signaturmodifikation dennoch als Verfälschung und folglich als Fehler erkannt werden. Bei einem negativen Signaturtest, einer zu spät eingetroffenen Nachricht ($t_d$ überschritten) oder wenn die Nachricht protokollkonform über eine andere Welle bereits eingetroffen ist, wird die empfangene Nachricht verworfen. Andernfalls wird die Nachricht akzeptiert und zum gewünschten Zeitpunkt $t_d$ verarbeitet.

Die Verzögerung zwischen tatsächlichem Empfang der Nachricht und die Verarbeitung zu dem in der Nachricht festgelegten Zeitpunkt $t_d$ ist deshalb notwendig, um sicherzugehen, dass alle Empfänger die Nachricht zum gleichen Zeitpunkt, gemessen am lokalen Zeitverständnis jedes Empfängers, und somit folglich auch alle empfangenen Nachrichten in der gleichen Reihenfolge verarbeiten. Dies ist eine zwingende Voraussetzung um die Eigenschaften *Atomicity*, *Order* und auch *Termination* der Definition des Atomic Broadcasts für synchrone Systeme, Definition 2.14, zu gewährleisten. Kleinere Abweichungen der lokalen Uhren der Empfänger sind hierbei nicht relevant, da diese keinen Einfluss auf die Reihenfolge der zu verarbeitenden Nachrichten haben, sondern lediglich, global gesehen, kleinere Abweichungen zwischen den Verarbeitungszeitpunkten entstehen können. Sollten Nachrichten ein und des selben Senders zur gleichen Zeit verarbeitet werden, so wird die Verarbeitung gemäß Sequenznummer $n_S$ vollzogen. Analoges gilt für Nachrichten unterschiedlicher Sender mit gleicher Zustellzeit $t_d$, wobei in diesem Fall gemäß einer globalen Totalordnung der Sender ID $i_S$, beispielsweise lexikographische oder numerische Ordnung, die Reihenfolge der Verarbeitung festgelegt wird.

**Erkennung der Bridge-Rolle** Empfängt eine Bridge eine FABAN-Nachricht, so muss diese erkennen, ob sie für diese Nachricht eine Distributing, Checking oder Forwarding Bridge ist. Ein ursprünglicher Vorschlag für die Realisierung dieser Entscheidung war die Einführung eines Hopzählers in das in Abbildung 3.3 vorgestellte Nachrichtenformat. Ein Hopzähler als alleinige Referenz zur Weiterleitung von Nachrichten ist allerdings nur in speziellen Graphen wie zum Beispiel Ringen ausreichend. In diesem konkreten Beispiel kann jede Bridge lokal entscheiden, gemäß welcher Rolle sie sich zu verhalten hat und die Nachricht auf dem (bezüglich des Rings) komplementären Port des Eingangsports weiterleiten. Im Allgemeinen ist dies jedoch nicht möglich, wenn Bridges nur geringfügig komplexer vernetzt sind und Netzwerke keine speziellen Struktureigenschaften aufweisen.

Aus diesem Grund werden im Allgemeinen Routing-Tabellen benötigt, in welchen mit nur geringfügig erhöhtem Speicherbedarf die Information über die Bridge-Rollen gespeichert werden kann. Im Zusammenhang mit der Toleranz von Mehrfachfehlern gewinnt der Hopzähler - neben dem Nutzen als redundante Information - jedoch einen zusätzlichen Nutzen, welcher nicht durch Routing-Tabellen abgedeckt werden kann. Dies wird in Kapitel 6 genauer thematisiert.

### 3.1.3. Anforderungen an Wellen

Da das Konzept der redundanten Verteilung von Nachrichten über Wellen den Kernaspekt von FABAN darstellt, ist es naheliegend, dass Wellen gewissen Regeln unterliegen müssen. Hierbei steht das Ziel im Vordergrund, dass eine fehlerhafte Komponente des Systems entweder dazu führt, dass gar keine FABAN-Nachricht verteilt wird, oder aber höchstens eine der beiden Wellen beeinflusst, so dass alle fehlerfreien Empfänger über die andere Welle die Nachricht empfangen. Dieses Ziel führt zu einer Reihe von Regeln, mit Hilfe derer Wellen für gegebene Netzwerktopologien konstruiert werden können.

**Regel 1** Es darf keine gemeinsame, parallele Kante zwischen zwei benachbarten Bridges in beiden Wellen vorkommen.

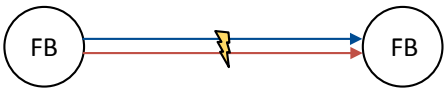



Abbildung 3.5.: Regel 1

**Regel 2** Nach der Initiierung der Wellen in den jeweiligen Checking Bridges, muss die jeweilige Welle die Distributing Bridge und die Checking Bridge der anderen Welle erreichen, darf von diesen aber nicht weiter verteilen.

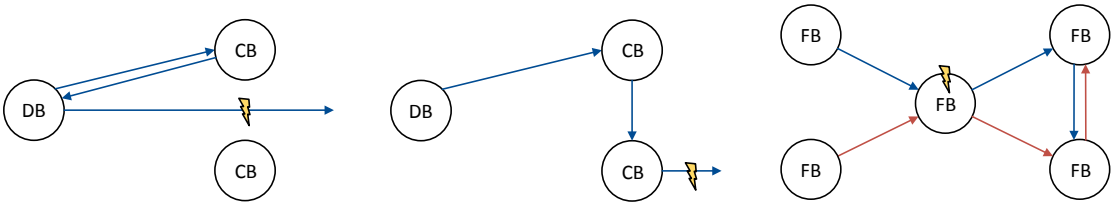



Abbildung 3.6.: Visualisierung der Verletzungen der Wellenregeln 2 und 3

**Regel 3** Zwischen den jeweiligen Checking Bridges beider Wellen und einer beliebigen anderen Bridge des Systems, darf auf beiden Pfaden keine gemeinsam passierte Bridge existieren.

Ein vollständiges Beispiel zweier korrekter Wellen, welche die obigen Regeln einhalten, ist in Abbildung 3.7 zu sehen.

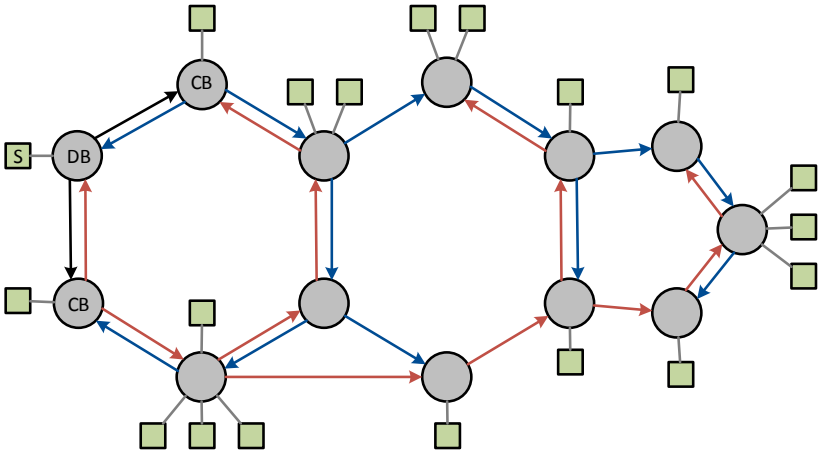



Abbildung 3.7.: Vollständiges Beispiel zweier Wellen für festgelegte Sender, Distributing und Checking Bridges

### 3.1.4. Toleranz von Fehlern

Durch die Kombination verschiedener Redundanzmechanismen ist FABAN in der Lage alle Fehlerarten mit nur wenigen Ausnahmen zu tolerieren. Hierbei werden die folgenden Ausnahmen festgestellt, wobei erstere natürlicherweise nie vermeidbar ist und letztere auf die an FABAN zugeschnittene Signaturmodifikation zurückzuführen ist.

**Ausnahme 1** Nachrichtenverfälschungen, welche nicht durch das gewählte Signaturverfahren erkannt werden können, werden von FABAN nicht toleriert.

Die Erkennung von Nachrichtenverfälschungen ist vollständig von den Eigenschaften des benutzten Signaturverfahrens abhängig. Da Signaturen Abbildungen von Nachrichten auf eine im Vergleich zur Nachrichtenlänge stark reduzierte Wertemenge sind, können selbstverständlich unterschiedliche Nachrichten auf ein und denselben Wert abgebildet werden, so dass solche speziellen Nachrichtenverfälschungen durch das Signaturverfahren

unerkannt bleiben würden. Es gibt allerdings Signaturverfahren und Prüfsummen, welche für bestimmte Fälle das Erkennen von Bitflips garantieren können. Ein Beispiel hierfür ist die Prüfsumme CRC, welche abhängig vom gewählten CRC-Polynom sowie der CRC-Länge unterschiedliche Eigenschaften bzgl. der Erkennungswahrscheinlichkeiten von Bitflips aufweist. Neben CRC-Polynomen, durch welche zu 100% sowohl zwei als auch eine beliebige ungerade Anzahl von Bitflips erkannt werden können, können Polynome für größere CRC-Längen gefunden werden, welche auch eine höhere Anzahl von Bitflips garantiert erkennen [RG88; KC04; RK06]. Des Weiteren können CRC und Signaturen kombiniert werden, indem beispielsweise die Signatur einer Nachricht als Verkettung einer CRC-Berechnungsfunktion sowie einer (Fehlertoleranz-)Signaturfunktion berechnet wird.

**Ausnahme 2** Sollte eine Bridge derart fehlerhaft sein, dass diese die Signaturmodifikationen der Distributing Bridge und der Checking Bridge kombiniert in dieser Reihenfolge anwendet, dann kann FABAN diesen Fehler nicht tolerieren.

Sollte dieser Fehler eintreten, dann würde sich eine Bridge als eine Kombination aus DB und CB ausgeben, was vom Protokoll unbemerkt bleiben würde. Hierbei ist diese Bridge folglich fehlerhaft und es ist nicht mehr gewährleistet, dass die Nachrichtenausbreitung entweder unterdrückt, oder aber erfolgreich abgeschlossen werden kann. Da die Signaturmodifikationen an die eingehenden und ausgehenden Ports der Bridges gebunden sind, kann dieser Fehler als Folge von technischen Fehlern als extrem unwahrscheinlich angesehen werden.

Die Fehlertoleranz wird bei FABAN durch Kombinationen unterschiedlicher Redundanzarten realisiert. In allen Fällen handelt es sich um statische Redundanz, welche wiederum einerseits aus der funktionellen Redundanz in Form der Separierung der Rollen, der Duplikaterkennung in den Empfängern und den Signaturmodifikationen und andererseits aus der Informationsredundanz sowohl in Form des redundanten Routings entlang der Wellen als auch der angehängten Prüfsumme bzw. Signatur besteht. Tabelle 3.1 stellt einen Überblick über die Redundanzmittel dar, welche in ihrer Kombination die Toleranz der jeweiligen Fehlerart ermöglichen.

| | | Checking Bridges | Redundantes Routing | Duplikaterkennung | Signaturmodifikation | Signatur (+CRC) |
|---|---|---|---|---|---|---|
| Zeitfehler | Fail-Stop | | ✓ | | | |
| | Ausfallfehler | | ✓ | | | |
| | Unterlassungsfehler | | ✓ | | | |
| | Verzögerungsfehler | ✓ | ✓ | | | |
| Logikfehler | Fehlerhafte Weiterleitung | | ✓ | | (✓) | (✓) |
| | Babbling Component | ✓ | ✓ | | ✓ | ✓ |
| | Duplikation | | | ✓ | | |
| Datenfehler | Bitflips | | | | | ✓ |
| | Feldmanipulation | | | | | ✓ |
| | Maskerade | | ✓ | | ✓ | ✓ |

Tabelle 3.1.: Übersicht der Redundanzarten, durch welche FABAN die entsprechenden Fehler tolerieren kann. Fehlerhaft weitergeleitete Nachrichten können auch über die Signatur bzw. die Signaturmodifikation erkannt werden, falls die fehlerhafte Bridge die Distributing Bridge oder die Checking Bridge ist (gekennzeichnet durch Häkchen in Klammern).

Zuletzt ist anzumerken, dass nicht nur solche Fehler toleriert werden, die exakt und klar einer einzigen Fehlerart zuzuordnen sind, sondern jegliche Form des Byzantinischen Fehlers, ausgenommen die zuvor genannten Spezialfälle. Konkret bedeutet dies, dass jeder Fehler, der eine willkürliche Kombination aus den in Tabelle 3.1 aufgelisteten Fehlerarten darstellt, ebenfalls toleriert wird.

## 3.2. Probleme und offene Fragen

In der hier beschriebenen Phase der Protokollspezifikation, sind noch eine Reihe von Problemen und offenen Fragen vorhanden bzw. unbeantwortet. Ohne sie an dieser Stelle der Arbeit zu tief zu thematisieren, folgt ein kleiner Überblick darüber, bezüglich welcher Fragen Anschlussforschung betrieben wurde und was im Laufe der Arbeit tiefgründiger

aufgegriffen wird.

**Formale Verifikation der Wellenregeln** Die in Abschnitt 3.1.3 vorgestellten Wellenregeln sind formal nicht verifiziert worden. Zum einen muss der Vollständigkeit halber formal überprüft werden, dass die drei Wellenregeln tatsächlich hinreichende Bedingungen für Wellen darstellen und zum anderen sollte untersucht werden, ob diese Regeln nicht zu streng sind und tatsächlich auch gleichzeitig notwendige Bedingungen darstellen.

**Generierung von Wellen** Für kleinere bzw. auch größere, aber überschaubare Topologien, lassen sich relativ leicht Wellen händisch erzeugen. Dies wird für größere und unüberschaubare Topologien deutlich mühseliger. Des Weiteren ist das händische Erzeugen von Wellen in jedem Fall eine schlechte Lösung, so dass die Frage nach einer automatisierten Generierung und der Möglichkeit der automatischen Verifizierung von Wellen im Raum steht.

**Toleranz von Mehrfachfehlern** Im Detail wurde FABAN in der hier thematisierten Form nur für Einfachfehler entwickelt. Für die Toleranz von Mehrfachfehlern wurden einige Ideen skizziert, allerdings nie vollständig spezifiziert. [EF16] Des Weiteren haben sich die meisten als fehlerhaft herausgestellt bzw. erforderten grundlegende Anpassungen. Das Hauptproblem hierbei stellt die Toleranz eines Byzantinischen Fehlers dar, welcher die Distributing Bridge betrifft. Bestimmte Konstellationen von Fehlern in Kombination mit einem Fehler in der Distributing Bridge können gegebenenfalls zur Inkonsistenz des Atomic Broadcasts führen, was nicht ohne zusätzliche Redundanz gelöst werden kann.

**Ausarbeitung eines Implementierungskonzepts** Neben der theoretischen Ausarbeitung einer Protokollspezifikation hat ein Testen in einer Simulationsumgebung oder auf echter Hardware einen deutlichen Mehrwert. Dabei könnten Laufzeitevaluationen durchgeführt, Protokolleigenschaften zusätzlich überprüft und das Protokoll auf allgemeine Praktikabilität getestet werden. Weiterhin geben Implementierungen in einer Simulation oder auf echter Hardware Aufschluss über mögliche zusätzliche, in der Theorie nicht bedachte Anforderungen an das System bzw. das Protokoll sowie über mögliche Anreize zur Verbesserung des Protokolls.

Kapitel 4

# Stand der Technik

Das im Fokus dieser Arbeit stehende Protokoll *Fault-Tolerant Atomic Broadcast and Agreement in Bridge-Connected Networks* bedient im Kern thematisch sowohl das Gebiet der Übereinstimmungsprotokolle als auch das Gebiet der unteilbaren Rundsprüche. Dieses Kapitel gibt einen Überblick über den aktuellen Stand der Technik beider Themengebiete.

## 4.1. Übereinstimmungsprotokolle

Das Problem der Übereinstimmung in verteilten Systemen ist auf das Problem der *Byzantinischen Generäle* von Leslie Lamport zurückzuführen [LSP82]. Lamport hat in der selben Publikation neben der Definition des Problems mehrere Lösungen vorgeschlagen. Das Protokoll *Oral Messages* (OM) basiert auf Kommunikation zwischen Komponenten ohne der Nutzung zusätzlicher Signaturen und benötigt bei der Toleranz von $f \geq 1$ Fehlern mindestens $n \geq 3f + 1$ Komponenten. Zugleich wurde bewiesen, dass $3f + 1$ die Mindestanzahl von Komponenten für beliebige fehlertolerante Übereinstimmungsprotokolle ohne Signaturen ist. Komponenten müssen in global festgelegten Zeitintervallen ihre jeweiligen Nachrichten senden bzw. empfangen, um ausbleibende Nachrichten detektieren zu können und benötigen insgesamt $f + 1$ Phasen, das heißt jede Nachricht jeder Komponente muss bis zu $f + 1$ mal weitergeleitet werden, bevor Empfänger eine Entscheidung auf Basis der empfangenen Daten treffen können. Das Protokoll *Signed Messages* (SM) verwendet zusätzlich Signaturen, um Verfälschungen von Nachrichteninhalten erkennbar zu machen und benötigt als Folge dessen nur $n \geq f + 1$ Komponenten, um ebenfalls in $f + 1$ Phasen Übereinstimmung erzielen zu können. Beide Lösungsvorschläge implizieren die Möglichkeit der direkten Kommunikation der einzelnen Komponenten, wie beispielsweise in Feldbussystemen, und erfordern einen gewissen Grad

der Synchronität. Dolev und Strong haben beweisen, dass tatsächlich mindestens $f+1$ Phasen notwendig sind, um Übereinstimmung mit signierten Nachrichten gemäß SM erreichen zu können [DS83]. Mit dem *Pendelprotokoll* wurde ein weiteres Protokoll für Systeme mit direkter Kommunikation vorgestellt, welches nur dann mehr Nachrichten benötigt, als mindestens notwendig, wenn tatsächlich Fehler während der Synchronisation auftreten [Ech89]. Weitere Erkenntnisse und Protokollvarianten auf Grundlage von Systemen mit direkter Kommunikationsmöglichkeit zwischen allen Komponenten und synchronisiertem Sendeverhalten und allgemeiner Synchronität wurden in den nachfolgenden Jahren und Jahrzehnten publiziert. Dabei wurden Verbesserungen bezüglich dem Nachrichtenoverhead, dem Kommunikationsaufwand und der Effizienz im Allgemeinen erzielt [LA86; Di +88; DRS90; GMY95; GM98]. Die Möglichkeiten Übereinstimmung auf drahtlosen verteilten Systemen, welche näherungsweise als Systeme mit direkten Verbindungen angenommen werden können, zu erzielen, wurde ebenfalls in jüngster Vergangenheit untersucht und vielversprechende Ansätze publiziert [Bou15a; Bou16].

Betrachtet man vollständig asynchrone Systeme, so haben Fischer, Lynch und Paterson 1983 gezeigt, dass das Problem der Übereinstimmung nicht lösbar ist, da bei beliebig wenigen fehlerhaften Komponenten $f \geq 1$ stets die Möglichkeit besteht, dass deterministische Protokolle nicht terminieren [FLP85]. Mit einem probabilistischen, nichtdeterministischen Lösungsansatz hat Ben-Or ein Protokoll vorgestellt, welches mit Wahrscheinlichkeit 1 Übereinstimmung in einem vollständig asynchronen System mit mehr als $n \geq 5 \cdot f$ Komponenten erzielt [Ben83]. Die Garantie des Erfolgs der Übereinstimmung geht dabei auf Kosten der Effizienz, da diese mit steigendem Anteil an fehlerhaften Komponenten exponentiell wächst. Ist dabei die Anzahl fehlerhafter Komponenten $f$ höchstens von der Ordnung $\mathcal{O}(\sqrt{n})$, so ist Übereinstimmung in konstanter Zeit zu erreichen und stellt eine deutliche Verbesserung zu den deterministischen Protokollen OM und SM dar. Bracha konnte die Anforderung der Anzahl der Komponenten reduzieren auf $n \geq 3 \cdot f$, indem zuverlässige Rundsprüche verwendet werden [Bra87].

Ein zu den Übereinstimmungsprotokollen gehörendes Forschungsgebiet ist die fehlertolerante Synchronisation von lokalen Uhren in verteilten Systemen. Uhrensynchronisationsprotokolle lösen das Übereinstimmungsproblem mit speziellem Anwendungszweck und sind seit den frühen 80er Jahren Forschungsthema [LM84; Hal+84; DHS86]. Die nebenläufige Ausführung eines fehlertoleranten Uhrensynchronisationsprotokolls ist eine Grundvoraussetzung für FABAN, weil die Checking Bridges entscheiden müssen, ob eine Nachricht noch rechtzeitig zugestellt werden kann. Im Hinblick auf bestimmte Netzwerktopologieklassen existieren aktuelle effiziente Lösungen, welche die Uhrensynchronisation speziell auf Ringtopologien realisieren. Diese haben durchaus Relevanz bei realer Anwendung von FABAN auf Ringtopologien [EM17; Moz18].

## 4.2. Unteilbare Rundsprüche

Übereinstimmung auf Netzwerken, die keine direkte, zuverlässige Kommunikation zwischen zwei beliebigen Komponenten des Systems ermöglichen, wird in der Wissenschaft ebenfalls seit Mitte der 80er Jahre vermehrt thematisiert. Im Allgemeinen werden zur Umsetzung von Übereinstimmungsprotokollen in Netzwerken, welche zum Beispiel wie hier über zu einer beliebigen Topologie verbundenen Bridges kommunizieren, Rundspruch-Routinen unterschiedlicher Art verwendet.

Eine der ersten Publikationen zu diesem Thema - und im Hinblick auf die Gemeinsamkeiten zu FABAN wichtigste Referenz - war die Arbeit von Flaviu Cristian, welche in der ersten Version bereits 1984 erschienen ist [Cri+95]. In dieser Publikation wurden drei Protokolle vorgestellt, welche aufeinander aufbauten und den Umfang der zu tolerierenden Fehler von Ausfall- über Verzögerungs- bis zu byzantinischen Fehlern erhöhten, aber auch die Komplexität der Implementierung der Protokolle steigerten. Die Grundidee dabei ist, Übereinstimmung durch unteilbare Rundsprüche zu realisieren, wobei der unteilbare Rundspruch durch das Prinzip der Nachrichtenflutung umgesetzt werden soll. Die für das Protokoll festgelegten Grundannahmen unterscheiden sich kaum von den Annahmen für FABAN:

- Das über Bridges verbundene Netzwerk muss zusammenhängend sein. Des Weiteren darf jedes Teilnetzwerk aus fehlerfreien Komponenten ebenfalls nicht geteilt sein.
- Lokale Uhren der jeweiligen Komponenten müssen (bis auf eine minimale, festgelegte zulässige Abweichung) synchronisiert sein.
- Die maximale Verzögerung einer Übertragung zwischen zwei Komponenten muss nach oben beschränkt sein.

Abhängig von der Größe der Netzwerktopologie wird wie bei FABAN eine Zustell- oder Verarbeitungszeit einer Nachricht berechnet, zu welcher die Nachricht bei allen Empfängern verarbeitet werden soll. Eine Rundspruchnachricht wird dann an alle benachbarten Komponenten gesendet. Jede Komponente, die eine Nachricht empfängt, versendet - falls die Nachricht erstmals in dieser Komponente empfangen wurde - diese ebenfalls an alle benachbarten Komponenten. Rekursiv wiederholt sich dieser Vorgang in allen Komponenten, bis das gesamte Netzwerk die Nachricht erhalten hat und das Netzwerk vollständig *geflutet* wurde. Diese Art und Weise der Verteilung der Nachrichten an alle Empfänger, ist im Allgemeinen mit höherem Nachrichtenaufwand verbunden, als dies bei FABAN durch die gezieltere Verteilung entlang redundanter Wellen der Fall ist. Durch zusätzliche Überprüfung, ob eine Nachricht nicht nur innerhalb der Zeitspanne zwischen dem Senden und der berechneten Verarbeitungszeit empfangen wurde, sondern auch ob der letzte Hop entsprechend nicht zu stark verzögert wurde, werden auch Verzögerungsfehler toleriert. Als letzter zusätzlicher Mechanismus, ermöglichen Signaturen die Toleranz von beliebigen

byzantinischen Fehlern. Ausgenommen hierbei sind natürlich solche Verfälschungen, die Signaturen brechen und folglich nicht durch Signaturprüfungen erkannt werden können. Während in FABAN für jeden Rundspruch exakt eine Signatur erstellt wird und diese nur in Checking Bridges und der Distributing Bridge durch rudimentäre Operationen modifiziert wird, co-signiert jede weiterleitende Komponente jede Nachricht.

Die breite Toleranz nahezu aller Fehlerarten wird bei Cristian u. a., sowie dem hier im Fokus stehenden Protokoll FABAN, durch Synchronität, Signaturen und strikten Systemannahmen ermöglicht. Im Hinblick auf die Vergleichbarkeit zur vorliegenden Arbeit, hinsichtlich vergleichbarer Netzwerke, Art und Menge der tolerierten Fehler, Systemannahmen und Anwendungsgebiete existieren - nach sehr gründlicher Literaturrecherche - keinerlei weitere Lösungen bzw. Publikationen. Im Gegensatz dazu wurden jedoch in den letzten Dekaden auch oft asynchrone Systeme bzw. Systeme mit deutlich abgeschwächten Anforderungen behandelt, zu welchen im Folgenden ein Überblick gegeben wird.

Chang und Maxemchuk entwickelten und publizierten 1984 eine Familie von zuverlässigen Rundspruchprotokollen (Reliable Broadcast) für nicht zuverlässige Netzwerke [CM84]. Zusätzlich garantieren diese Protokolle eine einheitliche Totalordnung der gesendeten Nachrichten bei allen fehlerfreien Empfängern. Die Grundidee der Protokolle für beliebige Netzwerke basiert dabei auf der Kombination vereinfachter Realisierungen auf Systemen mit nur einem Sender und Systemen mit nur einem Empfänger. Dabei werden alle Rundsprüche aller Sender über einen einzigen primären Empfänger, die so genannte *token site*, zu allen übrigen Empfängern weitergeleitet. Die Rolle der *token site* wird im Betrieb periodisch an andere Empfänger übergeben, um bei Auftreten eines Fehlers in der *token site* einen Totalausfall des Systems zu verhindern. Bei der Fehlerannahme wurden hierbei jedoch nur Ausfallfehler und Unterlassungsfehler betrachtet. Verzögerungen, Verfälschungen sowie allgemeines byzantinisches Verhalten werden nicht toleriert. Im Gegenzug werden an das System keine weiteren Anforderungen gestellt.

Einen spezielleren Anwendungsfall verfolgten Birman und Joseph mit dem Ziel fehlertolerante Rundsprüche in Systemen zu ermöglichen, welche in so genannte fehlertolerante Prozessgruppen separiert sind [BJ87]. Die Separierung in Gruppen wurde dabei nicht erstmals erwähnt, sondern wurde bereits in anderen Arbeiten untersucht [CZ85; Coo85]. Rundsprüche in solchen Systemen sind dabei so zu verstehen, dass nicht alle Teilnehmer des Systems, sondern alle Teilnehmer einer Prozessgruppe adressiert werden. Der wissenschaftliche Beitrag beinhaltete verschiedene Varianten von Rundsprüchen, unter anderem einen unteilbaren Rundspruch, welcher sowohl für Local-Area-Networks (LANs) als auch für Wide-Area-Networks (WANs) ausgelegt ist und einen hohen Grad an Nebenläufigkeit aufweist. Im Bezug auf die Fehlertoleranz haben die Autoren, wie die zuvor besprochene Lösung, ebenfalls nur Unterlassungsfehler bzw. Ausfallfehler untersucht. Peterson, Buchholz und Schlichting publizierten eine neue Lösung für die Interprozesskommunikation von Prozessen in verteilten Systemen, wobei neben der Zuverlässigkeit der Rundsprüche

auch eine globale, partielle Ordnung von Nachrichten erzielt wurde [PBS89]. Die Idee basiert hierbei auf der Kommunikation zwischen Prozessen über ein gemeinsames Medium wie zum Beispiel einem gemeinsamen Speicher und der Realisierung eines so genannten gemeinsamen Nachrichtenraums (*shared message space*), welcher durch das Protokoll sowohl die Zuverlässigkeit als auch die Ordnung garantiert. Betrachtet wurden dabei ebenfalls nur einfache Ausfall- und Unterlassungsfehler.

Zusätzlich zu der typischen Betrachtung von Ausfall- und Unterlassungsfehlern, haben Melliar-Smith, Moser und Agrawala Verzögerungsfehler betrachtet [MMA90]. Der Beitrag besteht aus zwei Protokollen, dem sogenannten *Trans-Protokoll* sowie dem *Total-Protokoll*. Das Trans-Protokoll ist ein effizientes, zuverlässiges Rundspruch-Protokoll, welches die nötigen Eigenschaften über positive und negative Bestätigungen (Acknowledgements) realisiert. Das besondere an dem Ansatz ist die Idee, nicht jeden Rundspruch von allen Empfängern zu bestätigen, sondern Bestätigungsnachrichten zusammen mit eigenen, künftigen Rundsprüchen zu versenden und dabei rekursiv festzustellen, ob eine implizite Bestätigung durch andere Komponenten bereits vorliegt und gegebenenfalls den Transferaufwand zu reduzieren. Dadurch entsteht eine auf der einen Seite hohe Effizienz des Protokolls ohne weitere Anforderungen an das zu Grunde liegende System zu stellen. Auf der anderen Seite erschwert diese Art der Bündelung von Informationen die Toleranz von Nachrichtenverfälschungen, welche im Rahmen der Arbeit nicht angestrebt wurde. Das Trans-Protokoll impliziert bei allen fehlerfreien Empfängern partielle Ordnungen von verteilten Nachrichten. Eine Totalordnung wird durch fehlerhafte Komponenten, und damit resultierenden, (noch) fehlenden Bestätigungen von Nachrichten, verhindert. Unterschiede der partiellen Ordnungen unterschiedlicher Empfänger sind in der Menge der jüngst versendeten Rundsprüche zu finden. Das Trans-Protokoll greift diese Eigenschaft auf und ermöglicht eine Totalordnung mit Verzögerungen. Die erwartete Anzahl an zusätzlichen Nachrichten, die notwendig sind, um eine Rundspruch-Nachricht im Kontext der Totalordnung mit hoher Wahrscheinlichkeit sehen zu dürfen, wurde mit 7.5 ermittelt. Eine Garantie dafür wurde aber nicht gegeben.

Zuletzt sei erwähnt, dass zu allen zuvor beschriebenen Szenarien aller Publikationen neben zum Teil fortführenden Arbeiten, auch viele Abwandlungen entstanden sind. Varianten wie zum Beispiel der *Optimistic Atomic Broadcast* [PS98] oder der *Probabilistic Broadcast* [FP02] sind nur zwei Beispiele von einem sehr in die breite gewachsenen Forschungsfeld. Eine detaillierte Auseinandersetzung mit all diesen abgewandelten Problemszenarien ist jedoch für diese Arbeit nicht relevant und wird deshalb nicht weiter verfolgt. Ebenfalls anzumerken ist, dass auch für Multi-Feldbussysteme Rundspruchprotokolle entwickelt wurden, welche ebenfalls Übereinstimmung erzielen. Masum und Echtle haben ein fehlertolerantes, zuverlässiges Rundspruchprotokoll entwickelt, welches Fehlerarten bis zu (wenigen) Byzantinischen Fehlern sehr effizient tolerieren kann. Dabei ist der Overhead im fehlerfreien Fall nahezu nicht vorhanden. Die Publikation

beschreibt auch, dass Möglichkeiten zur Umsetzung eine Totalordnung denkbar wären, welche gegebenenfalls durch rückwirkende Vetos angepasst werden müsste [EM96].

Insgesamt zeigt sich, dass die Arbeit von Flaviu Cristian nicht nur die erste wichtige Referenz bezüglich FABAN darstellt, sondern tatsächlich die einzige Referenz mit großen Schnittmengen ist und somit als Vorgänger dieser Arbeit gesehen werden kann. Zum einen lässt sich das damit erklären, dass Kernmechanismen wie die Berechnung der Wartezeit in den Empfängern als auch die redundante Ausbreitung der Nachrichten kaum bis gar nicht zu verbessern sind. Insbesondere hat die Ausbreitung der Nachrichten über Flutungen zwar einen geringen, jedoch vorhandenen Zusatzaufwand, welcher dadurch entsteht, dass die Flutung in Komponenten erst dann gestoppt wird, wenn doppelte Nachrichten empfangen wurden, so dass an dieser Stelle Verbesserungen denkbar sind. Zum anderen spielt Fehlertoleranz bei Automatisierungssystemen eine große Rolle, bei welchen in der Vergangenheit hauptsächlich Bussysteme verwendet wurden. Aus diesem Grund standen Bridge-Connected-Networks nicht im Fokus, so dass in den meisten Arbeiten Feldbussysteme behandelt wurden. Zunehmend ist jedoch die Automatisierung auf Bridge-Connected-Networks mit Ethernet-Varianten wie TSN von Interesse, so dass die Entwicklung von neuen Protokollen sinnvoll ist.

## 4.3. Fehlerhafter und erhöhter Nachrichtenverkehr

Ein problematisches Thema, welches keinen direkten Zusammenhang zu FABAN oder fehlertoleranten Protokollen generell hat, sondern viel mehr Kommunikation in Netzwerken im Allgemeinen betrifft, ist die Erhöhung von Datenströmen über erlaubte Begrenzungen aufgrund unterschiedlichster Ursachen, so dass eine Überlast entsteht und es zu Nachrichtenverlusten kommen kann. Die Ursachen können dabei in externen Angriffen bis hin zu fehlerhaften Komponenten innerhalb des Netzwerks liegen. Floyd und Fall haben beispielsweise Router-Mechanismen vorgestellt, welche auf der Basis von vergangenen Paketverlusten in Warteschlangen Bandbreitenregulierungen zur Vermeidung von Aufstauung ermöglichen [FF97]. Jiang und Dovrolis entwickelten einen weiteren Router-Mechanismus mit dem Namen *Guardian*, welcher als ein Modul des Routers eingehenden Verkehr auf Überlast überwacht und bei Bedarf diesen filtert bzw. drosselt [JD02]. Insbesondere im Bezug auf fehlerhafte Komponenten des Netzwerks als Ursache für zusätzlichen Verkehr sind Ansätze wie zum Beispiel von Mahoney und Chan von Interesse, bei welchen eingehende Nachrichten auf offensichtliche Fehler in den Paket-Headern überprüft und gegebenenfalls verworfen werden [MC01]. Zuletzt sei noch erwähnt, dass auch für echtzeitfähige Ethernet-Varianten wie TSN diese Probleme untersucht und geeignete Lösungen bereits publiziert wurden [Mey+20].

## 4.4. Zusammenfassung

Der Überblick über Publikationen, die Relevanz für das hier im Fokus stehende Protokoll FABAN haben, macht deutlich, dass eine Einordnung der Arbeit aufgrund rarer Publikationen bzw. oft zu weit entfernter Annahmen, sich schwierig gestaltet. Mit den drei Flutungsprotokollen von Cristian u. a. existiert nur eine einzige, über 30 Jahre alte Gruppe von Protokollen, welche eine große Schnittmenge zu FABAN haben. Andere Arbeiten betrachten meist asynchrone Systeme mit deutlich geringeren Anforderungen und möchten im Allgemeinen eine geringere Fehlertoleranz erzielen, wodurch sich eklatante Unterschiede der Anwendungsgebiete ergeben. Gründe wurden dafür am Ende des vorangehenden Abschnitts diskutiert. Damit ist FABAN neben Cristians Protokollen die erste Lösung, welche Übereinstimmung durch unteilbare Rundsprüche erzielt, flexibles und unabhängiges Sendeverhalten der Sender erlaubt und byzantinische Fehler toleriert und in dieser Kategorie die erste Lösung überhaupt, welche auf flexiblen Netzwerktopologien mit $f + 1$ redundanten Pfaden zwischen jedem Sender-/Empfängerpaar anwendbar ist und ein effizientes Nachrichtenrouting bietet. Im Vergleich zur Flutung kann somit das Netzwerk je nach Topologie deutlich entlastet werden. Zudem ist FABAN nicht auf Bridge-Connected-Networks beschränkt, sondern kann ohne weiteres auf Netzen angewendet werden, dessen Komponenten wie bei Cristians Flutungsprotokollen direkt und nicht über Bridges miteinander kommunizieren.

Kapitel 5

# FABAN
# Vertiefung und Erweiterung

Dieses Kapitel behandelt jegliche Vertiefungen und Erweiterungen für das Protokoll FABAN im Zusammenhang mit der Toleranz von Einfachfehlern. Die in Kapitel 3 vorgestellten Regeln für das Festlegen des redundanten Routings des FABAN Protokolls werden nach einer Einführung von mathematischen Grundlagen sowie Notation mittels einem mathematischen Ansatz analysiert, beschrieben und anschließend formal bewiesen. Mit Hilfe dieser Ergebnisse wird im nachfolgenden Abschnitt eine Lösung zur Verifikation und Generierung von Wellen für gegebene Topologien in der Form eines Algorithmus vorgestellt und anschließend eine Implementierung präsentiert sowie eine Auswertung vorgenommen.

Zum Schluss wird eine detaillierte Evaluation FABAN-kompatibler Topologien durchgeführt, mit dem Ziel, FABAN im Hinblick auf Praxistauglichkeit und optimalem Verhältnis zwischen der Anzahl von Bridges und Länge der Routingwege zu analysieren und zu bewerten. Hierbei werden sowohl formale als auch experimentelle Ansätze angewandt und Schlussfolgerungen daraus gezogen.

## 5.1. Formale Analyse

Die nachfolgenden Untersektionen stellen zunächst mathematische Grundlagen und Notationen vor, an welche nachfolgend die formale Beschreibung der Anforderungen des redundanten Routings bzw. der Wellen von FABAN anschließt.

### 5.1.1. Grundlagen und Notation

Das in dieser Arbeit stehende Protokoll FABAN ist entwickelt worden für sogenannte Bridge-Connected-Networks, welche bereits in Kapitel 3 erklärt wurden und für die Beispiele gezeigt worden sind. Folglich werden im weiteren Verlauf dieser Arbeit BCNs eine zentral wichtige Rolle spielen. Für die Analyse ist es daher von enormer Wichtigkeit, solche Netzwerke in formaler mathematischer Notation zu modellieren.

Netzwerke dieser Art lassen sich auf simple Weise je nach Modellanforderungen als gerichtete und ungerichtete Graphen modellieren. Dies erlaubt es, Analysen und Formulierungen von Eigenschaften des Protokolls in einer einheitlich formulierten und vereinfachten Sprache zu vollziehen. Zuerst wird die Definition von endlichen ungerichteten Graphen eingeführt und anschließend wird dieser Begriff für den Anwendungszweck der hier betrachteten Bridge-Connected-Networks adaptiert:

**Definition 5.1** (Graph/Netzwerkgraph)

1. Ein Tupel $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}, n \in \mathbb{N}$ und $E \subseteq V^2$ heißt **endlicher gerichteter Graph** mit Knotenmenge $V$ und Kantenmenge

$$E \subseteq \{(x, y) | x, y \in V\}. \tag{5.1}$$

   Jeder **ungerichtete endliche Graph** kann durch einen gerichteten endlichen Graphen $G = (V, E)$ ausgedrückt werden, indem zu jeder gerichteten Kante die entgegen gerichtete Kante der Kantenmenge hinzugefügt wird, d.h. es gilt

$$\forall (x, y) \in E : \ (y, x) \in E. \tag{5.2}$$

2. Ein Tripel $G := (\mathcal{B}, \mathcal{N}, \bar{\mathcal{L}})$ mit der Bridge-Menge $\mathcal{B}$, der Netzwerkknotenmenge $\mathcal{N}$ und der Link-Menge $\bar{\mathcal{L}}$ heißt **Netzwerkgraph**, falls $(\mathcal{B} \uplus \mathcal{N}, \bar{\mathcal{L}})$ ein endlicher gerichteter Graph ist mit

$$\bar{\mathcal{L}} \subseteq \left\{(x, y) \;\middle|\; (x, y) \in \mathcal{B}^2 \cup (\mathcal{N} \times \mathcal{B}) \cup (\mathcal{B} \times \mathcal{N})\right\}. \tag{5.3}$$

3. Ein aus einem Netzwerkgraphen $G := (\mathcal{B}, \mathcal{N}, \bar{\mathcal{L}})$ induzierter Graph $G' = (\mathcal{B}, \mathcal{L})$ heißt **vereinfachter Netzwerkgraph**, falls dieser aus der Reduktion des Netzwerkgraphen um die Sender/Empfänger der Menge $\mathcal{N}$ und betroffener Kanten entsteht, d.h. falls:
$$\mathcal{L} \subseteq \{(x, y) \in \bar{\mathcal{L}} \mid x, y \in \mathcal{B}\} \tag{5.4}$$
gilt.

Mittels der Definition des *Netzwerkgraphen* lassen sich Netzwerke, bestehend aus einer Menge von Bridges $\mathcal{B}$, einer Menge von Netzwerkknoten $\mathcal{N}$ sowie einer Menge von Verbindungen $\mathcal{L}$ als Graphen beschreiben. Da in der Beschreibung des Protokolls FABAN, wie bereits in Kapitel 3 gesehen, sowie auch in nachfolgenden Ansätzen, die Protokollbeschreibung sich auf die Kommunikation zwischen Bridges fokussiert und Netzwerkknoten als Sender bzw. Empfänger eine untergeordnete Rolle spielen, wurde in vorangehender Definition zusätzlich der *vereinfachte Netzwerkgraph* eingeführt, um die Netzwerkknoten aus der Beschreibung auszuschließen und eine formale Beschreibung und Analyse auf simpleren Mengen zu ermöglichen. Des Weiteren werden mit den Begriffen Sender und Empfänger stets Netzwerkknoten bezeichnet und nicht sendende oder empfangende Bridges bei einzelnen Zwischensendevorgängen.

Im Folgenden werden ausschließlich Netzwerkgraphen gemäß Definition 5.1 betrachtet, wobei im Allgemeinen eine Betrachtung von induzierten vereinfachten Netzwerkgraphen ausreichend ist und jegliche, im Laufe der Arbeit beschriebenen Formalisierungen simplifiziert. Eine detaillierte Modellierung von Netzwerkknoten ist hierbei im Zusammenhang der Fehlertoleranz überflüssig, da Fehler in Netzwerkknoten als spezielles Fehlverhalten von Bridges betrachtet werden können. In mathematischen Formulierungen ist es ungewöhnlich von Bridges und Verbindungen zu sprechen, so dass im Folgenden, insbesondere auf mathematischer Ebene, von Knoten und Kanten gesprochen wird, wobei im Zusammenhang mit vereinfachten Netzwerkgraphen stets Knoten synonym zu Bridges verwendet werden. Abbildung 5.1 visualisiert den Zusammenhang zwischen einem Netzwerkgraphen und dessen vereinfachter Variante.

Neben der vereinfachten Betrachtung von Netzwerkgraphen kann für die Modellierung von Netzwerken in den hier betrachteten Szenarien angenommen werden, dass weder parallele Verbindungen, noch in sich gerichtete Verbindungen, auch *Schlingen* genannt, existieren, d.h. für $G = (\mathcal{B}, \mathcal{L})$ gilt:

$$\begin{gathered} \forall l_1 = (b_1, b_2), l_2 = (b_1', b_2') \in \mathcal{L} : \; b_1 = b_1' \Rightarrow b_2 \neq b_2' \\ \forall l = (b_1, b_2) \in \mathcal{L} : \; b_1 \neq b_2 \end{gathered} \tag{5.5}$$

Teile von Graphen werden hier als Subgraphen wie folgt definiert:

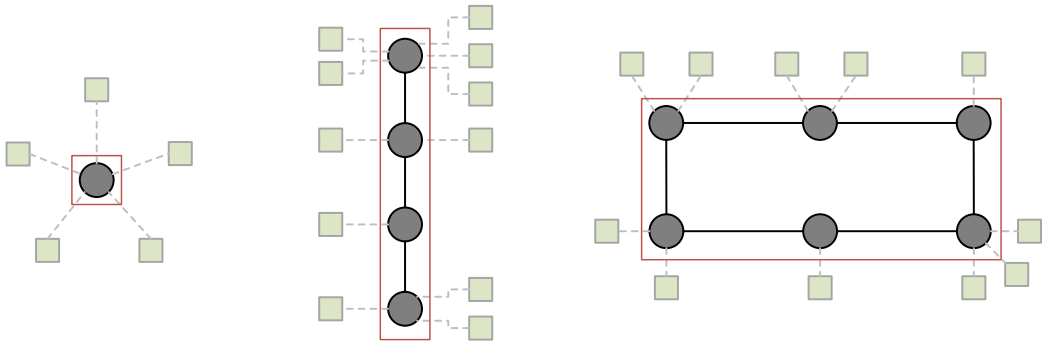


Abbildung 5.1.: Beispiele vereinfachter Netzwerkgraphen (rot eingerahmt), induziert aus vollständigen Netzwerkgraphen

**Definition 5.2** (Teilgraph/Subgraph)
Sei $G = (V, E)$ ein endlicher Graph. Ein Graph $G' = (V', E') \subset G$ heißt **Teilgraph**/**Subgraph** von $G$, falls $V' \subset V$, $E' \subset E$ und

$$\forall e = (v_1, v_2) \in E' : \; v_1, v_2 \in V' \tag{5.6}$$

gilt. Insbesondere kann ein Teilgraph mit nicht reduzierter Knotenmenge $V' = V$ durch eine Kantenteilmenge $E'$ beschrieben werden, welche den Teilgraphen $(V, E')$ induziert.

Für die nachfolgenden formalen Beschreibungen und Analysen, werden einige zentrale, aus der Graphentheorie wohl bekannte Begriffe benötigt, welche hier in einer angepassten Form definiert werden. Aufgrund der Überführbarkeit der Definition der ungerichteten Graphen in die Definition von gerichteten Graphen, sind die nachfolgenden Definitionen für beide Arten von endlichen Graphen gültig. Deshalb wird im Folgenden nicht zwischen gerichteten und ungerichteten Graphen differenziert.

Ein Kantenzug eines Graphen wird definiert durch [Die06]:

**Definition 5.3** (Kantenzug)
Sei $G = (V, E)$ ein endlicher Graph und $\tilde{E} \subseteq E$ eine Teilmenge von Kanten.

1. Ein **Kantenzug** der Länge $n \in \mathbb{N}$ zwischen zwei Knoten $a, b \in V$ auf $\tilde{E}$ ist definiert als ein $(n+1)$-Tupel von Knoten

$$\bar{p}_{\tilde{E}}(a, b) = (v_0, v_1 \ldots, v_{n-1}, v_n) \tag{5.7}$$

mit $v_0 := a$, $v_n := b$, falls gilt:

$$(v_i, v_{i+1}) \in \tilde{E} \quad \forall 0 \leq i \leq n-1. \tag{5.8}$$

Die **Länge des Kantenzugs** $n$ wird hierbei mit $|\bar{p}_{\tilde{E}}(a,b)|$ bezeichnet und entspricht der Anzahl der Kanten. Insbesondere wird für $\bar{p}_{\tilde{E}}(a,a) := (a)$ festgelegt:

$$|\bar{p}_{\tilde{E}}(a,a)| = 0. \tag{5.9}$$

2. Der **innere Kantenzug** eines Kantenzugs $\bar{p}_{\tilde{E}}(a,b)$ der Länge $n \in \mathbb{N}$ ist definiert als das $(n-1)$-Tupel der *inneren Knoten*

$$p_{\tilde{E}}(a,b) := (v_1, \ldots, v_{n-1}). \tag{5.10}$$

Die **Länge des inneren Kantenzugs** ist definiert als die Länge des Kantenzugs, aus dem der innere Kantenzug induziert wurde

$$|p_{\tilde{E}}(a,b)| = |\bar{p}_{\tilde{E}}(a,b)| = n, \tag{5.11}$$

d.h. der innere Kantenzug ist nicht kürzer als der Kantenzug aus dem dieser induziert ist.

Analog zum Kantenzug wird $p_{\tilde{E}}(a,a) := (\varnothing)$ mit $|p_{\tilde{E}}(a,a)| = 0$ festgelegt. Zusätzlich wird für benachbarte Knoten $a, b \in V$ der innere Kantenzug der direkten Verbindung zwischen beiden Knoten definiert durch $p_{\tilde{E}}(a,b) := (\varnothing)$ mit $|p_{\tilde{E}}(a,b)| = 1$. Bemerke, dass die Länge eines inneren Kantenzugs für diese beiden Fälle nicht allein aus seiner vektoriellen Darstellung bestimmt werden kann.

3. Die **Menge aller Kantenzüge** zwischen zwei Knoten $a, b \in V$ auf $\tilde{E}$ wird bezeichnet als

$$P_{\tilde{E}}(a,b) \tag{5.12}$$

und für $B \subseteq V$ wird

$$P_{\tilde{E}}(a,B) = \bigcup_{b \in B} P_{\tilde{E}}(a,b). \tag{5.13}$$

definiert. Weiterhin wird stets mit $\bar{p} \in P_{\tilde{E}}(a,b)$ ein Kantenzug von $a$ nach $b$ und mit $p \in P_{\tilde{E}}(a,b)$ der vom Kantenzug $\bar{p}$ induzierte innere Kantenzug bezeichnet.

Definition 5.3 führt alle Begriffe auf einer Teilmenge der Kantenmenge des zugrunde gelegten Graphen $G = (V, E)$ ein, um eine Flexibilität der Beschreibung von Routings in den nachfolgenden Kapitel zu ermöglichen. Beachte weiterhin, dass die Definition der Länge eines Kantenzugs $\bar{p}_{\tilde{E}}(a,b)$ bzw. eines inneren Kantenzugs $p_{\tilde{E}}(a,b)$ in (5.7) nicht durch die Anzahl der Elemente des jeweiligen Vektors sondern durch die Anzahl der

durch $\bar{p}_{\tilde{E}}(a,b)$ repräsentierten Kanten definiert ist. Hierbei stellt der innere Kantenzug lediglich eine kürzere Schreibweise des Kantenzugs dar, ohne die obligatorischen Anfangs- und Endknoten mit zu notieren.

Weiterhin werden einige weitere grundlegende Bezeichnungen für Graphen benötigt. Die Begriffe Nachbarknoten, Distanz bzw. Abstand sowie der Grad eines Knotens werden in nachfolgender Definition eingeführt.

**Definition 5.4** (Grundlegende Begriffe aus der Graphentheorie)
Sei $G = (V, E)$ ein Graph.

1. Die Menge aller **Nachbarknoten** eines Knotens $v \in V$ ist definiert als

$$N_G(v) := N_{(V,E)}(v) := N_E(v) := \{w \in V \backslash \{v\} \mid (v,w) \in E \vee (w,v) \in E\}. \quad (5.14)$$

2. Die **Distanz** bzw. der **Abstand** zwischen zwei Knoten $a, b \in V$ bezüglich einer Menge von Kanten $\tilde{E} \subseteq E$ ist definiert als die Länge des kürzesten Kantenzugs zwischen diesen beiden Knoten auf $\tilde{E}$, d.h. für $P_{\tilde{E}}(a,b) \neq \varnothing$

$$\mathrm{dist}_{\tilde{E}}(a,b) := \min\{|p| \mid p \in P_{\tilde{E}}(a,b)\} \quad (5.15)$$

und $\mathrm{dist}_{\tilde{E}}(a,b) := \infty$ für $P_{\tilde{E}}(a,b) = \varnothing$. Des Weiteren wird $\mathrm{dist}_{\tilde{E}}(a,a) := 0$ festgelegt.

Analog wird die Distanz zwischen einem Knoten $a \in V$ und einer Menge von Knoten $B \subseteq V$ als

$$\mathrm{dist}_{\tilde{E}}(a,B) := \min\{\mathrm{dist}_{\tilde{E}}(a,b) \mid b \in B\} \quad (5.16)$$

definiert.

3. Der **Grad** $\deg(v)$ eines Knotens $v \in V$ ist definiert als die Anzahl von Knoten, welche vom Knoten $v$ direkt erreicht werden können d.h.

$$\deg_G(v) := |\{w \in V \backslash \{v\} \mid (v,w) \in E\}|. \quad (5.17)$$

Insbesondere gilt für ungerichtete Graphen:

$$\deg_G(v) = |N_G(v)|. \quad (5.18)$$

Von besonderem Interesse sind im folgenden disjunkte Kantenzüge, d.h. Kantenzüge, welche keine Schnittpunkte haben. Ein Paar von Kantenzügen hat genau dann keinen Schnittpunkt, wenn dieses keine gemeinsamen inneren Knoten besitzt bzw. die durch die Kantenzüge induzierten, ungeordneten Mengen der inneren Knoten eine leere Schnittmenge haben.

**Definition 5.5**
Für zwei Kantenzüge $\bar{p} \in P_E(a,b)$, $\bar{q} \in P_E(a',b')$ mit $a,a',b,b' \in V$ eines Graphen $G = (V,E)$ wird der Schnitt definiert als:

$$\bar{p} \cap \bar{q} := p \cap q := \{v \in V \mid v \in p \wedge v \in q\}\,. \tag{5.19}$$

Die beiden Kantenzüge $\bar{p}$ und $\bar{q}$ heißen **disjunkt**, falls gilt:

$$\bar{p} \cap \bar{q} \equiv p \cap q = \varnothing \tag{5.20}$$

Beachte hierbei, dass bei dem Schnitt zweier (innerer) Kantenzüge die Endpunkte nicht betrachtet werden. Eine große Rolle wird vor allem die Größe spielen, welche die maximale Anzahl von disjunkten Pfaden ausdrückt:

**Definition 5.6** (Maximal disjunkte Pfade)
Sei $G = (V,E)$ ein endlicher Graph, $\tilde{E} \subseteq E$ eine Teilmenge von Kanten und seien $a \in V$ und $B \subseteq V$ ein Anfangsknoten und eine Menge von Endknoten. Die maximale Anzahl von disjunkten Kantenzügen von Knoten $a$ zu einem beliebigen Knoten aus $B$ ist definiert durch

$$\max P_{\tilde{E}}(a,B) := \max\{|Q| \mid Q \in \wp(P_{\tilde{E}}(a,B)) \ \wedge \ \forall p,q \in Q:\ p \cap q = \varnothing\}\,, \tag{5.21}$$

wobei $\wp$ die Potenzmenge bezeichnet.

Bemerke, dass die größte Menge von disjunkten Pfaden im Allgemeinen nicht eindeutig bestimmt ist, wohingegen die maximale Anzahl von disjunkten Pfaden stets ermittelt werden kann.

### 5.1.2. Formale Beschreibung der Anforderungen des Routings

Mit den im vorangehenden Abschnitt eingeführten Begriffen, lässt sich nun das in Kapitel 3 angeschnittene FABAN Routing formal beschreiben und ebenfalls die vorgestellten unbewiesenen Regeln für das Erstellen des FABAN Routings in Form der Wellen formalisieren und beweisen. Hierbei wird im Folgenden bewiesen, dass es sich bei dem hier formulierten Resultat sowohl um hinreichende als auch um notwendige Bedingung für die Erstellung der Wellen handelt.

Bevor das Theorem formuliert werden kann, wird der Begriff des FABAN-Graphen eingeführt:

**Definition 5.7** (FABAN-Graph)
Sei mit $G = (\mathcal{B}, \mathcal{L})$ ein vereinfachter Netzwerkgraph gegeben. Für eine gegebene Distributing Bridge $d \in \mathcal{B}$ und gegebene Checking Bridges $c_1, c_2 \in \mathcal{B}$ heißt

$$G_F := G_F(d, c_1, c_2) := (\mathcal{B}, \mathcal{L}, d, c_1, c_2) \tag{5.22}$$

**FABAN-Graph**, falls $c_1 \neq c_2$, $(d, c_1), (c_1, d) \in \mathcal{L}$ und $(d, c_2), (c_2, d) \in \mathcal{L}$ gilt.

Bevor in FABAN Fehlertoleranz thematisiert werden kann, sollte das Fehlermodell, bestehend aus Fehlerart sowie den Fehlerbereichen, exakt festgelegt werden. Die zu tolerierenden Fehlerarten sowie die wenigen Ausnahmen wurden bereits in Abschnitt 3.1.4 detailliert beschrieben. Gemäß Definition 2.3 werden die Fehlerbereiche wie folgt festgelegt:

- Ein *Perfektionskern* ist nicht vorhanden und nicht erforderlich. Das System soll autonom arbeiten und jegliche Fehler sollen entweder toleriert werden oder Nachrichten vollständig unterdrückt, so dass keine Inkonsistenz des Rundspruchs entstehen kann. In beiden Fällen würde fehlertolerantes Verhalten realisiert werden, welches die Anforderungen des *unteilbaren Rundspruchs* erfüllt.
- Für ein System mit Bridges $\mathcal{B} = \{b_1, \ldots, b_n\}$ und der Verbindungsmenge $\mathcal{L}$ wird für jede Bridge exakt ein *Einzelfehlerbereich* $E_i$, $1 \leq i \leq n$, definiert. Neben der jeweiligen Bridge werden alle zu dieser angrenzenden Netzwerkknoten sowie die Verbindungen zwischen der Bridge und dem jeweiligen Netzwerkknoten ebenfalls zum jeweiligen Einzelfehlerbereich gezählt, so dass bei fehlerhaftem Verhalten eines Senders, dieses als Fehlverhalten der DB modelliert werden kann. Zuletzt können auch Verbindungen zwischen Bridges Fehler verursachen, welche allerdings nicht eindeutig den bisher definierten $n$ Einzelfehlerbereichen zugeordnet werden können. Aus diesem Grund wird für jede Verbindung $(b_i, b_j) \in \mathcal{L}$ zwischen zwei Bridges ein zusätzlicher Einzelfehlerbereich $E_{i,j} \equiv E_{j,i}$ definiert. Für die Toleranz von einem Fehler, werden daraus nun Fehlerbereiche wie folgt festgelegt:

$$F_i := E_i \cup \left[ \bigcup_{(b_i, b_j) \in \mathcal{L}} E_{i,j} \right]. \tag{5.23}$$

  Diese Fehlerbereiche umfassen jeweils eine Bridge und alle dort eingehenden und ausgehenden Verbindungen. Die Toleranz eines Fehlers im System, stets zurückführbar auf das Fehlverhalten einer Bridge, wird mit dieser Fehlerbereichsannahme gemäß des *binären Fehlermodells* modelliert, d.h. unter diesen Annahmen wird maximal $f = 1$ Fehler toleriert, falls höchstens ein Fehlerbereich zur gleichen Zeit von einem Fehler betroffen ist (vgl. Definition 2.4).

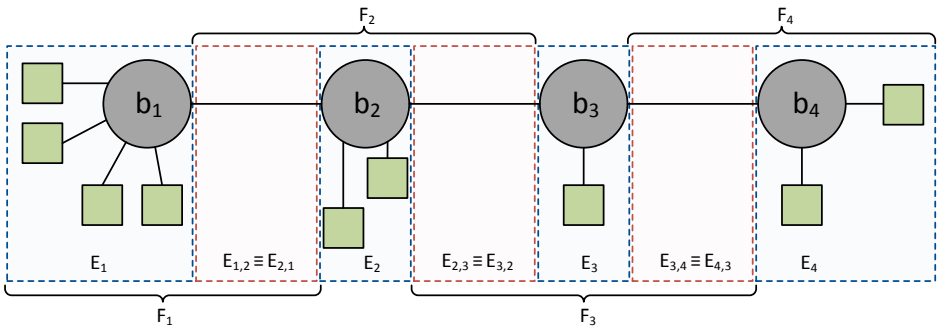



Abbildung 5.2.: FABAN Fehlerbereichsannahme für einen Fehler.

Ein FABAN-Graph in Kombination mit zwei Kantenteilmengen $W_1, W_2 \subseteq \mathcal{L}$, welche Teilgraphen ausgehend von einer festgelegten Distributing Bridge $d$ induzieren und Wellen repräsentieren, kann bei passend gewählten Wellen ein gültiges FABAN-Routing für alle Sender, die mit $d$ verbunden sind, für den Fall $f = 1$ darstellen. Zusammengefasst ergibt sich folgende Definition:

**Definition 5.8**
Für einen gegebenen FABAN-Graphen $G_F = (\mathcal{B}, \mathcal{L}, d, c_1, c_2)$ und gegebene Wellen $W_1, W_2 \subseteq \mathcal{L}$ wird das Tripel

$$(G_F, W_1, W_2) \tag{5.24}$$

als *1-fehlertolerant* bezeichnet, genau dann wenn $W_1$ und $W_2$ ein gültiges Wellenpaar bezüglich dem FABAN-Protokoll für die Toleranz von einem Fehler sind.

In vorangehender Definition ist der Begriff *Gültigkeit* bezogen auf die in Kapitel 3 formulierten Wellenregeln. Diese werden in folgendem Theorem formal zusammengefasst und sowohl als hinreichende als auch als notwendige Bedingungen bewiesen.

**Theorem 5.9**
Sei $G_F = (\mathcal{B}, \mathcal{L}, d, c_1, c_2)$ ein FABAN-Graph und seien

$$W_1 := \{w_1^1, w_1^2, \ldots, w_1^n\} \subseteq \mathcal{L} \tag{5.25}$$

und

$$W_2 := \{w_2^1, w_2^2, \ldots, w_2^n\} \subseteq \mathcal{L} \tag{5.26}$$

zwei Kantenteilmengen mit $n = |\mathcal{B}|$. Das Tripel $(G_F, W_1, W_2)$ ist *1-fehlertolerant*, genau dann wenn die beiden folgenden Eigenschaften erfüllt sind:

1. *(Konnektivität von Distributing und Checking Bridges)*

$$\begin{aligned}(d, c_1) &\in W_1 \\ (d, c_2) &\in W_2\end{aligned} \tag{5.27}$$

2. *(Redundante Erreichbarkeit)*
   Für jede Bridge $b \in \mathcal{B}$ existieren eindeutige Kantenzüge $\bar{p}_1 \in P_{W_1}(c_1, b)$ und $\bar{p}_2 \in P_{W_2}(c_2, b)$, so dass sowohl die Distributing Bridge als auch die Checking Bridges nicht in den inneren Kantenzügen $p_1$ und $p_2$ enthalten sind und diese keine gemeinsamen Elemente haben, also disjunkt sind, d.h.

$$d, c_1, c_2 \notin (p_1 \cup p_2) \tag{5.28}$$

$$p_1 \cap p_2 = \varnothing. \tag{5.29}$$

Bevor das Theorem bewiesen wird, ist noch anzumerken, dass ausgehend von der Distributing Bridge, $n-1$ Verbindungen gebraucht werden, um alle anderen $n-1$ Bridges zu erreichen. Da FABAN zusätzlich fordert, dass die Distributing Bridge eine Nachricht zu Empfängern weiterleiten darf, wenn diese zunächst von einer Checking Bridge überprüft wurde, wird eine weitere Verbindung benötigt. Somit sind und somit folglich insgesamt $n$ Verbindungen pro Welle erforderlich.

*Beweis.*
Der Beweis der Äquivalenz erfolgt auf klassischem Wege, indem separat die Richtungen „⇒“ und „⇐“ gezeigt werden.

1. *Zu zeigen: FABAN ist fehlertolerant* ⇒ (5.27), (5.28), (5.29)*:*
   Seien $W_1, W_2 \subseteq \mathcal{L}$ gültige Wellen bezüglich dem Protokoll FABAN, welche die Wellenregeln aus Kapitel 3 erfüllen und damit gemäß Definition 5.8 $(G_F, W_1, W_2)$ *1-Fehlertoleranz* garantiert. Betrachtet man gemäß Protokollspezifikation die Übertragung von der Distributing Bridge zur Checking Bridge als Teil der beiden Wellen, dann impliziert dies genau Bedingung (5.27).

   Ebenfalls folgt unmittelbar aus der Wellenregel 2 in Kapitel 3, dass die Verteilung über eine fremde Checking Bridge oder die Distributing Bridge nicht erlaubt ist und somit $d, c_1, c_2 \notin (p_1 \cup p_2)$ gilt, was (5.28) impliziert.

   Die Existenz eindeutiger Kantenzüge von der jeweiligen Checking Bridge zu jeder beliebigen Bridge $b \in \mathcal{B}$ folgt aus der Voraussetzung $|W_1| = |W_2| = |\mathcal{B}|$ und dass jede Bridge erreicht werden muss. O.B.d.A sei nun angenommen, dass für eine Bridge $b \in \mathcal{B}$ zwei Kantenzüge in $W_1$ existieren. Dann gilt

$$\exists (v_1, v_2), (w_1, w_2) \in W_1 : \; v_2 = w_2 = b \tag{5.30}$$

und somit dass nur noch höchstens $|\mathcal{B}| - 2$ andere Bridges erreicht werden können. Damit würde eine Bridges nicht erreicht werden, was ein Widerspruch zu den Voraussetzungen bedeutet.

Betrachte zunächst eine beliebige FB $b \in \mathcal{B}\backslash\{d, c_1, c_2\}$. Seien $\bar{p}_1 \in P_{W_1}(c_1, b)$ und $\bar{p}_2 \in P_{W_2}(c_2, b)$ die zwei eindeutigen Kantenzüge von der jeweiligen Checking Bridge zu Bridge $b$. Angenommen $p_1 \cap p_2 \neq \varnothing$, so dass gilt:

$$\exists b' \in \mathcal{B} : \; b' \in p_1 \cap p_2. \tag{5.31}$$

Ist nun $b'$ fehlerhaft, dann würde $b$ nicht mehr erreichbar sein und damit die Annahme der *1-Fehlertoleranz* verletzen, so dass per Widerspruch Eigenschaft 5.29 folgt.

Sei nun ohne Beschränkung der Allgemeinheit $b = c_1$. Dann ist $\bar{p}_1 = (c_1)$ bzw. $p_1 = (\varnothing)$ und folglich $p_1 \cap p_2 = \varnothing$ für einen beliebigen, aber regelkonformen Pfad $p_2 \in P_{W_2}(c_2, b)$, so dass trivialerweise Bedingung (5.29) erfüllt ist. Für $b = d$ müssen zwei Fälle unterschieden werden. Sollte $(c_1, d) \in W_1$ erfüllt sein, ist mit dem Pfad $\bar{p}_1(c_1, d) = (c_1, d)$ und beliebigem regelkonformen Kantenzug $\bar{p}_2(c_2, d) \in P_{W_2}(c_2, d)$

$$p_1(c_1, d) \cap p_2(c_2, d) = \varnothing \tag{5.32}$$

erfüllt und somit auch Bedingung (5.29). In dem Fall $(c_1, d) \notin W_1$ muss folglich der Pfad zu $d$ über andere Forwarding Bridges führen. Die Argumentation für diesen Fall ist identisch zu der bereits diskutierten Argumentation in dem Fall $b \in \mathcal{B}\backslash\{d, c_1, c_2\}$, also den Pfaden zu beliebigen Forwarding Bridges.

2. *Zu zeigen:* (5.27), (5.28), (5.29) $\Rightarrow$ *FABAN ist fehlertolerant:*
Zu zeigen ist, dass ein beliebiger Einzelfehler mit den gegebenen Voraussetzungen toleriert wird, d.h. dass garantiert entweder alle Empfänger eine Nachricht erhalten oder eine Nachricht unterdrückt wird. Dies wird im Folgenden durch Fallunterscheidung des Fehlerortes $b_f \in \mathcal{B}$ gezeigt.

   Sei $b_f = d$. Sollte die Nachricht eine oder beide Checking Bridges trotz fehlerhafter Distributing Bridge erreichen, wird die Nachricht bei nicht im Toleranzbereich liegenden Verzögerungen von diesen unterdrückt. Andernfalls würde eine Verteilung der Nachricht entlang mindestens einer Welle initiiert werden, so dass mit (5.28) und (5.29) garantiert ist, dass alle Empfänger die Nachricht rechtzeitig erreichen. Sollte die Nachricht auf andere Weise von der Distributing Bridge verfälscht worden sein, würde dies durch die Signatur bzw. die Signaturmodifikation in den Empfängern erkannt werden, so dass damit die *1-Fehlertoleranz* folgt.

   Sei o.B.d.A. $b_f = c_1$. Dann sind $d$ und $c_2$ aufgrund der *1-Fehlerannahme* fehlerfrei, so dass mit (5.27), (5.28) und (5.29) die Erreichbarkeit bei allen fehlerfreien Bridges gewährleistet ist und die Behauptung damit folgt.

Sei schließlich $b_f \in \mathcal{B} \backslash \{d, c_1, c_2\}$. Dann ist nach (5.29) $b_f \notin p_1$ oder $b_f \notin p_2$, so dass mindestens eine Welle die Nachricht fehlerfrei weiterleitet, wodurch ebenfalls die Behauptung zusammen mit (5.27) und (5.28) impliziert wird.

□

## 5.2. Lösung zur automatisierten Generierung von Wellen

Mit Hilfe des Hauptresultats aus dem vorherigen Abschnitt, Theorem 5.9, wird in diesem Unterkapitel ein Algorithmus zur Generierung von Wellen für den Fall $f = 1$ entwickelt. Der Algorithmus hat die Eigenschaft, dass einerseits erfolgreiche Ausführungen des Algorithmus valide Wellenpaare gemäß Definition 5.8 erzeugen, andererseits nicht erfolgreiche Versuche der Generierung von Wellen für gegebene Topologien implizieren, dass keine gültigen Wellenpaare existieren. Diese Eigenschaft wird im nachfolgenden Abschnitt formal nachgewiesen, bevor dieses Unterkapitel mit experimentellen Untersuchungen hinsichtlich unterschiedlicher interessanter Aspekte, darunter Laufzeiteigenschaften und Effektivität, abgeschlossen wird.

### 5.2.1. Algorithmus zur Verifikation und Generierung von Wellen

Wellen für gegebene Topologien zu generieren führt unmittelbar zu der Frage, wie komplex diese Problemstellung tatsächlich ist. Klassische Graphenexplorationsalgorithmen sind aufgrund des komplexen Zusammenhangs beider Wellen grundlegend problematisch [Cor+09]. Beispielsweise kann eine Suche nach gültigen Wellen mit Hilfe von Breitendurchläufen erfolgreich sein. Hierbei würde zunächst mit Hilfe eines Breitendurchlaufs eine mögliche Welle garantiert gefunden werden, falls die Grundvoraussetzung eines nicht partitionierten Netzwerks erfüllt ist, woraufhin anschließend aus den noch erlaubten Verbindungen, also solchen die nicht gegen die Eigenschaften aus Theorem 5.9 verstoßen, versucht werden würde, wieder mittels eines Breitendurchlaufs auf einer durch die erste Welle eingeschränkten Verbindungsmenge, eine zweite Welle zu finden. Aufgrund dessen, dass beim Erstellen der ersten Welle nicht darauf geachtet wird, nur solche Verbindungen zu verwenden, die die Möglichkeit, eine zweite Welle zu finden, wahren, ist der Erfolg dieser Methode (bei einmaliger, gezielter Exploration) weder garantiert, noch vielversprechend hoch. Beispiele an einfachen Netzwerkstrukturen untermauern diese Behauptung.

Erfolgreich gefundene Wellenpaare haben allerdings die Eigenschaft, dass die Wellenlängen, also die maximale Entfernung zwischen der Distributing Bridge und einer beliebigen anderen Bridge, tendenziell eher minimal ist, das heißt die erste Welle hat tatsächlich

minimale Wellenlänge, während die zweite Welle minimale Wellenlänge auf der durch die erste Welle eingeschränkten Verbindungsmenge hat.

Analoges gilt für den Erfolg bei einem Ansatz mit Hilfe des Tiefendurchlaufs, bei welchem ebenfalls sequentiell Wellen mit Hilfe des Tiefendurchlaufs gesucht werden. Im Vergleich zu der guten Eigenschaft der minimalen Entfernung des Breitendurchlaufs, ist beim Tiefendurchlauf natürlicherweise das genaue Gegenteil der Fall, d.h. mindestens eine der beiden Wellen hätte maximale Länge von der Distributing Bridge zu einer anderen Bridge.

Systematische Wiederholung der genannten Explorationsalgorithmen in Form von Brute-Force Ansätzen garantieren sicheren Erfolg der Wellengenerierung auf Kosten sehr hohen Suchaufwandes, falls der Graph diese Voraussetzung erfüllt. Für größere Graphen sind diese allerdings nicht in akzeptabler Zeit umsetzbar, da der Aufwand für steigende Graphengrößen exponentiell wächst, d.h. für einen Graphen $G = (\mathcal{B}, \mathcal{L})$ mit $n$ Bridges und $k$ Kanten beträgt der Aufwand maximal

$$\prod_{b \in \mathcal{B}} \deg_G(b) \left(\deg_G(b) - 1\right), \tag{5.33}$$

da sowohl für die erste als auch die zweite Welle für jede Bridge exakt eine eingehende Kante ausgewählt werden muss, wobei der Grad jedes Knotens bei der Wahl der Kanten für die zweite Welle durch die Einschränkung der ersten Welle um 1 verringert ist.

Aufgrund der unbefriedigenden Eigenschaften dieser trivialer Ansätze, wurde ein systematisches Vorgehen entwickelt. Der daraus entstandene Algorithmus besteht aus zwei Teilen, Algorithmus 5.1 und 5.3, welcher sequentiell die beiden Wellen generiert, falls der erste Teil, also Algorithmus 5.1, erfolgreich durchläuft. Im Folgenden werden beide Teile des Algorithmus im Detail erklärt.

### Generierung der ersten Welle

Als Eingabe des Algorithmus wird ein FABAN-Graph mit festgelegter Distributing Bridge und Checking Bridges erwartet. Zu Beginn wird überprüft ob grundlegende, triviale Voraussetzung für die Generierung von Wellen erfüllt sind. Zum einen muss zwingend die Distributing Bridge mit den beiden Checking Bridge verbunden sein, zum anderen darf der Graph $(\mathcal{B}, \mathcal{L})$ nicht partitioniert sein, d.h. es muss eine (in)direkte Verbindung zwischen beliebigen Knoten existieren. Ist dies nicht erfüllt, wird der Algorithmus ohne Erfolg abgebrochen.

Nach dieser trivialen Validierung wird die erste Welle $W_1$ schrittweise konstruiert. Gestartet wird mit zwei Verbindungen, von der DB zur CB $c_1$ und umgekehrt, sowie zwei Mengen von gerichteten Kanten $\overline{W}_1$ und $\overline{W}_2$, welche Obermengen der zu konstruierenden

Wellen darstellen. Letztere repräsentieren im Fortschritt des Algorithmus veränderliche Mengen von möglichen gerichteten Kanten, welche Kandidaten für die Konstruktion beider Wellen enthalten. Sie werden wie folgt initialisiert:

$$\begin{aligned}\overline{W}_1 &:= \Big\{(x,y) \in \mathcal{L} \;\Big|\; x \notin \{d, c_2\} \wedge y \neq d\Big\} \\ \overline{W}_2 &:= \Big\{(x,y) \in \mathcal{L} \;\Big|\; x \notin \{d, c_1\} \wedge y \neq d\Big\}.\end{aligned} \tag{5.34}$$

Anschließend startet die schrittweise Konstruktion der ersten Welle, wobei in jedem Schritt exakt eine Verbindung $l \in \mathcal{L}$ der Welle $W_1$ hinzugefügt und den Obermengen $\overline{W}_1$ und $\overline{W}_2$ entnommen wird. Solch eine Verbindung $l$ muss dabei folgende Bedingungen erfüllen:

1. Sie muss aus der Menge der noch erlaubten Kanten kommen, d.h. $l \in \overline{W}_1$, und zu einem Knoten führen, der noch nicht über Kanten aus $W_1$ erreicht wird
2. Nach der Hinzunahme von $l$ muss $W_1$ einen zusammenhängende Pfadmenge bilden, das heißt für alle $b \in \{b' \in \mathcal{B} \mid \exists (x,y) \in (W_1 \cup \{l\}) : \; b' = x \;\vee\; b' = y\}$ gilt:

$$P_{W_1 \cup \{l\}}(c_1, b) \neq \varnothing. \tag{5.35}$$

3. Die Möglichkeit eine zweite Welle mit Kanten aus $\overline{W_2} \backslash \{l\}$ und den Restriktionen der in Konstruktion befindlichen Teilwelle $W_1 \cup \{l\}$, zu konstruieren, muss gewährleistet sein. Die Realisierung dieser Überprüfung wird in den nachfolgenden Abschnitten im Detail beschrieben.

Im Allgemeinen darf eine Verbindung, welche die zuvor aufgelisteten Bedingungen erfüllt, vollkommen zufällig gewählt werden, was nicht-deterministische Ergebnisse zur Folge hätte. Alternativ kann auf deterministische Weise eine Verbindung aus einer Menge von geeigneten, geordneten Verbindungen gewählt werden. Dies würde zu einer deterministischen Umsetzung des Algorithmus führen, jedoch keinerlei Geschwindigkeits- bzw. Effizienzvorteile bringen. Ein potentieller Vorteil einer deterministischen Lösung ist, dass man die maximale Länge der Wellen beeinflussen kann und mittels geeigneter Ordnungen der Menge der Verbindungen Wellen konstruiert, deren maximale Länge tendenziell nahe dem möglichen Minimum liegt. Die Vor- und Nachteile beider Lösungen wurden anhand von Implementierungen untersucht und werden im nachfolgenden Abschnitt thematisiert.

Die Menge der geeigneten Verbindungen, aus welchen sukzessive eine neue Verbindung der Welle hinzugefügt wird, ist eine Teilmenge aller Verbindungen, die sowohl zu noch nicht erreichten Nachbarknoten führen, die durch die bisher konstruierte Welle erreicht werden, als auch in $\overline{W}_1$ liegen. Dies lässt sich ausdrücken durch: Wähle eine Verbindung aus

$$N_{\mathcal{L}}(W_1) \cap \overline{W}_1, \tag{5.36}$$

wobei $N_{\mathcal{L}}(A)$ für $A \subset \mathcal{L}$ analog zur Definition des Nachbarknotens in Definition 5.4 die Menge aller wegführenden Nachbarkanten beschreibt, d.h.

$$N_{\mathcal{L}}(A) := \left\{(x,y) \in \mathcal{L} \;\middle|\; x \in dest(A) \;\wedge\; y \notin dest(A)\right\}, \tag{5.37}$$

mit der Menge aller Knoten, zu denen Verbindungen aus $A$ führen:

$$dest(A) := \{b \in \mathcal{B} \mid \exists x \in \mathcal{B} :\; (x,b) \in A\}\,. \tag{5.38}$$

Verbindungen in (5.37) erfüllen die dritte Bedingung für die sukzessive Wahl von Verbindungen nicht garantiert. Um die dritte Bedingung formal zu beschreiben, wird die Relation $A \models\!\!\!\leftrightarrows B$ für $A, B \subseteq \mathcal{L}$ auf $G_F$ für zu $A$ und $B$ zugehörige Checking Bridges $c_A$ und $c_B$ wie folgt eingeführt

$$A \models\!\!\!\leftrightarrows B :\Leftrightarrow \forall b \in dest(A)\; \exists p_1 \in P_A(c_A, b)\; \exists p_2 \in P_B(c_B, b) :\; p_1 \cap p_2 = \varnothing, \tag{5.39}$$

das heißt, alle über Kanten aus $A$ erreichbare Bridges sind auch über Kanten aus $B$ erreichbar und es gibt mindestens ein Paar von Pfaden in $A$ bzw. $B$, welches disjunkt ist. Damit wird sichergestellt, dass die erste Welle in $A$ eine zweite Welle in $B$ nicht unmöglich macht. Diese Relation ist nicht transitiv und folglich auch keine Äquivalenzrelation. Abbildung 5.3 zeigt jeweils ein Beispiel für eine erfüllte und nicht erfüllte Relation.

Damit kann nun die dritte Bedingung wie folgt formuliert werden

$$[W_1 \cup \{l\}] \models\!\!\!\leftrightarrows [\overline{W}_2 \backslash \{l\}]. \tag{5.40}$$

Umgesetzt in die Pseudocode-Formulierung des Algorithmus ist die Relation $\models\!\!\!\leftrightarrows$ durch die Routine `subWaveValid`, welche in Algorithmus 5.2 beschrieben wird.

Die sukzessive Konstruktion der ersten Welle wird so lange durchgeführt, bis entweder $|W_1| = |\mathcal{B}|$ gilt und somit erfolgreich die Konstruktion beendet wurde oder aber $\overline{W}_1 = \varnothing$

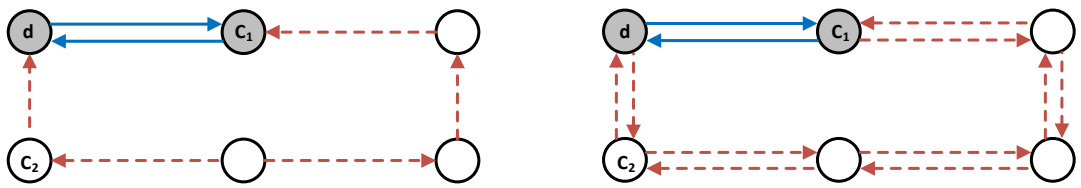



Abbildung 5.3.: Beispiele für die im Wellengenerierungsalgorithmus verwendete Relation zwischen (Teil)Wellen. Das linke Beispiel zeigt eine gültige Relation $A \models\!\!\!\leftrightarrows B$, das rechte Beispiel eine nicht gültige Relation. Die Kantenmenge $A$ ist hierbei blau/durchgezogen dargestellt und die Kantenmenge $B$ ist rot/gestrichelt dargestellt. Die grau markierten Bridges entsprechen $dest(A)$.

ist. In letzterem Fall bedeutet dies, dass für den gegebenen FABAN-Graph kein gültiges Wellenpaar konstruiert werden kann. Diese Behauptung wird im nachfolgendem Kapitel verifiziert. Das hier beschriebene Verfahren ist in Algorithmus 5.1 in Pseudocode formuliert.

**Algorithmus 5.1** Generierung der ersten Welle

```
1  Input: G_F = (B, L, d, c_1, c_2)

2  if(dist_L(d, c_1) ≠ 1 or dist_L(d, c_2) ≠ 1 or c_1 = c_2 or !isConnected(G_F))
3      then return ∅
4  end if

5  W_1 = {(d, c_1), (c_1, d)}
6  W̄_1 := W̄_2 := ∅

7  for l = (b_1, b_2) in L do
8      if b_1 ≠ d and b_2 ≠ d and b_1 ≠ c_2 then
9          W̄_1.add(l)
10     end if
11     if b_1 ≠ d or b_2 ≠ d or b_1 ≠ c_1 then
12         W̄_2.add(l)
13     end if
14 end for

15 while |W_1| < |B| and N_L(W_1) ∩ W̄_1 ≠ ∅ do
16     l = selectEdge(N_L(W_1) ∩ W̄_1)
17     if subWaveValid(W_1, W̄_2, c_1, c_2, l) then
18         W_1.add(l)
19         W̄_2.remove(l)
20     end if
21     W̄_1.remove(l)
22 end while

23 if |W_1| = |B| then
24     return W_1
25 else
26     return ∅
27 end if
```

Im Pseudocode von Algorithmus 5.1 bezeichnet `isConnected(`$G_F$`)` eine Funktion zur Überprüfung, ob der übergebene Graph zusammenhängend ist und `selectEdge(L)` bezeichnet eine beliebige Funktion zur deterministischen oder indeterministischen Auswahl einer Kante. Die Funktion `subWaveValid(`$W_1, \overline{W}_2, c_1, c_2, l$`)` zur Überprüfung der Relation $\not\leftrightarrows$ ist in nachfolgendem Pseudocode (Algorithmus 5.3) beschrieben. Dabei wird mit $W.getInnerPath(a, b)$ die Funktion zur Bestimmung eines inneren Pfades zwischen den Bridges $a$ und $b$ auf Kanten aus $W$ bezeichnet. Nach Konstruktion der Kantenmengen $W$ in Algorithmus 5.1 wird sichergestellt, dass stets exakt ein innerer Pfad zwischen Knoten $c_1$ und $c_2$, welche als Parameter übergeben werden, existiert. Schließlich bezeichnet die Funktion $\overline{C}$`.removeEdgesWithVertices(`$p_W$`)` eine Operation zur Entfernung aller Kanten aus $\overline{C}$, welche zu Knoten auf dem Pfad $p_W$ führen bzw. von solchen ausgehen.

Zuletzt sei erwähnt, dass in Abhängigkeit der Größe des zu Grunde liegenden Graphen, in jedem sukzessiven Iterationsschritt der Konstruktion der Welle, die Auswahl einer neuen gültigen Verbindung im Allgemeinen mehrere Versuche benötigen kann. Wird eine Verbindung gefunden, so ist die Wahl endgültig, so dass kein Backtracking stattfindet.

**Algorithmus 5.2** Algorithmus `subWaveValid` zur Validierung von Teilwellen

```
1 Input: W, C̄ ⊂ ℒ, c1, c2 ∈ ℬ, l = (b1, b2) ∈ ℒ

2 C̄.remove(l)
3 W.add(l)
4 pW := W.getInnerPath(c1, b2)
5 C̄.removeEdgesWithVertices(pW)

6 return C̄.getInnerPath(c2, b2) ≠ ∅
```

### Generierung der zweiten Welle

Nachdem die erste Welle erfolgreich konstruiert wurde, kann auf eine ähnliche Art und Weise die zweite Welle konstruiert werden. Der zweite Teil des Algorithmus benötigt als Eingabe zusätzlich zu dem FABAN-Graphen die zuvor konstruierte erste Welle. Während im ersten Teil des Algorithmus der erfolgreiche Ausgang der Routine nicht gewährleistet ist, sondern von dem gegebenen Netzwerkgraphen abhängt, ist die Garantie der erfolgreichen Konstruktion der zweiten Welle gegeben. Dies folgt aus der Tatsache, dass vor jeder sukzessiven Erweiterung der in Konstruktion befindlichen Welle überprüft wird, ob eine zweite Welle noch konstruiert werden kann. Würde diese Überprüfung fehlschlagen, würde der gesamte Algorithmus abbrechen und der zweite Teil nicht mehr ausgeführt.

Diese recht rechenkomplexe Überprüfung muss bei der Konstruktion der zweiten Welle nicht mehr stattfinden, sondern es muss lediglich überprüft werden, ob die hinzukommenden Verbindungen im Bezug auf die erste Welle korrekt ausgewählt werden.

Nachdem die selben trivialen und grundlegenden Voraussetzungen für die Generierung von Wellen überprüft worden sind, wird die zweite Welle $W_2$ schrittweise konstruiert.

**Algorithmus 5.3** Generierung der zweiten Welle

```
1  Input: G_F = (B, L, d, c_1, c_2), W_1

2  if(dist_L(d, c_1) ≠ 1 or dist_L(d, c_2) ≠ 1 or c_1 = c_2 or !isConnected(G_F))
3      then return ∅
4  end if

5  W_2 = {(d, c_2), (c_2, d)}
6  W̄_2 := ∅

7  for l = (b_1, b_2) in L do
8      if b_1 ≠ d and b_2 ≠ d and b_1 ≠ c_1 and l ∉ W_1 then
9          W̄_2.add(l)
10     end if
11 end for

12 while |W_2| < |B| and N_L(W_2) ∩ W̄_2 ≠ ∅ do
13     l = selectEdge(N_L(W_2) ∩ W̄_2)
14     if subWaveValid(W_2, W_1, c_2, c_1, l) then
15         W_2 = W_2.add(l)
16     end if
17     W̄_2.remove(l)
18 end while

19 return W_2
```

Initialisiert wird diese dabei analog zum ersten Teil des Verfahrens mit den beiden Verbindungen von der DB $d$ zur CB $c_2$. Als Hilfsobermenge für mögliche hinzukommende Verbindungen, wird $\overline{W}_2$ wie folgt definiert:

$$\overline{W}_2 := \left\{(x,y) \in \mathcal{L} \,\middle|\, x \notin \{d, c_1\} \wedge y \neq d\right\} \backslash W_1 \tag{5.41}$$

In der nachfolgenden schrittweisen Konstruktion der zweiten Welle wird wieder eine Verbindung $l \in \mathcal{L}$ der Welle $W_2$ hinzugefügt und der Obermenge $\overline{W}_2$ entnommen. Hierbei muss $l$ die folgenden Bedingungen erfüllen:

1. Sie muss aus der Menge der noch erlaubten Kanten kommen, d.h. $l \in \overline{W}_2$
2. Für alle $b \in \{b' \in \mathcal{B} \mid \exists (x, y) \in (W_2 \cup \{l\}) : \ b' = x \ \vee \ b' = y\}$ gilt:

$$P_{W_2 \cup \{l\}}(c_2, b) \neq \varnothing. \tag{5.42}$$

3. $\mathcal{W}_2 \cup \{l\}$ ist im Bezug auf $W_1$ eine gültige Teilmenge einer zweiten Welle.

Die ersten beiden Bedingungen werden von allen Verbindungen erfüllt in

$$N_{\mathcal{L}}(W_2) \cap \overline{W}_2. \tag{5.43}$$

Die letzte Bedingung kann wieder durch die im ersten Teil eingeführte Relation ausgedrückt werden:

$$[W_2 \cup \{l\}] \not\leftrightarrows [W_1]. \tag{5.44}$$

Da $W_1$ aber bereits vollständig erzeugt wurde, reicht es in diesem Fall zu überprüfen, ob die nach Hinzunahme von $l$ neu erreichte Bridge disjunkt zum Pfad in $W_1$ ist. Ebenfalls ist die Existenz eines passenden $l$ durch die Überprüfungen während der Generierung von $W_1$ garantiert. Die sukzessive Konstruktion wird wiederholt bis $|W_2| = |\mathcal{B}|$ gilt. Ein Fehlschlagen der Konstruktion der zweiten Welle ist, wie bereits zu Beginn erwähnt, ausgeschlossen.

#### Erzeugen eines Routings für das FABAN Protokoll

Der zuvor vorgestellte Algorithmus konstruiert ein Wellenpaar, falls möglich, für einen gegebenen Netzwerkgraphen $(\mathcal{B}, \mathcal{L})$ und festgelegte DB und CB. Da im Allgemeinen jede Komponente des Systems dazu in der Lage sein sollte, FABAN-Rundsprüche zu versenden, muss folglich der Algorithmus für jede Bridge des Systems als DB angewendet werden. Die Wahl der CBs kann auf algorithmische Weise nicht ohne Mehraufwand gelöst werden, so dass aus allen möglichen Kombinationen von CBs aus der Nachbarschaft der jeweiligen DB gewählt werden muss, bis geeignete gefunden werden. Ebenfalls ist es möglich, unterschiedliche Checking Bridge Paare zu finden, von denen einige „bessere“ Wellenpaare haben könnten als andere. Die Frage nach einem Vergleich von Wellenpaaren wird in Kapitel 5.3 thematisiert.

### 5.2.2. Formale Verifikation

In diesem Abschnitt wird formal bewiesen, dass einerseits konstruierte Wellenpaare für gegebene FABAN-Graphen gültig sind gemäß Definition 5.8 und andererseits, dass keine gültigen Wellenpaare existieren, falls der Algorithmus fehlschlägt.

Für die Formulierung und den Beweis der Behauptung, wird die Klasse aller gültigen Wellenpaaren eines FABAN-Graphen $G_F$ wie folgt definiert:

$$\mathcal{F}(G_F) := \left\{ (W_1, W_2) \;\middle|\; W_1, W_2 \subseteq \mathcal{L} :\ |W_1| = |W_2| = |\mathcal{B}| \ \wedge\ W_1 \models\!\!\!\leftrightarrows W_2 \right\} \tag{5.45}$$

Hierbei ist anzumerken, dass für ein gültiges Wellenpaar $(W_1, W_2)$ sowohl $W_1 \models\!\!\!\leftrightarrows W_2$ als auch $W_2 \models\!\!\!\leftrightarrows W_1$ erfüllt ist, obwohl es sich um keine Äquivalenzrelation handelt. Insbesondere für gültige und vollständige Wellenpaare ist $\models\!\!\!\leftrightarrows$ der passende Indikator.

Mit dieser Notation kann folgende Aussage formuliert werden:

**Theorem 5.10**
Sei $G_F = (\mathcal{B}, \mathcal{L}, d, c_1, c_2)$ ein FABAN-Graph mit festgelegter Distributing Bridge und festgelegten Checking Bridges und seien weiterhin $W_1$ und $W_2$ die Ergebnisse des Algorithmus aus Abschnitt 5.2.1. Dann gilt:

(i) $\mathcal{F}(G_F) = \varnothing \Rightarrow W_1 = W_2 = \varnothing$

(ii) $W_1 \neq \varnothing \wedge W_2 \neq \varnothing \Leftrightarrow (W_1, W_2) \in \mathcal{F}(G_F)$

*Beweis.*
Hat der gegebene FABAN-Graph nach Annahme keine gültigen Wellenpaare, d.h. ist $\mathcal{F}(G_F) = \varnothing$, dann terminiert der Algorithmus ohne Erfolg, da in jedem Erweiterungsschritt überprüft wird, ob jede Bridge immer noch über zwei disjunkte Pfade erreicht werden kann, was de erste Eigenschaft (i) impliziert. Mit dem gleichen Argument folgt die „⇒"-Richtung der zweiten Eigenschaft (ii). Die Rückrichtung „⇐" ist offensichtlich nach der Definition in (5.45). □

Deutlich weniger offensichtlich ist die Frage, ob ein gültiges Wellenpaar für einen gegebenen FABAN-Graphen existieren kann, falls der Algorithmus fehlschlägt. Bevor bewiesen wird, dass dies nicht möglich ist, wird noch einige zusätzliche Notation benötigt.

Die beiden Algorithmen für die Generierung beider Wellen benötigen exakt $|\mathcal{B}|$ Schritte bis diese erfolgreich terminieren. Hierbei ist unter einem Schritt das Hinzufügen von exakt einer neuen Verbindung der in Konstruktion befindlichen Welle gemeint. Für $0 \leq t \leq |\mathcal{B}|$ wird die in Konstruktion befindliche Welle als Teilwelle im Generierungsschritt $t$ bezeichnet und mit $W_{1/2}^{(t)}$ notiert. Die zugehörigen Kantenobermengen der beiden Teilwellen werden analog mit $\overline{W}_{1/2}^{(t)}$ notiert. Nach Definition gilt

$$\begin{aligned} \left| W_1^{(t)} \right| &= t, \\ \left| \overline{W}_2^{(t)} \right| &\geq |\mathcal{B}| - 2, \end{aligned} \tag{5.46}$$

wobei letztere Ungleichung damit begründbar ist, dass die zweite Welle zu jedem Zeitpunkt der Generierung der ersten Welle, stets jede Bridge mit Ausnahme der obligatorischen Distributing Bridge $d$ und Checking Bridge $c_2$ erreichen muss. Analog zur Definition der Nachbarkanten in (5.37) wird die Menge der Nachbarbridges einer Kantenmenge $A \subset \mathcal{L}$ definiert durch

$$N_{\mathcal{B}}(A) := \left\{ b \in \mathcal{B} \;\middle|\; b \notin dest(A) \;\wedge\; \exists b' \in dest(A) :\; \mathrm{dist}_{\mathcal{L}}(b, b') = 1 \right\}. \tag{5.47}$$

Bevor die zentrale Behauptung dieses Abschnitts bewiesen werden kann, wird das folgende Lemma benötigt:

**Lemma 5.11**
Sei $G_F = (\mathcal{B}, \mathcal{L}, d, c_1, c_2)$ ein FABAN-Graph mit festgelegter Distributing Bridge und festgelegten Checking Bridges mit der Eigenschaft, dass die Wellen $W_1$ und $W_2$ existieren, d.h.:

$$\mathcal{F}(G_F) \neq \varnothing. \tag{5.48}$$

Weiterhin seien $W_1^{(t)}$ die Teilwelle der ersten Welle und $\overline{W}_2^{(t)}$ die Kantenobermenge der zweiten Welle nach $t \in \{2, \ldots, |\mathcal{B}| - 1\}$ Schritten des ersten Teils des Algorithmus, so dass gilt:

$$W_1^{(t)} \not\leftrightarrows \overline{W}_2^{(t)}. \tag{5.49}$$

Dann existiert eine Kante $l \in N_{\mathcal{L}}\left(W_1^{(t)}\right)$, so dass folgende Bedingung erfüllt ist:

$$W_1^{(t+1)} := \left(W_1^{(t)} \cup \{l\}\right) \not\leftrightarrows \left(\overline{W}_2^{(t)} \backslash \{l\}\right) =: \overline{W}_2^{(t+1)}. \tag{5.50}$$

*Beweis.*
Sei $t \in \{2, \ldots, |\mathcal{B}| - 1\}$ mit $W_1^{(t)} \not\leftrightarrows \overline{W}_2^{(t)}$. Der Beweis erfolgt durch Fallunterscheidung in folgende drei Fälle (dabei wird stets Bezug auf Abbildung 5.4 genommen). Dabei müssen sich die Fälle 1 und 2 in einem Schritt nicht zwangsläufig ausschließen.

*Fall 1:* Es existieren Bridges $x \in N_{\mathcal{B}}\left(W_1^{(t)}\right)$ und $y_1, y_2 \in dest\left(W_1^{(t)}\right)$, so dass

$$\mathrm{dist}_{\mathcal{L}}(x, y_1) = \mathrm{dist}_{\mathcal{L}}(x, y_2) = 1 \tag{5.51}$$

gilt. In diesem Fall ist sowohl $l_1 = (y_1, x) \in \overline{W}_2^{(t)}$ als auch $l_2 = (y_2, x) \in \overline{W}_2^{(t)}$ und wegen $W_1^{(t)} \not\leftrightarrows \overline{W}_2^{(t)}$ existieren Pfade $p(c_2, y_1), p(c_2, y_2)$ sowie eindeutige Pfade $p(c_1, y_1), p(c_1, y_2)$, so dass

$$p(c_1, y_1) \cap p(c_2, y_1) = p(c_1, y_2) \cap p(c_2, y_2) = \varnothing \tag{5.52}$$

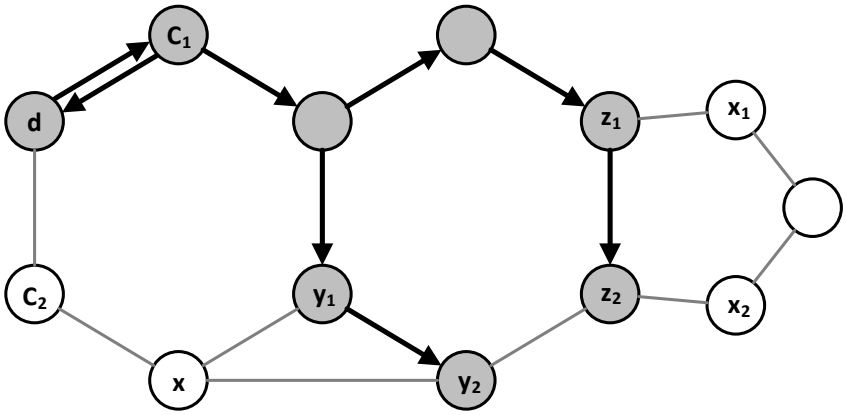



Abbildung 5.4.: Darstellung des Stands der Wellengenerierung nach $t = 8$ Schritten. Graue Bridges sind bereits durch die Teilwelle $W_1^{(8)}$ erreicht.

gilt. Außerdem ist mindestens entweder $y_1 \notin p(c_1, y_2)$ (Szenario A) oder $y_2 \notin p(c_1, y_1)$ (Szenario B), so dass mindestens entweder $l_2$ (Szenario A) oder $l_1$ (Szenario B) der Teilwelle $(W_1^{(t)})$ hinzugefügt werden kann, wobei die zweite Welle $x$ mindestens über $l_1$ (Szenario A) bzw. $l_2$ (Szenario B) erreichen kann, so dass für das entsprechend ausgewählte $i \in \{1, 2\}$ folgt:

$$W_1^{(t+1)} := \left(W_1^{(t)} \cup \{l_i\}\right) \not\leftrightarrows \left(\overline{W}_2^{(t)} \backslash \{l_i\}\right) := \overline{W}_2^{(t+1)}. \tag{5.53}$$

*Fall 2:* Es existieren Knoten $x_1, x_n \in N_{\mathcal{B}}(W_1^{(t)})$ und $z_1, z_n \in dest(W_1^{(t)})$ für ein $n \geq 2$, so dass

$$\mathrm{dist}_{\mathcal{L}}(x_1, z_1) = \mathrm{dist}_{\mathcal{L}}(x_n, z_n) = 1 \tag{5.54}$$

und

$$\mathrm{dist}_{\mathcal{L}}(x_1, x_n) \geq 1. \tag{5.55}$$

In diesem Fall existiert wegen $\mathcal{F}(G_F) \neq \varnothing$ ein Pfad

$$\bar{p}_{\mathcal{L}}(z_1, z_n) = (z_1, x_1, \ldots, x_n, z_n) \tag{5.56}$$

mit entsprechendem Rückpfad

$$\bar{p}_{\mathcal{L}}(z_n, z_1) = (z_n, x_n, \ldots, x_1, z_1), \tag{5.57}$$

so dass $(x_j, x_{j+1}), (x_{j+1}, x_j) \in \overline{W}_2^{(t)}$ für alle $j \in \{1, \ldots, n\}$ gilt. Dann kann, mit gleichem Argument wie in Fall 1, entweder $l_1 = (z_1, x_1)$ oder $l_n = (z_n, x_n)$ zur Teilwelle $W_1^{(t)}$ hinzugefügt werden, wobei die zweite Welle $x_n$ über (5.57) oder (5.56), d.h. den entsprechend

anderen Pfad, erreichen kann. Damit folgt für $i \in \{1, n\}$:

$$W_1^{(t+1)} := \left(W_1^{(t)} \cup \{l_i\}\right) \not\leftrightarrows \left(\overline{W}_2^{(t)} \backslash \{l_i\}\right) := \overline{W}_2^{(t+1)}. \tag{5.58}$$

*Fall 3:* Weder Fall 1, noch Fall 2 sind erfüllt, während $N_{\mathcal{L}}(W_1^{(t)}) \neq \varnothing$ gilt (in Abbildung 5.4 nicht dargestellt). Da mindestens ein Knoten noch nicht von der Welle erreicht wird, müssen Knoten $x \in N_{\mathcal{L}}(W_1^{(t)})$ und $y \in dest(W_1^{(t)})$ existieren, so dass $dist_{\mathcal{L}}(x, y) = 1$ und für alle $v \in dest(W_1^{(t)})$ und $p \in P_{\mathcal{L}}(x, v)$

$$y \in p \tag{5.59}$$

gilt, d.h. $x$ könnte mit bereits von der Welle erreichten Knoten nur über die Verbindung zu $y$ kommunizieren. Somit wäre dies ein Widerspruch zur Voraussetzung, dass mindestens ein Wellenpaar existiert in $G_F$. Folglich kann dieser Fall nicht eintreten.

Für alle anderen Fälle, z.B. in denen das Netzwerk partitioniert ist, ist es offensichtlich unmöglich Wellenpaare zu finden.

□

Mit Lemma 5.11 kann nun folgendes Hauptresultat formuliert und bewiesen werden.

**Theorem 5.12**
Sei $G_F = (\mathcal{B}, \mathcal{L}, d, c_1, c_2)$ ein FABAN-Graph mit festgelegter Distributing Bridge und festgelegten Checking Bridges und seien weiterhin $W_1$ und $W_2$ die Resultate des Algorithmus zur Generierung von Wellen aus dem vorherigen Abschnitt. Dann gilt

$$W_1 = W_2 = \varnothing \Rightarrow \mathcal{F}(G_F) = \varnothing \tag{5.60}$$

*Beweis.*
Der Beweis erfolgt durch den Beweis der negierten Implikation

$$\mathcal{F}(G_F) \neq \varnothing \Rightarrow W_1 \neq W_2 \neq \varnothing. \tag{5.61}$$

Um dies zu zeigen, muss nachgewiesen werden, dass der Algorithmus zwei endliche Folgen von Teilmengen

$$\begin{aligned} \varnothing = W_1^{(0)} \subset W_1^{(1)} \subset W_1^{(2)} \subset \cdots \subset W_1^{(|\mathcal{B}|)} = W_1, \\ \mathcal{L} = W_2^{(0)} \supset W_2^{(1)} \supset W_2^{(2)} \supset \cdots \supset W_2^{(|\mathcal{B}|)} = W_2 \end{aligned} \tag{5.62}$$

schrittweise generiert, mit

$$W_1^{(t)} \not\leftrightarrows W_2^{(t)} \quad \forall t \in \{0, \ldots, |\mathcal{B}|\}, \tag{5.63}$$

wobei im Anschluss die zweite Welle garantiert konstruiert werden kann, so dass

$$(W_1, W_2) \in \mathcal{F}(G_F) \tag{5.64}$$

gilt. Diese Behauptung allerdings folgt durch einfache Induktion über $t \in \{0, \ldots, |\mathcal{B}|\}$, wobei der Induktionsstart offensichtlich erfüllt ist für $t \in \{0, 1, 2\}$ und der Induktionsschritt aus Lemma 5.11 hervorgeht, welches besagt, dass in jedem Generierungsschritt der ersten Welle eine neue Kante zu dieser sicher hinzugefügt werden kann. □

### 5.2.3. Anwendung in realen Systemen

Der vorgestellte Algorithmus erfordert, dass vor einem realen Einsatz von FABAN die Netzwerktopologie bekannt ist, der Algorithmus vorab für jede Bridge als Distributing Bridge das Routing ermittelt und anschließend die relevanten Routing-Informationen für jede Bridge lokal gespeichert wird. Letzteres bedeutet, dass jede Bridge statische relevante Weiterleitungsinformationen jedes potentiellen Senders benötigt. Zudem impliziert diese Notwendigkeit, dass Systeme ebenfalls statisch sein müssen und nicht dynamisch erweitert werden dürfen. Sollte zur Laufzeit eines Systems eine Änderung stattfinden, wenn zum Beispiel eine Komponente entfernt oder hinzugefügt wird, dann muss das System rekonfiguriert werden.

### 5.2.4. Analytische und experimentelle Untersuchungen

Im Folgenden wird der in den vorherigen Abschnitten formulierte Algorithmus im Bezug auf die Effizienz und Laufzeit untersucht. Anschließend wird eine qualitative Bewertung der Ergebnisse des Algorithmus auf experimentelle Art und Weise vorgenommen und interpretiert.

#### Komplexitäts- und Laufzeitanalyse

Um eine analytische Abschätzung des Wachstums der Laufzeit im Bezug auf die Größe der für den Algorithmus zugrunde liegenden Netzwerktopologien, zu erhalten, wird der Algorithmus in einzelne Schritte unterteilt und separat die Komplexität ermittelt, woraus sich schlussendlich die Komplexitätsklasse des Algorithmus ergibt. Dabei wird von geeigneten Datenstrukturen ausgegangen, welche keinen nennenswerten Zusatzaufwand verursachen. Beginnend mit Algorithmus 5.1 sind die Zeilen 1 bis 6 sowie die Zeilen 23 bis 27 unabhängig von dem gegebenen Netzwerkgraphen, so dass sich eine Komplexität konstanter Ordnung $\mathcal{O}(1)$ ergibt. Die for-Schleife in den Zeilen 7 bis 14 benötigt exakt $|\mathcal{L}|$ Iterationen, so dass sich eine Komplexität der Ordnung $\mathcal{O}(|\mathcal{L}|)$ ergibt. Die while-Schleife

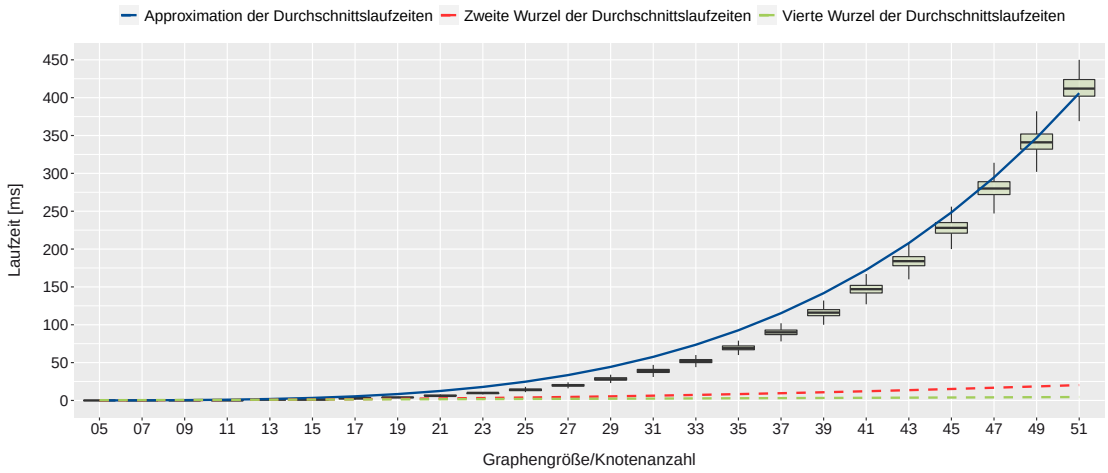



Abbildung 5.5.: Experimentell ermittelte Algorithmus Laufzeiten für unterschiedlich große Graphen. Hierbei wurden zufällige Topologien bestimmter Größe gleichverteilt erzeugt, auf welche im Anschluss der Algorithmus angewandt wurde. Pro festgelegte Knotenanzahl wurden jeweils $2^{15}$ Netzwerkgraphen erstellt, welche FABAN-kompatibel waren. Dabei gilt ein Graph als FABAN-kompatibel, wenn ein FABAN-Routing für jede Distributing Bridge gefunden werden konnte. Der in blau dargestellte Graph ist eine Approximation der gesammelten Durchschnittslaufzeiten durch ein Monom der ungefähren Ordnung 4. Die Ordnung lässt sich grob an der grün gestrichelten Kurve abschätzen, welche die vierte Wurzel der durchschnittlichen Laufzeiten visualisiert und nahezu linear verläuft.

in den Zeilen 15 bis 22 benötigt maximal $|\mathcal{L}|$ Iterationen und führt in jeder Iteration exakt einmal die Überprüfung der Relation $\not\leftrightarrows$ durch. Die Komplexität der Überprüfung dieser Relation in Algorithmus 5.2 ist abhängig von der Ermittlung der Pfade sowohl von $c_1$ zu einem beliebigen Knoten $b \in \mathcal{B}$ als auch von $c_2$ zu $b$ in den Zeilen 4 und 6, welche zusammengenommen eine Komplexität der Ordnung $|\mathcal{L}| + |\mathcal{B}|$ haben, wenn man für die Suche den Breitensuchlauf verwendet. Mit der selben Argumentation ergibt sich die gleiche Komplexitätsanalyse für den zweiten Algorithmus, wodurch sich zusammen eine Komplexität der Ordnung

$$\mathcal{O}\Big(|\mathcal{L}| \cdot \max\{|\mathcal{L}|, |\mathcal{B}|\}\Big) \tag{5.65}$$

ergibt. Mit der Annahme, dass $|\mathcal{L}| > |\mathcal{B}|$ gilt, was eine zwingende Voraussetzung für FABAN-kompatible Topologien ist, lässt sich die Komplexität zu

$$\mathcal{O}\Big(|\mathcal{L}|^2\Big) \tag{5.66}$$

vereinfachen.

Zuletzt sei noch zu untersuchen, wie groß die Auswirkung mehrfacher Ausführungen für das Konstruieren eines vollständigen Routings, d.h. die Ausführung des Algorithmus

für jede Bridge, ist. Im schlimmsten Fall muss der Algorithmus für eine DB $d \in \mathcal{B}$ alle Kombinationen von möglichen CBs durchlaufen. Diese Anzahl $N_{exec}$ lässt sich nach oben abschätzen durch

$$N_{exec} \leq \binom{|N_{\mathcal{B}}(d)|}{2} \tag{5.67}$$

Abbildung 5.5 zeigt eine Auswertung von Tests zur Verifikation der Laufzeitabschätzung des Algorithmus in Form von Boxplots. Hierbei lässt sich erkennen, dass die Laufzeit deutlich zunimmt mit steigender Anzahl von Knoten eines Netzwerkgraphen. Mit dem Ansatz, die Laufzeiten durch ein Monom unbekannten Grades zu approximieren, ergibt dies für die zwei grob gewählten Wertepaare $f(33) \approx 50$ und $f(80) \approx 2374$ (nicht in der Abbildung dargestellt) die approximierte Funktion

$$f(x) = 0,00008127189 \cdot x^{3,922852}. \tag{5.68}$$

Dieser Anstieg der Laufzeit ist vollkommen konform zur analytischen Schätzung in (5.66), auch wenn dies auf den ersten Blick widersprüchlich erscheint. Bei der Erzeugung zufälliger Graphen kann es bei $n \geq 3$ Knoten maximal $\frac{n}{2} \cdot (n-1)$ Kanten geben, wobei für jede Kante mit einer Wahrscheinlichkeit von 0.5 entschieden wird, ob diese Kante zum Graphen gehört oder nicht. Die Verteilung entspricht hierbei der Binomialverteilung, welche durch die Normalverteilung mit Erwartungswert $\mu = 0.5n \cdot (n-1)$ approximiert werden kann. Folglich werden im Durchschnitt Graphen mit Kantenanzahl $\mu$ erzeugt, so dass die Komplexität in Abhängigkeit der Knotenanzahl ca. im Bereich $\mathcal{O}(|\mathcal{B}|^4)$ liegt. Tatsächlich lässt sich hieran erkennen, dass in der realen Anwendung des Algorithmus die schlimmst-anzunehmende Komplexität deutlich nicht erreicht wird. Checking Bridges werden im Allgemeinen nicht beim ersten Versuch gefunden, so dass der Algorithmus pro FABAN-Graph mehrfach angewendet werden muss, was folglich im schlimmsten anzunehmenden Fall eine Erhöhung der Komplexität in (5.66) zur Folge hätte. Dies ist aber offensichtlich in der Praxis kaum der Fall.

### Qualitative Bewertung algorithmisch konstruierter Wellenpaare

Im Allgemeinen existieren für gegebene Netzwerkgraphen bzw. FABAN-Graphen keine eindeutigen Wellenpaare. Folglich stellt sich für den Fall $|\mathcal{F}(G_F)| \geq 2$ die Frage nach einem qualitativen Vergleich unterschiedlicher Wellenpaare und den qualitativen Vergleichsmerkmalen. Aufgrund des Protokolldesigns im Hinblick auf vollständige Toleranz durch Redundanz für maximal einen temporären Fehler zur gleichen Zeit bzw. maximal einem permanenten Fehler, sind im Hinblick der Fehlertoleranz alle gültigen Wellenpaare gleich gut.

Unterschiede zwischen Wellen gibt es einerseits in der maximalen Anzahl an Hops, die im entsprechenden Wellenpaar durchlaufen werden müssen und andererseits in sich

unterscheidenden Übertragungsdauern unterschiedlicher Verbindungen. Beides hat direkten Einfluss auf die maximale Übertragungsdauer einer Nachricht und schließlich nach Protokolldesign auch auf die Berechnung der *Zustellzeit* $t_d$. Dies kann bei der Wahl des nächsten Knotens bei der Konstruktion der ersten Welle beachtet werden, um möglichst „günstige“ Verbindungen auszuwählen.

Wie „gut“ Verbindungen im Bezug auf die Länge oder andere relevante Faktoren sind, lässt sich durch positive Kantengewichte ausdrücken, was sich am einfachsten durch eine Adjazenzmatrix

$$M_K := M_K(\mathcal{B}, \mathcal{L}) := (m_{i,j})_{1 \le i,j \le n} \tag{5.69}$$

für

$$\mathcal{B} = \{b_1, \ldots, b_n\} \tag{5.70}$$

realisieren lässt, wobei $m_{i,j} = 0$ ist für $(b_i, b_j) \notin \mathcal{L}$. Kantengewichte werden in der Literatur oft im Zusammenhang mit Kosten gesehen, so dass von nun an diese ebenfalls als Kosten und die Matrix $M_K$ als Kostenmatrix bezeichnet wird. Für das Senden einer Nachricht entlang einer Welle von der DB zu einer beliebigen anderen Bridge akkumulieren sich die Kosten der einzelnen Kanten, so dass bei der Konstruktion der Welle ein sich sukzessiv verändernder Kostenvektor für die DB $d$

$$K_d^{(t)} = \left(k_1^{(t)}, \ldots, k_n^{(t)}\right)^T \tag{5.71}$$

ergibt. Nach der Initiierung der Welle durch die Verbindungen von der DB zur CB sowie von der CB zur DB, kann der Kostenvektor für $t = 2$ initialisiert werden, wobei mit Ausnahme der DB und der CB alle anderen Elemente mit 0 initialisiert werden, d.h. für o.B.d.A. $d = b_k$ und $c_1 = b_l$ für $k, l \in \{1, \ldots, n\}$

$$K_d^{(2)} = (0, \ldots, 0, \underbrace{m_{k,l}}_{\text{CB}}, 0 \ldots, 0, \underbrace{m_{l,k} + m_{k,l}}_{\text{DB}}, 0, \ldots, 0). \tag{5.72}$$

Führt man nun Schritt $t+1$ der Konstruktion der ersten Welle $W_1^{(t)}$ mit dem Kostenvektor $K_d^{(t)}$ durch, dann kann die nächste Verbindung bezüglich der Kostenminimierung algorithmisch ausgewählt werden. Mit dem kanonischen Einheitsvektor $e_i$ für $1 \le i \le n$, werden zunächst bereits erreichte Bridges aus der Berechnung entfernt

$$M := M_K - M_K \cdot \left[ \sum_{b_i \in dest(W_1^{(t)})} e_i \cdot e_i^T \right], \tag{5.73}$$

anschließend die Kosten zu benachbarten, noch nicht erreichten Bridges in der Form einer Matrix (zu lesen wie eine Adjazenzmatrix) berechnet

$$A'_M = \left[ \sum_{b_i \in dest(W_1^{(t)})} e_i \cdot e_i^T \right] \cdot M =: (a'_{i,j})_{1 \le i,j \le n} \tag{5.74}$$

und schlussendlich die Kosten aus den vorherigen Schritten angerechnet, d.h.

$$A_M := (a_{i,j})_{1 \le i,j \le n} \quad \text{mit} \quad a_{i,j} := \begin{cases} 0 & , a_{i,j} = 0 \\ a'_{i,j} + k_j & , a_{i,j} > 0 \end{cases}. \tag{5.75}$$

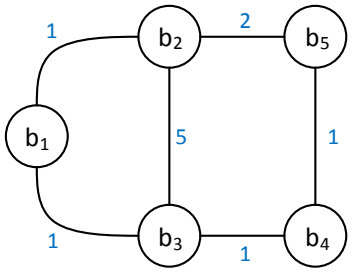
Positive Einträge der Matrix $A_M = (a_{i,j})_{1 \le i,j \le n}$ beschreiben die Kosten der Nachrichtenübertragung von der Distributing Bridge zu der noch nicht durch die Welle erreichten Bridge $b_j$. Aufsteigend sortiert, ergeben die positiven Einträge eine Kandidatenliste für die Auswahl der nächsten Kante bei gleichzeitiger Minimierung der Kosten. Kann eine Verbindung der Kandidatenliste nicht zur Welle hinzugefügt werden, beispielsweise weil die Wellenrelation $\nRightarrow$ nicht erfüllt ist, wird die nächste Verbindung der sortierten Kandidatenliste ausgewählt, überprüft und schließlich der Kostenvektor um die Kosten der neuen Verbindung aktualisiert. Dieses Verfahren wird so lange wiederholt, bis die Welle vollständig generiert ist, so dass schlussendlich für das resultierende Wellenpaar $(W_1, W_2)$ eines FABAN-Graphen $G_F = (\mathcal{B}, \mathcal{L}, d, c_1, c_d)$ der Kostenvektor

$$K_d = (k_{d,i})_{1 \le i \le |\mathcal{B}|}, \tag{5.76}$$

entsteht, wobei $k_{d,b}$ jeweils das Maximum der Kosten der Verbindungen über die Pfade $p_{W_1}(d,b)$ sowie $p_{W_2}(d,b)$ bezeichnet. Die tatsächlichen Kosten eines FABAN-Rundspruchs entsprechen den höchsten Kosten des Kostenvektors, da stets alle Empfänger vom Rundspruch erreicht werden müssen. Dieser Wert kann als Maß verwendet werden, um unterschiedliche Wellenpaare miteinander zu vergleichen.

*Hinweis:* Die Kriterien, nach welchen ein Vergleich stattfinden soll, kann von der maximalen Transferdauer, über die mittleren Transferdauern zu allen Empfängern oder gar komplexeren Metriken variieren.

**Beispiel:** Betrachte als Beispiel einen Netzwerkgraphen mit 5 Bridges gemäß nachfolgender Abbildung, sowie folgender Kostenmatrix:




$$M_K = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 5 & 0 & 2 \\ 1 & 5 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 2 & 0 & 1 & 0 \end{pmatrix}$$

Für die Distributing Bridge $b_1$, sowie die erste Checking Bridge $b_2$ für die Generierung der ersten Welle, ist der sich schrittweise aufbauende Kostenvektor in nachfolgender Tabelle

5.1 beschrieben. Die erste Spalte gibt hierbei den Kostenvektor im Anfangszustand (zwei Verbindung zwischen DB und CB) an, während die weiteren Spalten die Ergebnisse der nachfolgenden Rechenschritte zusammenfassen:

| | $t = 2$ | $t = 3$ | $t = 4$ | $t = 5$ |
|---|---|---|---|---|
| $b_1$ | **2** | 2 | 2 | 2 |
| $b_2$ | **1** | 1 | 1 | 1 |
| $b_3$ | 0 | 0 | 0 | **5** |
| $b_4$ | 0 | 0 | **4** | 4 |
| $b_5$ | 0 | **3** | 3 | 3 |

Tabelle 5.1.: Kostenvektor im Generationsschritt $t$. Fett gedruckte Zahlen heben die hinzugekommen Kosten im jeweiligen Schritt hervor.

**Schritt 1:**

$$M = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 0 & 2 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \quad A'_M = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \Rightarrow A_M = \begin{pmatrix} 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 6 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{5.77}$$

Damit sind die Kandidaten für die Wahl der nächsten Verbindung für die Welle die Menge der Verbindungen $\{(b_1, b_3), (b_2, b_5), (b_2, b_3)\}$, wobei die erste Verbindung die Überprüfung nicht besteht, weil sie über eine Verbindung der Distributing Bridge zur Checking Bridge der zweiten Welle führt, und anschließend die zweite gewählt wird.

**Schritt 2:**

$$M = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \quad A'_M = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \Rightarrow A_M = \begin{pmatrix} 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \end{pmatrix} \tag{5.78}$$

Auch im zweiten Schritten besteht die Menge der Kandidaten aus 3 Verbindungen, $\{(b_1, b_3), (b_5, b_4), (b_2, b_3)\}$, wobei wieder die Wahl auf die zweite Verbindung fällt.

**Schritt 3:**

$$M = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad A'_M = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \Rightarrow A_M = \begin{pmatrix} 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{5.79}$$

Im dritten und letzten Schritt stehen wieder 3 Kandidaten von Verbindungen zur Verfügung, $\{(b_1, b_3), (b_4, b_3), (b_2, b_3)\}$, von welchen wieder die zweite Verbindung ausgewählt wird, wodurch die Welle komplettiert und das Beispiel abgeschlossen wird. Die Konstruktion der Welle mit Hinblick auf Kostenminimierung lässt sich für die zweite Welle vollkommen analog durchführen.

Mit dieser Vorgehensweise ausgewählte Verbindungen garantieren die Konstruktion einer ersten Welle mit minimalen Kosten. Für die Konstruktion der zweiten Welle kann dieses Verfahren genauso gut angewendet werden, um auch hier eine zweite Welle mit minimalen Kosten zu konstruieren. Dies bedeutet aber nicht, dass dadurch ein optimales Wellenpaar generiert wird. Es kann nämlich durchaus auftreten, dass in der Konstruktion der ersten Welle die Bedingungen für die zweite Konstruktion so stark eingeschränkt werden, dass das Optimum nicht mehr gefunden werden kann. Dass dies in der Regel nicht der Fall ist, zeigen Auswertungen in Abbildung 5.6 von experimentellen Untersuchungen der Generierung von Wellenpaaren für zufällige Netzwerkgraphen, wobei diese analog zu den zufällig erzeugten Graphen in Abbildung 5.5 erstellt wurden. Hierbei wurden die Kosten aller Verbindungen auf konstant 1 festgelegt, um die Untersuchung auf das Wesentliche zu beschränken, d.h. es ist ausreichend die maximalen Hops der Wellenpaare zu betrachten und zu vergleichen.

Die Experimente in Abbildung 5.6 bestätigen zunächst die Vermutung, dass der Algorithmus mit der systematischen Kantenauswahl für die Nachfolgekanten nicht immer optimale Wellenpaare, d.h. Wellenpaare mit der kürzest möglichen maximalen Entfernung zwischen DB und allen anderen Bridges des zugrunde liegenden Netzwerkgraphen, erzeugen kann. Die schlechtesten Ergebnisse, die mit dem systematischen Ansatz entstanden sind, liegen hierbei aber dennoch deutlich unter den schlechtesten Ergebnissen, die mit dem zufälligen Ansatz erzeugt wurden (zum Teil weniger als halb so groß). Deutlich auffälliger ist der Unterschied beider Ansätze beim Vergleich der durchschnittlichen Wellenpaarlängen. Dabei lässt sich deutlich erkennen, dass beim systematischen Ansatz die durchschnittlichen Abweichungen zum optimalen Ergebnis einerseits unter dem Wert 1 liegen, andererseits der Anstieg der Abweichung beim zufälligen Ansatz deutlich steiler ist. Obwohl der Ansatz mit den zufälligen Nachfolgekanten bei der großen Menge von

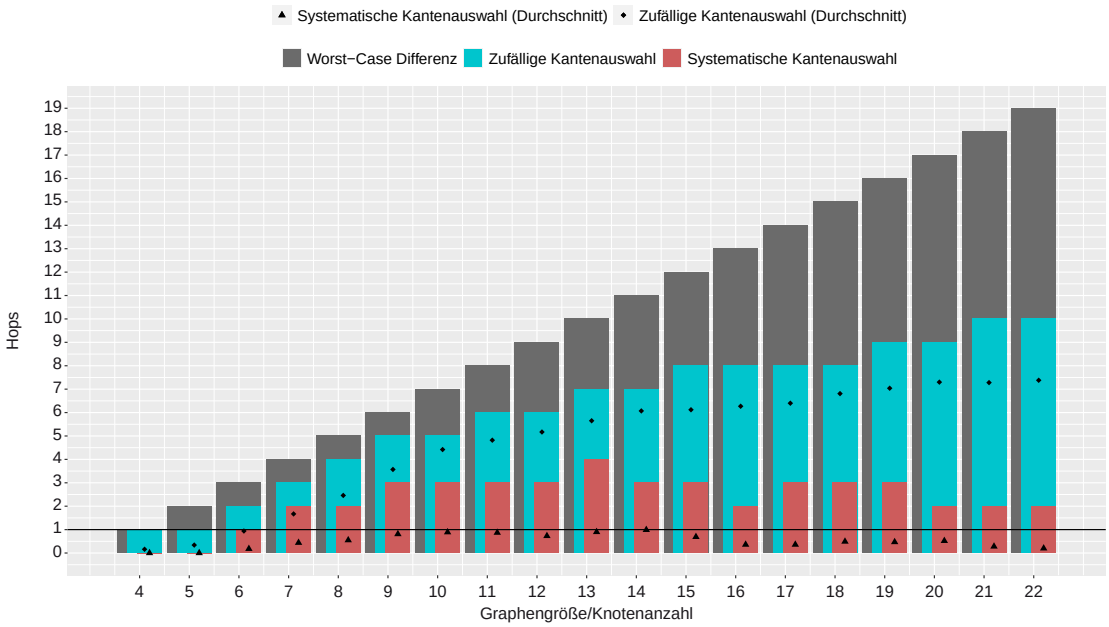



Abbildung 5.6.: Vergleich zwischen systematischer und zufälliger Auswahl von Nachfolger-Kanten während dem sukzessiven Aufbau der Wellen. Die Balken stellen jeweils die Differenz der Länge von Wellenpaaren zum kürzest möglichen Wellenpaar (approximiert durch Randomisierung) dar. Hierbei wird die maximal theoretisch mögliche Differenz (■), die maximale Differenz bei zufälliger Kantenauswahl (■) sowie die maximale Differenz der systematischen Kantenauswahl (■) dargestellt. Für jede Graphengröße/Kantenanzahl wurden 100 zufällige Graphen generiert, für welche jeweils einmal der Algorithmus mit der systematischen Kantenauswahl und 100.000 mal der Algorithmus mit zufälliger Kantenauswahl für Nachfolgekanten für jede Bridge als DB angewendet wurde. Die maximalen Werte wurden jeweils aus den vollständigen Ergebnissen der Experimente ermittelt. Zusätzlich ist für jede Graphengröße die durchschnittliche Differenz zum jeweiligen approximierten, optimalen Wellenpaar des systematischen Ansatzes (⧫) sowie des zufälligen Ansatzes (▲) markiert. Das heißt der jeweilige Wert zeigt, wie viel schlechter die algorithmisch ermittelten Wellenpaare im Vergleich zum (wahrscheinlich) besten Wellenpaar sind.

Wiederholungen oftmals genauso gut oder sogar besser war als beim systematischen Ansatz, spricht insbesondere das vorherige Argument deutlich für den Einsatz des systematischen Ansatzes, welcher mit deutlich höherer Wahrscheinlichkeit bessere Ergebnisse erzielt. Zudem sind in der Realität und insbesondere bei der bereits großen Menge eingesetzter Redundanzmechanismen, einige wenige zusätzliche Hops bei den Übertragungen vertretbar.

## 5.3. Evaluation von geeigneten Netzwerkgraphen

Die Möglichkeit des Einsatzes des Protokolls FABAN auf bereits existierenden Netzwerktopologien kann auf simple Art und Weise mit dem im vorherigen Abschnitt vorgestellten und bewiesenen Algorithmus überprüft werden. In dem Fall, dass eine vorliegende Netzwerkinfrastruktur nicht für die Ausführung von FABAN, aufgrund von mangelhafter redundanter Vernetzung, geeignet ist oder in dem Fall der Konstruktion eines speziell für den Einsatz von FABAN geeigneten Netzwerks stellt sich die Frage, ob Netzwerkgraphen sich derart konstruieren lassen, dass Sie sowohl eine möglichst geringe Anzahl an Verbindungen enthalten und gleichzeitig die Länge der Wellen ebenfalls möglichst gering ausfällt. Auch in diesem Zusammenhang stellt sich die Frage, von welchen Faktoren der Kompromiss zwischen Wellenlänge und der Anzahl der Verbindungen abhängt.

Im ersten Teil dieses Unterkapitels wird sowohl eine analytische Untersuchung der Möglichkeiten des Kompromisses zwischen Wellenlänge und Anzahl der Verbindungen durchgeführt, als auch ein Ansatz zur Findung von Netzwerkgraphen mit optimalen Wellen vorgestellt. Dies wird im zweiten Teil dieses Unterkapitels durch experimentelle Untersuchungen verifiziert.

### 5.3.1. Analytische Untersuchung von Wellenpaaren

Im Hinblick auf Wellenlänge und Kantenanzahl, spielen zwei besondere Graphenklassen eine wichtige Rolle:

**Ring** Die Klasse der Ring-Graphen ist die Klasse von Graphen mit der geringsten Anzahl von Kanten, auf welchen FABAN lauffähig ist. Jeder Knoten eines Rings hat den Knotengrad 2 und folglich hat ein Ring mit $n$ Knoten ebenfalls $n$ Kanten. Im Bezug auf FABAN-Eignung haben Ringe einerseits die kleinstmögliche Anzahl von Verbindungen

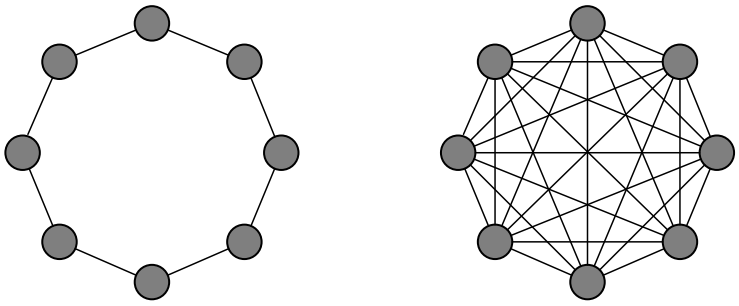


Abbildung 5.7.: Ring und vollvermaschtes Netz für 8 Knoten

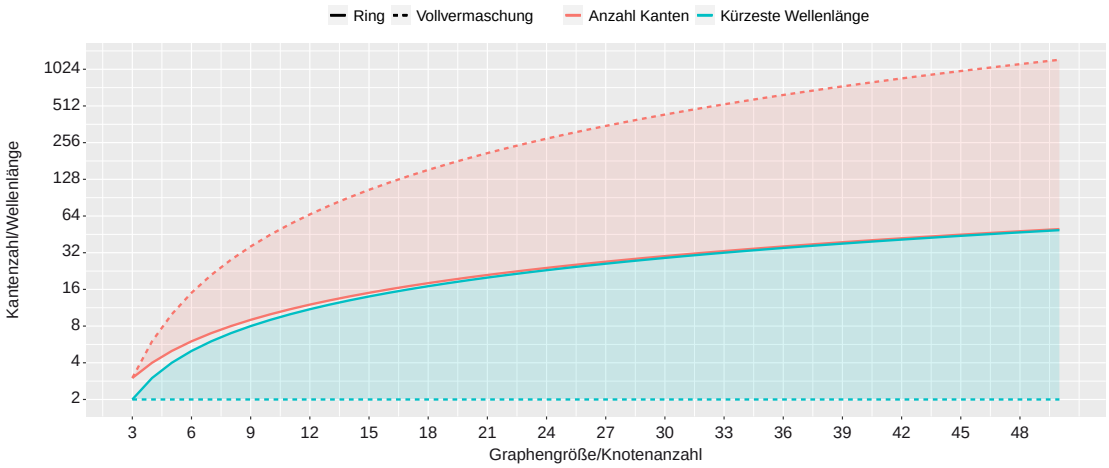



Abbildung 5.8.: Visualisierung des Verhältnisses von Kantenanzahl und kürzester Wellenlänge für Ring- und vollvermaschten Graphen in Abhängigkeit von festgelegter Anzahl an Knoten. Die rot und blau unterlegten Bereiche markieren jeweils die Bereiche, in denen für beliebige Graphen Kantenzahl und Wellenlänge liegen.

und andererseits, aufgrund der minimalen Verbindungsredundanz, die maximal mögliche Wellenlänge $n-1$ für Graphen mit $n$ Knoten. Des Weiteren ist ein Routing auf Ringen trivial und wäre, auch im Fall von FABAN, vollständig ohne jegliche Vorberechnungen umsetzbar. Nachrichten können auf den beiden Wellen im bzw. gegen den Uhrzeigersinn verschickt und die Rolle der Bridges anhand von Hopzählern innerhalb der Nachrichten ermittelt werden.

**Vollvermaschung** Die Klasse der vollvermaschten Graphen weist im Gegenzug zu den Ringen die höchstmögliche Anzahl von Kanten für eine festgelegte Anzahl von Knoten auf. Bei $n$ Knoten beträgt diese $0.5 \cdot n \cdot (n-1) = 0.5 \cdot (n^2 - n)$ Kanten. Im Bezug auf FABAN-Eignung haben vollvermaschte Netze maximale Größe und gleichzeitig garantierte minimal mögliche Wellenlängen.

Abbildung 5.7 zeigt ein Beispiel beider Graphen für 8 Knoten. Die Abwägung zwischen geringer Graphengröße, das heißt insbesondere geringeren Kosten, und kleinerer Wellenlänge, das heißt insbesondere geringeren Übertragungsdauern, ist im Allgemeinen eine komplexe Frage, die sich pauschal nicht beantworten lässt, sondern stets von den Anforderungen des Netzwerkkonstrukteurs an das Netzwerk abhängt. Abbildung 5.8 stellt die Wellenlängen und Kantenanzahl von Ringgraphen und vollvermaschten Graphen gegenüber. Ring und Vollvermaschung markieren die unteren und oberen Grenzen von beliebigen FABAN-Graphen hinsichtlich Wellenlänge und Kantenanzahl. In Abbildung 5.8 ist für beliebige FABAN-Graphen der Bereich aller möglichen Wellenlängen für eine festgelegte Knotenanzahl blau und der Bereich aller möglichen Anzahlen von Kanten

rot markiert. Optimale Graphen im Bezug auf FABAN sind solche, deren Kantenanzahl sowie Wellenlängen sich jeweils nah an der unteren Grenze beider Bereiche befinden. Dies ist trivialerweise unmöglich, ohne zusätzliche Verbindungen einzuführen, so dass es gilt, Kompromisse zwischen dem Zuwachs der Kantenanzahl und der Abnahme der Wellenlänge zu finden.

Die Abwägung zwischen der Kantenzunahme und der Wellenlängenabnahme eines gegebenen Graphen lässt sich auf unterschiedliche Weisen quantifizieren. Ein möglicher und vermutlich der intuitivste Ansatz ist die Definition von Vergleichsmaßen für das Messen von Unterschieden zweier Graphen. Die nachfolgende Definition führt zwei Maße für die relative und die absolute Differenz zweier Graphen ein, wobei die Grundidee darauf beruht, Unterschiede der Wellenlängen sowie Graphengrößen durch Differenzen bzw. Quotienten auszudrücken und diese dann durch Konstanten in Relation zu setzen:

**Definition 5.13**
Seien $G_1 = (\mathcal{B}, \mathcal{L}_1), G_2 = (\mathcal{B}, \mathcal{L}_2)$ zwei Netzwerkgraphen mit identischer Anzahl von Bridges und $\alpha_{abs}, \alpha_{rel} >= 0$ positive reelle Konstanten. Weiterhin seien $l_1$ und $l_2$ die maximalen Wellenlängen aller Wellen der jeweiligen Netzwerkgraphen sowie $k_1 := |\mathcal{L}_1|$ und $k_2 := |\mathcal{L}_2|$ die jeweilige Anzahl der Verbindungen beider Graphen.

1. Die *relative Differenz* zwischen $G_1$ und $G_2$ wird definiert durch

$$Q_{rel}(G_1, G_2) := \left(\frac{k_1}{k_2}\right)^{\alpha_{rel}} \cdot \frac{l_1}{l_2}, \tag{5.80}$$

   wobei $Q_{rel}(G_1, G_2) > 1$ besagt, dass $G_2$ ein günstigeres Kanten/Wellen-Verhältnis zu $G_1$ hat.

2. Die *absolute Differenz* zwischen $G_1$ und $G_2$ wird definiert durch

$$Q_{abs}(G_1, G_2) := \alpha_{abs}(k_1 - k_2) + (l_1 - l_2), \tag{5.81}$$

   wobei $Q_{abs}(G_1, G_2) > 0$ besagt, dass $G_2$ ein günstigeres Kanten/Wellen-Verhältnis zu $G_1$ hat.

Die Konstanten $\alpha_{rel}$ und $\alpha_{abs}$ legen zusätzlich die Proportionalität zwischen Verbindungszunahme/-abnahme und Wellenlängenreduktion/-erhöhung fest. Grundlegende Fälle werden in Tabelle 5.2 für $\alpha := \alpha_{rel} = \alpha_{abs}$ aufgelistet.

Beide Abstandsgrößen sind zudem transitiv. Dies ist ohne Verifikation ersichtlich. Beide Größen liefern grundlegend nach Definition unterschiedliche Ergebnisse, haben jedoch beide je nach Anforderung ihre Relevanz. Dies wird im Folgenden anhand von Untersuchungen von Graphenklassen deutlich. Die Ergebnisse für die relative sowie die

| | Relative Differenz | Absolute Differenz |
|---|---|---|
| $\alpha = 0$ | Änderungen der Anzahl der Verbindungen sind irrelevant. Jegliche Reduktion der Wellenlänge unabhängig von der Verbindungsanzahl ist eine Verbesserung | |
| $\alpha = 1$ | Änderung der Anzahl der Verbindungen um den Faktor $a \Rightarrow$ Wellenlängenänderung um den Faktor $1/a$ | Änderung der Anzahl der Verbindungen um $+a \Rightarrow$ Wellenlängenänderung um $-a$ |
| $0 < \alpha < 1$ oder $\alpha > 1$ | Änderung der Anzahl der Verbindungen um den Faktor $a \Rightarrow$ Wellenlängenänderung um den Faktor $(1/a)^{\alpha}$ | Änderung der Anzahl der Verbindungen um $+a \Rightarrow$ Wellenlängenänderung um $-a \cdot \alpha$ |

Tabelle 5.2.: Erläuterung der notwendigen Änderungen von Wellenlänge und Verbindungsanzahl in Abhängigkeit der Proportionalitätskonstante $\alpha := \alpha_{rel} = \alpha_{abs}$ um keine Veränderung der absoluten/relativen Differenz zu erhalten.

absolute Differenz sind wie folgt zu interpretieren:

- $Q_{res}(G_1, G_2) > 1 : G_2$ hat mit $\alpha$ gewichtet eine stärkere Reduktion der Wellenlänge als Zuwachs der Verbindungsanzahl oder eine stärkere Reduktion der Verbindungsanzahl als Zuwachs der Wellenlänge im Vergleich zu $G_1$.
- $Q_{abs}(G_1, G_2) > 0 : G_2$ hat mit $\alpha$ gewichtet eine stärkere Reduktion der Wellenlänge als Zuwachs der Verbindungsanzahl oder eine stärkere Reduktion der Verbindungsanzahl als Zuwachs der Wellenlänge im Vergleich zu $G_1$.

### 5.3.2. Optimierung von Wellenpaaren

Im vorherigen Abschnitt wurden die zwei wichtigen Eigenschaften zur Bewertung von Wellen bereits diskutiert: Wellenlänge und Verbindungsanzahl. Mit der Klasse der Ringe und der vollvermaschten Netze wurden ebenfalls zwei Klassen vorgestellt, welche jeweils gegensätzlich optimale und schlimmstmögliche Eigenschaften besitzen. Gibt es eine Möglichkeit, auf systematische Art und Weise Graphen mit besseren Eigenschaften zu erzeugen?

Um gegebene Graphen für die Nutzung von FABAN zu verbessern, gibt es intuitiv zwei Ansätze, die im Folgenden diskutiert werden.

### Ansatz 1: Hinzufügen von Verbindungen um die Wellenlänge zu verringern

Durch systematische Graphen-abhängige Erweiterungen um Kanten können Graphen gemäß dem Maß der Graphen-Vergleichsfunktion $Q$ verbessert werden. Ausgehend von den speziellen Eigenschaften von Ringen und vollvermaschten Graphen im Bezug auf Graphengröße und Wellenlänge, wie in Abbildung 5.8 dargestellt, lässt sich dieser Ansatz auf Ringe anwenden. Die Idee ist, nur die minimal nötigsten Verbindungen zu ergänzen und gleichzeitig die Wellenlänge deutlich zu reduzieren.

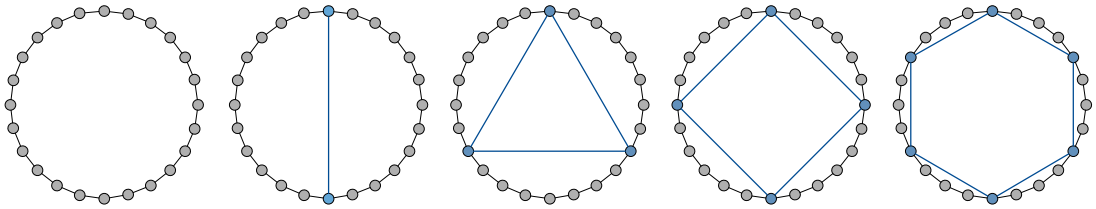


Abbildung 5.9.: Darstellung äquidistanter Erweiterungen von Ringen durch zusätzliche $k \in \{1,3,4,6\}$ Verbindungen. Blau markierte Knoten bilden den so genannten „Innenring“, während die zusammenhängenden Gruppen von grauen Knoten als „Außenringfragmente“ bezeichnet werden.

Abbildung 5.9 zeigt die möglichst äquidistante Erweiterung durch Verbindungen, um die Wege zwischen beliebigen Knoten zu verkürzen. Vorteile bringt dieser Ansatz bereits ab einer Erweiterung von $k = 1$ Verbindung, welche die Distanz zwischen einigen, jedoch nicht allen Knotenpaaren verkürzt. Eine Verkürzung der Distanzen zwischen allen Knotenpaaren entsteht durch die Erweiterung um mindestens $k \geq 3$ zusätzlichen Verbindungen. Sowohl die Erweiterung von $k = 1$ sowie von $k \in \{3,4,6\}$ Verbindungen sind in Abbildung 5.9 dargestellt. Durch $k \geq 3$ Erweiterungen werden die Komponenten des Rings in die Gruppe des Innenrings $\mathcal{B}_I$ sowie $k$ Gruppen von Außenringfragmenten $\mathcal{B}_O^{(i)}$, $1 \leq i \leq k$ unterteilt. Eine Verbindung zwischen zwei beliebigen Knoten besteht folglich entlang Verbindungen im Innenring, sowie entlang Verbindungen in exakt 2 Außenringfragmenten, so dass Verbindungen in $k-2$ Außenringfragmenten vollständig bei der Kommunikation nicht beteiligt sind. Insgesamt ergibt sich damit für einen Ring mit $n$ Knoten und $k$ zusätzlichen Verbindungen die maximale Anzahl von Knoten in den Außenringfragmenten zu

$$\max\left\{\left|\mathcal{B}_O^{(i)}(n,k)\right| \;\middle|\; 1 \leq i \leq k\right\} = \left\lceil \frac{n-k}{k} \right\rceil \tag{5.82}$$

und mit $|\mathcal{B}_I(n,k)| = k$ die sich daraus ergebende Wellenlänge von

$$\begin{aligned} f_n(k) :&= 2 \cdot \max\left\{\left|\mathcal{B}_O^{(i)}(n,k)\right| \;\middle|\; 1 \leq i \leq k\right\} + (|\mathcal{B}_I(n,k)| - 1) \\ &= 2 \cdot \left\lceil \frac{n-k}{k} \right\rceil + k - 1 = k - 3 + 2 \cdot \left\lceil \frac{n}{k} \right\rceil. \end{aligned} \tag{5.83}$$

Mit der Bezeichnung $R_{n+(k)}$ für eine Erweiterung eines Rings mit $n$ Kanten um $k$ zusätzliche Verbindungen, lässt sich nun mit Hilfe der Abstandsgrößen

$$\begin{aligned} Q_{rel}\left(R_n, R_{n+(k)}\right) &= \left(\frac{n}{n+k}\right)^{\alpha} \cdot \frac{n-1}{k-3+2\left\lceil\frac{n}{k}\right\rceil} \\ Q_{abs}\left(R_n, R_{n+(k)}\right) &= n+2-(\alpha+1)k-2\cdot\left\lceil\frac{n}{k}\right\rceil \end{aligned} \tag{5.84}$$

untersuchen, ob abhängig von der Ringgröße optimale Erweiterungsmöglichkeiten existieren.

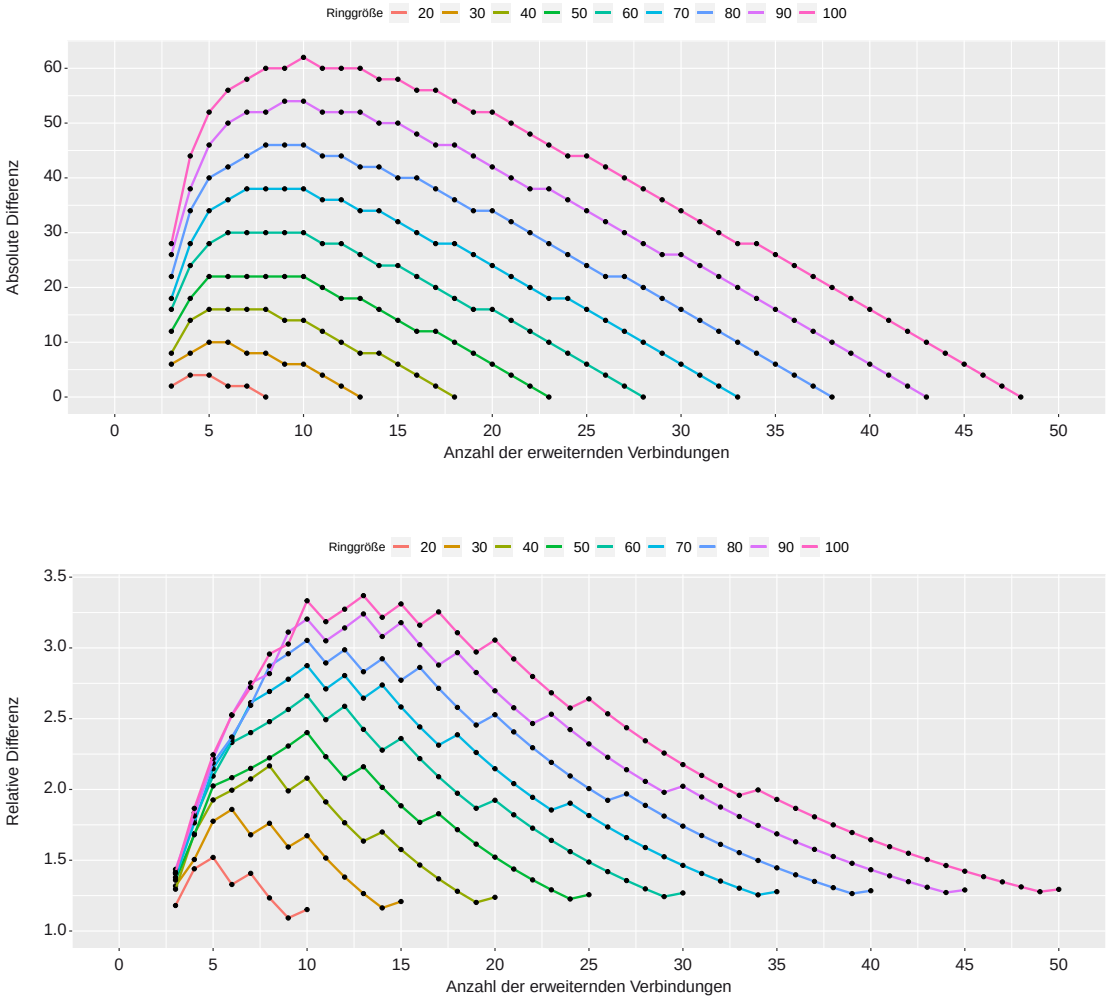



Abbildung 5.10.: Darstellung der Ergebnisse der Wellenpaar-Vergleichsfunktion zwischen Ringen und jeweiliger Erweiterungen um bis zu $n/2$ Verbindungen. Sprünge in allen Graphen entstehen durch den nicht stetigen Verlauf der Gaußschen Rundungsklammer im Ausdruck für die Berechnung der resultierenden Wellenlängen der Ringerweiterungen.

Abbildung 5.10 visualisiert die relative sowie die absolute Differenz zwischen ursprünglichen Ringen und Ringerweiterungen von bis zu $n/2$ zusätzlichen Verbindungen. Deutlich erkennbar ist die mit der Ringgröße zunehmende Sinnhaftigkeit der Ringerweiterungen, sowie der enorme Effekt der Reduktion der Wellenlängen durch die Zunahme weniger Verbindungen. Numerische Berechnungen, sowie Skalierungen und Approximationen ergeben, dass die optimale Wellenlänge, d.h. das Maximum der absoluten Differenz mit $\alpha = 1$ in Abhängigkeit von $n$, für größere Ringgrößen $n \geq 10$ durch die Funktion

$$f^{abs}_{approx}(n) \approx 3.02 \cdot \sqrt{n} \tag{5.85}$$

sowie die benötigten Zusatzverbindungen durch

$$k^{abs}_{approx}(n) \approx 0.93 \cdot \sqrt{n} \tag{5.86}$$

approximiert werden können. Wird die relative Differenz zugrunde gelegt, entstehen ähnliche Ergebnisse mit

$$f^{rel}_{approx}(n) \approx 2.8 \cdot \sqrt{n} \tag{5.87}$$

sowie die benötigten Zusatzverbindungen durch

$$k^{rel}_{approx}(n) \approx 1.37 \cdot \sqrt{n} \tag{5.88}$$

Abbildung 5.11 stellt die exakt berechneten Werte und die in (5.85) bis (5.88) formulierten Approximationen der Wellenlänge sowie der notwendigen Zusatzverbindungen für Ringgrößen bis 10000 dar.

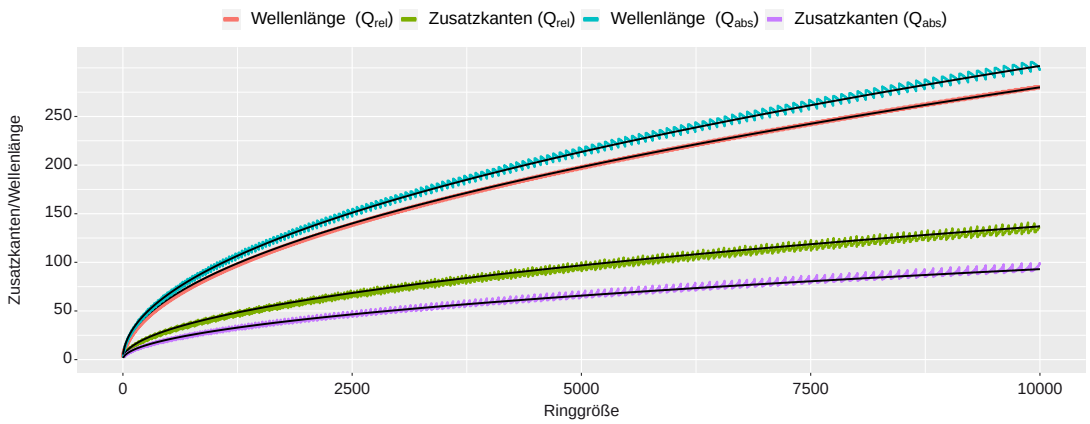



Abbildung 5.11.: Gegenüberstellung approximierter und exakter Ergebnisse von optimalen Ringerweiterungen, gemessen anhand relativer sowie absoluter Differenzen. Exakte Ergebnisse sind in unterschiedlichen Farben gemäß der Legende dargestellt, während die Approximationen zwecks Übersichtlichkeit jeweils durch die durchgehenden, schwarzen Kurven dargestellt sind. Beide Größen sind dabei jeweils von der Ordnung $\mathcal{O}(\sqrt{n})$.

Ohne näher darauf einzugehen sei noch angemerkt, dass die Distanz-Reduzierung durch Innenringe rekursiv fortgeführt werden kann, indem auf die entstehenden Außenringfragmente wieder eine analoge Erweiterung angewandt wird. Abbildung 5.12 zeigt ein Beispiel dafür.

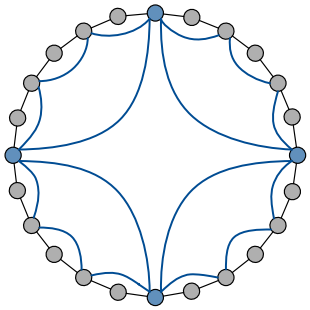


Abbildung 5.12.: Beispiel einer rekursiven Ringerweiterung durch wiederholt zusätzliche Verbindungen.

### Ansatz 2: Entfernen von Verbindungen um die Graphengröße zu reduzieren

Ausgehend von Graphen mit zu vielen redundanten und insbesondere nutzlosen Verbindungen, ist es gegebenenfalls möglich, durch systematische Vorgehensweise diese zu identifizieren und zu entfernen, so dass dadurch kein schädlicher Einfluss auf die kürzestmögliche Wellenlänge entsteht, gleichzeitig allerdings die Graphengröße bezüglich der Verbindungsanzahl abnimmt. Diese Idee lässt sich auf die Klasse der vollvermaschten Graphen anwenden und die Anzahl der Verbindungen von quadratischer Ordnung $\mathcal{O}(n^2)$ zu linearer Ordnung $\mathcal{O}(n)$ reduzieren und gleichzeitig die bestmögliche Wellenlänge 2 der vollvermaschten Graphen bewahren.

Betrachtet man ein beliebiges vollvermaschtes Netz mit $n$ Knoten, so können für jede Distributing Bridge beliebige Checking Bridges gewählt werden, so dass mit lediglich einem weiteren Hop alle Bridges die zu übertragende Nachricht direkt von der Checking Bridge erhalten. Eine im Hinblick auf FABAN effizientere Möglichkeit der Verteilung von Nachrichten basiert auf der Separierung von 3 Knoten, so dass folgende disjunkte Zerlegung der Menge der Knoten

$$\mathcal{B} = \mathcal{B}_O \cup \mathcal{B}_C \tag{5.89}$$

mit $|\mathcal{B}_C| = 3$ und $|\mathcal{B}_O| = n - 3$ entsteht. Die Reduktion der Verbindungsmenge erfolgt durch das Entfernen aller Verbindungen zwischen Knoten aus $\mathcal{B}_O$, so dass nur noch

$$|\mathcal{L}| = 3(n-3) + 3 = 3n - 6 \tag{5.90}$$

Verbindungen übrig bleiben. In solchen resultierenden Graphen kann nun jede Distributing Bridge eine Nachricht zunächst an 2 Checking Bridges aus $\mathcal{B}_C$ senden, woraufhin diese auf

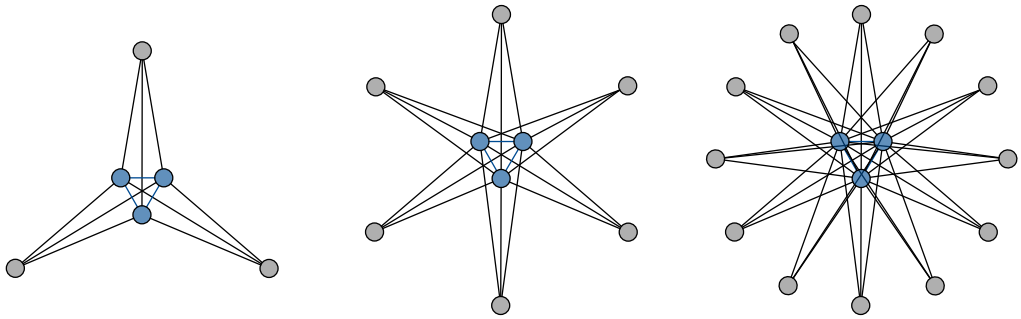


Abbildung 5.13.: Darstellung redundanter Sterntopologien, konstruiert aus vollvermaschten Graphen durch Reduktion um nicht zwingend erforderliche Verbindungen für 6, 9 und 15 Knoten.

direktem Wege mit nur einem zusätzlichen Hop alle Knoten im Netzwerk erreichen können. Die Notwendigkeit von 3 Bridges in $\mathcal{B}_C$ ist eine Folge des möglichen Szenarios, dass eine Bridge in $\mathcal{B}_C$ die Rolle der Distributing Bridge annehmen muss. Mit der Annahme, dass die Knoten in $\mathcal{B}_C$ keine verbundenen Rechenkomponenten besitzen, sondern lediglich zum Routing existieren, lässt sich die Verbindungsanzahl noch weiter reduzieren, so dass eine Zerlegung $\mathcal{B} = \mathcal{B}'_O \cup \mathcal{B}'_C$ entsteht mit $|\mathcal{B}'_C| = 2$, $|\mathcal{B}'_O| = n - 2$ sowie $|\mathcal{L}'| = 2(n-2)$.

Sowohl die Berechnung der relativen als auch der absoluten Differenz zwischen vollvermaschten Graphen und der optimierten Variante ist überflüssig, da nur eine Verbesserung, jedoch keinerlei Verschlechterung stattfindet. Aufgrund der starken optischen Ähnlichkeit zu Sterntopologien, welche naturgemäß keinerlei Redundanz aufweisen, wird diese Klasse von Graphen hier als die Klasse der *redundanten Sterntopologien* bezeichnet. Abbildung 5.13 zeigt einige Beispiele redundanter Sterne auf.

Abschließend zeigt Abbildung 5.14, in welchen Bereichen sich sowohl optimierte Ringe als auch redundante Sterne bezüglich Wellenlänge und Verbindungsanzahl in Abhängigkeit von der Knotenanzahl $n$ befinden. Ein Vergleich zwischen diesen beiden Graphenklassen sowie die Frage, inwieweit Verbesserungen darüber hinaus möglich sind, wird im nachfolgenden Abschnitt zum Abschluss dieses Kapitels untersucht.

Die in diesem Abschnitt erarbeiteten Resultate sind im Hinblick auf die Anforderungen - auch harte Echtzeit-Anforderungen - an ein zu konstruierendes System nutzbar. Einfache Ringe lassen sich durch Hinzunahme von wenigen Verbindungen derart erweitern, dass zum Beispiel Anforderungen an maximale Distanzen nahezu beliebig umgesetzt werden können. Auf diese Weise lassen sich die Kommunikationsdauern an harte Echtzeitanforderungen anpassen.

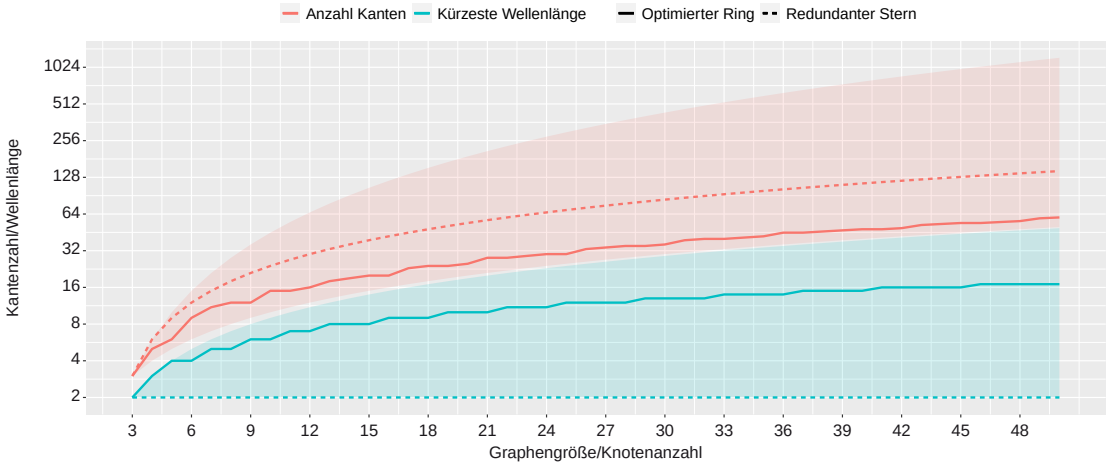



Abbildung 5.14.: Visualisierung des Verhältnisses von Kantenanzahl und kürzester Wellenlänge für optimierte Ringe und redundante Sterntopologien in Abhängigkeit der Knotenanzahl $n$. Die rot und blau unterlegten Bereiche markieren wie in Abbildung 5.8 jeweils die Bereiche, in denen die Kantenanzahl und die Wellenlänge für beliebige Graphen liegen.

### 5.3.3. Experimentelle Validierung

Im vorherigen Abschnitt haben die Überlegungen gezeigt, dass die Eignung von Graphen für FABAN abhängig ist von dem zugrunde gelegten Maßstab für die Bewertung. Dies wurde mit den Klasse der Ring- und der Klasse der vollvermaschten Graphen, sowie jeweiliger Abwandlungen beider Graphenklassen näher untersucht. Hierbei lässt sich auf einfache Weise nachrechnen, dass bezüglich der relativen Differenz, die Klasse der redundanten Sterne besser ist als die Klasse der optimierten Ringe und bezüglich der absoluten Differenz dies genau andersherum gilt.

Um zu verifizieren bzw. eine Tendenz für eine Aussage zu erhalten, ob mit diesen beiden Graphenklassen bereits optimale Klassen bezüglich beider Abstandsgrößen gefunden wurden, wurde eine Reihe von experimentellen Generierungen von Graphen durchgeführt, mit folgenden Parametern:

- Knotenanzahl von 3 bis 50
- Systematische Generierung aller möglichen Graphen für $3 \leq n \leq 8$
- Zufällige Generierung von Graphen für $9 \leq n \leq 50$, wobei
  - jede mögliche Verbindung mit Wahrscheinlichkeit $p = 0.3$ existiert
  - für jedes $n$, $9 \leq n \leq 50$, $2^{20}$ Graphen (ohne Erkennung von Duplikaten) generiert wurden

Für jeden generierten Graphen wurde anschließend der Algorithmus zur Generierung von

Wellen (mit Kostenminimierung bei konstant gleichen Kosten jeder Verbindung) ausgeführt und bei Erfolg Statistiken über Wellenlänge und Verbindungsanzahl gesammelt. Zum Schluss wurden die Ergebnisse ausgewertet, indem für jede gefundene Kombination aus Wellenlänge und Verbindungsanzahl die relative Differenz zur redundanten Sterntopologie derselben Knotenanzahl, sowie die absolute Differenz zum optimierten Ring der selben Knotenanzahl berechnet.

Die Abbildungen 5.15 sowie 5.16 fassen die Ergebnisse zusammen und beweisen für $3 \leq n \leq 8$ die Behauptung, dass der redundante Stern und der optimierte Ring bezüglich relativer und absoluter Differenz bestmögliche Graphen sind. Für $n \geq 9$ ist die Behauptung damit natürlich nicht bewiesen, zeigt allerdings eine eindeutige Tendenz. Jedoch stößt der experimentelle probabilistische Ansatz für $n > 50$ an Laufzeitgrenzen. Hierbei hat ein Durchlauf des Experiments für $n = 50$ Knoten $8,23$ Stunden gebraucht, wobei maximal mögliche Parallelisierung mit 32 Threads auf 2 Intel Xeon E5-2560 Prozessoren ausgenutzt wurde. [Cor12] Eine Erhöhung der Anzahl der Experimente würde folglich wenig Sinn machen und wenig vielversprechend sein.

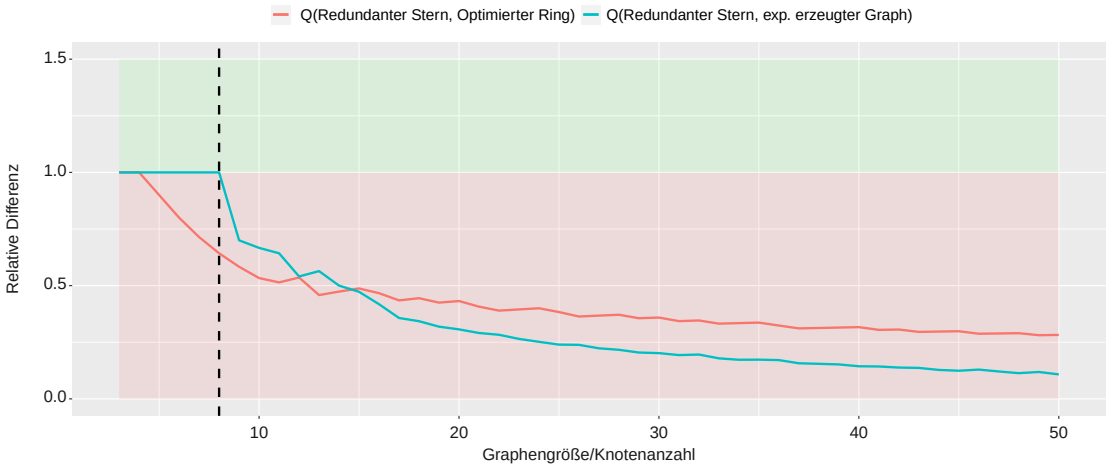



Abbildung 5.15.: Vergleich der relativen Differenz zwischen dem redundanten Stern und dem optimierten Ring sowie dem redundanten Stern und dem besten, auf experimentelle Art generierten Graphen. Der grüne/rote Bereich kennzeichnet den Bereich, indem eine Verbesserung/Verschlechterung im Vergleich zum redundanten Stern stattfinden würde. Die vertikale gestrichelte Linie kennzeichnet den Beginn zufälliger Graphengenerierungen.

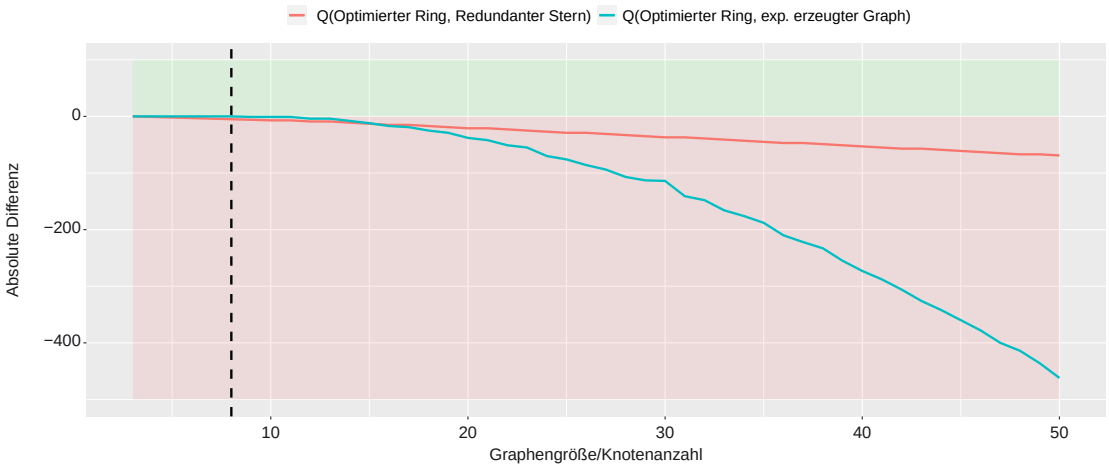



Abbildung 5.16.: Vergleich der absoluten Differenz zwischen dem optimierten Ring und dem redundanten Stern sowie dem optimierten Ring und dem besten, auf experimentelle Art generierten Graphen. Der grüne/rote Bereich kennzeichnet den Bereich, indem eine Verbesserung/Verschlechterung im Vergleich zum redundanten Stern stattfinden würde. Die vertikale gestrichelte Linie kennzeichnet den Beginn zufälliger Graphengenerierungen.

## 5.4. Zusammenfassung

Das Protokoll FABAN wurde ursprünglich informell entwickelt. Insbesondere die Anforderungen an die Wellen wurden durch gründliche Überlegungen in Form von Regeln spezifiziert und anhand von systematisch ausgewählten Beispielen auf Korrektheit und Plausibilität überprüft. Des Weiteren ermöglichen die formulierten Regeln lediglich eine heuristische, algorithmisch nicht umsetzbare Methode zur Erstellung von Wellen für gegebene Netzwerktopologien.

Die informelle Beschreibung der Wellenanforderungen wurde in diesem Kapitel im Fall der Einfehlerannahme um eine formale Analyse erweitert, indem die Routing-Anforderungen formalisiert und anschließend mathematisch bewiesen wurden. Dazu wurden aus der Graphentheorie bekannte Strukturen verwendet und um neu eingeführte, FABAN-spezifische Begriffe erweitert. Zusätzlich wurde ein Algorithmus entwickelt, der zuverlässig und effizient für eine gegebene Netzwerktopologie ein FABAN-Routing ermittelt, falls FABAN auf dieser angewendet werden kann. Die Korrektheit des Algorithmus wurde formal verifiziert und die Laufzeiteigenschaften in einer Komplexitäts- und Laufzeitanalyse untersucht. Des Weiteren wurde durch systematische Generierungen von Routings durch den Algorithmus gezeigt, dass die durch den Algorithmus generierten Wellenpaare im Hinblick auf die Länge der Routingwege fast optimale Ergebnisse liefern und diese für steigende Topologiegrößen tendenziell besser werden.

Abgeschlossen wurde die Arbeit durch eine ausführliche Evaluation von geeigneten Netzwerkgraphen, indem verschiedene Topologieklassen untersucht und miteinander verglichen wurden. Dabei hat sich deutlich gezeigt, dass Ringe mit zusätzlichen Zwischenverbindungen, welche auch als Verbund mehrerer einzelner Ringe gesehen werden können, für die Anwendung von FABAN den optimalen Kompromiss zwischen zusätzlichen Komponenten (Verbindungsredundanz) und geringen maximalen Distanzen zwischen Sender- und Empfängerpaaren darstellen.

Kapitel 6

# ExFABAN
# Toleranz von Mehrfachfehlern

Dieses Kapitel setzt sich mit den Möglichkeiten der Toleranz von Mehrfachfehlern des Protokolls FABAN auseinander. Nachdem das Problem und die besondere Herausforderung der Erweiterung des Protokolls zur Toleranz von Mehrfachfehlern im ersten Abschnitt dieses Kapitels erörtert worden ist, wird im nachfolgenden Abschnitt eine vollständige und detaillierte Beschreibung einer Erweiterung zur Toleranz von beliebig vielen Fehlern erfolgen. Das dadurch resultierende Protokoll wird als *Extended Fault Tolerant Atomic Broadcast and Agreement in Bridge Connected Networks for Multiple Faults (ExFABAN)* bezeichnet. Die Spezifikation von ExFABAN beinhaltet einerseits eine erweiterte Separierung von Rollen der Bridges, in Form der Einführung der Prechecking Bridge (PB) und andererseits der damit als Folge notwendigen Abänderung der Signaturmodifikationen. Abgeschlossen wird die Protokollspezifikation mit der Erweiterung des redundanten Routings, welches aufgrund des Ziels, Mehrfachfehler zu tolerieren, deutlich an Komplexität zunimmt.

Zum Abschluss wird mit einem topologiebasierten Ansatz, die Anwendung der hier vorgestellten Protokollerweiterung auf realen Netzwerken untersucht und es werden geeignete Lösungen vorgestellt. Dafür werden neben der bekannten Klasse der vollvermaschten Netze mit den so genannten *mehrschichtigen* Ringen und *redundanten Sterntopologien mit Verteilungskern* neue Graphenklassen für ExFABAN vorgestellt. Anschließend wird die Skalierbarkeit bei wachsendem Grad der Fehlertoleranz analysiert.

## 6.1. Probleme und Herausforderungen der Toleranz von Mehrfachfehlern

Das Protokoll FABAN nutzt für die Toleranz von einem Fehler eine Reihe von Redundanzmechanismen, um die erwünschte Fehlertoleranz zu erzielen. Die Fehlertoleranz, die durch die Redundanzmechanismen *Duplikaterkennung*, *Signatur* sowie *Signaturmodifikation* (siehe Tabelle 3.1) gewährleistet ist, lässt sich ohne grundlegende Anpassung auf die Toleranz von Mehrfachfehlern übertragen. Das *redundante Routing* kann in unveränderter Form lediglich einen Fehler tolerieren, da zu jedem Empfänger eines Netzwerks eine Nachricht entlang exakt zwei disjunkter Pfade geleitet wird. Um eine Erhöhung des Fehlertoleranzgrades auf $f$ Fehler zu erzielen, muss folglich die Anzahl der redundanten bzw. disjunkten Pfade zu jedem Empfänger von 2 auf $f+1$ erhöht werden. Die Notwendigkeit von $f+1$ redundanten Pfaden impliziert wiederum die Erhöhung der Anzahl der Checking Bridges ebenfalls auf $f+1$.

### Problem 1: Ungenügende Verzögerungsüberprüfungen

Bei der simplen Erweiterung des Protokolls um genügend zusätzliche redundante Pfade in der Form der Einführung weiterer Checking Bridges, welche direkt mit der Distributing Bridge verbunden sind, wird der erwünschte Grad der Fehlertoleranz $f > 1$ tatsächlich erzielt, allerdings nur unter der Annahme, dass die Distributing Bridge fehlerfrei ist. Dies ist allerdings in einem verteilten System, in welchem jede Bridge, mit der Netzwerkknoten verbunden sind, als Distributing Bridge handeln kann, nicht realistisch und würde im Allgemeinen zu inkonsistenter Nachrichtenübertragung führen, falls die Distributing Bridge von bestimmten Fehlern betroffen ist. Abbildung 6.1 zeigt eine beispielhafte Darstellung für ein Szenario zur Toleranz von zwei Fehlern mit drei Checking Bridge sowie folglich drei Wellen. Sollte die Distributing Bridge derart fehlerhaft sein, dass diese die Nachricht an die ersten beiden Checking Bridges unterdrückt und an die dritte Checking Bridge stark verzögert sendet, kann es passieren, dass ein weiterer Fehler in der dritten Checking Bridge dazu führt, dass die Überprüfung der Verzögerung fehlerhaft ist oder gar nicht stattfindet und in Folge dessen eine Welle initiiert wird, welche nicht mehr rechtzeitig alle Empfänger erreichen kann.

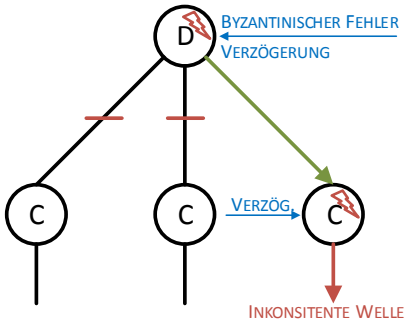



Abbildung 6.1.: Problemszenario 1

### Problem 2: Initiierung inkonsistenter Wellen

Das erste Problem lässt sich beheben, indem man zusätzliche Überprüfungen bzgl. Verzögerungen vor der Initiierung jeder Welle einführt. Dadurch wird zwar das Problem eines Verzögerungsfehlers byzantinischer Art der Distributing Bridge behoben, es ist aber dennoch nicht gewährleistet, dass entweder keine inkonsistenten Wellen initiiert werden oder zumindest genügend fehlerfreie Wellen initiiert werden, welche die Inkonsistenz aufheben können. Abbildung 6.2 verdeutlicht das Problem anhand eines Beispiel für die Toleranz von 2 Fehlern. Auch hier kann ein Fehler der Distributing Bridge dazu führen, dass unabhängig von der Anzahl der zu tolerierenden Fehler $f$ bei $f+1$ Wellen nur eine einzige Checking Bridge eine zu verteilende Nachricht erhält und somit ein weiterer Fehler, insbesondere Verzögerungsfehler, ausreicht um Inkonsistenz der Übertragung zu verursachen.

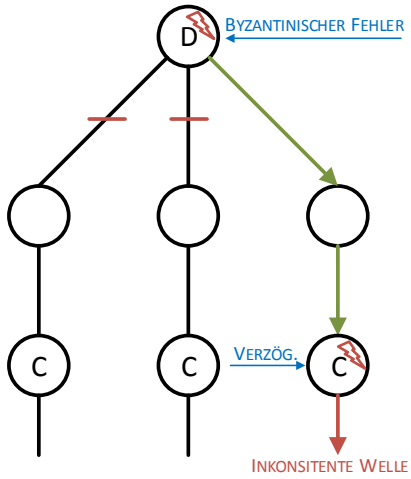



Abbildung 6.2.: Problemszenario 2

### Lösungsansatz: Gewährleistung der Initiierung genügend vieler Wellen

Um das zuvor beschriebene Problem zu verhindern, muss folglich gewährleistet werden, dass in dem Fall, dass inkonsistente Wellen entstehen können, genügend fehlerfreie Wellen initiiert werden. Konkret bedeutet dies, dass für einen zu erzielenden Fehlertoleranzgrad $f$ mindestens $f+1-k$ Wellen initiiert werden müssen, falls bereits $0 \leq k \leq f$ Fehler in dem Bereich vor der Initiierung der Wellen auftreten (In Abbildung 6.3 gestrichelt umrandet). Abbildung 6.3 zeigt ein vollständiges Beispiel einer Lösung, welche garantiert genügend Wellen initiiert um die geforderte Fehlertoleranz zu erhalten. Im Falle einer fehlerhaften Distributing Bridge initiieren entweder mindestens zwei Checking Bridges Wellen oder gar keine. In beiden Fällen wird eine Inkonsistenz der Nachrichtenübertragung vermieden.

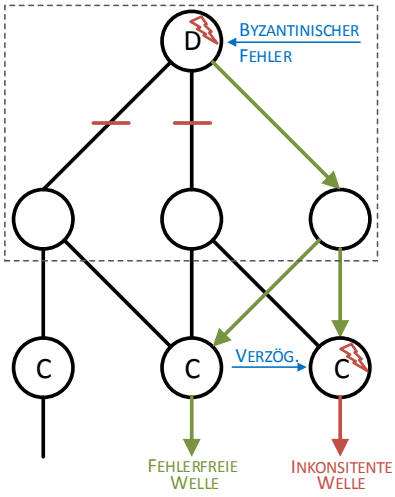



Abbildung 6.3.: Lösungsszenario

#### Herausforderung der Beschreibung sowie Ermittlung des Routings der Wellen

Eine besondere Herausforderung entsteht durch die Frage der Ermittlung eines konkreten Routings für gegebene Graphen bzw. der Ermittlung, ob ein geeignetes Routing existiert. Grund hierfür ist der enorme Anstieg der Komplexität der Aufgabe, drei oder mehr redundante Wellen entlang eines gegeben Graphen zu finden, ohne die Gewissheit zu haben, ob Lösungen überhaupt gefunden werden können. Ein anderer Ansatz für die Lösung dieses Problems ist es, geeignete Netzwerkgraphen für einen festgelegten Grad der Fehlertoleranz zu konstruieren. Beide Ansätze werden im Laufe des Kapitels genau untersucht und hier vorerst nicht weiter thematisiert. Zuletzt ist anzumerken, dass die Einbettung einer Struktur, wie zum Beispiel in Abbildung 6.3 dargestellt, in ein gegebenes Netzwerk topologisch anspruchsvoll ist.

## 6.2. Allgemeine Erweiterung des Protokolls

Die in diesem Abschnitt vorgestellte Erweiterung für das Protokoll FABAN mit der Bezeichnung ExFABAN ist vollständig kompatibel zur Protokollspezifikation in Kapitel 3 für die Toleranz von einem Fehler, während abhängig vom geforderten Fehlertoleranzgrad die Menge benötigter redundanter Mittel steigt. Folglich hat ExFABAN viele Gemeinsamkeiten zu FABAN, welche im Folgenden in kurzer Form zusammengefasst werden.

Die Annahmen an das anzuwendende System bleiben unverändert (siehe Kapitel 3.1.1 für Details):

- Bridge-Connected-Networks
- Synchrone Uhren
- Beschränkte pro Hop Latenz

Weiterhin unverändert bleibt die Funktionsweise sowohl der Rollen der Checking Bridge als auch der Forwarding Bridge, während die Distributing Bridge eine kleine Anpassung erfordert, die im Abschnitt zum *Primären Routing* genauer beschrieben wird.

Zusätzlich wird im Folgenden, nachdem die *Fehlerbereichannahme* angepasst worden ist, eine neue Rolle für Bridges mit der Rolle der *Prechecking Bridges* eingeführt. Aufgrund dieser neu eingeführten Bridge-Rolle, muss das Verfahren der Signaturmodifikation angepasst werden sowie das Format einer FABAN-Nachricht erweitert werden.

### 6.2.1. Fehlerbereichsannahme

Die Fehlerbereichsannahme für die Toleranz von Mehrfachfehlern kann aus Kapitel 5.1.2 übernommen werden, das heißt

$$F_i := E_i \cup \left[ \bigcup_{(b_i,b_j)\in\mathcal{L}} E_{i,j} \right], \tag{6.1}$$

wobei $E_i$ für $i \in \{1, \ldots, |\mathcal{B}|\}$ die Einzelfehlerbereiche, welche jeweils eine Bridge und die damit verbundenen Rechenkomponenten sowie die zugehörigen Verbindungen umfassen, und $E_{i,j}$ die Einzelfehlerbereiche sind, welche nur aus Verbindungen zwischen Bridge $b_i$ und $b_j$ bestehen, beschreiben. Nach Definition 2.4 dürfen höchstens $f$ Fehlerbereiche zur gleichen Zeit von Fehlern betroffen sein, falls $f$-Fehlertoleranz angenommen wird. Details wurden in Abschnitt 5.1.2 behandelt.

### 6.2.2. Erweiterung der Bridge-Rollen und der Signaturmodifikation

Zusätzlich zu den Rollen der *Distributing Bridge*, *Checking Bridge* sowie *Forwarding Bridge* wird die Rolle der *Prechecking Bridge* eingeführt. Die Aufgabe der Prechecking Bridge ist analog zur Checking Bridge die Überprüfung, ob eine empfangene Nachricht immer noch alle Empfänger im Netzwerk vor Verstreichen der errechneten Deadline, erreichen kann. Sollte dies nicht der Fall sein, wird die Nachricht unterdrückt, andernfalls wird die Nachricht gemäß des Routings weitergeleitet. Hierbei wird die Nachricht entweder zu einer weiteren Prechecking Bridge oder einer Checking Bridge weitergeleitet, allerdings noch keine Welle initiiert. Dies bleibt die alleinige Aufgabe der Checking Bridges.

Bei einer $f$-Fehlerannahme müssen zwischen der Distributing Bridge und jeder der $f+1$ Checking Bridges insgesamt mindestens $f-1$ Prechecking Bridges durchlaufen werden. Der Durchlauf von mehr als $f-1$ Prechecking Bridges ist nicht nötig, kann aber in bestimmten Fällen hilfreich sein, um geeignete Routings zu finden. Dies wird im Laufe des Kapitels thematisiert. Um nachvollziehen zu können, wie viele Prechecking Bridges tatsächlich eine Nachricht durchlaufen hat, wird das Nachrichtenformat einer FABAN-Nachricht um das Feld eines Hopzählers ergänzt, welcher in allen Bridges mit Ausnahme der Forwarding Bridges inkrementiert wird. Des Weiteren ist es notwendig zu unterscheiden, von welcher Checking Bridge eine Welle initiiert wurde, so dass das Nachrichtenformat auch um das Feld der ID der Checking Bridge erweitert wird. Einerseits wird im Laufe des Abschnitts deutlich, dass Nachrichten unterschiedlicher Wellen gegebenenfalls parallel zwischen zwei Bridges verschickt werden dürfen bzw. müssen und andererseits wird in Kapitel 7 durch Simulationen ein Problem auch im Einfehlerfall erkannt und thematisiert, welches durch die Information der initiierenden Checking Bridge behoben wird. Dies betrifft primär die reale Umsetzung des Protokolls, so dass

an dieser Stelle nicht weiter darauf eingegangen und auf nachfolgende Kapitel verwiesen wird. Abbildung 6.4 zeigt den Aufbau einer ExFABAN Nachricht.


Abbildung 6.4.: Nachrichtenaufbau einer ExFABAN Nachricht

Die Entscheidung, gemäß welcher Rolle eine Bridge sich verhalten muss, kann hierbei gegebenenfalls vom Hopzähler abhängig gemacht werden. Dies ist genau dann der Fall, wenn der Abstand zwischen Distributing Bridge und jeder Checking Bridge konstant ist. Eine andere Möglichkeit besteht in der Vorkonfigurierung aller Bridges um benötigte Routing-Informationen in Abhängigkeit von der Distributing Bridge. Unabhängig von der Entscheidungsvariante, wird der Hopzähler benötigt, um verifizieren zu können, dass eine Nachricht tatsächlich mindestens $f-1$ Prechecking Bridges durchlaufen hat. Dafür wird das in Kapitel 3.1.2 beschriebene Verfahren zur Signaturmodifikation verallgemeinert und erweitert, indem die Signaturmodifikation in Distributing Bridge, Prechecking Bridge und Checking Bridge durch folgende drei, den Bridges bekannte, bijektive Funktionen ausgedrückt wird, wobei jede Funktion wiederum eine Komposition zweier bijektiver Funktionen darstellt und $l$ der Länge der Signatur in Bits entspricht:

$$\begin{aligned}
\sigma_d, \sigma_d^{eg}, \sigma_d^{ing} &: \{0,1\}^l \to \{0,1\}^l, \quad \sigma_d(S) \mapsto \left(\sigma_d^{eg} \circ \sigma_d^{ing}\right)(S) \\
\sigma_p, \sigma_p^{eg}, \sigma_p^{ing} &: \{0,1\}^l \to \{0,1\}^l, \quad \sigma_p(S) \mapsto \left(\sigma_p^{eg} \circ \sigma_p^{ing}\right)(S) \\
\sigma_c, \sigma_c^{eg}, \sigma_c^{ing} &: \{0,1\}^l \to \{0,1\}^l, \quad \sigma_c(S) \mapsto \left(\sigma_c^{eg} \circ \sigma_c^{ing}\right)(S)
\end{aligned} \tag{6.2}$$

Abhängig von der Rolle der Bridge, wird im eingehenden Port die Funktion $\sigma^{ing}$ und im ausgehenden Port die Funktion $\sigma^{eg}$ auf die Signatur der Nachricht angewendet. Diese Trennung der Signaturmodifikation auf unterschiedliche, unabhängige Anwendungsorte hat denselben Grund wie bei FABAN, die Wahrscheinlichkeit für spezielle Fehler der bösartigen Kooperation zwischen ein- und ausgehendem Port zu minimieren. Nichtsdestotrotz kann solch ein Fehler theoretisch auftreten und muss folglich wie bei FABAN zur Ausnahme nicht tolerierbarer Fehler hinzugefügt werden (siehe Kapitel 3.1.4).

Jeder Empfänger der Nachricht, kann mit Hilfe des Hopzählers $h$ bestimmen, wie viele Prechecking Bridges diese Nachricht durchlaufen hat und kann damit die Signaturüberprüfung gemäß des zugrunde gelegten Signaturverfahrens für die empfangene Nachricht $m : S'$ durchführen, nachdem durch

$$\sigma_d^{-1} \circ \sigma_p^{-1} \circ \cdots \circ \sigma_p^{-1} \circ \sigma_c^{-1}(S') := \Omega(S') \tag{6.3}$$

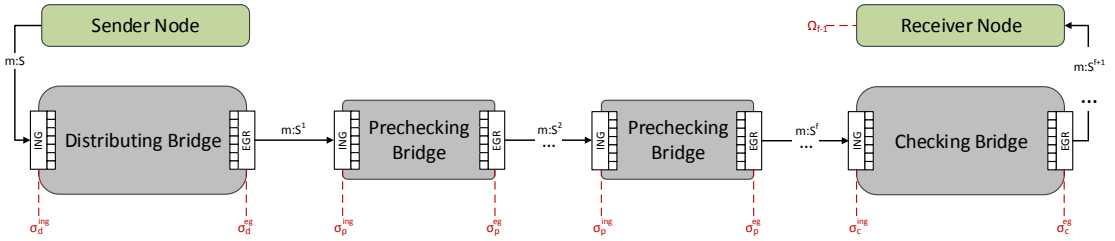



Abbildung 6.5.: Übersicht der ExFABAN-Signaturmodifikation

die ExFABAN-spezifischen Modifikationen rückgängig gemacht wurden. Hierbei ist es nicht nur ausreichend, sondern auch aus Sicherheitsgründen notwendig, dass die Empfänger nicht die einzelnen Funktion $\sigma_d, \sigma_p, \sigma_c$ mitsamt den Umkehrfunktionen $\sigma_d^{-1}, \sigma_p^{-1}, \sigma_c^{-1}$ kennen, sondern tatsächlich nur die Komposition der Umkehrfunktionen wie in (6.3) kennen. Der Anzahl der zu durchlaufenen Prechecking Bridges muss zwingend mindestens $f-1$ betragen, darf aber nach oben abweichen, ist im Allgemeinen aber nach oben beschränkt durch $n_p \geq f-1$. Folglich müssen in solchen Fällen bis zu $1+n_p-(f-1)$ Funktionen

$$\Omega_{f-1}, \ldots, \Omega_{n_p} : \{0,1\}^l \to \{0,1\}^l \tag{6.4}$$

den Empfängern zur Überprüfung der Signatur bereitgestellt werden, so dass eine Überprüfung einer Signatur $S'$ einer empfangenen Nachricht $m : S'$ genau dann erfolgreich ist, wenn mit exakt einer der Funktionen (6.4) die Signaturüberprüfung erfolgreich ist. Dabei muss ein Empfänger die Überprüfung nicht mit allen Funktionen durchführen, sondern kann anhand des Hopzählers $h$ der Nachricht gezielt die korrekte Funktion $\Omega_{h-2}$ zur Überprüfung auswählen. Es ist ausgeschlossen, dass mehrere Funktionen zu einer erfolgreichen Signaturprüfung einer Nachricht führen. Abbildung 6.5 visualisiert die hier beschriebene Signaturmodifikation.

Ein einfaches Beispiel lässt sich als eine Erweiterung der Signaturmodifikation von FABAN konstruieren. Zusätzlich zu den globalen konstanten Masken $d_{mask}$ und $c_{mask}$ wird eine globale Maske $p_{mask}$ gewählt, mit welcher die Signatur im ausgehenden Port XOR-verknüpft wird, nachdem diese im eingehenden Port um einen bit nach links geschoben wurde, d.h.

$$S' = \mathrm{ROL}(S,1) \veebar p_{mask}. \tag{6.5}$$

Die Empfängermaske $r_{mask}$ wird in diesem Fall ersetzt durch eine Menge von Masken $r^i_{mask}$ für $f-1 \leq i \leq n_p$, welche wie folgt vorab berechnet und von den Empfängern zur Laufzeit verwendet werden:

$$r^i_{mask} := \mathrm{ROR}(c_{mask}, i+2) \left[ \bigvee_{j=2}^{i+1} \mathrm{ROR}(p_{mask}, j) \right] \veebar \mathrm{ROR}(d_{mask}, 1) \tag{6.6}$$

Damit ergibt sich die vollständige Signaturmodifikation zu

$$\sigma_d^{ing}(S) = \sigma_p^{ing}(S) = \sigma_c^{ing}(S) = \mathrm{ROL}(S,1) \tag{6.7}$$

$$\begin{aligned} \sigma_d^{eg}(\sigma_d^{ing}(S)) &= \mathrm{ROL}(S,1) \veebar d_{mask} \\ \sigma_p^{eg}(\sigma_p^{ing}(S)) &= \mathrm{ROL}(S,1) \veebar p_{mask} \\ \sigma_c^{eg}(\sigma_c^{ing}(S)) &= \mathrm{ROL}(S,1) \veebar c_{mask} \end{aligned} \tag{6.8}$$

$$\Omega_i(S) = \mathrm{ROR}(S, i+2) \veebar r^i_{mask}, \quad \forall f-1 \leq i \leq n_p. \tag{6.9}$$

Dabei muss jeder Empfänger alle Funktionen in (6.9) kennen, für eine empfangene Nachricht jedoch nur die richtige Funktion der Funktionsmenge, welche am Hopzähler wie zuvor beschrieben zu erkennen ist, verwenden. Fehlerbedingte Verfälschungen des Hopzählers würden vom Empfänger erkannt werden, indem entweder gar keine Funktion zur Überprüfung der Signatur bereit stünde oder aber eine falsche Funktion verwendet werden würde, mit welcher keine erfolgreiche Überprüfung der Nachricht möglich wäre.

### 6.2.3. Protokollspezifikation

Gemäß Definition 2.7 eines vollständigen Kommunikationsprotokolls sind in vorherigem Abschnitt mit Bezügen zu Kapitel 3 bereits *Dienst*, *Annahmen*, *Nachrichtenmenge* sowie *Nachrichtenformat* von ExFABAN klar definiert. Mit steigender Anforderung an den Grad der Fehlertoleranz steigt auch die Komplexität des Nachrichtenroutings eines Protokolls, so dass in Folge dessen der letzte fehlende Aspekt von Definition 2.7, die *Ablaufregeln*, in diesem Abschnitt genauer beschrieben werden.

Vorgestellt wird hier der in Abbildung 6.3 skizzierte Lösungsansatz, welcher ausgearbeitet und hier komplettiert wird. Betrachtet wird stets ein vereinfachter Netzwerkgraph $G = (\mathcal{B}, \mathcal{L})$ eines Netzwerkgraphen $(\mathcal{B}, \mathcal{N}, \mathcal{L})$ mit einem Sender $s \in \mathcal{N}$, sowie der mit dem Sender verbundenen Bridge $d \in \mathcal{B}$, welche die Funktion der Distributing Bridge ausüben muss. Der Einfachheit halber wird im Folgenden o.B.d.A, wie in der restlichen Arbeit, nur der vereinfachte Netzwerkgraph betrachtet. Gemäß der Idee des Lösungsansatzes wird eine Nachricht von $s$ an die Distributing Bridge $d$ gesendet, woraufhin diese über mindestens $f+1$ Mengen von jeweils mindestens $f-1$ Prechecking Bridges an Bridges weitergeleitet werden, welche schlussendlich als Checking Bridges agieren und bis zu $f+1$ unabhängige, redundante Wellen erzeugen, so dass im Fall einer fehlerfreien Distributing Bridge jeder Empfänger $f+1$, im schlimmsten anzunehmenden Fall jedoch mindestens eine Kopie der Nachricht erhält. Die Verteilung der Nachricht bis zu den Checking Bridges wird als *Primäres Routing*, die Verteilung über die Wellen ab den Checking Bridges als *Sekundäres Routing* bezeichnet. Für die Beschreibung des Routings wird eine disjunkte Zerlegung der Menge der Bridges in die Menge der Distributing Bridge, der Menge der

Prechecking Bridges, der Menge der Checking Bridges sowie der Menge der Forwarding Bridges der Form

$$\mathcal{B} = \{d\} \cup [\mathcal{P} \cup \mathcal{C}] \cup \mathcal{F} \tag{6.10}$$

festgelegt, sowie die Menge der *primären Bridges* definiert als

$$\mathcal{B}_p := \{d\} \cup [\mathcal{P} \cup \mathcal{C}] = \mathcal{B} \backslash \mathcal{F}. \tag{6.11}$$

### Primäres Nachrichtenrouting

Das primäre Nachrichtenrouting muss sicherstellen, dass genügend Checking Bridges in $\mathcal{C}$ Wellen initiieren, so dass keine Inkonsistenz bei der Nachrichtenübertragung entstehen kann. Konkret bedeutet dies, dass eine Teilmenge von fehlerfreien Checking Bridges $C_{sub} \subseteq \mathcal{C}$ mit $|C_{sub}| \geq f - f_p + 1$ existieren muss, welche die Nachricht rechtzeitig und unverfälscht erhalten, während in $\mathcal{B}_p$ insgesamt $f_p \in \{0, \ldots, f\}$ Bridges fehlerhaft sind.

Folgendes Theorem formuliert die *Primäre Bedingung* von ExFABAN, welche beschreibt, unter welchen Umständen ein Nachrichtenrouting ein gültiges Primäres Routing für ExFABAN für die Toleranz von $f$ Fehlern darstellt. Dies gewährleistet hierbei allerdings nicht, dass auch ein *Sekundäres Routing* existiert. Darauf wird im nachfolgenden Abschnitt eingegangen. Dabei handelt es sich um eine Weiterentwicklung eines Theorems aus [FE18].

**Theorem 6.1** (Primäre Bedingung)
Seien $G = (\mathcal{B}, \mathcal{L})$ ein vereinfachter Netzwerkgraph, $d \in \mathcal{B}$ eine Distributing Bridge, eine Zerlegung der Menge der Bridges der Form

$$\mathcal{B} = \{d\} \cup [\mathcal{P} \cup \mathcal{C}] \cup \mathcal{F}, \tag{6.12}$$

sowie die Verbindungsmenge $\mathcal{L}_p \subset \mathcal{L}$ gegeben, welche nur Verbindungen zwischen Bridges in $\mathcal{B}_p = \{d\} \cup [\mathcal{P} \cup \mathcal{C}]$ enthält.

Eine Menge von Kantenzügen $Q \subseteq P_{\mathcal{L}_p}(d, \mathcal{C})$ repräsentiert ein gültiges ExFABAN Routing innerhalb $\mathcal{B}_p$ für die Toleranz von $f \geq 1$ Fehlern, mit der Garantie, dass Nachrichten nur bei fehlerhafter Distributing Bridge unterdrückt werden dürfen, genau dann, wenn für alle Checking Bridges $c \in \mathcal{C}$ sowie für alle $p(d, c) = (b_1, \ldots, b_{n-1}) \in Q$ für $n \in \mathbb{N}$ gilt:

$$|p(d, c)| = n \geq f \tag{6.13}$$

$$\forall b \in \bar{p}(d, c) : \ \max P_Q(b, \mathcal{C}) \geq f + 1 - \mathrm{dist}_{p(d,c)}(d, b), \tag{6.14}$$

wobei $\mathrm{dist}_{p(d,c)}$ die Distanz entlang Verbindungen des Kantenzugs $p(d, c)$ sowie der

Ausdruck $\max P_Q(b, \mathcal{C})$ die maximale Anzahl disjunkter Pfade zwischen dem Knoten $b$ sowie einem beliebigen Knoten aus $\mathcal{C}$ entlang Verbindungen in $Q$ bezeichnen.

*Hinweis: Vergleiche Notation mit den Definitionen in* (5.11), (5.13) *und* (5.21).

Abbildung 6.6 visualisiert die Gleichungen (6.13) und (6.14) für $f = 2$.

*Beweis.*
*(Theorem 6.1)* Der Beweis der Äquivalenz erfolgt auf klassischem Wege, indem separat die Richtungen „⇒“ und „⇐“ gezeigt werden.

⇒: Seien für $\mathcal{B}$ eine disjunkte Zerlegung der Menge der Bridges wie in (6.12) sowie mit $Q \subseteq P_{\mathcal{L}_p}$ ein gültiges primäres ExFABAN- Routing innerhalb $\mathcal{B}_p$ gegeben. Im trivialen Fall einer derart fehlerhaften Distributing Bridge, dass keine einzige Checking Bridge eine Nachricht erhält und somit auch keine Welle initiiert wird, entsteht keine Inkonsistenz der Nachrichtenübertragung, so dass folglich nichts zu zeigen ist.

Ohne Beschränkung der Allgemeinheit wird nun angenommen, dass ein fehlerhaftes Verhalten der Distributing Bridge $d$ zu Einschränkungen der Nachrichtenübertragungen des primären Routings (Nachrichten werden fehlerhaft oder nicht übertragen) führt, so dass nur ein einziger Pfad $p_{\mathcal{L}_p}(d, c) \in P_{\mathcal{L}_p}(d, c)$ existiert, über den exakt eine Checking Bridge $c \in \mathcal{C}$ erreicht wird. Eine Beeinträchtigung von $c$ kann nun dazu führen, dass eine Verteilung der Nachricht entlang einer Welle initiiert wird,

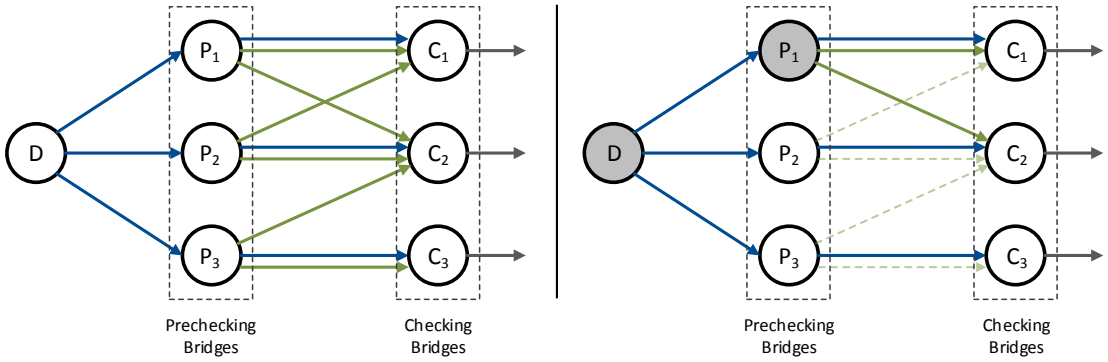



Abbildung 6.6.: Visualisierung der Primären Bedingung von ExFABAN an einem gültigen Beispiel für $f = 2$. Die linke Grafik zeigt an einem vollständigen Beispiel die Routing-Pfade des primären Routings, wobei blaue Pfeile die Pfade von der Distributing Bridge zu den Checking Bridges kennzeichnen und grüne Pfeile die Pfade von den Prechecking Bridges zu den Checking Bridges, d.h. parallele Verbindungen sind die selben physikalischen Verbindungen. Die rechte Grafik verdeutlicht die Bedingung (6.14) von der Distributing Bridge $D$ (blaue Pfeile) sowie der Prechecking Bridge $P_1$ (grüne durchgehende Pfeile).

obwohl nicht alle Empfänger diese Nachricht rechtzeitig erhalten werden. Um dieses Problem zu beheben, müssen $f-1$ zusätzliche Zeit-Überprüfungen in Prechecking Bridges zwischen Distributing Bridge und Checking Bridges stattfinden. Folglich muss der Kantenzug von der Distributing Bridge $d$ über die Prechecking Bridges bis zur Checking Bridge $c$ die Mindestlänge $f$ aufweisen, was (6.13) zeigt.

Ist die Fehlfunktion in $c$ beliebig, kann dies ebenfalls zu inkonsistenter Nachrichtenverteilung führen. Betrachtet man nun eine Bridge $b$ eines beliebigen Kantenzugs $p(d,c) \in Q$ und unterteilt diesen Kantenzug in zwei Teile

$$p(d,c) = p(d,b) \cup p(b,c), \tag{6.15}$$

so können im schlimmsten Fehlerfall bereits alle Bridges vor $b$ in $p(d,b)$ fehlerhaft sein, so dass $b$ die einzige Bridge ist, welche noch eine fehlerfreie Nachricht zu einer Menge von Checking Bridges weiterleiten kann. Aus diesem Grund muss sichergestellt werden, dass noch mindestens

$$f+1-|p(d,b)| = f+1-\mathrm{dist}_{p(d,c)}(d,b) \tag{6.16}$$

Checking Bridges ausgehend von $b$ erreicht werden, damit im schlimmsten anzunehmenden Fall von weiteren $f-|p(d,b)|$ Fehlern noch mindestens eine fehlerfreie Checking Bridge erreicht wird und damit eine sichere Verteilung der Nachricht gewährleistet ist. Die Garantie, dass bei fehlerfreier Distributing Bridge keine vollständige Unterdrückung der Nachricht entsteht und genügend fehlerfreie Checking Bridges Wellen initiieren, zeigt schließlich (6.14) und beweist die hinführende Richtung für den nicht-trivialen Fall.

$\Leftarrow$: Um die Gegenrichtung zu zeigen, wird zunächst die Distributing Bridge $d$ betrachtet. Für $b = d$ folgt mit (6.13) und (6.14), was

$$\max P_{p(d,c)}(d,\mathcal{C}) \geq f+1-\mathrm{dist}_{p(d,c)}(d,d) = f+1 \tag{6.17}$$

impliziert, dass $f+1$ disjunkte Pfade mit einer Länge $\geq f$ von $d$ zu paarweise verschiedenen Checking Bridges existieren.

Nimm nun an, dass $b \in \mathcal{B}_p$ die einzige Bridge in $\mathcal{B}_p$ ist mit

$$\mathrm{dist}_{p(d,c)}(d,b) = f_p \in \{1,\dots,f\}, \tag{6.18}$$

welche eine fehlerfreie Nachricht von $d$ erhält. Wäre nun mindestens eine Bridge auf dem Pfad von $d$ nach $b$ oder $d$ selbst fehlerfrei, so würde nach (6.14) mindestens eine weitere Bridge $b' \in \mathcal{B}_p$ mit $\mathrm{dist}_{p(d,c)}(d,b) = f_p$ existieren, was ein Widerspruch zur Annahme wäre. Folglich müssen alle Bridges in $p(d,b)$ sowie die Distributing Bridge $d$ selbst fehlerhaft sein. Nach (6.14) ist

$$\max P_Q(b,\mathcal{C}) \geq f+1-\mathrm{dist}_{p(d,c)}(d,b) = f+1-f_p, \tag{6.19}$$

das heißt, es existieren gemäß (6.13) mindestens $f+1-f_p$ disjunkte Pfade der Länge $f-f_p$ zu verschiedenen Checking Bridges.

Damit folgt, dass es trotz bis zu $f$ Fehlern garantiert ist, dass mindestens $f+1-f_p$ Checking Bridges eine fehlerfreie Nachricht rechtzeitig erhalten und somit fehlertolerantes Verhalten erreicht wird. Damit ist die Rückrichtung bewiesen.

□

**Bemerkung 6.2**
Die Bedingung in (6.14) ist sehr stark, allerdings notwendig um zu gewährleisten, dass eine Unterdrückung einer Nachricht nur im Fall einer fehlerhaften Distributing Bridge auftreten darf. Dies zusätzlich zu verhindern ist nicht ohne Weiteres möglich und erfordert die redundante Verbindung von Rechenkomponenten zu mehreren Bridges. Eine Abschwächung der Bedingung (6.14) ist möglich, hätte aber zur Folge, dass auch andere Fehlerkonstellationen dazu führen könnten, dass eine Nachricht unterdrückt wird.

Fehlerhafte Bridges in $\mathcal{B}_p$ können verhindern, dass einige fehlerfreie Checking Bridges eine unverfälschte und nicht übermäßig verzögerte Nachricht erhalten. Im Folgenden ist es notwendig formal quantifizieren zu können, wie viele Wellen im fehlerfreien, aber auch im nicht fehlerfreien Fall mindestens initiiert werden. Dazu sei nun folgende abstrakte Funktion definiert

$$\xi : \wp(\mathcal{B}_p) \to \wp(\wp(\mathcal{C})), \tag{6.20}$$

welche eine Menge von fehlerhaften Bridges in $\mathcal{B}_p$ auf die Menge aller Mengen von fehlerfreien Checking Bridges abbildet, welche eine zu verteilende Nachricht unverfälscht und rechtzeitig erhalten. Weiterhin wird für $f_p$ fehlerhafte Bridges in $\mathcal{B}_p$, $0 \leq f_p \leq f$, die Funktion

$$\begin{aligned} \Xi : \{0,\ldots,f\} &\longrightarrow \wp(\wp(\mathcal{C})) \\ f_p &\longmapsto \bigcup_{b_1,\ldots,b_{f_p} \in \mathcal{B}_p} \xi\left(\{b_1,\ldots,b_{f_p}\}\right) \end{aligned} \tag{6.21}$$

definiert, welche eine Anzahl fehlerhafter Bridges in $\mathcal{B}_p$ auf alle möglichen Mengen von fehlerfreien Checking Bridges abbildet, welche eine unverfälschte Nachricht rechtzeitig erhalten könnten. Bemerke, dass die Menge der fehlerfreien Checking Bridges, welche eine unverfälschte Nachricht erhalten, von konkreten Fehlertypen und Fehlerorten der fehlerhaften Bridges abhängt. Zusätzlich können gleiche Fehlertypen und Fehlerorte trotzdem zu indeterministischem Verhalten führen. Diese Möglichkeiten sind in $\xi$ sowie $\Xi$ bedacht.

Theorem 6.1 besagt, dass entweder keine Checking Bridge Wellen initiiert (trivialer Fall ≡ vollständig unterdrückende Distributing Bridge) oder mehr Checking Bridges

Wellen initiieren, als die Anzahl möglicher Fehler im sekundären Routing beträgt, so dass unter diesen Wellen mindestens eine fehlerfrei ist (nicht trivialer Fall). Mit $f_p$ Fehlern im primären Routing folgt damit für jedes $C \in \Xi(f_p)$, dass

$$|C| \geq f + 1 - f_p \tag{6.22}$$

im nicht-trivialen Fall und $|C| = 0$ im trivialen Fall gilt. Im nicht-trivialen Fall ist die Größe

$$\min \Xi(f_p) := \min \{|X| \mid X \in \Xi(f_p)\} \tag{6.23}$$

ein Maß für die minimale Anzahl von Wellen bei $0 \leq f_p \leq f$ Fehlern in $\mathcal{B}_p$, welche garantiert initiiert werden.

### Sekundäres Nachrichtenrouting

Nachdem eine Nachricht, durch die Verteilung entlang des *primären Routings*, einer Menge von Checking Bridges zugestellt wurde, muss diese Nachricht sicher zu allen Empfängern $b \in \mathcal{B}$ weitergeleitet werden. In dem Fall von $f = f_p + f_s$ Fehlern, wobei $f_p$ die Fehler in $\mathcal{B}_p$ und $f_s$ die Fehler in $\mathcal{B} \backslash \mathcal{B}_p = \mathcal{F}$ bezeichnet, garantiert die primäre Bedingung, dass mindestens $f_s + 1$ fehlerfreie Checking Bridges erreicht werden.

Die Idee des sekundären Routings basiert auf der Notwendigkeit, die Nachricht entlang $f_s + 1$ paarweiser disjunkter Pfade zu allen Empfängern zu verschicken. Während Bridges in $\mathcal{F}$ beim *primären Routing* nicht benutzt wurden und somit im *sekundären Routing* unproblematisch zur Weiterleitung von Nachrichten benutzt werden können, trifft dies auf Bridges in $\mathcal{B}_p$ im Allgemeinen nicht zu. Ist eine Bridge in $\mathcal{B}_p$ fehlerhaft, hat dies zur Folge, dass weniger Wellen initiiert werden können. Sollte eine Welle zur Verbreitung der Nachricht diese fehlerhafte Bridge benutzen, würde dieser Einzelfehler doppelten Einfluss auf eine Nachricht haben und somit genauso viel Gewicht wie zwei Fehler in unterschiedlichen Bridges haben, wodurch die gezielte Fehlertoleranz nicht mehr garantiert werden könnte. Dies erhöht im Allgemeinen die Komplexität der Aufgabe, ein geeignetes *sekundäres Routing* zu finden.

Ein einfacher und funktionierender Ansatz besteht darin, das *sekundäre Routing* auf Bridges in $\mathcal{F}$ zu limitieren, d.h. nur Bridges in $\mathcal{F}$ dürfen Nachrichten an andere Bridges weiterleiten, während Bridges in $\mathcal{B}_p$ in der Phase des *sekundäre Routings* Nachrichten lediglich zu verbundenen Rechenkomponenten zustellen dürfen. Dadurch wird die Menge der verwendbaren Verbindungen für das Routing in dieser zweiten Phase begrenzt und somit grundlegend die Suche nach geeigneten Pfaden erschwert, mehrfach Einflussnehmende Einzelfehler allerdings ausgeschlossen. Dieser Ansatz führt zu folgendem Theorem, welches als *sekundäre Bedingung* bezeichnet wird.

**Theorem 6.3** (Sekundäre Bedingung)
Seien $G = (\mathcal{B}, \mathcal{L})$ ein vereinfachter Netzwerkgraph, $d \in \mathcal{B}$ eine Distributing Bridge,

$$\mathcal{B} = \{d\} \cup \mathcal{P} \cup \mathcal{C} \cup \mathcal{F} = \mathcal{B}_p \cup \mathcal{F}. \tag{6.24}$$

eine gegebene disjunkte Zerlegung der Menge der Bridges, sowie die Annahme, dass die *primäre Bedingung* erfüllt ist für $f \geq 1$. Eine Menge $\mathcal{W} = \{W_1, \ldots, W_n\}$ von Mengen von Verbindungen $W_i \subset \mathcal{L}$, initiiert von einer Menge von Checking Bridges $C = \{c_1, \ldots, c_n\}$ für $n \geq f + 1$, repräsentiert eine gültige Menge von Wellen für das *sekundäre Routing* mit Fehlertoleranzgrad $f$, falls die folgenden Bedingungen erfüllt sind:

(i) *(Eingeschränkte Konnektivität)*
Für alle $1 \leq i \leq n$ gilt:

$$\forall (b_k, b_l) \in W_i : \ (b_k, b_l) \in (\{c_i\} \cup \mathcal{F}) \times \mathcal{B} \tag{6.25}$$

(ii) *(Redundante Erreichbarkeit)*
Für jede Bridge $b \in \mathcal{B}$ existiert eine Teilmenge $\{i_1, \ldots, i_{f+1}\} \subseteq \{1, \ldots, n\}$, so dass die Wellen $W_{i_1}, \ldots, W_{i_{f+1}}$ insgesamt $f + 1$ disjunkte Pfade zwischen der jeweiligen Checking Bridge und Bridge $b$ bereitstellen, d.h.

$$\forall 1 \leq k < l \leq f + 1 : \ p_{i_k} \cap p_{i_l} = \varnothing, \tag{6.26}$$

wobei $p_{i_j} \in P_{W_{i_j}}(c_{i_j}, b),\ 1 \leq j \leq f + 1$, Pfade von der Checking Bridge $c_{i_j}$ zu Bridge $b$ entlang der Welle $W_{i_j}$ bezeichnen.

*Hinweis: Vergleiche Notation mit der Definition in* (5.19).

Dieser Ansatz ist eine direkte Erweiterung der Bedingungen aus Theorem 5.9 für den Fall $f = 1$, laut welchem es nicht erlaubt ist, eine Checking Bridge der anderen Welle für die Verteilung einer Nachricht zu verwenden. Ein wichtiger Unterschied liegt darin, dass dieses Resultat lediglich eine Folgerung und keine Äquivalenz darstellt, das heißt, dass es sich um eine hinreichende, aber nicht notwendige Bedingung handelt. Dies erlaubt es, zusätzliche Flexibilität bei der Konstruktion der Wellen auszunutzen, was einerseits die Anzahl der Nachrichtenübertragungen erhöhen, allgemeine Kosten allerdings senken kann. Im Allgemeinen muss dies mit größter Vorsicht geschehen, da das mehrfache Benutzen von Bridges die Komplexität des Problems drastisch erhöhen kann. Im nachfolgenden Kapitel wird in speziellen Graphenklassen ein sekundäres Routing vorgestellt, welches 5.9 verletzt.

*Beweis (von Theorem 6.3).* Sei $\mathcal{W} = \{W_1, \ldots, W_n\}$, $W_i \subset \mathcal{L}$ für $1 \leq i \leq n$, eine Menge von

Wellen, welche die Bedingungen (6.25) und (6.26) erfüllen. Weiterhin sei die primäre Bedingung erfüllt für $f \geq 1$, das heißt dass bei $1 \leq f_p \leq f$ Fehlern in $\mathcal{B}_p$,

$$\min \Xi(f_p) \geq f - f_p + 1 \tag{6.27}$$

gilt und somit $f - f_p + 1$ fehlerfreie Checking Bridges eine unverfälschte Nachricht rechtzeitig erhalten und Wellen initiieren. Während $f - f_p$ weitere Fehler zu Beeinträchtigungen der Wellen führen können, ist dies für die $f_p$ Fehler in $\mathcal{B}_p$ nicht der Fall. Mit (6.25) und (6.26) folgt schließlich, dass mindestens

$$\min \Xi(f_p) - (f - f_p) \geq 1 \tag{6.28}$$

Pfade nicht von Fehlern betroffen sind und somit die Behauptung gezeigt ist. □

Beispiele für das *sekundäre Routing* werden im Folgenden im Rahmen der Untersuchung spezieller Graphenklassen vorgestellt. Losgelöste Beispiele lassen sich nicht nur schwer konstruieren, sondern sind im Allgemeinen für $f > 1$ auch schlecht darstellbar.

## 6.3. Topologiebasierter Ansatz zur Realisierung komplexer Redundanz für Mehrfachfehler

In diesem Teil der Arbeit werden topologiebasierte Ansätze zur Anwendung des Protokolls ExFABAN auf realen Systemen untersucht, das heißt es werden Klassen von Graphen entwickelt, die für die Anwendung von ExFABAN geeignet sind und für unterschiedliche Anforderungen an die Fehlertoleranz skaliert werden können.

Die nachfolgenden Unterkapitel behandeln die Klassen der *vollvermaschten Graphen*, der *mehrschichtigen Ringe* sowie der *redundanten Sterntopologien*. Abschließend werden die Klassen durch die Gegenüberstellung relevanter Größen miteinander verglichen.

### 6.3.1. Vollvermaschte Graphen

Vollvermaschte Graphen bieten das größtmögliche Maß an Verbindungsredundanz, so dass folglich auf vollvermaschten Graphen die Anwendung von ExFABAN am einfachsten ist. Folgendes Theorem fasst die Möglichkeiten der Anwendung von ExFABAN auf vollvermaschten Graphen zusammen:

**Theorem 6.4**
ExFABAN kann auf einem vollvermaschten Graphen $G_V(N) = (\mathcal{B}, \mathcal{L})$ mit $N \in \mathbb{N}$ Bridges genau dann bis zu $f \in \mathbb{N}$ Fehler tolerieren, wenn $N \geq 1 + f \cdot (f+1)$ gilt.

*Beweis.* Ausgehend von einer beliebigen Distributing Bridge $d \in \mathcal{B}$ müssen laut Theorem 6.1 mindestens $f+1$ disjunkte Pfade zu $f+1$ verschiedenen Checking Bridges existieren, wobei jede Checking Bridge eine Mindestdistanz zur Distributing Bridge von $f$ hat. Damit ergibt sich die Mindestanzahl von Bridges zu $N \geq 1 + f \cdot (f+1)$.

Umgekehrt lassen sich in einem vollvermaschten Graphen der Mindestgröße von $N \geq 1 + f \cdot (f+1)$ Bridges die in Theorem 6.1 geforderten disjunkten Pfade zu $f+1$ Checking Bridges stets finden, so dass von diesen aus direkt die Nachricht an alle anderen Bridges übertragen werden kann. □

### 6.3.2. Mehrschichtige Ringe

Die kleinste Klasse von Netzwerkgraphen mit zwei redundanten Pfaden zwischen jedem Knotenpaar, ist die Klasse der Ringgraphen, welche bereits in Kapitel 5.3.1 für die Toleranz von einem Fehler mittels FABAN detailliert untersucht und der Klasse der vollvermaschten Netze gegenübergestellt wurde.

Ringe haben den großen Vorteil, dass sie durch geeignete Verbindungen einzelner Ringe gleicher oder verschiedener Größen, zu größeren Graphen, den so genannten *Ringnetzen*, zusammengesetzt werden können. Hierbei kann ein Ring mit einem anderen Ring oder Ringnetz verschmolzen werden, indem eine beliebige Verbindung zwischen zwei Knoten $a$ und $b$ des hinzuzufügenden Rings entfernt wird und die Knoten $a$ und $b$ mit zwei beliebigen benachbarten Knoten des anderen Rings bzw. des Ringnetzes verbunden werden. Abbildung 6.7 zeigt ein Beispiel eines Ringnetzes, welches aus insgesamt sieben Ringen unterschiedlicher Größen zusammengesetzt ist. Insbesondere ist hervorzuheben, dass für Ringnetze dieser Art immer ein geeignetes FABAN Routing für jede Bridge als Distributing Bridge für die Toleranz eines Fehler gefunden werden kann. Dies lässt sich darauf zurückführen, dass ein Ringnetz stets einen einzelnen Ring, der durch alle Bridges führt, als Teilgraphen enthält.

Die in Kapitel 5.3.2 diskutierten erweiterten Ringe stellen einen Spezialfall von Ringnetzen dar. Eine Erweiterung eines Rings um $k$ Verbindungen in möglichst äquidistanten Abständen (vgl. Abbildung 5.9) ist ein Ringnetz, bestehend aus einem zentralen Ring mit $k$ Knoten sowie $k$ damit an paarweise benachbarten Knoten des zentralen Rings verbundenen Nebenringen, welche sich in der Größe höchstens um 1 voneinander unterscheiden.

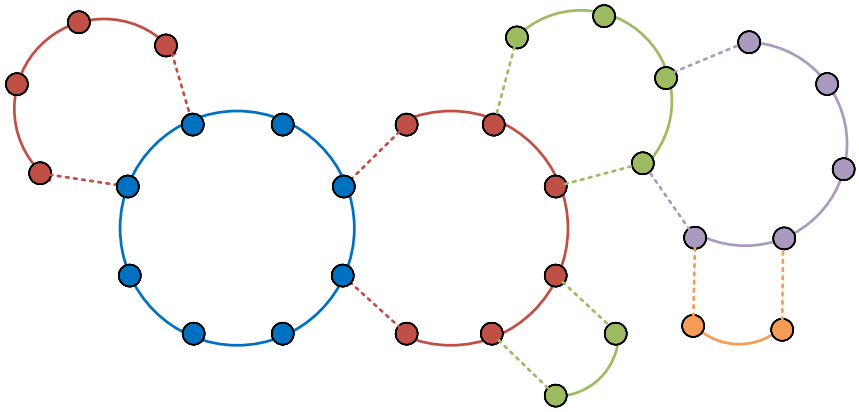


ABBILDUNG 6.7.: Beispiel eines Ringnetzes, zusammengesetzt aus sieben Ringen teilweise unterschiedlicher Größen. Gestrichelte Verbindungen symbolisieren die Verbindungsstellen zwischen einzelnen Ringen.

Für die Toleranz von Mehrfachfehlern mit ExFABAN bieten einfache Ringe sowie einfache Ringnetze nicht genug Redundanz. Die Grundidee ist es, die Struktur von Ringen sowie Ringnetzen beizubehalten und gleichzeitig um Redundanz wie folgt zu erweitern: Ausgehend von einem Ring der Größe $k \in \mathbb{N}$ mit den Knoten $b_0^{(1)}, \ldots, b_{k-1}^{(1)}$, wird eine zweite Schicht des Rings durch Duplikation des ursprünglichen Rings, mit den Knoten $b_0^{(2)}, \ldots, b_{k-1}^{(2)}$ erzeugt und mit dem ursprünglichen Ring verbunden, indem jeweils eine Verbindung zwischen $b_i^{(1)}$ und $b_i^{(2)}$ für $0 \leq i \leq k-1$ hinzugefügt wird. Das Resultat wird als *zweischichtiger Ring* der Größe $k$ bezeichnet. Sukzessive können auf diese Weise *$m$-schichtige Ringe* konstruiert werden. Folgende Definition fasst dies formal zusammen:

**Definition 6.5** (Mehrschichtiger Ring)
Ein Graph $G_R(k,n) = (\mathcal{B}, \mathcal{L})$ mit $k, n \in \mathbb{N}$ heißt *$n$-schichtiger Ring* genau dann, wenn die Menge der Knoten $\mathcal{B}$ von der Form

$$\mathcal{B} = \left\{ b_i^{(j)} \;\middle|\; 0 \leq i \leq k-1 \;\wedge 1 \leq j \leq n \right\} \tag{6.29}$$

ist und die Menge der Verbindungen eine Permutation von

$$\tilde{\mathcal{L}} := \left\{ \left( b_i^{(j)}, b_{i'}^{(j)} \right) \;\middle|\; 1 \leq j \leq n \;\wedge\; i' = (i+1) \mod k \right\} \cup \tag{6.30}$$

$$\left\{ \left( b_i^{(j)}, b_i^{(j')} \right) \;\middle|\; 1 \leq j, j' \leq n \;\wedge\; j \neq j' \;\wedge\; 0 \leq i \leq k-1 \right\} \tag{6.31}$$

ist.

Die Menge in (6.30) beschreibt in obiger Definition die Verbindungen entlang jeder Schicht der mehrschichtigen Rings, während die Menge in (6.31) die Verbindungen

zwischen den Ringschichten formuliert. Abbildung 6.8 zeigt Beispiele von je einem ein-, zwei- und dreischichtigen Ring.

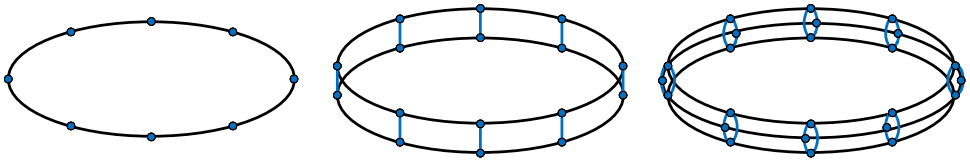


Abbildung 6.8.: Mehrschichtige Ringe für 1, 2 und 3 Schichten.

Analog zu den zu Beginn dieses Abschnittes eingeführten Ringnetzen, welche aus einfachen Ringen bestehen, können *mehrschichte Ringnetze* konstruiert werden, wobei die Verbindung zweier mehrschichtiger Ringe bzw. Ringnetze auf allen Schichten der entsprechenden benachbarten Knoten geschehen muss. Abbildung 6.9 zeigt ein Beispiel eines dreischichtigen Ringnetzes, welches aus 5 dreischichtigen Ringen gleicher Größe zusammengesetzt ist.

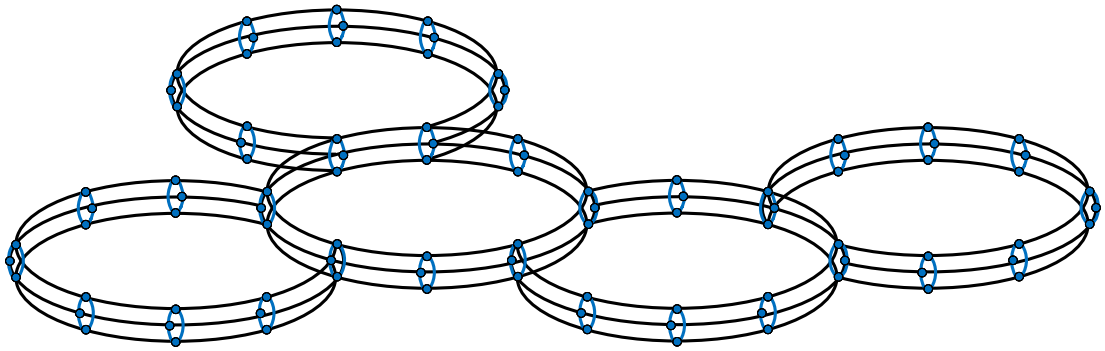


Abbildung 6.9.: Beispiel eines dreischichtigen Ringnetzes.

Die Klasse der mehrschichtigen Ringe bzw. Ringnetze stellt, analog zu den Ringen für die Toleranz *eines* Fehlers, die einfachste Klasse von Graphen dar, auf welchen ExFABAN Anwendung finden kann. Der Grad der möglichen Fehlertoleranz sowie eine Anforderung an die Größe mehrschichtiger Ringe wird in folgendem Lemma formuliert und bewiesen:

**Lemma 6.6**
ExFABAN kann auf einem mehrschichtigen Ring $G_R(k,n)$ für $n \in \mathbb{N}$ bis zu $f = n$ Fehler tolerieren, falls $k \geq 1 + 2f$ gilt.

*Beweis.* Gemäß Theorem 6.1 muss die Distanz zwischen der Distributing Bridge sowie jeder Checking Bridge höher sein als $f$, d.h.

$$\mathrm{dist}_{\mathcal{L}_p}(d, \mathcal{C}) \geq f. \tag{6.32}$$

Eine Möglichkeit für das primäre Routing auf mehrschichtigen Ringen ist die Weiterleitung der Nachrichten in beide Richtungen auf allen Schichten, was mindestens $1+2f$ Bridges pro Schicht voraussetzt. Das Routing im Detail wird im Nachfolgenden angegeben, bewiesen und anhand eines Beispiels visualisiert, wodurch sich die Behauptung ergeben wird. □

Alle weiteren Bridges in größeren mehrschichtigen Ringen handeln einfach als Forwarding Bridge. Im Beispiel in Abbildung 6.8 mit 1, 2 sowie 3 Schichten (und somit $f = 1$, $f = 2$ bzw. $f = 3$) würden 5, 6 sowie 3 Bridges als Forwarding Bridges handeln, unabhängig davon, welche Bridge die Distributing Bridge ist.

**Bemerkung 6.7**
Unmittelbar aus Lemma 6.6 folgt, dass die Voraussetzung für die Toleranz von $f$ Fehlern auf einem $n$-schichtigen Ringnetz darin besteht, dass $f \leq n$ gilt sowie jede Schicht des Ringnetzes mindestens $1+2f$ Bridges enthält.

Bevor in nachfolgendem Theorem das primäre Routing spezifiziert wird, werden zunächst offene, geschlossene sowie halboffene Intervalle auf dem Restklassenring

$$\mathbb{Z}_k := \{0, \ldots, k-1\} \tag{6.33}$$

für $k \in \mathbb{N}$ für $a, b \in \mathbb{Z}_k$ definiert:

$$i \in (a,b) :\Leftrightarrow 0 < [(i-a) \mod k] < [(b-a) \mod k] \tag{6.34}$$
$$i \in (a,b] :\Leftrightarrow 0 < [(i-a) \mod k] \leq [(b-a) \mod k] \tag{6.35}$$
$$i \in [a,b) :\Leftrightarrow 0 \leq [(i-a) \mod k] < [(b-a) \mod k] \tag{6.36}$$
$$i \in [a,b] :\Leftrightarrow 0 \leq [(i-a) \mod k] \leq [(b-a) \mod k] \tag{6.37}$$

Zuletzt sei noch erwähnt, dass Addition und Subtraktion von Indizes in $\mathbb{Z}_k$ ebenfalls den Rechenregeln des Restklassenrings unterliegen, das heißt die Ergebnisse werden stets modulo $k$ gerechnet. Beachte außerdem, dass $[a,b]$ komplementär zu $(b,a)$ ist, d.h. insbesondere ist $[a,b] \cup (b,a) = \mathbb{Z}_k$ für beliebige $a, b \in \mathbb{Z}_k$.

**Theorem 6.8** (Primäres Routing mehrschichtiger Ringe)
Seien $G_R(k,n) = (\mathcal{B}, \mathcal{L})$ ein mehrschichtiger Ring mit $k \geq 1+2n$ und $n \in \mathbb{N}$ sowie $d := b_\lambda^{(\mu)} \in \mathcal{B}$ eine Distributing Bridge für beliebige $0 \leq \lambda \leq k-1$ und $1 \leq \mu \leq n$. Dann ist die Menge von Verbindungen $\mathcal{L}_p \subset \mathcal{L}$, definiert durch

$$\mathcal{L}_p := \mathcal{L}_p^{(\Updownarrow)} \cup \mathcal{L}_p^{(\leftarrow)} \cup \mathcal{L}_p^{(\rightarrow)} \tag{6.38}$$

mit

$$\mathcal{L}_p^{(\Updownarrow)} := \left\{ \left( b_i^{(j)}, b_i^{(j')} \right) \;\middle|\; \begin{array}{c} i \in [\lambda - f, \lambda + f] \backslash \{\lambda\} \wedge j \neq j' \wedge 1 \leq j, j' \leq n \\ \vee \;\; i = \lambda \wedge j = \mu \wedge 1 \leq j' \leq n \end{array} \right\} \tag{6.39}$$

$$\mathcal{L}_p^{(\leftarrow)} := \left\{ \left( b_i^{(j)}, b_{i-1}^{(j)} \right) \;\middle|\; i \in (\lambda - f, \lambda] \;\wedge\; 1 \leq j \leq n) \right\} \tag{6.40}$$

$$\mathcal{L}_p^{(\rightarrow)} := \left\{ \left( b_i^{(j)}, b_{i+1}^{(j)} \right) \;\middle|\; i \in [\lambda, \lambda + f) \;\wedge\; 1 \leq j \leq n) \right\} \tag{6.41}$$

ein gültiges primäres ExFABAN Routing für die Toleranz von bis zu $f = n$ Fehlern.

*Beweis.*
Um die Behauptung zu beweisen, muss nachgewiesen werden, dass sowohl (6.13) als auch (6.14) erfüllt sind.

Betrachte zunächst ohne Beschränkung der Allgemeinheit eine Bridge $b_i^{(j)}$ mit $i \in (\lambda - f, \lambda]$ und $1 \leq j \leq n$. Nach Konstruktion ist die Entfernung von $b_i^{(j)}$ zu der näher liegenden Checking Bridge derselben Schicht exakt $i - (\lambda - f)$, so dass sich die Abschätzung

$$\mathrm{dist}_{\mathcal{L}_p}\left(b_i^{(j)}, \mathcal{C}\right) \geq i - (\lambda - f) = f - (\lambda - i) \geq f - \mathrm{dist}_{\mathcal{L}_p}\left(d, b_i^{(j)}\right) \tag{6.42}$$

ergibt. Aufgrund des symmetrischen Aufbaus der primären Bridges folgt mit analoger Argumentation für $b_i^{(j)}$ mit $i \in (\lambda + f, \lambda]$ Eigenschaft (6.13).

Für die zweite Eigenschaft werden zwei Fälle differenziert betrachtet. Betrachte zunächst die Distributing Bridge. Entlang der „vertikalen“ Verbindungen aus $\mathcal{L}_p^{(\Updownarrow)}$ empfangen $n-1$ Prechecking Bridges die Nachricht direkt von der Distributing Bridge und leiten diese dann entlang von $f-1$ disjunkten Pfaden zu $f-1$ unterschiedlichen Checking Bridges entweder auf der linken oder der rechten Seite weiter. Weiterhin existieren zwei weitere Checking Bridges auf der Schicht, auf welcher sich die Distributing Bridge befindet, sowie folglich zwei zueinander disjunkte Pfade zu diesen beiden Checking Bridges. Diese sind zudem paarweise disjunkt zu den Pfaden der ersten $f-1$ Checking Bridges, so dass insgesamt $f+1$ paarweise disjunkte Pfade zu verschiedenen Checking Bridges existieren und somit (6.13) für $d$ gezeigt wurde.

Betrachtet man nun eine beliebige andere Bridge $b_i^{(j)}$ mit $i \in (\lambda - f, \lambda]$ und $1 \leq j \leq n$, dann können über alle $n - 1 = f - 1$ benachbarten Bridges der anderen Schichten mindestens $f$ Checking Bridges entlang disjunkter Pfade erreicht werden. Damit folgt mit der Abschätzung $\mathrm{dist}_{\mathcal{L}_p}(d, b_i^{(j)}) \geq 1$:

$$\max P_{Links_p}\left(b_i^{(j)}, \mathcal{C}\right) \geq f \geq f + 1 - \mathrm{dist}_{\mathcal{L}_p}\left(d, b_i^{(j)}\right), \tag{6.43}$$

was schlussendlich (6.14) zeigt. □

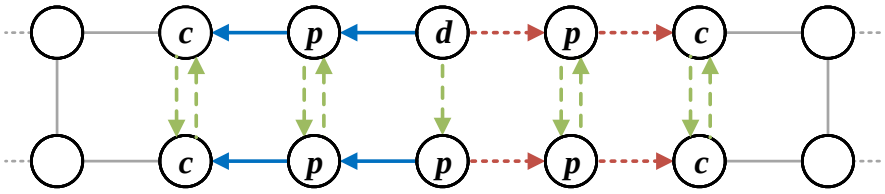



Abbildung 6.10.: Primäres ExFABAN Routing auf einem zweischichtigen Ring für die Toleranz von bis zu $f = 2$ Fehlern. $\mathcal{L}_p^{(\updownarrow)}$ sind hierbei mit vertikalen (grünen) gestrichelten, $\mathcal{L}_p^{(\leftarrow)}$ mit horizontalen (blauen) durchgehenden und $\mathcal{L}_p^{(\rightarrow)}$ mit horizontalen (roten) gestrichelten Pfeilen dargestellt.

Abbildung 6.10 visualisiert das Primäre Routing für ExFABAN für die Toleranz von $f = 2$ Fehlern. Die Beschreibung des Routings wird mit folgendem Theorem durch die Beschreibung des sekundären Routings komplettiert.

**Theorem 6.9** (Sekundäres Routing mehrschichtiger Ringe)
Seien $G_R(k,n) = (\mathcal{B},\mathcal{L})$ ein mehrschichtiger Ring mit $k \geq 1+2n$ und $n \in \mathbb{N}$ sowie $d := b_\lambda^{(\mu)} \in \mathcal{B}$ eine Distributing Bridge für beliebige $0 \leq \lambda \leq k-1$ und $1 \leq \mu \leq n$. Seien weiterhin $f = n$, $\mathcal{C}$ die Menge der Checking Bridges definiert durch

$$\mathcal{C} := \left\{ b_{\lambda \pm f}^{(i)} \;\middle|\; 1 \leq i \leq n \right\}, \tag{6.44}$$

sowie die Menge der Wellen, initiiert durch die Checking Bridges, bezeichnet durch

$$\mathcal{W} := \left\{ W\left(b_{\lambda \pm f}^{(i)}\right) \;\middle|\; 1 \leq i \leq n \right\}. \tag{6.45}$$

Die Menge der Wellen $\mathcal{W}$, definiert durch

$$\begin{aligned}
W\left(b_{\lambda \pm f}^{(i)}\right) := &\left\{ \left(b_j^{(i)}, b_{j\pm1}^{(i)}\right) \;\middle|\; 1 \leq j \leq k-1 \;\wedge\; j \neq (\lambda + f \mp 1) \right\} \cup && (6.46)\\
&\left\{ \left(b_{j\pm1}^{(i)}, b_j^{(i)}\right) \;\middle|\; j \in (\lambda - f, \lambda + f) \right\} \cup && (6.47)\\
&\left\{ \left(b_j^{(i)}, b_j^{(i')}\right) \;\middle|\; 1 \leq j \leq k-1 \;\wedge\; i' \in \{1,\dots,n\} \backslash \{i\} \right\} \cup && (6.48)\\
&\left\{ \left(b_j^{(i')}, b_\lambda^{(i')}\right) \;\middle|\; j \in \{\lambda - 1, \lambda + 1\} \;\wedge\; i' \in \{1,\dots,n\} \backslash \{\mu\} \right\} && (6.49)
\end{aligned}$$

repräsentiert eine gültige Menge von Wellen für das sekundäre ExFABAN Routing für die Toleranz von bis zu $f$ Fehlern. Bemerke, dass das Vorzeichen von $\pm 1$ in (6.46) sowie (6.47) durch das Vorzeichen von $\pm f$ im Index der Checking Bridge $b_{\lambda+f}^{(i)}$ oder $b_{\lambda-f}^{(i)}$, welche die Welle initiiert, bestimmt wird.

Abbildung 6.11 visualisiert eine Welle des sekundären ExFABAN Routings für den Fall $f = 2$.

*Beweis.*
Der Beweis erfolgt durch Fallunterscheidung:

*Fall 1:* Angenommen, die Distributing Bridge ist fehlerfrei und auf jeder Schicht des Ringnetzes ist exakt eine Bridge fehlerhaft, das heißt

$$\mathcal{B}_{fh} := \left\{ b_{j_1}^{(1)}, \ldots, b_{j_n}^{(n)} \right\} \tag{6.50}$$

für bestimmte $j_1, \ldots, j_n \in \{0, \ldots, k-1\}$. Betrachte nun für $1 \leq i \leq n$ die fehlerhafte Bridge $b_{j_i}^{(i)}$ auf Schicht $i$. Ist $b_{j_i}^{(i)} \in \mathcal{B}_p$, dann gilt

$$b_{\lambda-1}^{(i)} \in \xi\left(\mathcal{B}_p \cap \mathcal{B}_{fh}\right) \ \vee \ b_{\lambda+1}^{(i)} \in \xi\left(\mathcal{B}_p \cap \mathcal{B}_{fh}\right) \tag{6.51}$$

und mit 6.46 und 6.47 folgt, dass jede fehlerfreie Bridge auf Schicht $i$ eine fehlerfreie Kopie der gesendeten Nachricht innerhalb der erlaubten Zeit empfängt. Ist andererseits $b_{j_i}^{(i)} \in \mathcal{B} \backslash \mathcal{B}_p$, das heißt eine Forwarding Bridge ist fehlerhaft, dann gilt

$$b_{\lambda-1}^{(i)}, b_{\lambda+1}^{(i)} \in \xi\left(\mathcal{B}_p \cap \mathcal{B}_{fh}\right), \tag{6.52}$$

so dass wieder mit 6.46 und 6.47 folgt, dass jede fehlerfreie Bridge auf Schicht $i$ eine fehlerfreie Kopie der gesendeten Nachricht innerhalb der erlaubten Zeit empfängt. Weil dies auf allen Schichten gilt, ist somit die Behauptung für diesen Fall bewiesen.

*Fall 2:* Angenommen die Distributing Bridge ist fehlerfrei und es existiert eine Schicht $i$, $1 \leq i \leq n$, so dass jede Bridge auf dieser Schicht fehlerfrei ist. Aus (6.48) und (6.46) folgt, dass jede fehlerfreie Bridge eine fehlerfreie Kopie der gesendeten Nachricht innerhalb der erlaubten Zeit direkt von Bridges aus Schicht $i$ empfängt, unabhängig davon welche Bridges auf den anderen Schichten von Fehlern betroffen sind.

*Fall 3:* Sei nun angenommen, dass die Distributing Bridge fehlerhaft ist. Im trivialen Fall, in dem $d$ die Nachricht entweder an niemanden weiterleitet oder derart stark verzögert,

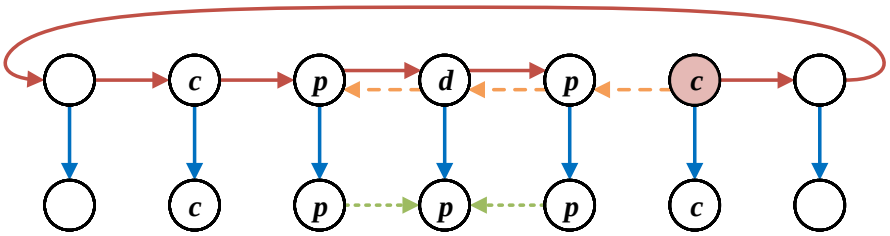



Abbildung 6.11.: Darstellung einer Welle des sekundären ExFABAN Routings für den Fall $f = 2$. Die vier Teilmengen der Welle aus Theorem 6.9 sind hier als horizontal durchgehende/rote (6.46), horizontal grob gestrichelte/orangefarbene (6.47), vertikal durchgehende/blaue (6.48) sowie horizontal fein gestrichelte/grüne (6.49) Pfeile dargestellt.

dass keine Checking Bridge eine Welle initiiert, ist nichts weiter zu zeigen. Ist $d$ derart fehlerhaft, dass nur eine einzige Prechecking Bridge die Nachricht von $d$ erhält, gilt die gleiche Argumentation für den Fall, dass die Prechecking Bridge ebenfalls fehlerhaft ist. Diese Argumentationskette kann induktiv für alle Prechecking Bridges fortgeführt werden.

Ohne Beschränkung der Allgemeinheit nimm an, dass unabhängig vom Fehlverhalten der fehlerhaften Distributing Bridge $d$, die Bridge $b_{\lambda+1}^{(\mu)}$ fehlerfrei ist und eine fehlerfreie Kopie der Nachricht innerhalb der erlaubten Zeit empfängt. Während $f-1$ weitere Bridges fehlerhaft sein können, garantiert Theorem 6.1, dass mindestens eine Checking Bridge in $\{b_{\lambda+f}^{(1)}, \ldots, b_{\lambda+f}^{(n)}\}$ eine Welle initiiert. Die genaue Zahl der initiierten Wellen hängt von der Anzahl der Fehler $f_p$ in $\mathcal{B}_p$ ab. Da $d$ im angenommenen Fall fehlerhaft ist, gilt für $1 \leq f_p \leq f$:

$$\min \Xi(f_p) \geq f - (f_p - 1) = f - f_p + 1. \tag{6.53}$$

Folglich können höchstens $f - f_p$ Fehler innerhalb der Forwarding Bridges auch höchstens $f - f_p$ Wellen beeinträchtigen, so dass mindestens eine Welle nicht von Fehlern beeinträchtigt wird und die gesendete Nachricht zu allen Bridges fehlerfrei und innerhalb der erlaubten Zeit weitergeleitet wird, während die Verbindungen gemäß (6.47) und (6.49) mehrfachen Einfluss von Einzelfehlern verhindern. Damit ist der Beweis abgeschlossen. □

Die hier bewiesenen Resultate bezügliche des primären und sekundären ExFABAN Routings können ebenfalls auf Ringnetzen angewendet werden. Hierbei muss/sollte folgendes beachtet werden:

- Das primäre Routing darf nur entlang der Verbindungen der *Außenhülle* des Ringnetzes stattfinden, also ausgenommen den Verbindungen, welche zwischen zwei Verknüpfungspunkten zweier Ringe/Ringnetze liegen. Sonst garantiert Theorem 6.3 keine Existenz eines sekundären Routings mit der erwünschten Fehlertoleranz. Abbildung 6.12 hebt die Außenhülle eines einschichtigen Ringnetzes hervor.
- Das sekundäre Routing darf sowohl entlang der *Außenhülle*, als auch entlang der Zwischenverbindungen erfolgen. In der ersten Variante kann Theorem 6.3 direkt angewendet werden, während in der zweiten Variante das Theorem auf den einzelnen Ringen separat angewendet werden kann, wobei darauf geachtet werden muss, dass die Verteilung in jedem Ring einer Schicht stets in entgegengesetzte Richtungen erfolgen muss. Abbildung 6.13 verdeutlicht das Prinzip des sekundären Routings für die Toleranz von einem Fehler auf einem einschichtigen Ringnetz.

Die Redundanz, welche durch das hier beschriebene primäre Routing entsteht, wird in diesem Maße nur benötigt, falls die Distributing Bridge fehlerhaft in einer byzantinischen Art ist. Sollte dies tatsächlich der Fall sein, so garantiert diese Redundanz, dass ein Broadcast entweder unterbunden oder erfolgreich durchgeführt wird. In dem Fall, dass

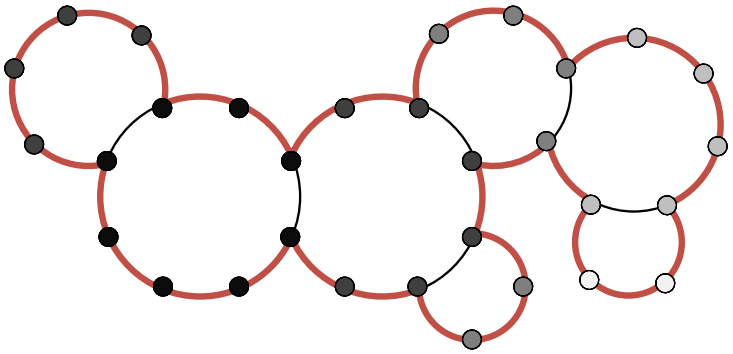


Abbildung 6.12.: Darstellung der Außenhülle eines einschichtigen Ringnetzes

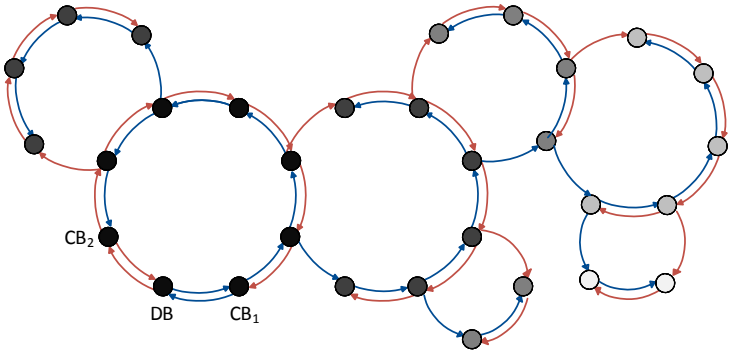



Abbildung 6.13.: Optimale Verteilung von Nachrichten auf Ringnetzen im Hinblick auf kurze Wellenlängen

die Distributing Bridge fehlerfrei ist, bietet das zuvor beschriebene Routing einen höheren Grad der Fehlertoleranz für das Fehlverhalten aller anderen Bridges. Nach Theorem 6.8 sendet die Distributing Bridge die zu verteilende Nachricht an zwei Prechecking Bridges der selben Schicht, sowie $f-1$ Prechecking Bridges auf jeweils einer der anderen Schicht, so dass auf jeder der $f$ Schichten des mehrschichtigen Rings, die Nachricht von jeder der $f-1$ Prechecking Bridges der anderen Schichten, sowie von beiden Prechecking Bridges der selben Schicht zu allen $2f$ Checking Bridges führen. Bei bis zu $f$ Fehlern in $\mathcal{B}_p$ können folgende zwei Szenarien eintreten:

- Alle Fehler betreffen Prechecking/Checking Bridges nur auf einer Seite der Distributing Bridge (in der gleichen oder in einer anderen Schicht als die Distributing Bridge), so dass entweder links oder rechts von der Distributing Bridge keine Checking Bridge eine fehlerfreie Welle initiieren kann. In diesem Fall ist aber die rechte bzw. linke Seite der Prechecking Bridges vollständig fehlerfrei, so dass $f$ Wellen von fehlerfreien Checking Bridges für die Verteilung einer unverfälschten

Nachricht initiiert werden.

- Sollten Fehler sowohl Bridges links, als auch rechts von der Distributing Bridge betreffen, dann existiert zu jeder Checking Bridge jeweils ein fehlerfreier Pfad von der Distributing Bridge, so dass mindestens $2f - f$ fehlerfreie Checking Bridges Wellen für die Verteilung einer unverfälschten Nachricht initiieren. Diese untere Abschätzung erhält man, wenn alle $f$ Fehler Checking Bridges betreffen und somit $2f - f = f$ fehlerfreie Checking Bridges Wellen initiieren.

In beiden Fällen können folglich zusätzlich zu den bereits $f$ fehlerhaften Bridges in $\mathcal{B}_p$, weitere $f-1$ Bridges in $\mathcal{B}\backslash\mathcal{B}_p$ fehlerhaft sein, so dass dennoch garantiert werden kann, dass die Nachricht gemäß der Definition des Atomic Broadcast bei allen fehlerfreien Empfängern zugestellt werden kann. Vergrößert man die Schwelle der Anzahl fehlerhafter Bridges in $\mathcal{B}_p$ noch weiter, dann können bestimmte Fehlerkombinationen zu Netzwerkpartitionierung führen, so dass ein definitionskonformer Atomic Broadcast nicht mehr möglich ist.

Bemerke jedoch, dass dies den Grad der Fehlertoleranz im Allgemeinen nicht tatsächlich erhöht, allerdings in der Praxis von Vorteil sein könnte, um bestimmte Fehlerkombinationen höheren Grades tolerieren zu können.

### 6.3.3. Redundante Sterntopologien mit Verteilungskern

Dieser Abschnitt stellt eine weitere Klasse von Graphen als Alternative zu vollvermaschten Netzen und mehrschichtigen Ringen vor, um auf systematische Art und Weise, Netzwerke für die Anwendung von ExFABAN aufbauen zu können. Die Idee basiert analog zu dem Ansatz der redundanten Sterntopologien in Kapitel 5.3.2 darauf, jede mögliche Distributing Bridge über einen zentralen Kern von Bridges zu allen Empfängern mit möglichst wenig Nachrichtenübertragungen weiterzuleiten. Während dazu im Fall der zu tolerierenden Fehler von $f = 1$ lediglich zwei Übertragungen von der Distributing Bridge zu jeder beliebigen Bridge im Netzwerk notwendig sind, werden im allgemeinen Fall gemäß Theorem 6.1 mehr Hops benötigt.

Im Allgemeinen gestaltet es sich als äußerst schwierig, Netzwerkgraphen für ExFABAN zu finden, welche einerseits möglichst wenig Verbindungen aufweisen und andererseits Symmetrieeigenschaften besitzen, so dass mit jeder Bridge Sender und Empfänger verbunden sein dürfen. Mit den mehrschichtigen Ringen wurde im vorangehenden Unterkapitel eine solche Klasse von Topologien vorgestellt. Insbesondere das sekundäre Routing hat gezeigt, dass die Hauptschwierigkeit darin liegt, die im primären Routing bereits verwendeten Bridges zu erreichen ohne übermäßig viele zusätzliche Verbindungen zu fordern. Aus diesem Grund ist die Idee entstanden, die Struktur der redundanten Sterntopologien aus Kapitel 5.3.2 aufzugreifen und anstelle von vielen zusätzlichen redundanten Verbindungen wenige zusätzliche Bridges bereitzustellen, welche nur für das primäre Routing

verwendet werden und mit denen keinerlei Netzwerkknoten verbunden sein dürfen. Die daraus resultierende Klasse von Graphen wird im Folgenden *redundante Sterntopologie mit Verteilungskern* bezeichnet. Mögliche Anwendungen könnten solche Topologien in Bereichen finden, wo zentrale redundante Rechenblöcke denkbar sind.

Nachfolgendes Lemma formuliert die Grundvoraussetzung für die Anwendbarkeit von ExFABAN auf redundanten Sterntopologien mit Verteilungskern.

**Lemma 6.10**
Der *Verteilungskern* einer redundanten Sterntopologie für die Toleranz von $f \geq 1$ Fehlern muss $f \cdot (f+1)$ Bridges umfassen.

*Beweis.* Folgt unmittelbar aus Theorem 6.1. □

Mit dieser Grundvoraussetzung lässt sich nun folgende formale Definition formulieren:

**Definition 6.11** (Redundante Sterntopologie mit Verteilungskern)
Ein Graph $G_S(f,n) = (\mathcal{B}, \mathcal{L})$ mit $f, n \geq 1$ heißt *redundante Sterntopologie mit Verteilungskern*, falls eine Teilmenge von Bridges

$$\mathcal{B}_{Kern} = \mathcal{B}^P_{Kern} \dot\cup \mathcal{B}^C_{Kern} \subseteq \mathcal{B} \tag{6.54}$$

mit Prechecking Bridges $\mathcal{B}^P_{Kern}$ und Checking Bridges $\mathcal{B}^C_{Kern}$, definiert durch

$$\mathcal{B}^P_{Kern} := \{b_{i,j} \in \mathcal{B} \mid 0 \leq i \leq f \ \wedge \ 1 \leq j < f\} \tag{6.55}$$

$$\mathcal{B}^C_{Kern} := \left\{b_{i,f} \in \mathcal{B} \;\middle|\; 0 \leq i \leq f\right\} \tag{6.56}$$

und $|\mathcal{B} \backslash \mathcal{B}_{Kern}| = n$ existiert und die Verbindungsmenge von der Form

$$\mathcal{L} = \bigcup_{j=1}^{f-1} \bigcup_{i=0}^{f} \bigcup_{k=i}^{i+f-j} \left\{(b_{i,j}, b_{(k \mod f+1),(j+1)})\right\} \tag{6.57}$$

$$\cup \{(b, b_{i,j}) \mid b \in \mathcal{B} \backslash \mathcal{B}_{Kern} \ \wedge \ 0 \leq i \leq f \ \wedge \ j \in \{1, f\}\} \tag{6.58}$$

$$\cup \{(b_{i,j}, b) \mid b \in \mathcal{B} \backslash \mathcal{B}_{Kern} \ \wedge \ 0 \leq i \leq f \ \wedge \ j \in \{1, f\}\} \tag{6.59}$$

ist.

Die Verbindungsmenge in vorangehender Definition ist gerade so definiert, dass zwischen den Prechecking Bridges und den Checking Bridges nur die notwendigen Verbindungen vorhanden sind, welche von Theorem 6.1 für ein primäres ExFABAN Routing gefordert werden. Die Menge der Bridges $\{b_{i,f} \mid 0 \leq i \leq f\}$ repräsentiert hierbei die Menge der

Checking Bridges und kann alle Empfänger des Netzwerks, ausgenommen den Bridges aus $\mathcal{B}_{Kern}$, auf direktem Weg erreichen und die Nachricht jeweils entlang von 1-Hop Wellen weiterleiten. Abbildung 6.14 zeigt ein Beispiel eines redundanten Sterns mit Verteilungskern für die Toleranz von $f = 3$ Fehlern.

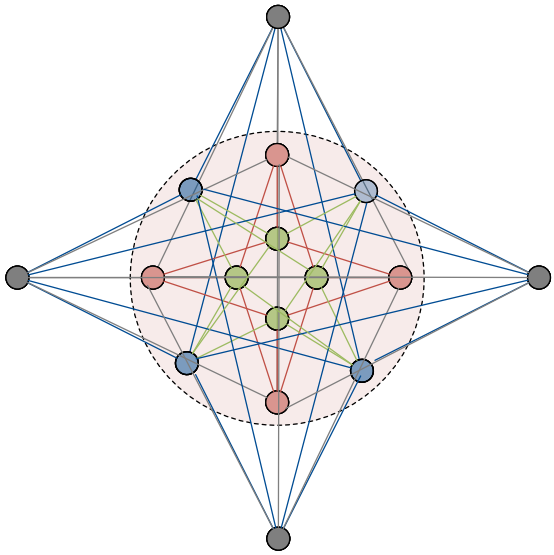


Abbildung 6.14.: Darstellung einer redundanten Sterntopologie mit Verteilungskern $G_S(3,4)$. Der Verteilungskern ist durch den roten Kreis gekennzeichnet. Die Prechecking Bridges $b_{i,1}$ sind in blau und $b_{i,2}$ in grün dargestellt, während die Checking Bridges rot markiert sind. Jede der grau markierten Bridges ist mit allen roten sowie blauen Bridges verbunden und kann als Distributing Bridge agieren. Die Menge der grauen Bridges kann um beliebig viele weitere Bridges auf analoge Art und Weise erweitert werden, wobei pro hinzukommende Bridge $2(f+1)$ Verbindungen hinzukommen. Die Kantenfarbe entspricht der Farbe der Bridges, zu welchen der Nachrichtenversand eines ExFABAN Rundspruchs stattfindet. Blaue sowie graue Kanten kennzeichnen die Verbindungsmenge (6.58) und (6.59), während grüne und rote Kanten die Verbindungen (6.57) innerhalb des Kerns entsprechen.

### 6.3.4. Realisierung der Uhrensynchronisation

Als Grundvoraussetzung für die Anwendbarkeit von FABAN sowie ExFABAN ist die Ausführung eines nebenläufigen und im gleichen Grad wie FABAN/ExFABAN fehlertoleranten Uhrensynchronisationsprotokolls. Im Zusammenhang mit den in den Abschnitten 6.3.1 - 6.3.3 vorgestellten Netzwerktopologien ist es deshalb wichtig sicherzustellen, dass Uhrensynchronisationsprotokolle mit dieser Anforderung für die jeweiligen Netzwerktopologien existieren. Das Protokoll von Halpern u. a. ist konzipiert für nicht zwingend vollvermaschte Netzwerke und kann beliebig viele Fehler tolerieren, so lange die fehlerfreien Komponenten weiterhin miteinander kommunizieren können [Hal+84]. Die

Topologieklassen der vollvermaschten Netze, der mehrschichtigen Ringe, der redundanten Sterntopologien mit Verteilungskern sowie allen anderen zu ExFABAN kompatiblen Netzwerktopologien erfüllen diese Anforderung, da sowohl FABAN als auch ExFABAN so konzipiert sind, dass zwischen allen Knotenpaaren mindestens $f+1$ redundante Pfade existieren müssen. Damit ist mindestens dieses Uhrensynchronisationsprotokoll stets anwendbar, so dass die Grundvoraussetzung der nebenläufig synchronisierten Uhren stets erfüllt wird.

### 6.3.5. Gegenüberstellung charakteristischer Eigenschaften

Während der Untersuchung und Entwicklung der in den vorherigen Abschnitten behandelten Graphenklassen für die Anwendung von ExFABAN, wurden eine Reihe von Ergebnissen erzielt, welche dazu genutzt werden können, um charakteristische Größen der Graphen, sowie des Protokolls ExFABAN gegenüberstellen zu können. Die nachfolgenden Ergebnisse ergeben sich mit Hilfe von Theorem 6.4, Definition 6.5, Lemma 6.6, Theorem 6.8 sowie Theorem 6.9.

Verglichen werden folgende charakteristische Größen, welche im Allgemeinen von der Graphengröße (Knotenzahl, Kantenzahl) sowie der Anzahl der zu tolerierenden Fehler $f$ abhängig sind:

- `Knotenanzahl` $|\mathcal{B}|$
- `Verbindungsanzahl` $|\mathcal{L}|$
- `Minimale Knotenanzahl` $\min|\mathcal{B}|$
- `Minimale Verbindungsanzahl` $\min|\mathcal{L}|$
- `Wellenlänge` $len_W$
- `Nachrichtenübertragungen pro Broadcast` $\#msg/bc$
- `Maximale Wellenanzahl` $\#W$

Die Berechnung der Größen wird dabei im folgenden nacheinander für die drei einzelnen Graphenklassen durchgeführt.

**Vollvermaschte Graphen** Ein vollvermaschter Graph $G_V(N)$ mit $|\mathcal{B}| = N$ Bridges besteht aus

$$|\mathcal{L}| = \sum_{k=1}^{N} k = \frac{1}{2}N(N-1) \tag{6.60}$$

Verbindungen. Die minimale Anzahl benötigter Bridges ergibt sich aus Theorem 6.4 zu

$$\min|\mathcal{B}| = f^2 + f + 1 \tag{6.61}$$

und damit zur Mindestanzahl von Verbindungen in Höhe von

$$\min|\mathcal{L}| = \frac{1}{2}(f^2+f+1)(f^2+f+1-1) = \frac{1}{2}(f^4+2f^3+2f^2+f). \tag{6.62}$$

Für die Ausführung von ExFABAN werden $\#W = f+1$ Wellen benötigt, wobei jede Welle ab jeder Checking Bridge alle anderen Bridges auf direktem Weg erreichen kann, so dass sich $len_W = f+1$ ergibt. Im Hinblick auf einen einzelnen Rundspruch eines Senders ergibt sich im fehlerfreien Fall die Gesamtanzahl der Nachrichtenübertragungen pro Broadcast wie folgt:

$$\#msg/bc = (f+1) + (f+1)\cdot\sum_{j=2}^{f} j + (f+1)(N-1) \tag{6.63}$$

$$= \frac{1}{2}\left[f^3+2f^2+(2N-1)f+(2N-2)\right] \tag{6.64}$$

**Mehrschichtige Ringe** Ein mehrschichtiger Ring $G_R(k,n)$ kann so viele Fehler tolerieren, wie er Schichten enthält, d.h. $f = k$, wobei die Ringgröße jedes einzelnen Rings nach unten beschränkt ist auf $n \geq 2f+1$ Bridges. Für einen beliebig großen mehrschichtigen Ring bei nur notwendig vielen Ringschichten, das heißt bei $k = f$, beträgt die Anzahl der Bridges $|\mathcal{B}| = nk$ und die Anzahl der Verbindungen ergibt sich als Summe der Verbindungen innerhalb der Ringe sowie den Verbindungen zwischen den Ringschichten:

$$|\mathcal{L}| = nf + n\sum_{j=1}^{f-1} j = \frac{1}{2}nf(f+1). \tag{6.65}$$

Die Mindestanzahl von Bridges für die Toleranz von $f$ Fehlern ergibt sich nach Lemma 6.6 mit $n = 2f+1$ zu

$$\min|\mathcal{B}| = f(2f+1) \tag{6.66}$$

sowie die Mindestanzahl von Verbindungen zu

$$\min|\mathcal{L}| = \frac{1}{2}(2f+1)\cdot f\cdot(f+1) = \frac{1}{2}(2f^3+3f^2+f). \tag{6.67}$$

Auf jedem einzelnen Ring des mehrschichtigen Rings werden 2 Wellen initiiert, so dass insgesamt $\#W = 2f$ Wellen benötigt werden. Während des primären Routings können dabei im schlimmsten anzunehmenden Fall $2f+1$ Nachrichtenweiterleitungen entstehen, bevor die jeweilige Welle über maximal $n$ weitere Hops die Nachricht allen anderen Bridges zustellt, so dass sich die maximale Wellenlänge von $len_W = n + [2f+1]$ ergibt. Betrachtet man abschließend einen einzigen Rundspruch, so lässt sich die Gesamtanzahl der Nachrichtenübertragungen zwischen Bridges mit Theorem 6.8 sowie Theorem 6.9 wie

folgt ermitteln:

$$\#msg/bc = 2f^2 + 2f^2(f-1) + (f-1) + \qquad \text{[Prim. Routing]} \tag{6.68}$$

$$+ 2f\left[(k-1) + (2f-1) + k(f-1) + 2(f-1)\right] \qquad \text{[Sek. Routing]} \tag{6.69}$$

$$= 2f^3 + (8+2k)f^2 - 7f - 1. \tag{6.70}$$

**Redundante Sterntopologien mit Verteilungskern** Eine redundante Sterntopologie mit Verteilungskern $G_S(f,n)$ besteht nach Lemma 6.10 aus einem Kern mit $f(f+1)$ Bridges sowie weiteren $n$ Bridges, so dass sich insgesamt $|\mathcal{B}| = f(f+1)+n$ ergeben. Mit der Anzahl der Verbindungen im Verteilungskern $(f+1)\sum_{k=2}^{f} k$, zusammen mit den Verbindungen der Bridges außerhalb des Verteilungskerns zu $f+1$ Prechecking Bridges sowie $f+1$ Checking Bridges, erhält man insgesamt

$$|\mathcal{L}| = 2n(f+1) + (f+1)\sum_{k=2}^{f} k = 2n(f+1) + \frac{1}{2}\left(f^3 + 2f^2 - f - 2\right) \tag{6.71}$$

Verbindungen. Neben dem Verteilungskern wird mindestens eine weitere Bridge benötigt, so dass sich die Mindestanzahl an Bridges von

$$\min|\mathcal{B}| = f^2 + f + 1 \tag{6.72}$$

ergibt und mit $n = 1$ in Gleichung (6.71) erhält man mit

$$\min|\mathcal{L}| = \frac{1}{2}(f^3 + 2f^2 + 3f + 2) \tag{6.73}$$

die kleinste Anzahl von Verbindungen, welche eine redundante Sterntopologie mit Verteilungskern benötigt. Analog zu den vollvermaschten Graphen werden auch hier für die Ausführung von ExFABAN $\#W = f+1$ Wellen benötigt, wobei jede Welle ab jeder Checking Bridge ebenfalls alle Bridges auf direktem Weg erreichen kann, so dass sich $len_W = f+1$ ergibt. Da bei redundanten Sterntopologien mit Verteilungskern das gleiche Verteilungsschema verwendet wird wie bei vollvermaschten Graphen, entspricht die Gesamtanzahl der Nachrichtenübertragungen zwischen Bridges dem gleichen Wert, d.h.

$$\#msg/bc = \frac{1}{2}\left[f^3 + 2f^2 + (2N-1)f + (2N-2)\right]. \tag{6.74}$$

Tabelle 6.1 simplifiziert die Ergebnisse und stellt die Konvergenzordnungen der zuvor beschriebenen charakteristischen Größen in Abhängigkeit von der geforderten Anzahl der zu tolerierenden Fehler gegenüber. Die tatsächliche Anzahl von Bridges $|\mathcal{B}|$ und Links $|\mathcal{L}|$ ist frei wählbar und folglich nicht von Interesse in dem Vergleich. Lediglich die unteren Schranken dieser Größen sind relevant und werden betrachtet.

| | Vollvermaschter Graph $G_V(N)$ | Mehrschichtiger Ring $G_R(k,n)$ | Red. Sterntopologie $G_S(f,n)$ |
|---|---|---|---|
| $\min\|\mathcal{B}\|$ | $\mathcal{O}(f^2)$ | $\mathcal{O}(f^2)$ | $\mathcal{O}(f^2)$ |
| $\min\|\mathcal{L}\|$ | $\mathcal{O}(f^4)$ | $\mathcal{O}(f^3)$ | $\mathcal{O}(f^3)$ |
| $len_W$ | $\mathcal{O}(f)$ | $\mathcal{O}(f)$ | $\mathcal{O}(f)$ |
| $\#W$ | $\mathcal{O}(f)$ | $\mathcal{O}(f)$ | $\mathcal{O}(f)$ |
| $msg/BC$ | $\mathcal{O}(f^3)$ | $\mathcal{O}(f^3)$ | $\mathcal{O}(f^3)$ |

Tabelle 6.1.: Vergleich charakteristischer Größenordnungen zwischen unterschiedlichen Graphenklassen.

Auffallend ist, dass beide neu eingeführten Klassen von Graphen insgesamt ähnliche Konvergenzordnungen haben und sich nur in einzelnen Aspekten voneinander unterscheiden. Dies ist aber nicht überraschend vor dem Hintergrund, dass diese Graphenklassen im Hinblick auf möglichst effiziente Nachrichtenverteilungen und Symmetrieeigenschaften entstanden sind. Eklatante Unterschiede lassen sich nur bei den vollvermaschten Netzen erkennen. Diese haben naturgemäß eine deutlich höhere Anzahl an Verbindungen, während die restlichen verglichenen Eigenschaften kaum bis gar keine Vorteile im Vergleich zu den beiden neu eingeführten Graphenklassen bieten, was wiederum die guten Eigenschaften der mehrschichtigen Ringe bzw. Ringnetze sowie der redundanten Sterne mit Verteilungskern unterstreicht.

Ein Vergleich zwischen diesen beiden Klassen zeigt, dass keine allgemeine Aussage darüber, welche Klasse besser ist, getroffen werden kann. Viel mehr spielt hierbei das Anwendungsszenario eine große Rolle. Während die redundanten Sterne mit Verteilungskern gut auf Szenarien appliziert werden können, in denen ein kompaktes Netzwerk mit ungefähr gleichmäßiger Anbindung zwischen einzelnen Komponenten gefordert wird, lassen sich mehrschichtige Ringe eher auf großflächige Netzwerke mit nicht gleichmäßiger Anbindung anwenden.

## 6.4. Zusammenfassung

FABAN wurde ursprünglich entwickelt um eine beliebige Anzahl von Fehlern zu tolerieren. Dabei hat sich herausgestellt, dass der ursprüngliche Ansatz, lediglich die Anzahl der Wellen sowie die Anzahl der Checking Bridges mit zunehmendem Fehlertoleranzgrad zu erhöhen, nicht ausreichend ist. Als Lösung dieses Problems wurde das Protokoll grundlegend erweitert, woraus sich das vollständig abwärtskompatible Protokoll ExFABAN

ergeben hat. Neben der Erweiterung um die zusätzliche Rolle der Prechecking Bridge, wurde das Routing aufgrund der zunehmenden Komplexität der Anforderungen in die zwei Phasen des primären sowie des sekundären Routings separiert und das Verfahren der Signaturmodifikationen um Modifikationen in den Prechecking Bridges erweitert. Des Weiteren wurde das Nachrichtenformat ergänzt durch einen Hop Counter sowie die Information, von welcher Checking Bridge die Ausbreitung der jeweiligen Nachricht initiiert wurde. Notwendige und hinreichende Fehlertoleranz-Bedingungen für die Netzwerktopologie, das primäre und sekundäre Routing sowie die Funktionalität der Bridges wurden formal beschrieben und mathematisch vollständig bewiesen.

Eine effiziente Weiterentwicklung des Algorithmus für die Generierung von Wellen in der Einfehlerannahme konnte für die allgemeine Fehlerannahme nicht gefunden werden und ist mit zunehmendem Fehlertoleranzgrad aufgrund der steigenden Anforderungen an die Netzwerktopologien von eher vernachlässigbarem Nutzen. Der praktikablere Ansatz ist die Erstellung von ExFABAN-geeigneten Netzwerktopologien für einen angestrebten Fehlertoleranzgrad. Dieser Ansatz wurde insoweit verfolgt, dass mit den vollvermaschten Graphen, den redundanten Sterntopologien und insbesondere den mehrschichtigen Ringen bzw. Ringnetzen geeignete Klassen von Netzwerktopologien beschrieben wurden, auf welchen eine beliebige Anzahl von Fehlern toleriert werden kann. Abgeschlossen wurde der Teil der Arbeit mit einem Vergleich charakteristischer Größen aller Topologieklassen, woraus sich ergeben hat, dass die mehrschichtigen Ringe bzw. Ringnetze im Bezug auf die meisten Netzwerk-Anforderungen die besten Eigenschaften, sowohl theoretisch als auch praktisch vorweisen.

Kapitel 7

# Simulative Bewertung

In diesem Teil der Arbeit werden die zuvor vorgestellten und theoretisch analysierten Protokolle mittels Computersimulation untersucht. Hierbei ist das Ziel zum einen die Korrektheit der Behauptungen aus den vorherigen Kapiteln auf diversitärem Wege zu validieren, indem die theoretischen Überlegungen simulativ überprüft werden, und zum anderen mögliche, bisher unvorhergesehene Probleme zu erfahren und gegebenenfalls zu lösen.

Im ersten Abschnitt dieses Kapitels wird zunächst die genutzte Simulationsumgebung, *Objective Modular Network Testbed in C++ (OMNeT++)*, vorgestellt und begründet. Das darauffolgende Unterkapitel stellt das Implementierungskonzept der Simulation vor. Hierbei werden die Implementierungen einzelner Komponenten wie der Netzwerkknoten bzw. der Bridges im notwendigen, aber geringstmöglichem Detailgrad sowie der Umfang der statistischen Erfassung als auch der Fehlerinjektionsmechanismen im Detail vorgestellt.

Die Protokollierung der durchgeführten Simulationen erfolgt durch die Beschreibung der zu simulierenden Netzwerktopologien und den zugehörigen Parametrisierungen. Zunächst werden Simulationen ohne Fehlerinjektionen durchgeführt, um Laufzeitstatistiken wie beispielsweise Queueauslastungen und Übermittlungszeiten zu ermitteln und die Korrektheit der Implementierung zu verifizieren. Anschließend werden Fehlerinjektionen hinzugezogen und, neben der wiederholten Erfassung der Laufzeitstatistiken, Ergebnisstatistiken der erfolgreichen/nicht erfolgreichen Broadcasts, der fehlerhaften oder der verlorenen Nachrichten gesammelt und ausgewertet.

## 7.1. Simulationsumgebung

Für die Realisierung einer Simulation von FABAN bzw. ExFABAN gibt es eine Vielzahl unterschiedlicher Möglichkeiten mit spezifischen Vor- und Nachteilen. Die Implementierung einer von Grund auf eigenen Simulation in einer Programmiersprache beliebiger Wahl wie zum Beispiel in C++ oder Java bietet auf der einen Seite die größten Freiheiten und die größten Möglichkeiten des Abstraktionsgrades. Diese reichen von der Wahl der grundlegenden Simulationsart, z.B. kontinuierliche Simulation oder diskrete Event Simulation, über die Logik der Simulation bis hin zu der Darstellung der Ergebnisformate einzelner Simulationsläufe [LP06; ÖB09; Ste18]. Auf der anderen Seite kann, abhängig vom angestrebten Abstraktionsgrad der Simulation, der mit der Implementierung rudimentärer Simulations- und Datenerfassungsmechanismen verbundene Aufwand, unverhältnismäßig werden. Mit zunehmender Nähe der Simulation an das reale System, d.h. mit abnehmendem Abstraktionsgrad, steigt der Aufwand der Abbildung realer Mechanismen in dem Simulationsmodell, so dass die Nachteile einer vollständig eigenen Simulation überwiegen könnten.

Um eine möglichst hohe Nähe zur Realität bei gleichzeitig hoher Flexibilität des Abstraktionsgrades zu gewährleisten, wurde für die Simulation von FABAN bzw. ExFABAN die Entscheidung getroffen, auf ein Simulations-Framework zurückzugreifen, welches einerseits grundlegende Mechanismen für Simulationen und Datenerfassung bereitstellt, andererseits genug Freiheiten in der Implementierung bietet, um das zu implementierende Protokoll auf gewünschte Eigenschaften untersuchen zu können. Hierbei ist die Wahl konkret auf das Simulations-Framework OMNeT++ gefallen [Ope20b]. Dabei handelt es sich um ein Framework für die Simulation von Rechnernetzen sowie Netzwerkprotokollen, welches eine Reihe von Leistungsmerkmalen bereitstellt, die im nachfolgenden punktuell aufgelistet und erläutert werden:

- **Diskrete Event Simulation:** Der zeitliche Verlauf wird nicht als stetiges Fortschreiten der Zeit, sondern ereignisbasiert modelliert, so dass unwichtige bzw. uninteressante Zeitabschnitte aus dem Fokus geraten sowie eine schnellere Simulation größerer Zeitspannen ermöglicht wird.
- **Programmiersprache und IDE:** Die Implementierung logischer Abläufe findet in der Programmiersprache C++ statt. Hierbei kann der gesamte Sprachraum benutzt werden, welcher durch sinnvolle, speziell für OMNeT++ entwickelte Funktionen und Klassen, erweitert wird. OMNeT++ wird hierbei als eine Erweiterung in Eclipse, sowohl in Form eines Gesamtpakets als auch in Form eines Plugins für bestehende Eclipse Installationen, angeboten.
- **Modularität:** Simulationen in OMNeT++ sind modular aufgebaut. Grundmodule (*SimpleModule*) können frei in C++ programmiert werden und können Nachrichten

erstellen, empfangen, manipulieren, versenden oder zerstören. Erweiterte Module (*CompoundModule*) bestehen aus Grundmodulen, welche über optional selbst implementierte Verbindungen miteinander kommunizieren können. Zuletzt dürfen Netzwerke aus Grundmodulen sowie erweiterten Modulen bestehen, welche als Grundlage für Simulationen dienen.

- **Nachrichtenformate:** Als Kommunikation zwischen einzelnen Grundmodulen sowie erweiterten Modulen dienen Nachrichten einer zwingend vorgegebenen Grundklasse (*cMessage*). Eigene Nachrichtenformate lassen sich mittels einer vereinfachten, OMNeT++ speziellen Syntax definieren, so dass durch interne OMNeT++ Mechanismen abgeleitete Nachrichtenklassen erzeugt werden, welche einerseits vollständige Kompatibilität zum Kommunikationssystem bieten und andererseits Individualität ermöglichen.
- **Parametrisierbare Simulationen:** Basismodule sowie auch erweiterte Module können durch Parameter eingestellt werden. Eine zentrale Konfigurierbarkeit von einzelnen Simulationen wird durch Konfigurationsdateien ermöglicht. Durch diese können einzelne bzw. alle Parameter von Modulen, welche das Netzwerk beinhaltet, überschrieben werden, so dass eine hohe Flexibilität der Parametrisierung einzelner Simulationsdurchläufe ermöglicht wird.
- **Graphische Darstellung:** Simulationen können in einer graphischen Simulationsoberfläche sowohl schrittweise als auch mit variablen Geschwindigkeiten ausgeführt werden. Auf diese Weise lassen sich Nachrichtentransfers und andere Abläufe im Detail nachvollziehen und gegebenenfalls Fehler in der Implementierung oder im Protokolldesign finden.
- **Cli Simulationen:** Finale Implementierungen können vollständig als Konsolenanwendung gestartet werden. Auf diese Weise können Mengen von Simulationen automatisiert ausgeführt werden.
- **Datenerfassung und Auswertung:** OMNeT++ bietet globale Mechanismen zum Erfassen von Daten. Hierbei können skalare Werte, Vektoren sowie Histogramme aufgezeichnet werden, welche nach Abschluss einer jeden Simulation in entsprechend eindeutig benannten Statistikdateien gespeichert werden. Die Auswertung der in einer Simulation gesammelten Daten kann sowohl in der OMNeT++-IDE als auch nach dem Exportieren in geeignete allgemeine Formate (CSV, SQLite, ...) in externen Anwendungen erfolgen. Im Rahmen dieser Arbeit werden Statistiken mit der Statistiksoftware *R* [Fou20] ausgewertet, welche mehr Freiheiten bietet.

Auf OMNeT++ aufbauend existieren eine Reihe von Erweiterungen in Form von Bibliotheken für allgemeinere bis hin zu spezielleren Anwendungsgebieten. Die prominenteste Modellbibliothek stellt das Framework *INET* dar, welches Modelle für den Internet Stack

(TCP, UDP, IPv4, IPv6 usw.), für Verbindungsprotokolle (Ethernet, PPP, IEEE 802.11 usw.) sowie diverse andere Protokolle und Modelle bereitstellt. [INE20] Für speziellere Anwendungsgebiete existieren mit *SimuLTE*, *Core4INET* oder *FiCo4Omnet* Bibliotheken, welche Erweiterungen für die Simulation von LTE, Realtime Ethernet (TTEthernet, AVB, TSN) oder Fieldbus Communication (FlexRay, CAN) Netzwerken bereitstellen. [Cor20; FiC20; FiC20; VSN14] Bei diesen Beispielen handelt es sich nur um einen sehr kleinen Auszug einer sehr langen Liste von Erweiterungsbibliotheken, welche im nachfolgenden aber keine Rolle spielen werden. Als Grundlage für die im Rahmen dieser Arbeit implementierte Simulation werden die Kernfunktionen von OMNeT++ genommen, ohne von weiteren Erweiterungen Gebrauch zu machen.

## 7.2. Implementierungskonzept

Die Implementierung einer Simulation von FABAN sowie ExFABAN in OMNeT++ ist ein vielschichtiger Prozess, der in den nachfolgenden Unterabschnitten strukturiert beschrieben und erklärt wird.

### 7.2.1. Allgemeine Überlegungen und ganzheitliche Konzeptidee

FABAN sowie ExFABAN sind für Netzwerke konzipiert worden, welche aus Netzwerkknoten sowie Bridges bestehen. Folglich muss das Verhalten sowohl eines Netzwerkknotens, welcher einerseits unter bestimmten, vorgegebenen Regeln FABAN-Rundsprüche initiieren und andererseits FABAN-Nachrichten empfangen und protokollkonform verarbeiten muss, als auch das Verhalten einer FABAN-fähigen Bridge umgesetzt werden. Letztere muss situationsbedingt alle FABAN-Rollen, d.h. die Rollen der Distributing Bridge, der Checking Bridge, der Prechecking Bridge sowie der Forwarding Bridge, annehmen können. Die Modularität von OMNeT++ wird hierbei ausgenutzt und eine soweit möglich unabhängige Implementierung von Netzwerkknoten und Bridges realisiert. Eine genauere Beschreibung der Umsetzung beider Module findet in den nachfolgenden Abschnitten statt.

Globale Aufgaben, sowie eine zentrale Schnittstelle für globale Konfigurationen, werden in Form eines zusätzlichen Moduls, dem so genannten *NetzwerkAdministrator*, realisiert. Die Kernaufgaben dieses Moduls werden im Folgenden erläutert.

- Um den Aufwand der Konfiguration durch den Benutzer zu minimieren, führt das Modul eine Graphenexploration durch, erstellt eine formale Beschreibung des Netzwerks und ermittelt im Fall der Toleranz von $f = 1$ Fehlern vollautomatisiert das FABAN-Routing mit Hilfe einer externen Java-Implementierung des Algorithmus aus Kapitel 5.2.1, welche bereits für Evaluationen in Kapitel 5 verwendet wurde.

Für den allgemeineren Fall $f > 1$ existiert keine effektive Lösung zur Ermittlung der Wellen, so dass in diesem Fall das Modul die Routing Informationen aus einer externen Datei lädt. Als mögliche Alternative könnte ein Brute-Force-Ansatz zur Ermittlung der Wellen in Betracht gezogen werden. Da für die Simulationen im Wesentlichen Graphen von Interesse sind, welche den Graphenklassen aus Kapitel 6.3 entsprechen und bei einem Brute-Force-Ansatz keine Garantie besteht, effektive Wellen zu erhalten, spielte dieser auch keine weitere Rolle. In jedem Fall verteilt das Modul die ermittelten bzw. eingelesenen Routing-Information an die entsprechenden Bridges.

- Für die Signaturmanipulationen in den einzelnen Bridges bzw. dem Empfänger erstellt bzw. berechnet das Modul zufällige Signaturmasken und stellt global zugreifbare Funktionen bereit, welche den Zugriff auf diese Werte erlauben.
- Zuletzt stellt das Modul weitere zentrale Funktionen bereit, welche es erlauben, dass einzelne Netzwerkknoten ihre individuellen, öffentlichen Signaturschlüssel unter allen Empängern des Netzwerks bereitstellen und die maximale Wellenlänge eines FABAN-Rundspruchs, basierend auf dem Ergebnis der algorithmischen Wellen-Ermittlung bzw. der eingelesenen Wellen-Informationen, eines Senders liefern.

Das Nachrichtenformat, welches in Kapitel 6.2.2 für den allgemeinen Fall $f \geq 1$ beschrieben wurde, wird als OMNeT++ eigenes Nachrichtenpaket definiert, wobei folgende Feldgrößen gewählt wurden:

| Feld | Datentyp | Größe |
|---|---|---|
| `senderID` | `uint16_t` | 16 bit |
| `seqNo` | `uint8_t` | 8 bit |
| `deliveryTime` | `uint64_t` | 64 bit |
| `hopCounter` | `uint8_t` | 8 bit |
| `checkingBridgeId` | `uint16_t` | 16 bit |
| `signature` | `uint32_t` | 32 bit |

Tabelle 7.1.: Übersicht der Feldgrößen einer FABAN-Nachricht in der Simulation

Dabei wurden die Größen so gewählt, dass man keine Einschränkungen bei der Netzwerkgröße oder den zu tolerierenden Fehler hinnehmen muss. Insbesondere das Feld für die Zustellzeit ist groß genug gewählt worden, so dass selbst in der verwendeten Nanosekunden-Genauigkeit bis zu 513 Jahre abgedeckt werden. Das Datenfeld kann des Weiteren beliebig groß gewählt werden, wobei der Dateninhalt zufällige Daten enthält und durch OMNeT++-spezifische Mittel verkapselt wird.

Im Vorfeld der Beschreibung des Implementierungskonzepts der Netzwerkknoten sowie der Bridges ist es zuletzt wichtig, die angestrebten Parameter der Simulationen festzulegen. Diese, allgemeinen Simulationsparameter, werden in folgender Tabelle beschrieben:

| Parameter | Beschreibung |
|---|---|
| `**.sendInterval` | Konstante oder Wahrscheinlichkeitsverteilung gemäß welcher die Abstände neuer FABAN-Nachrichten aller Sender bestimmt werden. |
| `**.dataSize` | Konstante oder Wahrscheinlichkeitsverteilung gemäß welcher die Größen der Datenfelder von FABAN-Nachrichten bestimmt werden. |
| `**.maxDataSize` | Konstante obere Schranke für die Größe des Datenfeldes. Diese ist gedacht für Wahrscheinlichkeitsverteilungen, die keine natürliche obere Schranke implizieren. |
| `**.srcGeneratorActive` | Boolesche Konstante, die festlegt, ob ein Netzwerkknoten als Generator für FABAN-Nachrichten aktiv ist. |
| `**.maxMessages` | Maximale Anzahl von Nachrichten, die von einem einzelnen Sender erzeugt werden darf. |
| `**.noPorts` | Konstante Anzahl der Ports der Bridges. |
| `**.faultassumption` | Die Anzahl der zu tolerierenden Fehler. |
| `**.datarate` | Verbindungsgeschwindigkeit(Linkspeed) in Bits/Sekunde (kann in Omnet++ mit Faktoren versehen werden, um bps, kbps, Mbps, Gbps usw. zu erhalten). |
| `**.queuesize` | Kapazität aller Warteschlangen der ausgehenden Ports in Bridges und Netzwerkknoten sowie dem internen Puffer der Bridges. |
| `**.computingtime` | Konstante oder Wahrscheinlichkeitsverteilung gemäß welcher die Berechnungszeit ( = Verweilzeit) eines Pakets innerhalb einer Bridge bestimmt wird. |
| `**.deliveryTimeFactor` | Konstanter Faktor zur Verringerung/Erhöhung der von den Sendern berechneten Zustellzeit von Nachrichten. Dieser dient hauptsächlich zu Debugging-Zwecken bzw. zur experimentellen Verifikation der Zustellzeiten. |

Tabelle 7.2.: Parameterübersicht von Simulationen von FABAN und ExFABAN.

Die Modellierung von Verweilzeiten auf den Links wurde nicht separat umgesetzt, da diese als Teil der Verweilzeiten in den Bridges modellierbar sind. Alle Parameter lassen sich global für alle Komponenten eines Systems oder individuell für einzelne oder Gruppen von Komponenten einstellen.

### 7.2.2. Netzwerkknoten

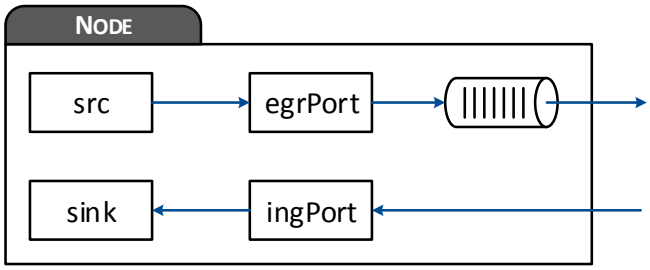



Abbildung 7.1.: Aufbau eines Netzwerkknoten in OMNeT++

Der Aufbau eines Netzwerkknoten ist so simpel wie möglich konzipiert worden und ist in Abbildung 7.1 dargestellt.

Ist ein Netzwerkknoten als Sender aktiv, so generiert dieser im Modul `src` gemäß den Parametern für die Abstände zwischen zwei aufeinanderfolgenden FABAN-Nachrichten und der Datengröße einer FABAN-Nachricht so lange Rundspruch-Nachrichten, bis die maximale Anzahl von zu erzeugenden Nachrichten versendet wurde bzw. die Simulation terminiert. Hierbei wird der Abstand zwischen zwei Nachrichten zusätzlich auf einen von der Größe der aktuellen Nachricht $length(m)$ abhängigen Mindestwert

$$\Delta_{min} = \frac{length(m)}{Linkspeed} \tag{7.1}$$

festgelegt, falls der durch den Parameter resultierende Abstand zwischen zwei Nachrichten zu gering ausfallen sollte. Dadurch soll verhindert werden, dass die nächste FABAN-Nachricht erzeugt wird, bevor die aktuelle Nachricht vollständig an der benachbarten Bridge, limitiert durch physikalische Einschränkungen durch Verbindungen, ankommen konnte. Alle Nachrichten werden gemäß Protokollspezifikation mit korrekter Sender ID sowie fortlaufender Sequenznummer modulo $2^8$ erzeugt.

#### Zeitmodell und Bestimmung der Zustellzeit

Bridges und Netzwerkknoten verfügen über vollständig synchronisierte Uhren, wobei diese nicht separat realisiert werden, sondern die Simulationszeit des OMNeT++-Zeitmodells

als globale Uhr in allen Komponenten verwendet wird. Das OMNeT++-Zeitmodell hat standardmäßig eine Genauigkeit von $10^{-12}s$ und kann bei Bedarf auf bis zu $10^{-18}s$ verringert werden. Individuelle lokale Uhren mit Abweichungen lassen sich leicht als Funktionen der globalen Simulationszeit realisieren, sind hier jedoch nicht von Interesse und wurden folglich nicht betrachtet.

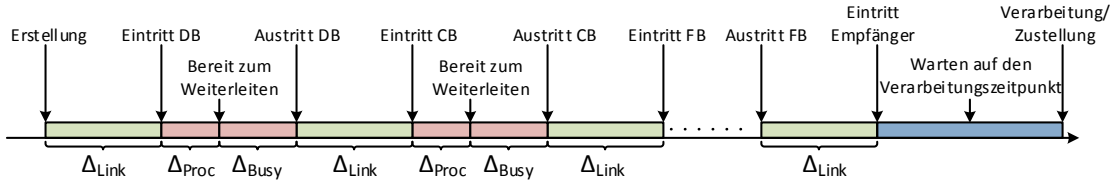



Abbildung 7.2.: Darstellung der zeitlichen Verzögerungen einer Nachricht zur Berechnung der Zustellzeit $t_d$.

Als Grundlage für die Berechnung der Zustellzeit einer Nachricht eines festen Senders mit einer gegebenen Menge von Wellen $\mathcal{W} = \{W_1, \ldots, W_k\}$ für $k \geq 2$, wird die Überlegung in Abbildung 7.2 betrachtet. Aus dieser ergeben sich mit dem Maximum aller Wellenlängen $H := \max\{|W_j| \mid 1 \leq j \leq k\}$ folgende von der Nachricht $m \in M$ abhängige Teilgrößen:

- $\Delta_{Link}(m)$ beschreibt die Dauer der Übertragung von einer Nachricht $m$ zwischen zwei direkt Verbundenen Komponenten und errechnet sich wie folgt:

$$\Delta_{Link}(m) := \frac{length(m)}{Linkspeed}. \tag{7.2}$$

- $\Delta_{Proc}(m)$ beschreibt die Verweilzeit der Nachricht innerhalb einer Bridge aufgrund von Verzögerungen durch interne Berechnungen, wie z.B. Operationen auf der Signatur, Zeitprüfungen oder der Ermittlung des Routings für das Weitersenden der Nachricht. Diese Größe wird durch den Parameter `**.computingtime` vorab der Simulation als Konstante oder als Wahrscheinlichkeitsverteilung festgelegt.

- $\Delta_{Busy}$ beschreibt die maximale zeitliche Verzögerung in einer Bridge, die durch hohen Nachrichtenverkehr in den Bridges entstehen kann. Im Gegensatz zu den vorherigen Größen ist dieser Wert nicht ohne Weiteres vorhersehbar, sondern ist von zugrunde liegenden Annahmen und Systemeigenschaften abhängig. Geht man von einem System mit beschränkten pro Hop Latenzen aus, so wie in den FABAN-Systemannahmen in Kapitel 3.1.1 gefordert, kann $\Delta_{Busy}$ von dieser Größe abgeleitet werden. Ist diese Eigenschaft des Systems nicht vorhanden, ist die Frage interessant, inwieweit mit anderen Mitteln, wie zum Beispiel der Beschränkung von Sendeströmen, eine sinnvolle Anwendung von FABAN erzielt werden kann. Dies wird während Simulationen im nachfolgenden näher untersucht und entsprechende

Abschätzungen für $\Delta_{Busy}$ vorgeschlagen. Zwecks Flexibilität ist deshalb diese Größe über Parameter einstellbar.

| Parameter | Beschreibung |
|---|---|
| `**.deltaBusy` | Zusätzliche maximale Verzögerung einer Nachricht in einer Bridge. |

Tabelle 7.3.: Ergänzung zur Parameterübersicht

Damit ergibt sich die absolute Zustellzeit, gemessen ab Simulationsstart zu Sekunde 0, einer Nachricht, welche über $H+1$ Bridges übertragen wird, zu:

$$t_d = (H+2) \cdot \Delta_{Link} + (H+1) \cdot (\Delta_{Proc} + \Delta_{Busy}) \tag{7.3}$$

sowie die Überprüfung der Zustellzeit in Prechecking Bridge und Checking Bridge zu

$$t_d \leq t_{now} + (H+2-h) \cdot \Delta_{Link} + (H+1-h) \cdot (\Delta_{Proc} + \Delta_{Busy}), \tag{7.4}$$

wobei $h$ den Wert des Hopzählers der zu übertragenen Nachricht und $t_{now}$ den aktuellen Zeitpunkt bezeichnet. Wie sich $\Delta_{Busy}$ in Abhängigkeit von gegebenen Topologien abschätzen lassen kann, wird in Kapitel 7.3.3 thematisiert.

### Signatur

Zuletzt wird über den gesamten Nachrichteninhalt der gängige CRC-32 (Generatorpolynom $0x04C11DB7$ [Koo02]), welcher unter anderem in Ethernet Verwendung findet, gebildet und im Anschluss mit dem Signaturverfahren *MsigOM* [Ech18] die Nachrichtensignatur erzeugt. Hierbei hätte in der Simulation auch jedes andere Signaturverfahren benutzt werden können bzw. man könnte auch auf ein konkretes Verfahren verzichten und den Mechanismus abstrahieren. Im Hinblick auf eine Implementierung auf echter Hardware, wurde die Wahl getroffen auch in der Simulation ein konkretes Verfahren zu benutzen, um Ergebnisse der Simulation und der Ausführung auf echter Hardware besser vergleichen zu können. Die Wahl auf *MSigOM* fiel deshalb, weil dieses Verfahren gute Eigenschaften bezüglich Fehlererkennung aufweist und keinen zusätzlichen Speicher zwingend benötigt.

### Empfang und Versand von Nachrichten

Auf diese Weise generierte Nachrichten werden über das Modul *egrPort* zu der mit dem Netzwerkknoten verbundenen Komponente gesendet und - falls eine vorherige Übertragung noch nicht abgeschlossen ist - in eine Warteschlange eingefügt. Letzterer Fall wird zwar

durch das Generierungsverfahren in der *src* verhindert, so dass die Warteschlange hinter dem *egrPort* stets leer sein sollte, wird im schlimmsten Fall dennoch erwartet, um mögliche Schwächen der Implementierung erkennen zu können.

Eingehende Nachrichten werden im Modul *ingPort* empfangen und unverändert an das Modul *sink* weitergeleitet. In diesem werden Nachrichten auf Korrektheit geprüft und im fehlerfreien Fall zugestellt. Eine Duplikaterkennung verhindert die mehrfache Zustellung redundanter Nachrichten über mehrere Wellen.

### 7.2.3. Bridges

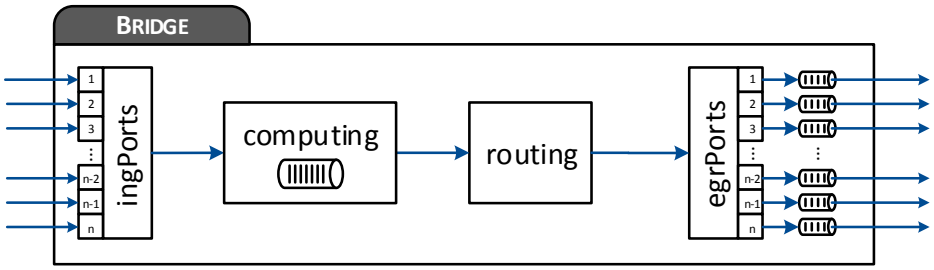



ABBILDUNG 7.3.: Aufbau einer Bridge in OMNeT++

Die Bridge ist so konzipiert, dass sie eine konfigurierbare Anzahl von Ports hat. Eingehende Nachrichten werden in einem der *n ingPorts* Module empfangen, welche abhängig von der Rolle der Bridge die Signaturmodifikation gemäß Protokollspezifikation durchführen. Anschließend werden Nachrichten in dem Modul *computing* zusammengeführt und eine Berechnungszeit gemäß Parametereinstellungen simuliert, realisiert durch Zwischenspeicherung in einer Warteschlange. Nach dem Ablaufen der Berechnungszeit werden Nachrichten an das Modul *routing* weitergeleitet und es wird gemäß den lokal bekannten Routing-Informationen die Menge der ausgehenden Ports bestimmt, auf welchen Kopien der Nachricht versendet werden sollen um sie anschließend an die korrekte Menge der ausgehenden Ports *egrPorts* weiterzuleiten. Jedem ausgehenden Port ist eine eigene Warteschlange zugeordnet, in welche Nachrichten einsortiert werden, falls die ausgehende Verbindung des jeweiligen Ports noch mit einer anderen Übertragung beschäftigt ist.

*Anmerkung:* Nachrichten werden erst dann weitergesendet, wenn sie vollständig empfangen wurden. In Kapitel 7.4 wird eine Duplikaterkennung in Bridges benötigt, welche den kompletten Inhalt der Nachricht benötigt, bevor eine Weiterleitung begonnen werden kann. Denkbar wäre, den Mechanismus zur Duplikaterkennung anzupassen, steht hier jedoch nicht im Fokus und wäre Inhalt von zukünftigen Arbeiten (siehe Kapitel 9.2).

### Routingformat

Die Informationen, gemäß derer Nachrichten weitergeleitet werden sollen, werden in Form einer Lookup-Tabelle organisiert, aus welcher abhängig von dem Sender der Nachricht, der initiierenden Checking Bridge der Welle, über welche die Verteilung stattfindet, sowie dem eingehenden Port der aktuellen Bridge die Menge der ausgehenden Ports für die weiterzuleitende Nachricht bestimmt wird.

| SenderID | CheckingBridgeId | ingPort | egrPorts |
|---|---|---|---|
| `int` | `int` | `int` | `int[]` |

Tabelle 7.4.: Aufbau der Routing-Lookup-Tabelle

## 7.2.4. Statistische Datenerfassung

Während der Simulationsdurchläufe, werden statistische Daten gesammelt, welche anschließend ausgewertet werden sollen. Nachfolgende Tabelle gibt einen detaillierten Überblick über die Art sowie den Ort, an dem statistische Daten erfasst werden. Zusätzliche, kürzere Bezeichnungen (blau) im Namen werden im Laufe dieser Arbeit in Tabellen und Graphen verwendet.

- *Name:* `NumTx` (#$\mathsf{Msgs_{sent}}$)
  *Typ:* `Scalar`
  *Ort:* `Node.src`
  *Beschreibung:* Anzahl versendeter FABAN-Nachrichten.
- *Name:* `NumRx` (#$\mathsf{Rx}$)
  *Typ:* `Scalar`
  *Ort:* `Node.sink`
  *Beschreibung:* Anzahl empfangener FABAN-Nachrichten, unabhängig davon, ob diese verfälscht wurden oder Duplikate sind.
- *Name:* `NumSuccessfulBroadcasts` (#$\mathsf{Bc_{Succ}}$)
  *Typ:* `Scalar`
  *Ort:* `Node.sink`
  *Beschreibung:* Anzahl erfolgreich empfangener FABAN-Rundsprüche.
- *Name:* `NumFfDuplicates` (#$\mathsf{Dup_{FF}}$)
  *Typ:* `Scalar`
  *Ort:* `Node.sink`
  *Beschreibung:* Anzahl empfangener fehlerfreie Duplikate von FABAN-Nachrichten.

- *Name:* `NumDiscardedDelay` ($\#\mathrm{Drop}_{\mathrm{Delay}}$)
  *Typ:* `Scalar`
  *Ort:* `Node.sink`
  *Beschreibung:* Anzahl verworfener FABAN-Nachrichten aufgrund von verspätetem Eintreffen.
- *Name:* `NumDiscardedFaulty` ($\#\mathrm{Drop}_{\mathrm{BF}}$)
  *Typ:* `Scalar`
  *Ort:* `Node.sink`
  *Beschreibung:* Anzahl verworfener FABAN-Nachrichten, aufgrund von erkanntem verfälschten Nachrichteninhalt.
- *Name:* `RxTimes`
  *Typ:* `Histogram`
  *Ort:* `Node.sink`
  *Beschreibung:* Histogramm über die Transferzeiten von FABAN-Nachrichten, gemessen durch das Zeitintervall zwischen Versand und Empfang.
- *Name:* `DiffToDeliveryTime`
  *Typ:* `Histogram`
  *Ort:* `Node.sink`
  *Beschreibung:* Histogramm über die Zeitdifferenzen empfangener FABAN-Nachrichten zwischen Verarbeitungs- und Empfangszeitpunkt im jeweiligen Empfänger. Positive Werte bedeuten einen Empfang vor der Verarbeitungszeit der Nachricht im Empfänger.
- *Name:* `NumRxFromSender`
  *Typ:* `Histogram`
  *Ort:* `Node.sink`
  *Beschreibung:* Anzahl empfangener FABAN-Nachrichten, unabhängig davon ob diese verfälscht wurden oder Duplikate sind, differenziert nach dem Sender der Nachrichten.
- *Name:* `QueueLoad`
  *Typ:* `Vector`
  *Ort:* `Node.egrQueue, Bridge.computing, Bridge.egrQueue`
  *Beschreibung:* Auslastung der jeweiligen Queue über die Zeit, gemessen in Bits und in der Anzahl der Nachrichten.
- *Name:* `QueueingTime`
  *Typ:* `Histogram`
  *Ort:* `Node.egrQueue, Bridge.computing, Bridge.egrQueue`
  *Beschreibung:* Histogramm über die Verweilzeiten von Nachrichten in der jeweiligen Queue.

- *Name:* `TimeCheckFailed`
  *Typ:* `Scalar`
  *Ort:* `Bridge.routing`
  *Beschreibung:* Anzahl unterdrückter Nachrichten aufgrund des Fehlschlagens der Überprüfung der Zustellzeit in einer Checking Bridge.

### 7.2.5. Fehlerinjektionen

Die Implementierung der Fehlerinjektionsmechanismen erfolgt ausschließlich im Modul der Bridge. Es werden alle Fehlerarten, welche in der Fehlerklassifizierung in Kapitel 2.5 thematisiert und in Abbildung 2.6 hierarchisch visualisiert wurden, umgesetzt mit den folgenden Ausnahmen:

- *Fail-Stop-Fehler* stellen eine Untermenge der *Ausfallfehler* dar, bei denen davon ausgegangen werden darf, dass alle betroffenen Komponenten Kenntnis des Eintretens des Fehlers haben. Diese Kenntnis ist für FABAN nicht von Relevanz und macht die separate Injektion von *Fail-Stop-Fehlern* überflüssig.
- *Datenmanipulation* unterscheidet sich von einem *Bitflip* lediglich in der Ursache des Fehlers. Während nach der in dieser Arbeit vorgenommenen Klassifizierung von Fehlern *Bitflips* auf technische Ursachen (physikalische Einflüsse, defekte Kabel, Strahlung) zurückzuführen sind, ist die Datenmanipulation auf Fehler in der Logik der Komponenten zurückzuführen. Da die Ursache bei der Modellierung keine Relevanz hat, wird davon abgesehen, die Datenmanipulation separat zu implementieren.
- Der Effekt der *Maskerade* wird bei passender Parametrisierung vollständig durch den Fehler der *Feldmanipulation* abgebildet, so dass auch hier auf eine separate Implementierung verzichtet wird.
- *Nicht feststellbare byzantinische Fehler* ist eine nicht modellierbare Menge von Fehlern, welche es experimentell herauszufinden gilt. Hiervon ausgenommen sind solche Nachrichtenverfälschungen, welche durch die Signatur natürlicherweise nicht erkannt werden können. Des Weiteren können in der Realität solche Fehler durch aufwändigere Signaturverfahren (fast) beliebig unwahrscheinlich gemacht werden. Deshalb ist es auch nicht von Interesse, diese nicht zu tolerierende Menge von Fehlern zu modellieren.

Die Realisierung der Fehlerinjektionsmechanismen erfolgte, soweit möglich, in einem separaten Fehlerinjektionsmodul (vgl. Abbildung 7.4 Modul *FI*), welches zwischen das *computing*- und das *routing*-Modul platziert wurde. Die einzige Ausnahme stellt hierbei die Fehlerart *Fehlerhafte Weiterleitung* dar. Bei dieser ist es zwingend notwendig, Änderungen

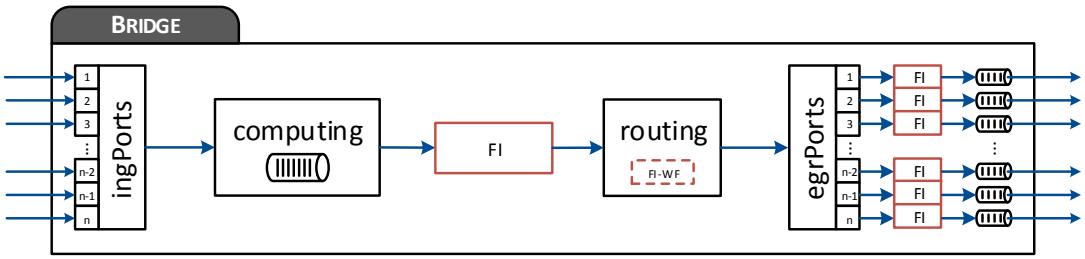



Abbildung 7.4.: Bridge Modul erweitert um Fehlerinjektionsmechanismen. Modulinstanzen mit der Bezeichnung *FI* sind Instanzen der selben Implementierung und sind verantwortlich für die Injektionen fast aller Fehlertypen. Die Ausnahme ist der Fehlertyp *Fehlerhafte Weiterleitung*, welcher als fester Bestandteil der Moduls *routing* implementiert wurde.

im Modul *routing* vorzunehmen, in welchem die korrekte Menge der ausgehenden Ports, auf welchen die Nachricht weitergeleitet werden soll, bestimmt wird und diese im Fehlerfall zu manipulieren (vgl. Abbildung 7.4 *FI-WF*). Um byzantinisches Verhalten zu generieren, wurde zwischen jeden ausgehenden Port und zugehöriger Nachrichtenwarteschlange jeweils eine weitere Instanz des selben Fehlerinjektionsmoduls zusätzlich platziert, um auf diese Weise portunabhängiges Fehlverhalten zu generieren. Änderungen an der Funktionsweise waren hierbei nicht nötig. Abbildung 7.4 visualisiert den resultierenden Aufbau der Bridge mit eingebauten Fehlerinjektionsmechanismen.

Über die Parameter

$$\texttt{**.faultinjector.active} \in \{\texttt{true,false}\} \tag{7.5}$$

$$\texttt{**.portFaultinjector[*].active} \in \{\texttt{true,false}\} \tag{7.6}$$

kann zunächst allgemein für jede Bridge individuell bzw. für Teilmengen von Bridges die Funktionalität der Fehlerinjektionsmodule aktiviert bzw. deaktiviert werden. Damit kann insbesondere die Menge der fehlerhaften Bridges sowie byzantinisches Verhalten gesteuert werden.

**Konfiguration 7.1** Ausschnitt einer Konfigurationsdatei für byzantinisches Verhalten

```
1 **.b[0..1].faultinjector.active = true
2 **.b[0].portFaultinjector[*].active = true
3 **.b[1].portFaultinjector[*].active = false
```

Konfiguration 7.1 zeigt als Beispiel einen Ausschnitt einer OMNeT++ Konfiguration, die folgendes festlegt:

- Bei Bridge $b[0]$ ist byzantinisches Fehlverhalten aktiviert, so dass sowohl im zentralen FI-Modul als auch in den FI-Modulen der ausgehenden Ports unabhängige

Fehlerinjektion stattfindet. Verbundene Bridges könnten vollkommen unterschiedliche Verfälschungen ein und derselben Nachricht erhalten.

- Bei Bridge $b[1]$ ist byzantinisches Fehlverhalten deaktiviert, so dass nur im zentralen FI-Modul die Fehlerinjektion stattfindet und folglich jegliche Verfälschung einer Nachricht bei allen unmittelbaren Empfängern konsistent ist.

Spezifiziert wird die Injektion der Fehler in den Fehlerinjektionsmodulen über eine Reihe von weiteren Parametern, welche im Folgenden im Rahmen eines detaillierten Überblicks über die Art und Weise der Implementierung der jeweiligen Fehlerart sowie über die Möglichkeiten ihrer Parametrisierungen beschrieben werden. Hierbei ist anzumerken, dass Parameter in OMNeT++ nicht auf konstante Werte beschränkt sind, sondern auch Zufallsgrößen annehmen können, d.h. man kann explizit Wahrscheinlichkeitsverteilungen angeben, gemäß derer zufällige Werte bestimmt werden. OMNeT++ unterstützt dabei die wichtigsten und gängigsten Wahrscheinlichkeitsverteilungen, welche dem offiziellen *Simulation Manual* zu entnehmen sind und auf welche hier nicht näher eingegangen wird. Kann einer der nachfolgenden Parameter eine Zufallsvariable sein, so wird diese im Folgenden mit $X$ oder $Y$ gekennzeichnet, wobei lediglich der Wertebereich der Zufallsvariable beachtet werden muss, falls dieser eingeschränkt ist [Ope20a].

### Ausfallfehler

Die Eintrittswahrscheinlichkeit eines *Ausfalls* einer Bridge wird durch den Parameter

$$\texttt{**.failurePropCrash} \in [0, 1] \tag{7.7}$$

festgelegt. Tritt der Ausfallfehler in einer Bridge ein, so werden ab dem Zeitpunkt des Fehlereintritts alle Nachrichten, welche an allen eingehenden Ports der betroffenen Bridge eintreffen, verworfen.

### Unterlassungsfehler

Die Eintrittswahrscheinlichkeit einer *Unterlassung* einer Nachricht wird durch den Parameter

$$\texttt{**.failurePropOmission} \in [0, 1] \tag{7.8}$$

festgelegt. Mit dieser Wahrscheinlichkeit wird in fehlerhaften Bridges im Modul *FI* für jede weiterzuleitende Nachricht festgelegt, ob diese unterdrückt oder korrekt weitergeleitet werden soll.

### Verzögerungsfehler

Die Eintrittswahrscheinlichkeit einer zeitlichen *Verzögerung* einer Nachricht wird durch den Parameter

$$\texttt{**.failurePropDelay} \in [0,1] \tag{7.9}$$

festgelegt. Mit dieser Wahrscheinlichkeit wird in fehlerhaften Bridges im Modul *FI* für jede weiterzuleitende Nachricht festgelegt, ob diese verzögert werden soll, wobei das Ausmaß möglicher Verzögerungen als Zufallsgröße durch den Parameter

$$\texttt{**.failureParamDelayDF} = X \geq 0 \tag{7.10}$$

bestimmt wird.

### Bitflips

Die Eintrittswahrscheinlichkeit von *Bitfehlern* einer Nachricht wird durch den Parameter

$$\texttt{**.failurePropBitflip} \in [0,1] \tag{7.11}$$

festgelegt. Mit dieser Wahrscheinlichkeit wird in fehlerhaften Bridges im Modul *FI* für jede weiterzuleitende Nachricht festgelegt, ob Bits verfälscht werden sollen, wobei die Anzahl der verfälschten Bits als Zufallsgröße durch den Parameter

$$\texttt{**.failureParamBitflipNumber} = X \in \mathbb{N}_0 \tag{7.12}$$

bestimmt wird.

### Feldmanipulation

Die Eintrittswahrscheinlichkeit von *Feldmanipulationsfehlern* wird durch den Parameter

$$\texttt{**.failurePropFieldManipulation} \in [0,1] \tag{7.13}$$

festgelegt. Mit dieser Wahrscheinlichkeit wird in fehlerhaften Bridges im Modul *FI* für jede weiterzuleitende Nachricht festgelegt, ob Feldmanipulationen stattfinden sollen, wobei die Anzahl der zu verfälschenden Felder als Zufallsgröße durch den Parameter

$$\texttt{**.failureParamFieldManipulationNumber} = X \in \mathbb{N}_0 \tag{7.14}$$

bestimmt wird. Zusätzlich lässt sich durch den konstanten Parameter

$$\texttt{**.failureParamFieldManipulationNumber} = \text{"}x_1x_2x_3x_4x_5x_6x_7\text{"} \tag{7.15}$$

einschränken, welche Felder von Fehlern betroffen sein dürfen (x=1) und welche nicht (x=0). Der Index des jeweiligen Indikators $x_i$, $1 \leq i \leq 7$ entspricht der Reihenfolge der Felder des Nachrichtenaufbaus in Abbildung 6.4. Die Verfälschung der Inhalte der betroffenen Felder wurden abhängig vom Typ des jeweiligen Feldes wie folgt umgesetzt:

- Die Sender ID und die Checking Bridge ID werden ersetzt durch eine andere, valide Sender ID bzw. Checking Bridge ID, so dass die Nachricht einem anderen Sender/Checking Bridge-Paar zugeordnet wird.
- Die Zustellzeit wird multipliziert mit einer zufälligen, normalverteilten Zufallsvariable $x \sim \mathcal{N}(1,4)$ mit Erwartungswert $\mu = 0$ und Varianz $\sigma^2 = 4$, welche auf positive Zahlen beschränkt wird, das heißt bei negativem Ergebnis wird stets eine neue Zufallszahl bestimmt. Dadurch wird sichergestellt, dass die Erhöhung bzw. Verringerung der Zustellzeit mit (fast) gleicher Wahrscheinlichkeit geschieht und nicht zu stark verändert wird.
- Die Sequenznummer sowie der Hopzähler werden um $1 + X$, mit $X \sim Poi(1)$, mit gleicher Wahrscheinlichkeit erhöht oder verringert. Die Poisson-Verteilung ist eine diskrete Verteilung für seltene Ereignisse, das heißt die Wahrscheinlichkeitsmasse verteilt sich ähnlich nah um den Erwartungswert herum, wie es zum Beispiel bei der Normalverteilung der Fall ist. Aus diesem Grund eignet sich deshalb diese Verteilung mit dem Parameter $\lambda = 1$ besonders um Fehler nachzubilden, welche kritische Beachtung bei kleinen Veränderungen erfordern bzw. bei denen größere Veränderungen zunehmend unwahrscheinlicher werden.
- Der Inhalt der Signatur bzw. des Datenfeldes wird vollständig zufällig neu gewürfelt.

### Duplizierung

Die Eintrittswahrscheinlichkeit eines *Duplizierungsfehlers* einer Nachricht wird durch den Parameter

$$\texttt{**.failurePropDuplication} \in [0,1] \tag{7.16}$$

festgelegt. Mit dieser Wahrscheinlichkeit wird in fehlerhaften Bridges im Modul *FI* für jede weiterzuleitende Nachricht festgelegt, ob die weiterzuleitende Nachricht in mehrfacher Ausführung weitergeleitet werden soll, wobei die Anzahl der zusätzlichen Duplikate als Zufallsgröße durch den Parameter

$$\texttt{**.failureParamDuplicationNumber} = X \in \mathbb{N}_0 \tag{7.17}$$

bestimmt wird.

### Fehlerhafte Weiterleitung

Die Eintrittswahrscheinlichkeit einer *Fehlerhaften Weiterleitung* einer Nachricht wird durch den Parameter

$$\texttt{**.failurePropWrongForwarding} \in [0,1] \tag{7.18}$$

festgelegt. Mit dieser Wahrscheinlichkeit wird in fehlerhaften Bridges im Modul *routing* bestimmt, ob eine fehlerhafte Weiterleitung stattfinden soll. Für eine weiterzuleitende Nachricht $m$ tritt in jedem Port unabhängig voneinander mit Wahrscheinlichkeit 0.5 der Fehler wie folgt ein:

- muss $m$ über den Port gesendet werden, so wird der Sendevorgang unterdrückt
- muss $m$ über den Port gemäß des ExFABAN-Routings nicht gesendet werden, so wird eine Kopie von $m$ gesendet.

### Babbling Component

Die Eintrittswahrscheinlichkeit eines Fehlers der Art *Babbling Component* wird durch den Parameter

$$\texttt{**.failurePropBabblingIdiot} \in [0,1] \tag{7.19}$$

festgelegt. Mit dieser Wahrscheinlichkeit wird in fehlerhaften Bridges im Modul *FI* bestimmt, ob eine Menge von zufällig generierten Nachrichten mit vollständig zufälligem Inhalt erzeugt und versendet werden soll. Die Anzahl der Nachrichten wird hierbei als Zufallsgröße durch den Parameter

$$\texttt{**.failureParamBabblingComponentNumber} = X \in \mathbb{N}_0 \tag{7.20}$$

und die zeitliche Differenz zwischen einzelnen Nachrichten als Zufallsgröße durch den Parameter

$$\texttt{**.failureParamBabblingComponentTimeInterval} = Y \geq 0 \tag{7.21}$$

bestimmt.

### Fehlerhafte Signaturmodifikation

Neben der zufälligen Verfälschung von Signaturen, welche bereits durch die Fehlerart *Feldmanipulation* abgedeckt wird, kann es Sinn machen, Signaturen systematisch gemäß der in Kapitel 3.1.2 bzw. in Kapitel 6.2.2 beschriebenen Methoden unerlaubt zusätzlich zu verändern um damit die Signaturmodifikationen zu validieren. Die Eintrittswahrscheinlichkeit dieser Fehlerart wird durch den Parameter

$$\texttt{**.failurePropAddValidSignManipulation} = X \in [0,1] \tag{7.22}$$

festgelegt, wobei die konkrete Art der Veränderung gleichverteilt aus folgender Liste von möglichen Signaturmodifikationen ausgewählt wird:

- Rotieren der Signatur um 1 Bit nach Links
- XOR-Verknüpfung mit der D-Maske
- XOR-Verknüpfung mit der C-Maske
- XOR-Verknüpfung mit der P-Maske (im Fall $f \geq 2$)

Die Anzahl der zusätzlichen Signaturmodifikationen kann als Zufallsgröße durch folgenden Parameter konfiguriert werden:

$$\texttt{**.failureParamAddValidSignManipulationNumber} = X \in \mathbb{N}. \tag{7.23}$$

Alle zuvor aufgeführten Parameter können mit OMNeT++ eigenen Mitteln individuell für einzelne Komponenten eingestellt werden. Ein Beispiel ist in Konfiguration 7.2 gegeben, gemäß welcher die beiden Bridges $b[0]$ und $b[1]$ byzantinisches Fehlverhalten aufweisen dürfen. Dabei können die Bridges vollständig ausfallen, einzelne Nachrichten unterdrücken, verfälschen, verzögern und/oder zusätzliche Signaturmodifikationen durchführen. Mit Ausnahme des Ausfallfehler sind alle anderen Fehlerarten für beide Bridges gleich parametrisiert.

**Konfiguration 7.2** Ausschnitt einer Konfigurationsdatei

```
1  **.b[0].faultinjector.active = true
2  **.b[0].portFaultinjector[*].active = true
3  **.b[7].faultinjector.active = true
4  **.b[7].portFaultinjector[*].active = true

6  **.b[0].failurePropCrash = 0.001
7  **.b[1].failurePropCrash = 0.1
8  **.failurePropOmission = 0.1
9  **.failurePropBitflip = 0.1
10     **.failureParamBitflipNumber = intuniform(1, 10)
11 **.failurePropDelay = 0.5
12     **.failureParamDelayDF = uniform(0s, 0.1s)
13 **.failurePropAddValidSignManipulation = 0.1
```

Eine komplette syntaktische Erklärung davon ist hier nicht von Relevanz und wird deshalb weggelassen.

## 7.3. Evaluation durch Simulationen für Einfachfehler

Mit Hilfe von systematisch durchgeführten Simulationen werden im Nachfolgenden Analysen mit unterschiedlichem Fokus durchgeführt. Bevor darauf näher eingegangen wird, wird ein Überblick über die zugrunde gelegten Netzwerktopologien gegeben, sowie deren Wahl begründet.

Zunächst wird mit Hilfe von Simulationen obligatorisch verifiziert, dass die Implementierung des Protokolls korrekt umgesetzt wurde. Dabei werden gezielt einzelne Mechanismen, die im Protokoll Verwendung finden, in den Fokus gestellt und geeignete Simulationsszenarien untersucht.

Im Anschluss daran wird analysiert, inwieweit die im vorherigen Abschnitt eingeführte Zeitdifferenz $\Delta_{Busy}$ abgeschätzt werden kann. Letzteres wird hierbei nicht ausschließlich experimentell ermittelt, sondern unterschiedliche Ansätze werden miteinander verglichen und bewertet, sowie Lösungen vorgestellt, welche mit Hilfe von Simulationen konsolidiert werden.

Simulationen mit der Injektion von Fehlern sollen anschließend in großem Umfang die Fehlertoleranzeigenschaften von FABAN überprüfen und das Ausmaß der Fehlertoleranz bekräftigen. Hierbei sollen zusätzlich mögliche Probleme in der Protokollspezifikation oder der Umsetzung erkannt und gegebenenfalls gelöst werden. Zunächst wird die Injektion von Einfachfehlern betrachtet, bevor anschließend die Erweiterung der Betrachtung auf Mehrfachfehler ausgeweitet wird.

### 7.3.1. Netzwerktopologien

#### Ring

Als einfachste Variante einer Netzwerktopologie mit redundanten Pfaden für die Toleranz von Einfachfehlern bietet sich die Klasse der Ringe an. Neben der maximalen Anzahl von Hops auf dem Weg vom Sender zu jedem Empfänger, ist auch ein hoher Grad von Kollisionen von Nachrichten auf einzelnen Verbindungen zu erwarten. Dies liegt daran, dass jede FABAN-Nachricht auf jeder Verbindung zwischen benachbarten Bridges exakt zwei mal gesendet wird, je eine Nachricht in jeder der beiden Wellen, und somit bei hohem Nachrichtenaufkommen Kollisionen nicht zu vermeiden sind. In folgenden Beschreibungen und Auswertungen werden Ringe, bestehend aus $n$ Bridges mit der Notation `Ring(n)` bezeichnet.

### Vollvermaschung

Als Klasse von Netzwerktopologien mit maximaler Verbindungsanzahl bietet sich diese an um mögliche andere Erkenntnisse zu gewinnen. Insbesondere die Tatsache, dass zwischen jedem Bridgepaar eine eigene physikalische Verbindung existiert, sollte Kollisionen aufgrund hohem Nachrichtenverkehr erheblich reduzieren. Lediglich auf Verbindungen zwischen Bridges und Empfängerknoten sowie aufgrund gemeinsam gewählter Checking Bridges für unterschiedliche Distributing Bridges dürften noch Kollisionen auftreten. Diese Klasse von Netzwerktopologien eignet sich hierbei für die Toleranz von beliebig vielen Fehlern bei genügend vielen Bridges. Vollvermaschte Graphen, bestehend aus $n$ Bridges, werden im Folgenden als `Vollvermascht(n)` bezeichnet.

### Ringnetz

Die größte Nähe zu realen Anwendungen hat die in Kapitel 6.3.2 eingeführte Klasse der *Ringnetze*. Durch flexible Verbindungen von einzelnen Ringen gleicher oder unterschiedlicher Größe, lassen sich skalierbare und an die Anforderungen anpassbare redundante Netzwerktopologien konstruieren. Hier lassen sich Beispiele in diversen Anwendungsgebieten finden, angefangen bei (großen) Firmengebäuden, in denen einzelne Abteilungen/Etagen/Bereiche in einzelnen Ringen vernetzt und untereinander zu einem Ringnetz verbunden werden, bis hin zu sicherheitskritischen verteilten Systemen mit Echtzeitanforderungen wie sie beispielsweise in Autos, Flugzeugen, Industrieanlagen, Kernkraftwerken oder im medizinischen Bereich zu finden sind [Kop+89; Dav+07]. Ringnetze lassen sich nicht durch eine einfache Notation wie bei Ringen oder vollvermaschten Netzen beschreiben, so dass aus diesem Grund die in den Abbildungen 7.6, 7.7 sowie 7.8 dargestellten Ringnetze unterschiedlicher Größe in nachfolgenden Simulation verwendet werden, wobei die verwendete Darstellungsart in Abbildung 7.5 verdeutlicht wird. Dabei wurden aus technischen Gründen Ringe derart miteinander verbunden, dass das Ergebnis stets eine zusätzliche Verbindung im Vergleich zu einem Ringnetz aufweist, diese jedoch beim Routing nie Verwendung findet, so dass die verwendeten Topologien effektiv Ringnetzen entsprechen.

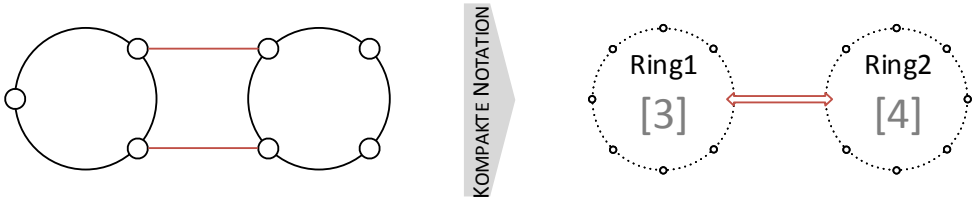



Abbildung 7.5.: Kompakte Darstellung von Ringnetzen mit einer zusätzlichen, ungenutzten Verbindung. Die Anzahl der Bridges in einem Ring wird in eckigen Klammern angegeben.


Abbildung 7.6.: Netz aus drei Ringen mit insgesamt 10 Bridges (*Ringnetz10*). Verbindungen zwischen zwei benachbarten Ringen bestehen stets zwischen benachbarten Bridgepaaren der jeweiligen Einzelringe.

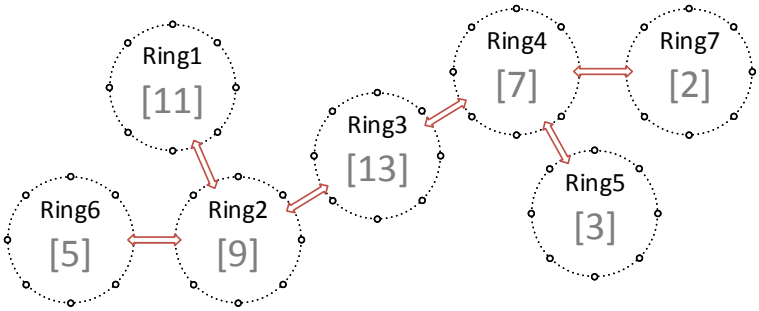



Abbildung 7.7.: Netz aus sieben Ringen mit insgesamt 50 Bridges (*Ringnetz50*). Verbindungen zwischen zwei benachbarten Ringen bestehen stets zwischen benachbarten Bridgepaaren der jeweiligen Einzelringe.

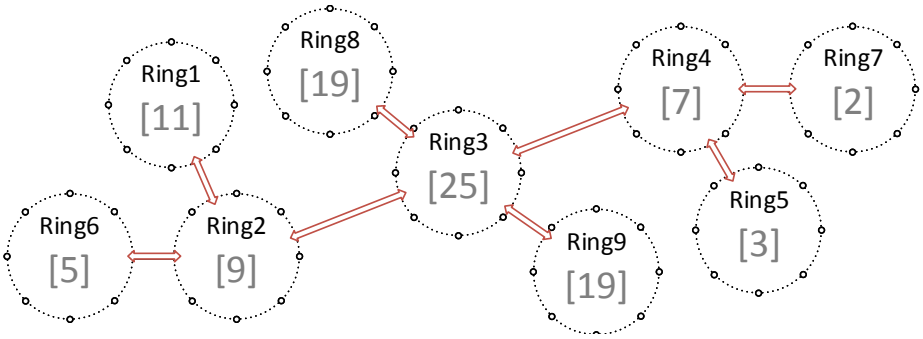



Abbildung 7.8.: Netz aus neun Ringen mit insgesamt 100 Bridges (*Ringnetz100*). Verbindungen zwischen zwei benachbarten Ringen bestehen stets zwischen benachbarten Bridgepaaren der jeweiligen Einzelringe.

### 7.3.2. Validierung der Implementierung

In diesem Unterkapitel wird mit Hilfe von Simulationen die obligatorische Richtigkeit der Implementierung des Protokolls validiert. Hierbei werden eine Ringtopologie, ein vollvermaschtes Netz sowie ein Ringnetz mit jeweils 50 Bridges, mit welchen jeweils exakt ein Netzwerkknoten verbunden sind, und unterschiedlichen Konfigurationen simuliert, um aus den gesammelten Ergebnissen die Korrektheit einzelner vom Protokoll verwendeter Schlüsselmechanismen zu folgern. Der grundlegende Ablauf von Simulationen und Auswertungen ist in Abbildung 7.9 dargestellt. Der in OMNeT++ geschriebene *FabanSimulator* benötigt für die Durchführung von Simulationen eine Netzwerktopologie (**.ned*-Datei) sowie eine zugehörige Konfiguration der Parameter (**.ini*-Datei). Während der Simulation werden Daten gesammelt (beschrieben in Kapitel 7.2.4) und in einem OMNeT++-eigenen Format (**.sca*- und **.vec*-Dateien) gespeichert. Diese Dateien lassen sich in OMNeT++ öffnen und simple Resultate daraus ablesen. Für eine komplexere Analyse von großen Datenmengen wurde zusätzlich in der Statistiksoftware *R* ein Script geschrieben, welches die von OMNeT++ erzeugten Dateien einliest und daraus Diagramme erstellt und in Form von PDF Dateien ausgibt. Jegliche Daten bzw. Diagramme, die im Folgenden in dieser Arbeit zu finden sind, sind Ergebnisse, die auf diese beiden Arten gewonnen wurden.

Die nachfolgenden Simulationsdurchläufe haben eine gemeinsame Basis von Parametern, um eine möglichst gute Vergleichsgrundlage verschiedener Simulationen zu erhalten. Zusätzlich ist die Parametrisierung so gewählt, dass möglichst einfache sowie eindeutige Ergebnisse entstehen. In allen Simulationen existiert exakt ein einzelner Sender, welcher insgesamt 100 FABAN-Nachrichten versendet. Die zeitliche Differenz zwischen zwei aufeinanderfolgenden Sendevorgängen ist gleichverteilt auf dem Intervall $(0.001s, 0.003s)$ und die Größe der FABAN-Nachrichten beträgt exakt $125byte$ (Datengröße $107byte$). Da es bei diesen ersten Simulationsdurchläufen nur um die Überprüfung reiner Funktionalitäten und nicht um Performanz geht, wird die berechnete Verarbeitungszeit der FABAN-Nachrichten bei den Empfängern verdoppelt und die Warteschlangengröße in den Bridges auf $1MB$ gesetzt. Letztere spielt hier keine weitere Rolle, da Warteschlangen stets leer

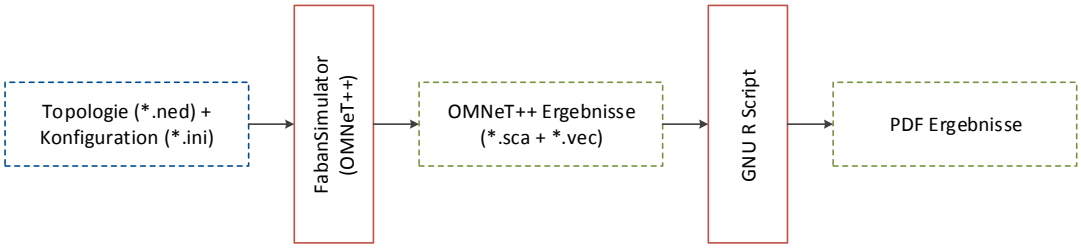



Abbildung 7.9.: Darstellung der Simulationstoolchain zur Gewinnung von statistischen Ergebnissen mit OMNeT++ und GNU R

bleiben aufgrund des hohen Sendeintervalls. Ebenfalls anzumerken ist, dass der Großteil der Simulationen dieses Abschnitts - Ausnahmen werden eindeutig gekennzeichnet - jeweils nur einmal durchgeführt wurden. Ein größerer Stichprobenumfang ist nicht erforderlich. Dies ist aus dem Grund ausreichend, weil der einzige Zufallsfaktor in der Parametrisierung einen genügend großen Mindestwert aufweist, so dass es zu keinen Einflüssen zwischen vielen einzelnen Sendevorgängen, die nacheinander ausgeführt werden, kommen kann. Tabelle 7.5 fasst die globalen Parameter nochmal zusammen.

| Parameter | Wert |
|---|---|
| `**.sendIntervall` | `uniform(0.001s, 0.003s)` |
| `**.dataSize` | `107byte` |
| `**.deliveryTimeFactor` | `2` |
| `**.maxMessages` | `100` |
| `**.dataRate` | `1000Mbps` |
| `**.queuesize` | `1MB` |

Tabelle 7.5.: Parameterübersicht der Simulationen zur Validierung der Richtigkeit der Protokollimplementierung.

Die erste Menge von Simulationen zielt darauf ab, das grundlegende Prinzip der redundanten Verteilung, initiiert von der Distributing Bridge, zu überprüfen. Tabelle 7.6 enthält die Ergebnisse der Simulationen und zeigt, dass jede FABAN-Nachricht dupliziert und von allen Empfängern in doppelter Ausführung empfangen wurde.

| Topologie | $\#\mathsf{Msgs}_{\mathsf{Sent}}$ | $\#\mathsf{Rx}$ | $\#\mathsf{Bc}_{\mathsf{Succ}}$ | $\#\mathsf{Dup}_{\mathsf{FF}}$ | $\#\mathsf{Drop}_{\mathsf{Delay}}$ | $\#\mathsf{Drop}_{\mathsf{BF}}$ |
|---|---|---|---|---|---|---|
| Ring(50) | 100 | 200 | 100 | 100 | 0 | 0 |
| Vollvermascht(50) | 100 | 200 | 100 | 100 | 0 | 0 |
| Ringnetz(50) | 100 | 200 | 100 | 100 | 0 | 0 |

Tabelle 7.6.: Ergebnisse von Simulationen für drei unterschiedliche Topologien mit jeweils einem Sender und ohne Injektion von Fehlern. Die Ergebnisse sind bei allen Empfängern identisch. *Hinweis:* Die Kürzel zur Benennung der Spalten wurden in Abschnitt 7.2.4 definiert.

Verfälschte Nachrichten werden mit Hilfe von Signaturen erkannt und gegebenenfalls bei den Empfängerknoten verworfen. Um das implementierte Signaturverfahren auf grundlegende Funktionsfähigkeit zu testen, wurde in eine Checking Bridge ein Fehler mit einer Wahrscheinlichkeit von 100% injiziert, der ein zufälliges Bit des Datenfeldes der weiterzuleitenden Nachricht toggelt. Die Ergebnisse in Tabelle 7.7 zeigen, dass alle Empfänger, die an fehlerfreie Bridges angeschlossen sind, 100 korrekte Nachrichten

empfangen und zugestellt haben. Die fehlerhaften Kopien der Nachrichten der redundanten Welle wurden ebenfalls empfangen, aber als fehlerhaft erkannt und in Folge dessen verworfen.

| Topologie | $\#\text{Msgs}_{\text{Sent}}$ | $\#\text{Rx}$ | $\#\text{Bc}_{\text{Succ}}$ | $\#\text{Dup}_{\text{FF}}$ | $\#\text{Drop}_{\text{Delay}}$ | $\#\text{Drop}_{\text{BF}}$ |
|---|---|---|---|---|---|---|
| Ring(50) | 100 | 200 | 100 (0) | 0 | 0 | 100 (200) |
| Vollvermascht(50) | 100 | 200 | 100 (0) | 0 | 0 | 100 (200) |
| Ringnetz(50) | 100 | 200 | 100 (0) | 0 | 0 | 100 (200) |

TABELLE 7.7.: Ergebnisse von Simulationen für drei unterschiedliche Topologien mit jeweils einem Sender und einer konstant fehlerhaften Checking Bridge (Bitflips). Die Ergebnisse sind bei allen Empfängern, die mit fehlerfreien Bridges verbunden sind, identisch. Die roten Werte sind die Ergebnisse des Empfängers, der mit der fehlerhaften Checking Bridge verbunden ist.

Die Erkennung von Verzögerungen ist essentiell sowohl in der Checking Bridge als auch bei den Empfängern. Für die Überprüfung beider Mechanismen, wurden wiederholt Fehler in eine der beiden Checking Bridges der zum Sender zugehörigen Distributing Bridge injiziert. Zum einen wird jede Nachricht zentral in der Checking Bridge vor dem *Routing* mit einer Wahrscheinlichkeit von 50% um $1s$ und zum anderen an allen ausgehenden Ports mit einer Wahrscheinlichkeit von 100% ebenfalls um $1s$ verzögert. Während die erste Injektion zur Folge hat, dass die Hälfte der Nachrichten von der Checking Bridge unterdrückt werden muss, weil diese nicht mehr alle Empfänger rechtzeitig erreichen kann, führt die Injektion an den ausgehenden Ports dazu, dass die Nachricht bei den Empfängern zu spät eintrifft und von diesen verworfen werden muss. Tabelle 7.8 bestätigt diese Erwartung auf Basis von Ergebnissen von 100 Wiederholungen des Simulationslaufs.

| Topologie | $\#\text{Msgs}_{\text{Sent}}$ | $\#\text{Rx}$ | $\#\text{Bc}_{\text{Succ}}$ | $\#\text{Dup}_{\text{FF}}$ | $\#\text{Drop}_{\text{Delay}}$ | $\#\text{Drop}_{\text{BF}}$ |
|---|---|---|---|---|---|---|
| Ring(50) | 100 | 150.8 | 100 (0) | 0 | 50.8 (150.8) | 0 |
| Vollvermascht(50) | 100 | 149.9 | 100 (0) | 0 | 49.9 (149.9) | 0 |
| Ringnetz(50) | 100 | 149.8 | 100 (0) | 0 | 49.8 (149.8) | 0 |

TABELLE 7.8.: Ergebnisse von Simulationen für drei unterschiedliche Topologien mit jeweils einem Sender und einer konstant fehlerhaften Checking Bridge (Verzögerung). Die Ergebnisse sind bei allen Empfängern, die mit fehlerfreien Bridges verbunden sind, identisch. Die roten Werte sind die Ergebnisse des Empfängers, der mit der fehlerhaften Checking Bridge verbunden ist.

Die Duplikaterkennung in den Empfängerknoten ist dafür verantwortlich, dass eine redundant verteilte Nachricht nicht mehrfach verarbeitet wird, das heißt dass auch in Abwesenheit von Fehlern diese Funktionalität bei jeder Übertragung benötigt wird. Um auch die Erkennung der fehlerhaften Duplizierung von Nachrichten zu überprüfen, wurde in eine der beiden Checking Bridges ein Fehler derart injiziert, dass eine weiterzuleitende

Nachricht stets doppelt geschickt wird. Tabelle 7.9 fasst die Ergebnisse zusammen, welche zeigen, dass jede Nachricht bei allen Empfängern in dreifacher bzw. vierfacher (Empfänger an der fehlerhaften Checking Bridge) Ausführung eintreffen.

| Topologie | #Msgs$_{\text{Sent}}$ | #Rx | #Bc$_{\text{Succ}}$ | #Dup$_{\text{FF}}$ | #Drop$_{\text{Delay}}$ | #Drop$_{\text{BF}}$ |
|---|---|---|---|---|---|---|
| Ring(50) | 100 | 300 (400) | 100 | 200 (300) | 0 | 0 |
| Vollvermascht(50) | 100 | 300 (400) | 100 | 200 (300) | 0 | 0 |
| Ringnetz(50) | 100 | 300 (400) | 100 | 200 (300) | 0 | 0 |

Tabelle 7.9.: Ergebnisse von Simulationen für drei unterschiedliche Topologien mit jeweils einem Sender und einer konstant fehlerhaften Checking Bridge (Duplizierung). Die Ergebnisse sind bei allen Empfängern, die mit fehlerfreien Bridges verbunden sind, identisch. Die roten Werte sind die Ergebnisse des Empfängers, der mit der fehlerhaften Checking Bridge verbunden ist.

Zuletzt wurde sichergestellt, dass eine fehlerhafte Signaturmodifikation, das heißt die Rotation der Signatur um ein Bit bzw. der XOR-Verknüpfung mit einer validen D/C/P-Maske außerhalb der Protokollspezifikation, nicht unerkannt bleibt und nicht zu unerwünschtem Verhalten führt. Auch in diesem Fall wurde der zugehörige Fehlertyp mit einer Wahrscheinlichkeit von 100% in eine der beiden Checking Bridges injiziert. Die Resultate sind mit den Ergebnissen der Simulationen mit Bitverfälschungen identisch, da die verfälschten Nachrichten der Signaturprüfung nicht standgehalten haben. Siehe Tabelle 7.10.

| Topologie | #Msgs$_{\text{Sent}}$ | #Rx | #Bc$_{\text{Succ}}$ | #Dup$_{\text{FF}}$ | #Drop$_{\text{Delay}}$ | #Drop$_{\text{BF}}$ |
|---|---|---|---|---|---|---|
| Ring(50) | 100 | 200 | 100 (0) | 0 | 0 | 100 (200) |
| Vollvermascht(50) | 100 | 200 | 100 (0) | 0 | 0 | 100 (200) |
| Ringnetz(50) | 100 | 200 | 100 (0) | 0 | 0 | 100 (200) |

Tabelle 7.10.: Ergebnisse von Simulationen für drei unterschiedliche Topologien mit jeweils einem Sender und einer konstant fehlerhaften Checking Bridge (Fehlerhafte Signaturmodifikation). Die Ergebnisse sind bei allen Empfängern, die mit fehlerfreien Bridges verbunden sind, identisch. Die roten Werte sind die Ergebnisse des Empfängers, der mit der fehlerhaften Checking Bridge verbunden ist.

### 7.3.3. Evaluation optimaler Zustellzeiten

Die Berechnung der Zustellzeit wurde bereits in Kapitel 7.2.2 kurz thematisiert, die Frage der Abschätzung der Teilgröße $\Delta_{Busy}$ konnte jedoch nicht abschließend geklärt werden und wird in diesem Abschnitt mit Hilfe von theoretischen Überlegungen in Kombination mit Simulationen evaluiert. Hierbei ist das Ziel, optimale Zustellzeiten, das heißt zugleich kleinstmögliche als auch genügend große Zustellzeiten, zu ermitteln.

### Garantierte Schranken für Übertragungszeiten

Eine essentielle Anforderung für ein Netzwerksystem, auf welchem FABAN Anwendung finden soll, ist die garantierte pro Hop Latenz, also eine obere Schranke für die Zeit, welche eine Nachricht für die Übertragung zwischen zwei benachbarten Komponenten benötigt. Ist diese Anforderung erfüllt, so kann trivialerweise die komplette Berechnung der Zustellzeit darauf reduziert werden, die garantierte pro Hop Latenz mit der maximalen Länge beider Wellen für jede Distributing Bridge zu multiplizieren.

Insbesondere im Hinblick auf die Realisierung des Protokolls auf möglichst einfacher bzw. handelsüblicher realer Hardware - Details folgen im nachfolgenden Kapitel - ist die Frage von besonderem Interesse, inwieweit die Anwendung ohne eine gegebene obere Schranke sinnvoll ist bzw. ob sich eine obere Schranke für eine pro Hop Latenz mit einfachen Mitteln bestimmen lässt. Aus diesem Grund wird im Folgenden der Fokus auf die Findung einer Lösung dieser Frage gelegt.

### Beschränkung der Senderaten

Die Idee für die Lösung des im vorherigen Abschnitt angesprochenen Problems liegt darin, die Senderaten von möglichen Sendern zu beschränken. Werden Senderaten von Senderknoten nicht beschränkt, so können in Bridges schnell Engpässe, verursacht durch die Datenrate der Verbindungen des Netzwerks, entstehen, was bei konstanten Sendeströmen zu stetig steigenden Übertragungszeiten von FABAN-Nachrichten führen würde.

Dabei sei angemerkt, dass die Überlegungen in diesem Teil der Arbeit grundlegende Mindestanforderungen im fehlerfreien Betrieb darstellen und nicht robust gegenüber willkürlichen Fehlern im System sind. Die Erhöhung des Netzwerkverkehrs aufgrund fehlerhafter Komponenten ist ein grundlegendes Problem, welches bereits in Kapitel 4 aufgegriffen wurde. Es lässt sich nicht ohne zusätzliche Mechanismen in den Komponenten des Netzwerks lösen. Mit den Überlegungen grundlegender Mindestanforderungen bezüglich des Sendeverhaltens soll der reibungslose Betrieb im fehlerfreien Fall ermöglicht werden und im späteren Verlauf dieses Kapitels evaluiert werden, inwieweit Fehlerinjektionen Probleme verursachen könnten.

Die Überlegung zu Beginn dieses Abschnittes führt zur folgenden ersten Mindestanforderung:

**Anforderung 7.1**
Die durch die Summe der (über die Zeit variablen) Sendeströme $R_S(t)$ aller Sender $S \in \mathcal{S}$ erzeugte Netzwerklast darf die Übertragungsrate $R_{Link}$ der Verbindungen des Netzwerks zu keinem Zeitpunkt überschreiten. Da bei FABAN jede Nachricht redundant über zwei

Wellen gesendet wird, muss folglich

$$2 \cdot \sum_{S \in \mathcal{S}} R_S(t) \leq R_{Link} \quad \forall t \geq 0 \tag{7.24}$$

gelten.

Auswirkungen von Senderaten gemäß vorangehender Bemerkung im Vergleich zu minimal erhöhten Senderaten werden in Abbildung 7.10 einander gegenübergestellt. Bei 10 Sendeknoten entsteht bei jedem Empfänger ein eingehender Datenstrom der 20-fachen Senderate, was bereits bei einer minimal erhöhten Senderate ($> 5Mbps$) das offensichtliche und erwartete Ergebnis der stetig steigenden Queueauslastung verdeutlicht.

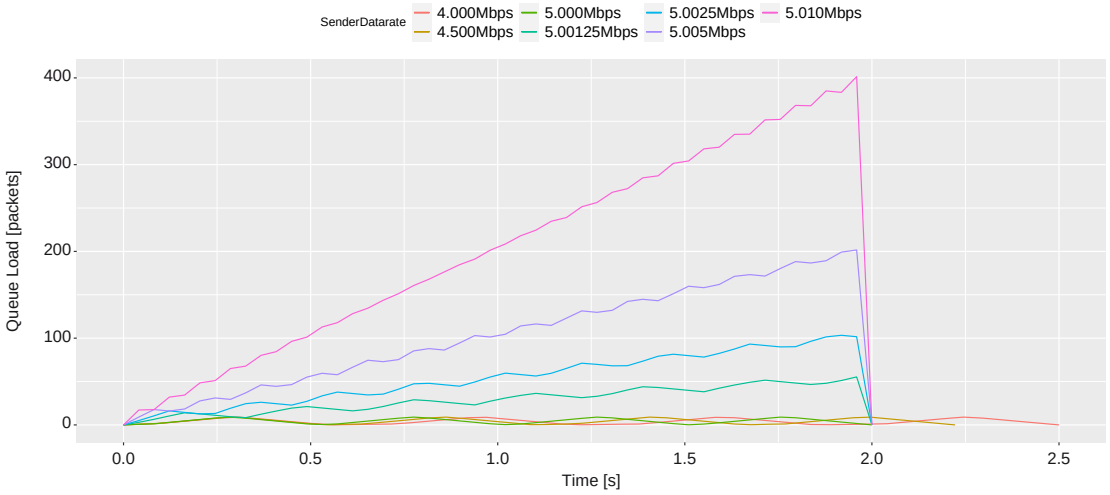



Abbildung 7.10.: Vergleich der Queueauslastungen (approximierte Werte) des mit einem Empfängerknoten verbundenen ausgehenden Ports einer beliebigen Bridge einer Ringtopologie der Größe 10. Simulationsdurchgänge wurden für unterschiedliche Senderaten der Sender durchgeführt und sind hier einander gegenübergestellt. In jedem Durchgang hatten alle Sender die gleiche Senderate.

Neben der zwingenden Beschränkung der Senderate ist es auch sinnvoll, überflüssige Kollisionen, soweit wie möglich, nicht entstehen zu lassen. Erstellt man Routings für Wellen unterschiedlicher Distributing Bridges komplett unabhängig voneinander, so kann es dazu führen, dass zwei Nachrichten von unterschiedlichen Sendern während der jeweiligen Sendevorgänge mehrfach miteinander kollidieren. Abbildung 7.11 verdeutlicht das Problem an einem ausführlichen Beispiel.

Das Problem in Abbildung 7.11 tritt auf, weil die Wellen von $D_1$ und $D_2$ nicht den gemeinsamen Weg zur Bridge $C$ haben, sich die getrennten Wege in $C$ aber wieder kreuzen. Dies lässt sich im Allgemein vermeiden, indem bei der Konstruktion von Wellen

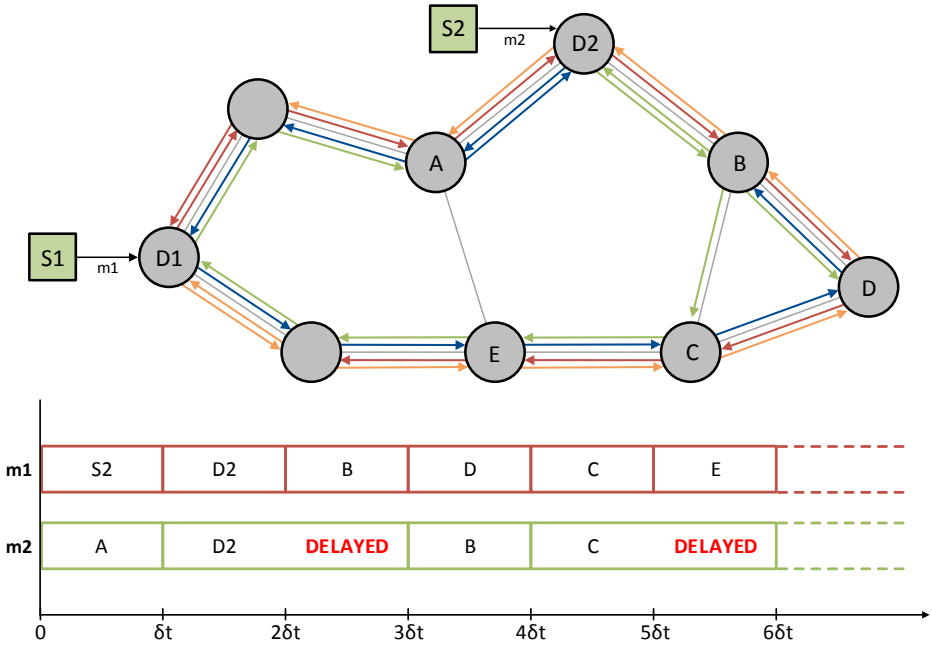



Abbildung 7.11.: Beispiel von mehrfachen Kollisionen zweier Nachrichten $m_1$ und $m_2$ der Sender $S_1$ und $S_2$. Mit roten/orangenen bzw. blauen/grünen Pfeilen sind die Wellenpaare der zu den Sender zugehörigen Distributing Bridges im oberen Teil der Grafik dargestellt. Der untere Teil der Grafik zeigt einen möglichen zeitlichen Verlauf beider Nachrichten und markiert den Bereich der Kollisionen. Der Einfachheit halber wird in dem Beispiel angenommen, dass beide Nachrichten identisch groß sind und pro Hop eine konstante Zeiteinheit $\delta t$ benötigen. Die Rechtecke in dem Diagramm stellen die von der Zeit abhängige Position inklusive der Zeitdauer der Übertragung zur nächsten Bridge der jeweiligen Nachricht im Netzwerk dar. Zum Zeitpunkt $t = 0$ befindet sich Nachricht $m_1$ in Bridge $A$ und Nachricht $m_2$ im Sender $S_2$.

unterschiedlicher Distributing Bridges darauf geachtet wird, dass die Verteilung nach dem ersten Aufeinandertreffen von Wellen, gemeinsam fortgeführt wird. Dies wird in folgender zweiter Anforderung zusammengefasst:

**Anforderung 7.2**

Gilt für beliebige Wellen $W_{d_1}, W_{d_2}$ unterschiedlicher Distributing Bridges $d_1, d_2 \in \mathcal{B}$ mit Checking Bridges $c_1^1, c_1^2, c_2^1, c_2^2 \in \mathcal{B}$

$$\exists b_1, b_2 \in \mathcal{B} : \ (b_1, b_2) \in W_{d_1} \cap W_{d_2} \tag{7.25}$$

und

$$\exists b_3 \in \mathcal{B} \setminus \left\{ d_1, d_2, c_1^1, c_1^2, c_2^1, c_2^2 \right\} : \ (b_2, b_3) \in W_{d_1} \cup W_{d_2}, \tag{7.26}$$

dann soll gelten:

$$(b_2, b_3) \in W_{d_1} \cap W_{d_2} \tag{7.27}$$

Anforderung 7.2 ist keine Veränderung des Routings, sondern eine Eingrenzung der Auswahl aus unterschiedlichen Wellenpaaren für ein Routing einer Distributing Bridge, in Abhängigkeit von einer bereits getroffenen Auswahl von Wellenpaaren für andere Distributing Bridges. Abbildung 7.12 verdeutlicht nochmal an dem Beispiel von Abbildung 7.11, inwieweit sich die zusätzliche Restriktion für die Auswahl von Wellenpaaren für unterschiedliche Distributing Bridges gemäß Anforderung 2 auf die Anzahl der Kollisionen zweier Nachrichten auswirken kann.

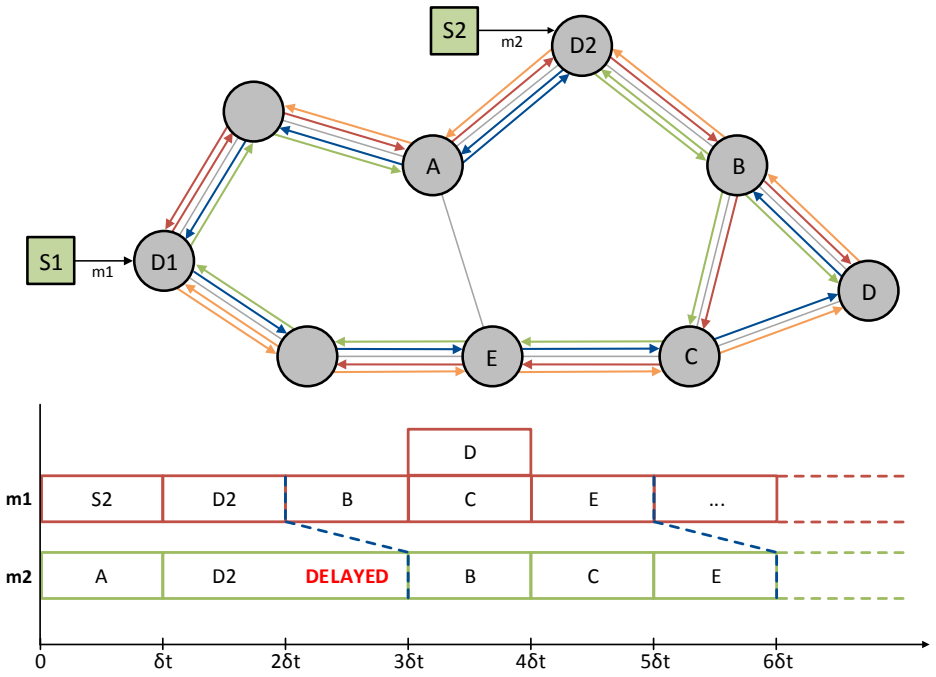



Abbildung 7.12.: Vermeidung von Mehrfachkollisionen zweier Nachrichten an dem Beispiel von Abbildung 7.11 durch Beachtung von Anforderung 2. Als Folge der Anpassung findet keine zweite Kollision statt, sondern die Nachrichten bewegen sich durch das Netzwerk unmittelbar aufeinanderfolgend.

Mit diesen Voraussetzungen kann nun die Analyse der Abschätzung der Teilgröße $\Delta_{Busy}$ erfolgen. Die Teilgröße $\Delta_{Busy}$ bezeichnet die maximal mögliche Verzögerung einer Nachricht im fehlerfreien Betrieb, welche durch Kollisionen an ausgehenden Ports in

Bridges entstehen können. Kollisionen können hierbei in unterschiedliche Fälle unterteilt werden, welche separat untersucht werden:

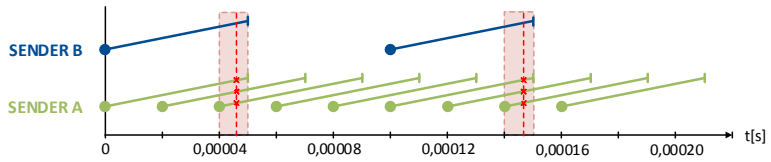



Abbildung 7.13.: Darstellung eines zeitlichen Szenarios der Übertragungen von FABAN-Nachrichten von zwei Sendern, welche mit konstanter, aber unterschiedlicher Senderate Nachrichten erzeugen und verschicken. Die roten/grünen Punkte markieren den Erstellzeitpunkt der Nachrichten beider Sender, die Linie den Übertragungszeitraum bis zum entferntesten Netzwerkknoten im Netzwerk, sowie die vertikalen Markierungen den Empfangszeitpunkt am entferntesten Netzwerkknoten. Hierbei wird in diesem Beispiel eine Linkspeed von $100MBit/s$ und eine konstante Nachrichtengröße von $1kBit$ angenommen. Sender A sendet hierbei Nachrichten mit einer Datenrate von $50MBit/s$ und Sender B mit einer Datenrate von $10MBit/s$. Die vertikale rot gestrichelte Linie sowie die darauf liegenden Kreuze markieren den Zeitpunkt der Kollision der Nachricht von Sender B mit maximal einer Nachricht von Sender A.

- **Fall 1** *Kollisionen mit Nachrichten von unterschiedlichen Sendern:*
  Eine Nachricht eines Senders $S_1$ kann nach Anforderung 2 in Lemma 7.2 mit einer Nachricht eines beliebigen anderen Senders $S_2$ höchsten einmal kollidieren. Ebenfalls kann ausgeschlossen werden, dass Kollisionen von $S_1$ mit mehreren aufeinander folgenden Nachrichten von $S_2$ auf der selben Welle auftreten. Dies kann damit begründet werden, dass Nachrichten von $S_2$ auf der jeweiligen Welle konstant auf den gleichen Pfaden versendet werden und folglich sequentiell entlang der Routing-Pfade übertragen werden. Durch eine Kollision reiht sich eine Nachricht von $S_1$ in diese Sequenz ein und kann von keiner nachfolgenden Nachricht von $S_2$ durch Kollisionen verzögert werden. Siehe dazu Abbildung 7.13. Die analoge Argumentation gilt folglich für die andere Welle des Senders $S_2$. Existieren in beiden Fällen keine Kollisionen durch Kreuzung der Routing-Pfade, so ist dies spätestens bei der letzten Übertragung zum Empfänger der Fall und muss in jedem Fall beachtet werden. Diese Überlegung ist unabhängig von der Länge der Nachrichten.
- **Fall 2** *Kollisionen mit eigenen Duplikaten:*
  FABAN-Nachrichten werden in der Distributing Bridge dupliziert und können bei der letzten Übertragung von einer Bridge zu einem Empfänger zu einer Kollision führen. Insbesondere tritt dieser Fall bei jeder Übertragung bei der Zustellung der Nachricht beim Sender selbst auf. Aufgrund der minimalen Anzahl der Übertragungen ($S \to DB \to CB \to DB \to S$) führt dies allein aber niemals zu Problemen. Abbildung 7.14 skizziert das Problem beispielhaft.

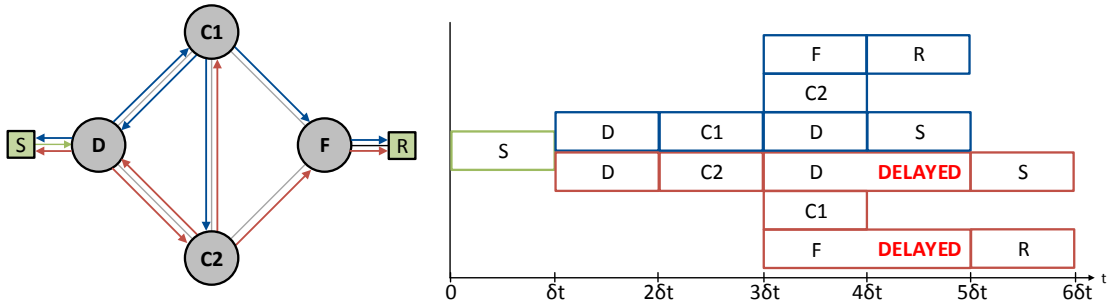



ABBILDUNG 7.14.: Beispielhafte Darstellung eines Szenarios mit Kollisionen mit dem eigenen Duplikat. Die nebenstehende Grafik zeigt die Positionen beider Duplikate der Nachricht abhängig von der Zeit und verdeutlicht die mögliche Verzögerung in der Distributing Bridge $D$ sowie Bridge $F$.

Durch Kombination beider Fälle erhält man schließlich folgende Abschätzung für die Teilgröße $\Delta_{Busy}$, zusammengefasst in folgendem Lemma:

**Lemma 7.3** (Abschätzung der Größe $\Delta_{Busy}$)
In einem Netzwerk beliebiger Größe mit Verbindungsgeschwindigkeit $R_{Link} > 0MBit/s$, $n > 0$ Senderknoten $S_1, \ldots, S_n \in \mathcal{S}$ und maximal einem verbundenen Sender pro Bridge kann eine FABAN-Nachricht im schlimmsten Fall maximal $(2 \cdot n - 1)$ mal durch Kollisionen mit anderen Nachrichten aufgehalten werden ($2(n-1)$ Kollisionen mit Nachrichten von Sendern anderer Distributing Bridges auf jeder Welle sowie maximal einer Eigenkollision). Die maximale Verzögerung einer Nachricht von einem Sender $S_i$, $1 \leq i \leq n$, durch Kollisionen kann dann wie folgt nach oben abgeschätzt werden:

$$(H+1) \cdot \Delta_{Busy}(S_i) \leq \left\lceil 2 \sum_{j=1}^{n} \frac{msgsize_{max}(S_j)}{R_{Link}} \right\rceil - \frac{msgsize_{max}(S_i)}{R_{Link}}, \tag{7.28}$$

wobei $msgsize_{max}(S_i)$ die maximal mögliche Nachrichtengröße einer Nachricht von Sender $S_i$ bezeichnet. Ist für alle Sender die maximal mögliche Nachrichtengröße gleich, so lässt sich die Ungleichung reduzieren zu:

$$(H+1) \cdot \Delta_{Busy} \leq (2n-1) \cdot \Delta_{Link}. \tag{7.29}$$

Ein formaler Beweis für diese Behauptung wird einerseits durch die theoretische Argumentation in der dem Lemma vorangehenden Diskussion ersetzt und zum anderen wird durch nachfolgende Simulationen die Richtigkeit evaluiert. Das nachfolgende Beispiel in Abbildung 7.15 verdeutlicht, dass ungünstige Szenarien tatsächlich derart existieren, so dass die Abschätzung in vorangehendem Lemma nicht weiter verringert werden kann.

In Lemma 7.3 wurde pro Bridge maximal ein verbundener Sender erlaubt. Sind mit manchen Bridges mehrere Sender verbunden, so werden diese im Rahmen von Lemma 7.3 als ein Sender gezählt und die Aussage kann folglich analog angewandt werden.

Zusätzlich zum konstruierten Szenario aus Abbildung 7.15, wurden eine Reihe von Simulationen auf Ring-Netzwerken unterschiedlicher Größen durchgeführt um zusätzlich zur Gültigkeit der oberen Schranke zu untersuchen, wie sich Nachrichtentransferdauern tatsächlich von der oberen berechneten Schranke unterscheiden. Hierbei wurde eine Linkrate von $100MBit/s$, eine konstante Nachrichtengröße von $1kBit$, sowie ein Sendeintervall festgelegt, so dass eine Gesamtankunftslast bei knapp unter $100MBit/s$ entsteht. Insgesamt sendet jeder Netzwerkknoten 10.000 Nachrichten. Das Experiment wurde für die Ringgrößen 3, 4, 5 und 10 durchgeführt. Die Ergebnisse sind in Abbildung 7.16 zusammengefasst.

Es ist weiterhin anzumerken, dass Kollisionen unabhängig von der Fehlertoleranz unvermeidlich sind. Allerdings wird durch die redundante Weiterleitung von Nachrichten entlang von Wellen der Kommunikationsaufwand und dadurch das Ausmaß der möglichen Kollisionen entsprechend erhöht.

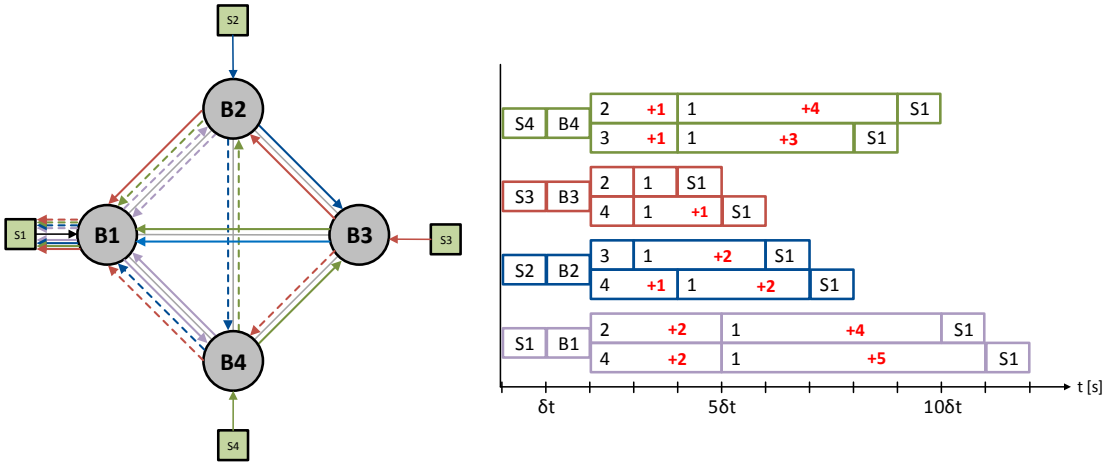



Abbildung 7.15.: Vollvermaschtes Netzwerk mit 4 Bridges. Die Sender $S1$, $S2$, $S3$ und $S4$ senden zum gleichen Zeitpunkt eine FABAN-Nachricht. Dargestellt sind nur die Abschnitte der Wellen, die zum Empfänger $S1$ führen. Wellen unterschiedlicher Sender werden hierbei in unterschiedlichen Farben dargestellt, wohingegen die Gegenwelle jeweils gestrichelt dargestellt ist. Der rechte Teil der Abbildung visualisiert den zeitlichen Verlauf der Nachrichten im Netzwerk.

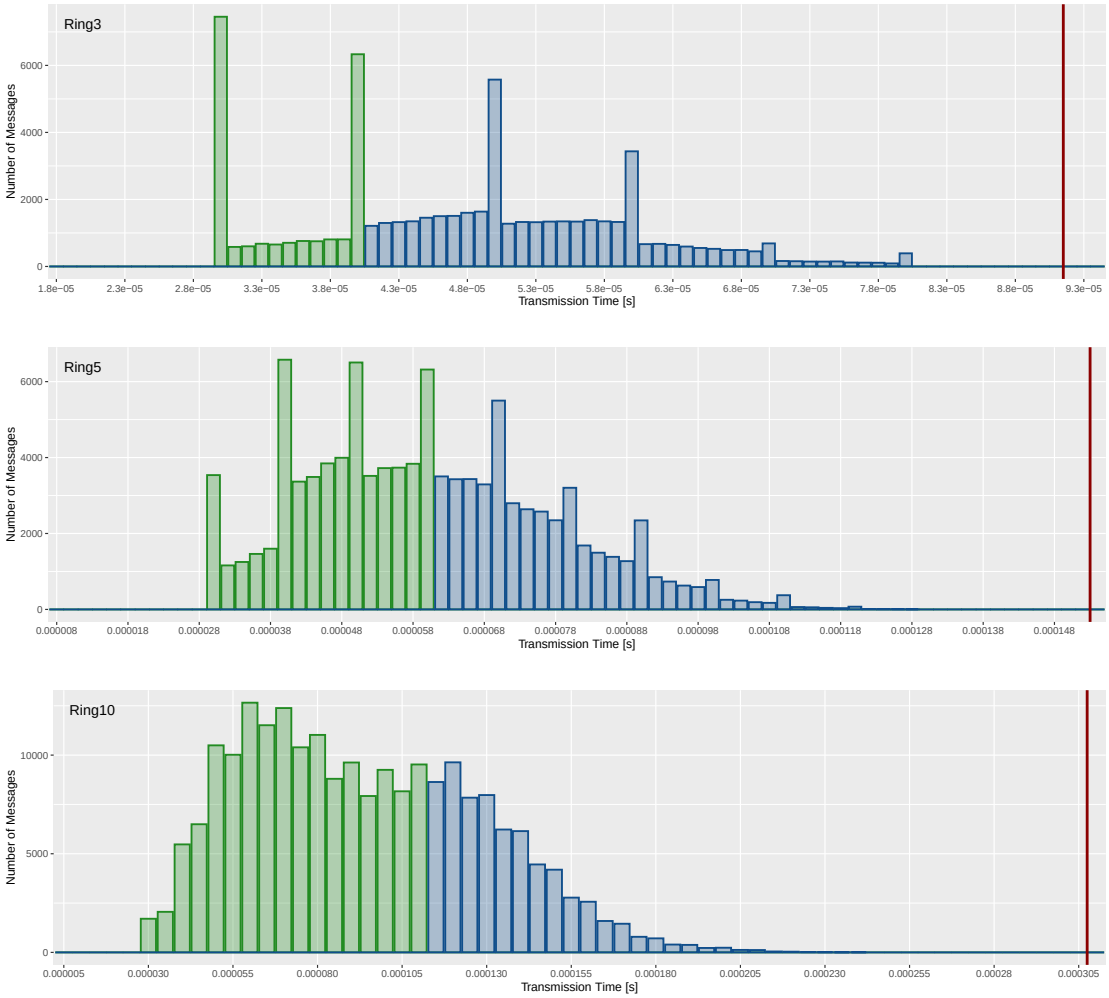



Abbildung 7.16.: Darstellung der Transferzeiten von Nachrichten vom Sender bis zum jeweiligen Empfänger in Form eines Histogramms. Die grünen Balken stellen Sendezeiten von Nachrichten dar, welche kleiner sind als die minimal nötige Zeit, welche eine Nachricht zu dem entferntesten Netzwerkknoten ohne Kollisionen benötigt. Die blauen Balken stellen Sendezeiten von Nachrichten dar, welche durch eine oder mehrere Kollisionen verzögert wurden. Die rote Linie markiert die maximal erlaubte Transferzeit, bestehend aus den Verzögerungen durch die physikalischen Übertragungen sowie der zusätzlichen Verzögerung durch die Abschätzung aus Lemma 7.3.

Die Ergebnisse bestätigen in dem durch Simulation möglichen Rahmen die in Lemma 7.3 formulierte Aussage. Auffällig hierbei ist, dass die kalkulierte obere Grenze gemäß (7.29) für Verzögerungen in allen Simulationen nicht erreicht wird bzw. die Transferzeiten von Nachrichten höchstens um 80% über der mindestens notwendigen maximalen Transferdauer liegt, gekennzeichnet durch die blauen Bereiche in Abbildung 7.16. Dies darf jedoch nicht als Zeichen gesehen werden, dass die Grenze für Verzögerungen zu hoch

geschätzt wurde, denn im Gegensatz zu dem Beispielszenario in Abbildung 7.15, ist bei Ringtopologien die Rollenverteilung der Distributing Bridges und der Checking Bridges vollständig gleichmäßig verteilt, so dass in diesen Szenarien der berechnete Fall deutlich unwahrscheinlicher ist und in manchen Topologien im fehlerfreien Fall sogar gänzlich ausgeschlossen werden kann. Als wichtigste Erkenntnis lässt sich aus den Simulationen folgern, dass die berechnete Zustellzeit zumindest im fehlerfreien Fall vollständig den Ansprüchen genügt. Inwieweit dies noch in der Präsenz von Fehlern der Fall ist, wird in nachfolgenden Abschnitten untersucht.

#### Berechnungszeiten

Die letzte auf Nachrichten Einfluss nehmende zeitliche Komponente ist die Größe der Verweilzeit von Nachrichten innerhalb von Bridges durch interne Berechnungen $\Delta_{Proc}(m)$ (vgl. Abbildung 7.2). Es ist offensichtlich, dass die Verweilzeit einer Nachricht maximal so hoch sein darf, dass die dadurch implizierte Bearbeitungsrate von Nachrichten innerhalb der Bridge maximal so hoch wie die Zuflussrate von Nachrichten ist. Andernfalls würde eine Aufstauung in den Bridges stattfinden und zwangsläufig sowohl zu unkontrollierten Verzögerungen sowie zu Nachrichtenverlusten aufgrund von Pufferüberläufen führen. Ist diese Bedingung erfüllt, so kann die obere Schranke der Verweilzeit als konstante Verzögerung $\Delta_{Proc}(m)$ jeder ExFABAN-Nachricht in jeder Bridge betrachtet werden, wodurch sich lediglich eine konstante Erhöhung der Zustellzeit ergibt und keine zusätzlichen Kollisionen oder sich verstärkende, unvorhersehbare Verzögerungen entstehen. Auch bei der Kollisionsbetrachtung in vorangehendem Abschnitt entsteht durch die Verweilzeit keine negative Auswirkung. Dies wird in den Simulationen der nachfolgenden Abschnitte implizit verifiziert, bei welchen eine Verweilzeit ebenfalls simuliert wird.

### 7.3.4. Simulationen mit Injektionen von Einfachfehlern

Nachdem die Implementierung validiert, die korrekte Funktionsfähigkeit im fehlerfreien Fall gezeigt sowie die Berechnung optimaler Zustellzeiten im fehlerfreien Fall evaluiert wurde, werden in diesem Unterkapitel Fehlerinjektionen hinzugezogen. Hierbei werden Simulationen mit unterschiedlichen Mengen von Fehlerarten durchgeführt und relevante Daten gesammelt, welche die volle Funktionsfähigkeit zeigen bzw. Probleme aufzeigen sollen. Vor der Durchführung der Simulationen, wurden die in Kapitel 7.2.5 beschriebenen Fehlerarten im Hinblick auf das Ausmaß möglicher Probleme bzw. Ursachen für Probleme in drei disjunkte Gruppen bzw. Mengen unterteilt, welche in den folgenden Unterabschnitten thematisiert werden.

Alle Simulationen wurden auf den drei Netzwerktopologieklassen, welche in Kapitel 7.2.5 beschrieben wurden, durchgeführt, wobei für Ringe und vollvermaschte Netze

Größen von 10, 50 sowie 100 Bridges gewählt wurden, sowie die ebenfalls in Kapitel 7.2.5 beschriebenen Ringnetze der selben drei Größen. Für jede Topologie wurden die Simulationen 10 mal wiederholt, wobei jeder an eine Bridge angeschlossene Netzwerkknoten in jedem Durchgang 1.000 Nachrichten versendet. Die Anzahl der ausreichenden Wiederholungen wurde statistisch bestimmt, indem bei fortlaufenden Wiederholungen 99.9%-Konfidenzintervalle erfolgreicher Broadcasts bestimmt wurden bis diese hinreichend klein wurden und valide Ergebnisse abdeckten [Law13]. Des Weiteren ist ein essentielles Kriterium, dass stets alle Empfänger an fehlerfreien Bridges alle Rundsprüche von Sendern fehlerfreier Bridges empfangen haben. Das erste Kriterium war im Schnitt bei 4 bis 5 Wiederholungen erfüllt, während das zweite Kriterium immer erfüllt war. Aus Gründen des geringen Mehraufwands wurde die Anzahl der Simulationen auf 10 festgelegt. Konkrete Berechnungen der Konfidenzintervalle sind zu jeder Simulation auf dem beiliegenden Datenträger zu finden. Die Linkrate wurde auf $100MBit/s$ festgelegt und die Senderate, abhängig von der Topologiegröße, gemäß Bemerkung 7.1 auf den höchstmöglich erlaubten Wert $R_L$ gesetzt, wobei die tatsächliche Rate nach jedem Sendevorgang zufällig im Intervall $[0.9R_L, R_L]$ bestimmt wurde, um eine möglichst breite Variation der Nachrichtenverläufe zu erhalten. Die Nachrichtengröße wurde zu Vergleichszwecken auf $1kBit$ und die Verweildauer einer Nachricht auf maximal $1\mu s$, was einer Verarbeitungsrate von $1GBit/s$ entspricht, festgelegt.

Aufgrund des vollständig symmetrischen Sendeverhaltens ist es ausreichend, in allen aufeinanderfolgenden Simulationen, eine feste Bridge als fehlerhafte Bridge zu konfigurieren. Auch die Ringtopologie sowie die Topologien der vollvermaschten Netz sind vollständig symmetrisch, so dass dieses Argument auch für diese beiden Topologieklassen gilt. Ringnetze wiederum sind nicht vollständig symmetrisch, können aber im Falle einer fehlerhaften Bridge, die keine Verbindungsstelle zwischen zwei Ringen ist, auf einen einzigen Ring reduziert werden. Folglich ist es ausreichend und sinnvoll, zwecks Gewinn weiterer Erkenntnisse eine Bridge, die als Verbindung zwischen zwei Ringen dient, als fehlerhafte Bridge zu konfigurieren. Sollten Simulationen keine unterschiedlichen Ergebnisse zwischen den einzelnen Topologien ergeben, so wird im Folgenden, um den Rahmen dieses Abschnittes in Grenzen zu halten, zur Analyse und Evaluation eine Netzwerktopologie stellvertretend gewählt. Wenn im nachfolgenden nicht explizit erwähnt, können alle injizierten Fehler auch byzantinisch sein.

#### Nicht-kritische Fehlerarten

**Beschreibung und Parametrisierung** Den nicht-kritische Fehlerarten wurden solche Fehlerarten zugeordnet, welche Nachrichtendurchsätze nicht erhöhen. Ohne eine Erhöhung der Last des Netzwerks, welche die Anforderungen aus Bemerkung 7.1 verletzen würden, sind keinerlei Probleme bzw. Anomalien zu erwarten. Im Detail wurden folgende

Fehlerarten als nicht-kritisch klassifiziert:

- *Ausfallfehler* und *Unterlassungsfehler* führen dazu, dass die fehlerhafte Bridge eine Menge von weiterzuleitenden Nachrichten nicht weiterleitet, was nicht zu einer Erhöhung, sondern zu einer Verringerung des Nachrichtendurchsatzes führt.
- *Bitflips*, *Feldmanipulationen* und *Fehlerhafte Signaturmodifikationen* verfälschen einzelne Nachrichten und haben weder eine Erhöhung noch eine Verringerung des Nachrichtendurchsatzes zur Folge.

Die Parametrisierung der einzelnen Fehlerarten wird dabei wie folgt spezifiziert:

- Ein totaler Ausfall der fehlerhaften Bridge tritt mit einer Rate von 100 Ausfällen pro Sekunde auf. Dies entspricht einer Fehlerwahrscheinlichkeit von 0.001 bei jeder weiterzuleitenden Nachricht.
- Unterlassungsfehler, Bitflips, Feldmanipulationen sowie die fehlerhafte Signaturmodifikation treten jeweils mit einer Wahrscheinlichkeit von 0.1 auf. Hierbei werden bei ...
  - Bitflip-Fehlern eine gleichmäßig verteilte Anzahl von Bits aus dem Intervall $\{1, \ldots, 10\}$ getoggelt.
  - jeder Feldmanipulation eine gleichmäßig verteilte Anzahl von Feldern aus dem Intervall $\{1, \ldots, 6\}$ manipuliert, d.h. alle Felder dürfen fehlerhaft sein.

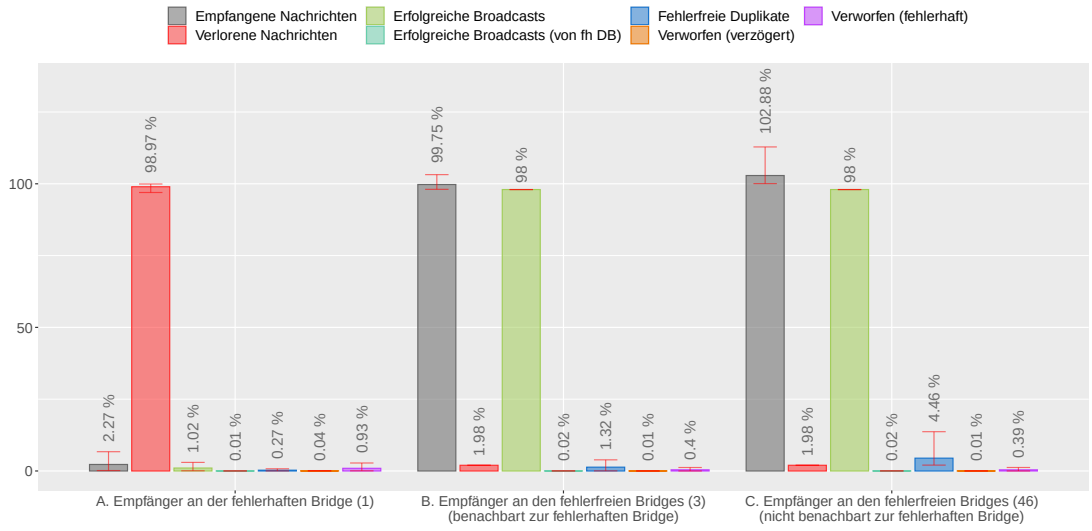



Abbildung 7.17.: Empfangsstatistiken für Simulationen mit nicht-kritischen Fehlern in einer Bridge auf dem Ringnetz der Größe 50.

**Evaluation** Die Simulationsergebnisse entsprechen vollständig den Erwartungen. Fehlverhalten der fehlerhaften Bridge verhindert die unverfälschte Zustellung von höchstens einer der beiden Kopien von Nachrichten der Sender, die an fehlerfreien Bridges angeschlossen sind, so dass jeder FABAN-Rundspruch erfolgreich bei allen Empfängern, ausgenommen dem Empfänger an der fehlerhaften Bridge, zugestellt wird, was die Anforderungen an einen fehlertoleranten Rundspruch in jedem Fall vollständig erfüllt. Rundsprüche, die über die fehlerhafte Bridge als Distributing Bridge gesendet werden, werden zu einem geringen Teil, jedoch konsistent erfolgreich zugestellt. Abbildung 7.17 stellt stellvertretend für alle neun Topologien, auf welchen Simulationen durchgeführt wurden, statistische Zusammenfassungen von empfangenen Nachrichten in den Empfängern dar, wobei über die Ergebnisse aller Wiederholungen die Mittelwerte gebildet wurden. Zusätzlich wurden die Ergebnisse von Empfängern der zur fehlerhaften Bridge benachbarten fehlerfreien Bridges sowie der nicht benachbarten fehlerfreien Bridges zu zwei Gruppen zusammengefasst, um eine kompakte Darstellung der Ergebnisse zu erhalten. Der vollständige Bereich der jeweiligen statistischen Größe um den Mittelwert, wird mittels vertikaler, roter Intervalllinien dargestellt.

### Kritische Fehlerarten: Verzögerungen

**Beschreibung und Parametrisierung** Verzögerungen verursachen, wie die zuvor thematisierte Gruppe der nicht-kritischen Fehler, ebenfalls keine Erhöhung des Gesamtnachrichtendurchsatzes, es ist allerdings vorstellbar, dass durch einzelne bzw. häufige Verzögerungen von Nachrichten in einer ungünstigen Abfolge, eine Verschiebung der Last aus einem unkritischen Zeitintervall in ein Zeitintervall entsteht, so dass durch die zusätzliche Last, Nachrichten nicht mehr rechtzeitig zugestellt werden können.

Um der Existenz solch einer ungünstigen Konstellation von Verzögerungen nachzugehen, wurden Simulationen mit variierenden Fehlerwahrscheinlichkeiten sowie variierenden Wahrscheinlichkeitsverteilungen für das Ausmaß der Verzögerungen durchgeführt. Konkret wurde die Fehlerwahrscheinlichkeit für das Eintreten einer Verzögerung für alle Wahrscheinlichkeiten $p \in \{0.5, 0.75, 0.95\}$ betrachtet, sowie einer gleichverteilten Verzögerung $D \sim \mathcal{U}\{(0s, 1s\}$. Damit wurden für jede Netzwerktopologie 30 Simulationsdurchläufe durchgeführt. Abbildung 7.18 zeigt stellvertretend für alle Topologien die Simulationsergebnisse für die Simulation auf dem Ringnetz der Größe 50.

**Evaluation** Die Haupterkenntnis aus den Simulationen ist, dass alle fehlerfreien Bridges stets alle FABAN-Rundsprüche von Sendern, welche über fehlerfreie Distributing Bridges senden, in mindestens einfacher Ausführung rechtzeitig erhalten. Auch der Anteil rechtzeitig erhaltener Duplikate sowie verspäteter Duplikate, inklusive der hohen Varianz dieser Werte durch die variierenden Fehlerparameter, ist vollkommen plausibel.

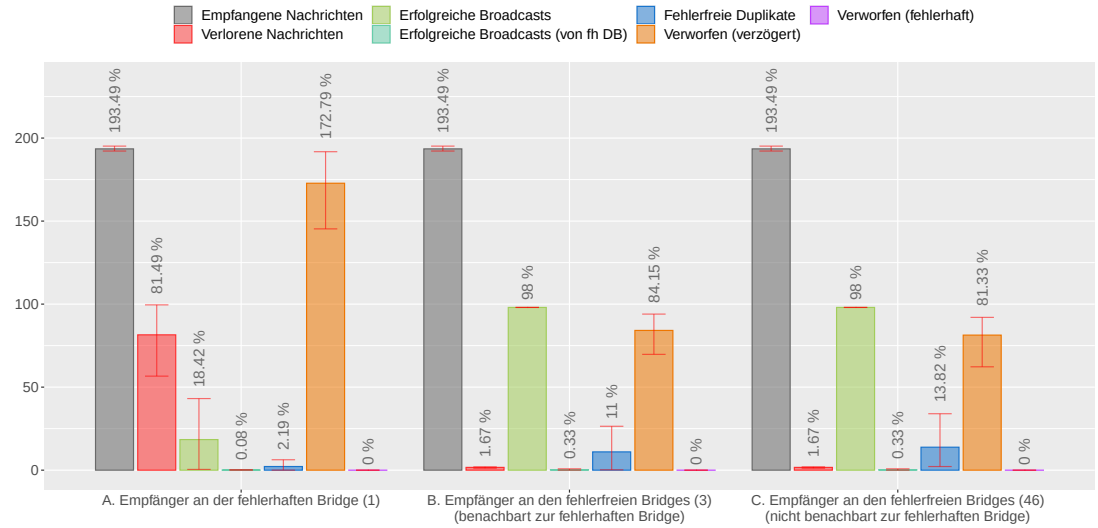



Abbildung 7.18.: Empfangsstatistiken für Simulationen mit Verzögerungsfehlern in einer Bridge auf dem Ringnetz der Größe 50.

### Kritische Fehlerarten: Fehlerhafte Weiterleitung

**Beschreibung und Parametrisierung** Eine fehlerhafte Weiterleitung einer Nachricht kann die Korrektheit dieser weder direkt verfälschen noch dafür sorgen, dass die Eigenschaften des unteilbaren Rundspruchs nicht erfüllt werden. Allerdings kann, sollte eine Nachricht zusätzlich zu den korrekten Weiterleitungen auf anderen ausgehenden Ports an weitere Bridges weitergeleitet werden, in ungünstigen Fällen nicht differenziert werden, dass eine falsch weitergeleitete Nachricht nicht weitergeleitet werden soll. Dieses Problem besteht bei dem ursprünglichen, in Kapitel 3.1.2 beschriebenen Nachrichtenformat. Die initial geplanten Informationen, welche eine FABAN-Nachricht enthalten soll, sind ausreichend, um ein Routing im fehlerfreien Fall zu erstellen, welches auf Basis des Senders und des eingehenden Ports einer jeden Bridge die Menge der ausgehenden Ports liefert. In dem Fall einer fehlerhaften Bridge, die Nachrichten fehlerhaft weiterleitet, stellte sich dies als nicht ausreichend heraus, da das Zurücksenden einer Nachricht zu der Bridge, von welcher die Nachricht übermittelt wurde (fehlerhafte Reflektion), dazu führen kann, dass die Nachricht als Nachricht der anderen Welle interpretiert und folglich die Anzahl der Nachrichtenströme konstant erhöht wird. Abbildung 7.19 skizziert das Problemszenario an einem Minimalbeispiel.

Eine weiteres Problem kann abhängig von der Art des Umgangs von Nachrichten in Forwarding Bridges entstehen. Empfängt eine Forwarding Bridge eine FABAN-Nachricht, so muss diese stets zu allen angeschlossenen Empfängern weitergeleitet werden. Es ist daher naheliegend, die Routing-Lookup-Tabelle zu reduzieren und auf explizite Einträge für die

Abbildung 7.19.: Erhöhung der Datenstromrate durch fehlerhafte Weiterleitungen. Grüne und blaue Pfeile beschreiben Nachrichtentransfers beider Wellen, wobei die roten Pfeile die durch fehlerhafte Reflektion entstandenen neuen Ströme darstellen.

Weiterleitung an benachbarte Empfänger zu verzichten. Dies führt jedoch ebenfalls zu einer Reduktion der Fehlertoleranzfähigkeiten, da die zuvor beschriebene fehlerhafte Reflektion bei zur fehlerhaften Bridge benachbarten Bridges ebenfalls zu einer Datenstromerhöhung führt, was wiederum zu Überfüllungen von Warteschlangen führen kann.

Für alle drei beschriebenen Varianten der Routing-Lookup-Tabellen wurden Simulationen durchgeführt, wobei die Fehlereintrittswahrscheinlichkeit $p \in \{0.1, 0.5, 0.95\}$ betrachtet wurden.

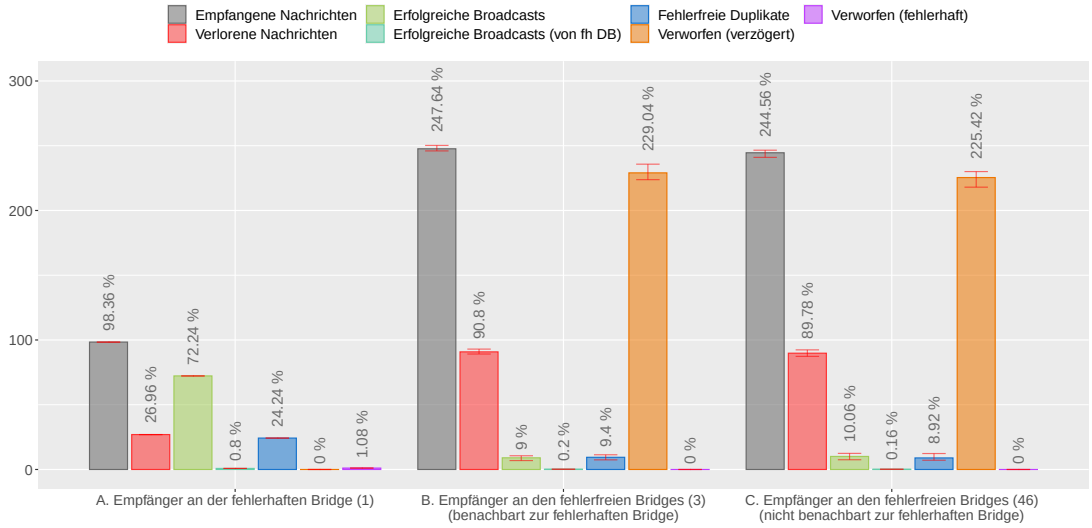



Abbildung 7.20.: Empfangsstatistiken für Simulationen mit fehlerhaften Weiterleitungen in einer Bridge auf dem Ringnetz der Größe 50 und der Ermittlung des Routings in Abhängigkeit von SenderID und eingehendem Port. Die Zustellung zu den Empfängern findet statisch in allen Forwarding Bridges statt.

**Evaluation** Abbildung 7.20 zeigt den Einfluss des Effekts der fehlerhaften Reflektion durch den Fehler der fehlerhaften Weiterleitung, wenn die Differenzierung von eingehenden Nachrichten lediglich durch die SenderID sowie den eingehenden Port erfolgt. Die Zustellung zu den Empfängern bedarf in diesem Fall keines Routings, sondern wird in Abhängigkeit vom Hop-Counter und somit von der Rolle der Bridge realisiert. Das zuvor beschriebene Problem aus Abbildung 7.19 macht sich bei allen Sendern in Form des geringen Anteils von erfolgreichen Rundsprüchen als Folge der fehlerhaften Erhöhung der Datenströme durch die fehlerhafte Bridge deutlich bemerkbar.

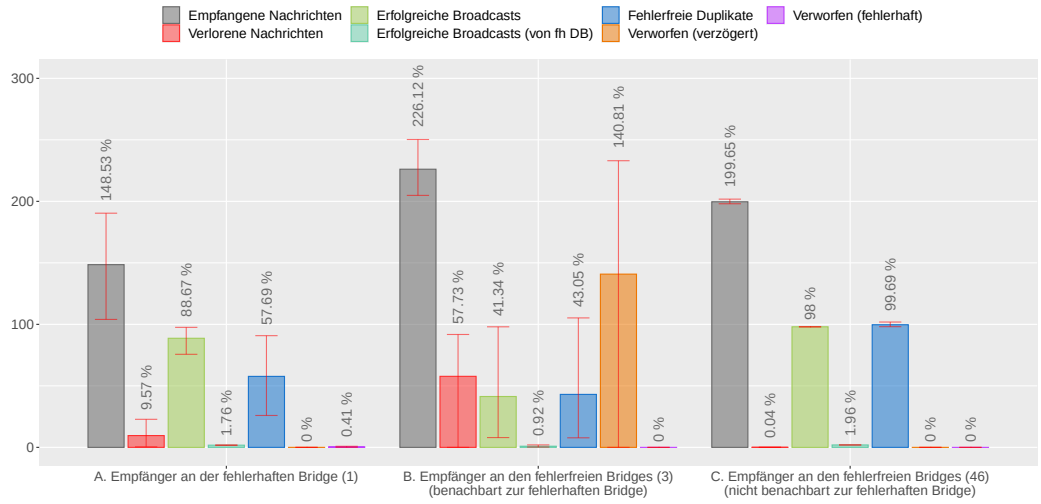



Abbildung 7.21.: Empfangsstatistiken für Simulationen mit fehlerhaften Weiterleitungen in einer Bridge auf dem Ringnetz der Größe 50 und der Ermittlung des Routings in Abhängigkeit von SenderID, der initiierenden Checking Bridge und dem eingehendem Port. Die Zustellung zu den Empfängern findet statisch in allen Forwarding Bridges statt.

Abbildung 7.21 zeigt die deutliche Verbesserung der Ergebnisse, wenn zusätzlich die initiierende Checking Bridge zur Differenzierung zur Rate gezogen wird. Einfluss auf den Empfang von Rundsprüchen bei Empfängern, die an fehlerfreien Bridges angeschlossen sind, besteht nur noch an Empfängern, die mit zur fehlerhaften Bridge benachbarten Bridges verbunden sind. Dort werden empfangene Nachrichten anhand der zusätzlichen Information der initiierenden Checking Bridge nicht mit Nachrichten der anderen Welle verwechselt und nicht zu anderen Bridges weitergeleitet. Lediglich die Weiterleitung zu verbundenen Empfängern, die nur anhand des Hop-Counters bestimmt wird, führt als Folge des Fehlers zu überflüssig redundanten Nachrichten und verursacht eine Überlastung der physikalischen Verbindung.

Abschließend zeigt Abbildung 7.22, dass es notwendig ist, die Zustellung von Nachrichten von Forwarding Bridges zu den Empfängern ebenfalls über Einträge in der Routing-Lookup-Tabelle zu realisieren. Erst bei dieser Variante wird die Fehlerhafte

Weiterleitung vollständig toleriert.

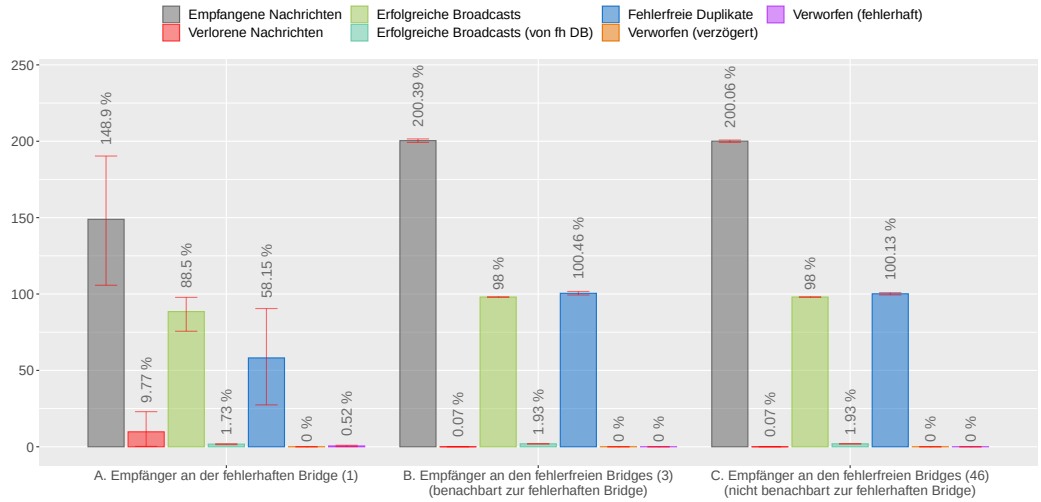



Abbildung 7.22.: Empfangsstatistiken für Simulationen mit fehlerhaften Weiterleitungen in einer Bridge auf dem Ringnetz der Größe 50 und der Ermittlung des Routings in Abhängigkeit von SenderID, der initiierenden Checking Bridge und dem eingehendem Port. Die Zustellung zu den Empfängern findet ebenfalls über Routing-Lookup-Tabellen Einträge statt.

### Kritische Fehlerarten: Babbling Component

**Beschreibung und Parametrisierung** Ist eine Bridge derart fehlerhaft, dass diese sich wie eine Babbling Component verhält, dann kann diese zu beliebigen Zeiten willkürliche Nachrichten generieren und an benachbarte Komponenten senden. Mit der vollständigen Routing-Lookup-Tabelle, wie sie bereits bei der Fehlerart Fehlerhafte Weiterleitung thematisiert wurde, sollte jede Bridge, welche eine willkürlich generierte Nachricht empfängt, diese als ungültig erkennen und unterdrücken. Eine Ausnahme stellen solche Nachrichten dar, welche zufällig valide Inhalte im Feld der SenderID sowie der initiierten Checking Bridge aufweisen. Dies ist allerdings einerseits sehr unwahrscheinlich und andererseits, sollte der unwahrscheinliche Fall dennoch auftreten, entsteht der Effekt eines Duplizierungsfehlers (in Kombination mit Nachrichtenverfälschung), welcher im Anschluss genauer thematisiert wird. Bei der Simulation dieses Fehlers wurde die Fehlerwahrscheinlichkeit $p = 0.25$ betrachtet, sowie bei jedem Fehlereintritt die Anzahl der zu erzeugenden willkürlichen Nachrichten gleichverteilt auf dem Intervall $\{1, 10\}$ ermittelt. Hierbei werden die Abstände $T$ dieser Nachrichten aus dem Intervall gemäß folgender Gleichverteilung bestimmt:

$$T \sim [1 \cdot 10^{-x}, 1.1 \cdot 10^{-x}] \quad \text{für } x \in \{3, 5, 7\}. \tag{7.30}$$

**Evaluation** Die relativ hohe Fehlerwahrscheinlichkeit von $p = 0.25$ mit durchschnittlich 50 zusätzlichen Nachrichten mit zufälligem Inhalt bei jeder weiterzuleitenden Nachricht führt dazu, dass der enorm erhöhte Netzwerkverkehr fehlerfreie Nachrichten zu stark verzögert und diese nicht mehr rechtzeitig zugestellt werden können. Die fehlerfreien Bridges sowie die mit ihnen verbundenen Empfänger sind von dem Fehlverhalten der Babbling Component nicht betroffen. Die Rundsprüche von Sendern, die nicht über die fehlerhafte Bridge als Distributing Bridge verteilen, werden vollständig von allen Empfängern an fehlerfreien Bridges empfangen und sind keiner Zusatzlast ausgesetzt.

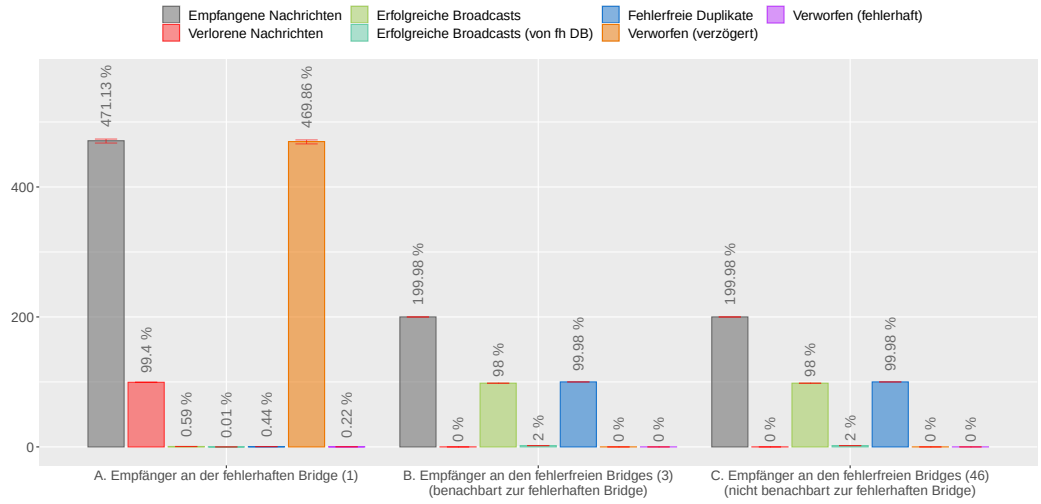



Abbildung 7.23.: Empfangsstatistiken für Simulationen mit einer als Babbling Component agierenden Bridge auf dem Ringnetz der Größe 50.

### Kritische Fehlerarten: Duplizierung

**Beschreibung und Parametrisierung** Das fehlerhafte Erzeugen und Versenden von Duplikaten kann, wie die zuvor beschriebenen Fehlerarten, indirekt Probleme verursachen. Wird eine Nachricht dupliziert, nicht weiter verfälscht und korrekt weitergeleitet, so erhält ein fehlerfreier Empfänger mehr als nur eine redundante Kopie einer korrekten FABAN-Nachricht. Aufgrund des zwingend notwendigen Mechanismus der Duplikaterkennung in den Empfängern, werden solche Nachrichten folglich ebenfalls erkannt und unterdrückt und führen zu keinen weiteren Verfälschungen. Wie bei den vorherigen Fehlerarten, erzeugen aber Duplikate eine Zusatzlast auf dem gesamten Netzwerk und können, anders als zuvor, nicht mit den FABAN-spezifischen Mitteln wie der Routing-Lookup-Tabelle, verhindert werden.

Um diesen Fehler vollständig zu tolerieren, ist es notwendig, dass das zugrunde liegende Netzwerk die Fähigkeit besitzt, den Effekt eines Denial of Service (DoS) zu erkennen und

zu unterdrücken [Nee93]. Eine weitere Möglichkeit besteht darin, eine Duplikaterkennung in Bridges zu realisieren, im Optimalfall in den eingehenden Ports, um Duplikate frühzeitig zu erkennen und zu unterdrücken [Pap+07]. Im nachfolgenden Abschnitt zur Evaluation von Simulationen von Mehrfachfehlern wird eine abstrahierte Duplikaterkennung als Ergänzung zur Implementierung beschrieben. Diese Duplikaterkennung wurde bei den Simulationen verwendet.

Um das Ausmaß von nicht erkannten und unterdrückten Duplikaten herauszufinden, wurde eine Simulation mit der konstanten Wahrscheinlichkeit von Duplizierungsfehlern von 0.1 durchgeführt sowie der Erstellung von konstant einem Duplikat bei Eintritt des Fehlers.

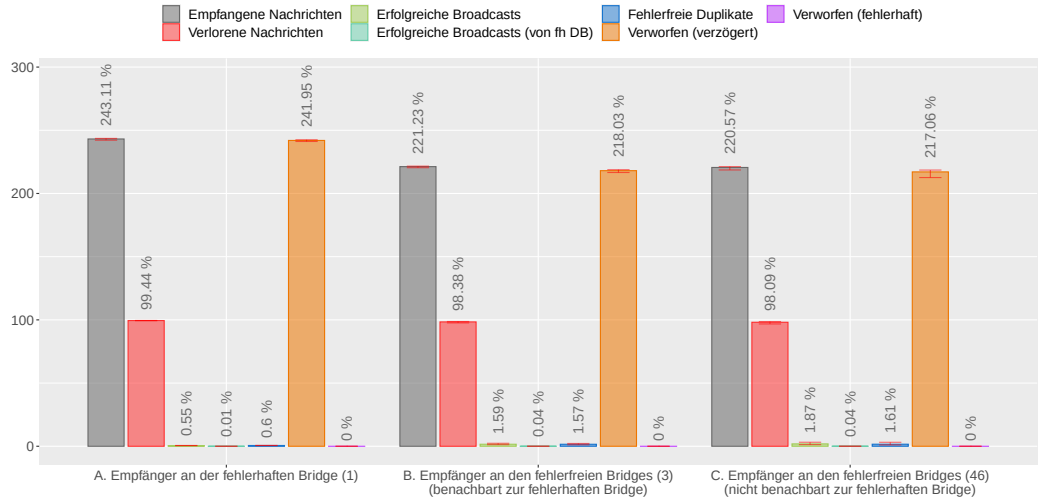



Abbildung 7.24.: Empfangsstatistiken für Simulationen mit einer fehlerhaften, Duplikat erzeugenden, Bridge auf dem Ringnetz der Größe 50.

**Evaluation** Die Simulationsergebnisse bestätigen die vorangehenden Überlegungen. Duplikate werden valide durch das Netzwerk weitergeleitet und in den Empfängern korrekt und exakt einmal zugestellt. Aufgrund der Erhöhung der Datenmenge im Netz ist diese jedoch aufgrund der hohen Senderaten der Sender schnell überlastet, so dass es zu enormen Verzögerungen führt und die Zustellzeiten nicht mehr eingehalten werden können. Simulationen mit einer Duplikaterkennung in den Bridges werden im nachfolgenden Kapitel evaluiert.

#### Fazit

Zusammenfassend lässt sich sagen, dass die Erwartungen bezüglich der Toleranz von Fehlern, so wie zuvor ausführlich in theoretischer Art und Weise untersucht und diskutiert, erfüllt wurden. Durch Simulation konnte verifiziert werden, dass alle zu tolerierenden Fehlerarten tatsächlich toleriert werden und somit die Spezifikation von FABAN erfüllt ist. Beobachtete Probleme im Zusammenhang mit Duplikaten bzw. genereller Überlast sind keine FABAN-spezifischen Probleme, sondern stellen allgemeine Probleme von Netzwerken dar, welche durch zusätzliche Mechanismen gelöst werden können.

## 7.4. Evaluation durch Simulation für Mehrfachfehler

Das vorangehende Unterkapitel beschäftigte sich intensiv mit der Validierung der Implementierung der Simulation bei Betrachtung von Einfachfehlern sowie einer detaillierten Evaluation des Protokolls FABAN anhand von Simulationen mit Fehlerinjektionen. Insbesondere wurden etwaige Szenarien unterschiedlicher Fehlerarten untersucht, welche zu Problemen führen könnten, sowie Lösungen diskutiert und zum großen Teil umgesetzt.

Dieses Unterkapitel erweitert die Analyse des vorangehenden Abschnitts um die Betrachtung von Mehrfachfehlern und schließt die Evaluation durch Simulationen von FABAN sowie ExFABAN ab. Hierbei wird der Fokus besonders darauf gelegt zu untersuchen, inwieweit die Aussagen von Bemerkung 7.1 (keine Nachrichtenüberlast) sowie Lemma 7.3 (begrenzte Anzahl von Kollisionen) auf den Fall von Mehrfehlertoleranz appliziert werden können.

### 7.4.1. Netzwerktopologien

In Kapitel 6.3 wurden drei Klassen von Netzwerktopologien vorgestellt, welche es erlauben für einen gewünschten Fehlertoleranzgrad passende sowie skalierbare Topologien zu erstellen. Während vollvermaschte Netze unpraktikabel sind und redundante Sterntopologien lediglich einen minimalen Sendeaufwand zwischen Sender und allen Empfängern implizieren, bietet sich die Klasse der mehrschichtigen Ringe bzw. Ringnetze am geeignetsten für reale Anwendungen an. Konkret werden im folgenden Simulationen für 2- bzw. 3-Fehlertoleranz auf mehrschichtigen Ringen mit 2 bzw. 3 Schichten sowie jeweils 10 Bridges pro Ring durchgeführt. Wie zuvor wird stets ein Netzwerkknoten pro Bridge betrachtet. Im Folgenden werden mehrschichtige Ringe mit $m$ Schichten und $n$ Bridges pro Schicht als `multiring_m×n` bezeichnet.

### 7.4.2. Erweiterung von Funktionen

Während bei der Einfehlerannahme Distributing Bridge und Checking Bridge direkt miteinander verbunden sind, sind diese in Systemen entsprechend einer Mehrfehlerannahme durch $f-1$ Prechecking Bridges getrennt. Nachrichten müssen im primären Routing folglich ebenfalls auf mehreren redundanten Pfaden von der Distributing Bridge zu einer Checking Bridge versendet werden, so dass auch im fehlerfreien Fall jede Checking Bridge mehrere Duplikate einer korrekten Nachricht erhält. Um zu verhindern, dass nicht mehrere redundante Wellen von ein und derselben (möglicherweise fehlerhaften) Checking Bridge initiiert werden und damit eine statisch erhöhte Netzwerklast entsteht, wurde die Simulation um eine Duplikaterkennung in den Bridges ergänzt. Hierbei werden exakt gleiche Nachrichten, die zuvor bereits weitergeleitet wurden unterdrückt, verfälschte Nachrichten jedoch als verschieden vom Original behandelt und nicht unterdrückt. Bei einer realen Implementierung können dabei ExFABAN-spezifische Informationen verwendet werden, um die Duplikaterkennung effizienter zu gestalten. Konkret kann die Zustellzeit in einer Nachricht verwendet werden, um gespeicherte Informationen vergangener Nachrichten bereits dann zu verwerfen, wenn ein nachrückende Kopie einer Nachricht die Überprüfung der Zustellzeit nicht mehr einhalten könnte.

### 7.4.3. Simulation und Evaluation

#### Einstellung der Senderaten und Ermittlung der Zustellzeiten

Die Grundvoraussetzung für eine problemfreie Ausführung von ExFABAN ist analog zum Einfehlerfall die Anforderung, dass die Summe der Senderaten, die durch die Rundsprüche entstehen, nicht die Verbindungsgeschwindigkeiten übersteigen. Während im Einfehlerfall jeder Sender jeden FABAN-Rundspruch entlang zwei redundanter Wellen verteilt, verteilt ExFABAN im allgemeinen Fall $f \geq 1$ Rundspruchnachrichten entlang mindestens $f+1$ Wellen. Folglich ergibt sich analog zu Bemerkung 7.1 im Mehrfehlerfall folgende Anforderung, wobei $N_S$ die Anzahl redundanter Ströme eines Senders $S \in \mathcal{S}$ bezeichnet:

$$\sum_{S \in \mathcal{S}} N_S \cdot R_S(t) \leq R_{Link}. \tag{7.31}$$

Nachfolgende Tabelle gibt eine Übersicht über die maximalen Senderaten für die hier betrachteten Topologien:

| Topologie | Linkrate | Ströme pro Sender | Senderate |
|---|---|---|---|
| `multiring_2×10` | 1000 $Mbps$ | 4 | 12.5 $Mbps$ |
| `multiring_2×10` | 100 $Mbps$ | 4 | 1.25 $Mbps$ |
| `multiring_3×10` | 1000 $Mbps$ | 6 | 5.55 $Mbps$ |
| `multiring_3×10` | 100 $Mbps$ | 6 | 0.55 $Mbps$ |

TABELLE 7.11.: Übersicht über maximale Senderaten auf mehrschichtigen Ringen

Analog zu der Anpassung der Senderaten an die erhöhte Anzahl redundanter Ströme, muss die Abschätzung für die Berechnung der Größe $\Delta_{Busy}$ ebenfalls an die erhöhte Anzahl der Ströme angepasst werden. Aus Lemma 7.3 ergibt sich damit für den vereinfachten Fall gleicher maximaler Nachrichtengrößen, die folgende Abschätzung:

$$(H+1)\cdot\Delta_{Busy} \leq [N_S\cdot(n\cdot m)-1]\cdot\Delta_{Link}. \tag{7.32}$$

### Simulation ohne Fehlerinjektion

Zur Validierung vorangehender Überlegungen wurden zunächst Simulationen ohne Fehlerinjektionen durchgeführt. Hierbei wurden auf den beiden exemplarisch ausgewählten Netzwerktopologien `multiring_2×10` und `multiring_3×10` Simulationen mit jeweils 100 Wiederholungen durchgeführt, wobei jeder Sendeknoten 1000 Nachrichten der Größe $1kBit$ gemäß den Senderaten aus Tabelle 7.11 generiert.

Die Simulationsergebnisse in Abbildung 7.25 bestätigen die Überlegungen, indem diese zeigen, dass die Senderaten nicht zu hoch gewählt sowie die $\Delta_{Busy}$ nicht zu gering berechnet wurden. Rundspruchnachrichten kommen bei allen Empfängern in exakt vier- bzw. sechs-facher Ausführung an und eine zusätzliche Netzwerklast wird durch die Duplikaterkennung in Bridges verhindert.

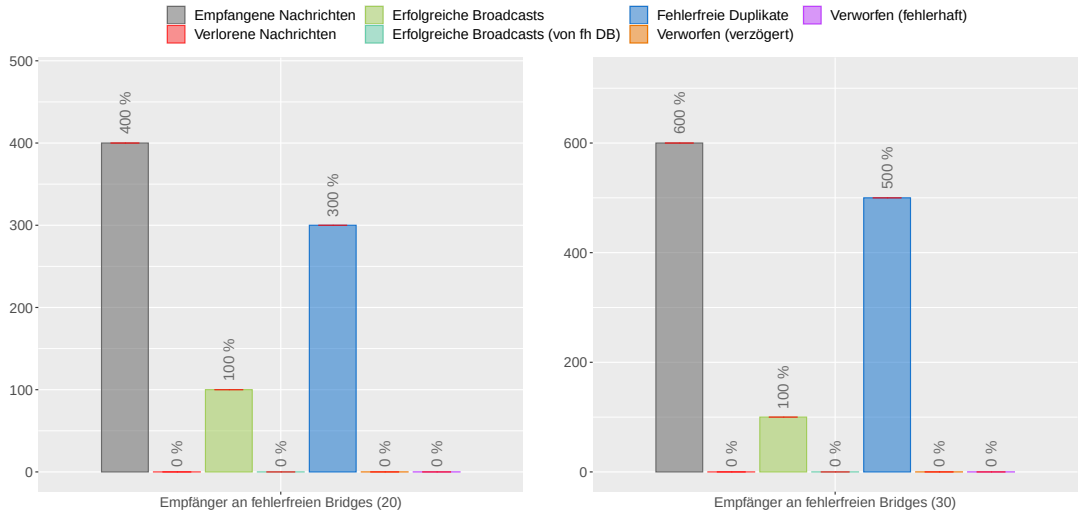



Abbildung 7.25.: Simulationsergebnisse von Simulationen ohne Fehlerinjektionen auf den Topologien `multiring_2×10` (links) und `multiring_3×10` (rechts).

### Simulation mit Fehlerinjektionen

Zieht man zusätzlich Fehlerinjektionen in Betracht, so werden Probleme sichtbar, welche im Protokollablauf bei der Einfehlerannahme nicht existieren. Betrachtet wird im folgenden explizit der Fall $f = 3$ auf der Topologie `multiring_3×10` und einer physikalischen Verbindungsgeschwindigkeit von $1000 Mbps$. Neben der gleichen Parametrisierung der fehlerinjektionslosen Simulationen des vorherigen Abschnitts wurden alle Fehlerarten mit einer Wahrscheinlichkeit von 0.2 aktiviert. Die konkrete Wahl der drei fehlerhaften Bridges wurde in separaten Simulationsdurchläufen variiert, weisen aber ebenfalls gleiche Ergebnisse auf, so dass es genügt, hier keine konkrete Unterscheidung zu treffen[1].

Zunächst wurden Simulationen mit einer Senderate von $5.55 Mbps$ pro Sender durchgeführt. Damit entsteht bei jedem Empfänger ein eingehender Datenstrom von

$$30 \cdot 6 \cdot 5.55 Mbps = 999 Mbps$$

und bietet somit nur minimalen Spielraum für zusätzliche Last. Abbildung 7.26 zeigt die Ergebnisse, welche mit einem Nachrichten-Verlustanteil von fast 90% eindeutig unbefriedigend sind.

Während des primären Routings werden Nachrichten von der Distributing Bridge zu jeder Checking Bridge jeweils auf mehreren redundanten Pfaden gesendet und durch

[1] Alle Simulationskonfigurationen sowie Ergebnisse können auf dem beigefügten Datenträger eingesehen werden

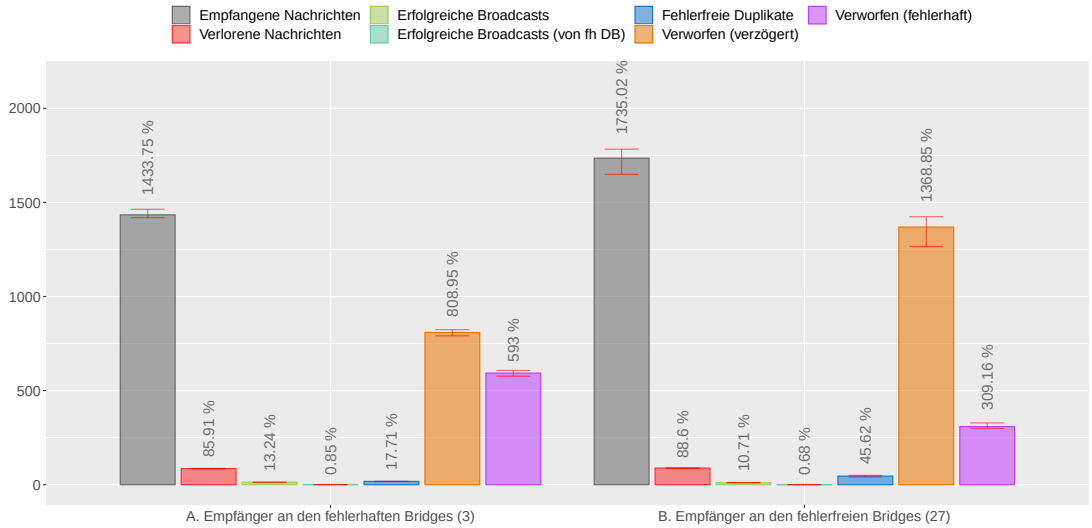



Abbildung 7.26.: Simulationsergebnisse von Simulationen mit allen Fehlerinjektionen, die möglich sind, auf der Topologie `multiring_3×10`, bei hohen Senderaten.

die Duplikaterkennung in den Bridges verhindert, dass gleiche Wellen mehrfach initiiert werden. Durch fehlerhafte Prechecking Bridges können Nachrichten mit verfälschtem Inhalt nicht mehr als Duplikat der fehlerfreien Nachricht erkannt werden, so dass ein Teil der verfälschten Nachrichten korrekt durch das Netzwerk geroutet werden kann und somit durch Initiierung von zusätzlichen Wellen eine zusätzliche Nachrichtenlast auf dem gesamten Netzwerk erzeugt. Des Weiteren ist es sicher, dass die Unteilbarkeit des Rundspruchs nicht eingehalten wurde. Jede Checking Bridge, die eine Welle mit einer unverfälschten Nachricht initiiert, leitet diese auch an Empfänger, die mit der Checking Bridge verbunden sind, weiter, welche die Nachricht mit höherer Wahrscheinlichkeit zustellen als Nachrichten von entfernten Sendern. Folglich dürften, sollte Unteilbarkeit eingehalten werden, keinerlei Rundsprüche verloren gehen, was nicht der Fall ist.

Abbildung 7.26 beweist ohne weitere Untersuchungen nicht, dass alle fehlgeschlagenen Rundsprüche auf diese Begründung zurückzuführen sind. Dies wird jedoch mit einer weiteren Reihe von Simulationen belegt, in welchen bei sonst gleichen Parameterwerten die Senderate auf 1/10 reduziert wurde. Abbildung 7.27 zeigt das Ergebnis der Simulationen, aus welchem klar hervorgeht, dass die fehlerhaft redundant erzeugte Wellen, keine Limitierung durch die begrenzte Verbindungsgeschwindigkeit hervorruft. Insbesondere kann aus beiden Simulationen gefolgert werden, dass sowohl die korrekte Funktionsfähigkeit der implementierten ExFABAN-spezifischen Mechanismen sowie der angestrebte Fehlertoleranzgrad grundlegend gewährleistet sind.

Das Problem der Erhöhung des Nachrichtenverkehrs durch Nachrichtenverfälschung

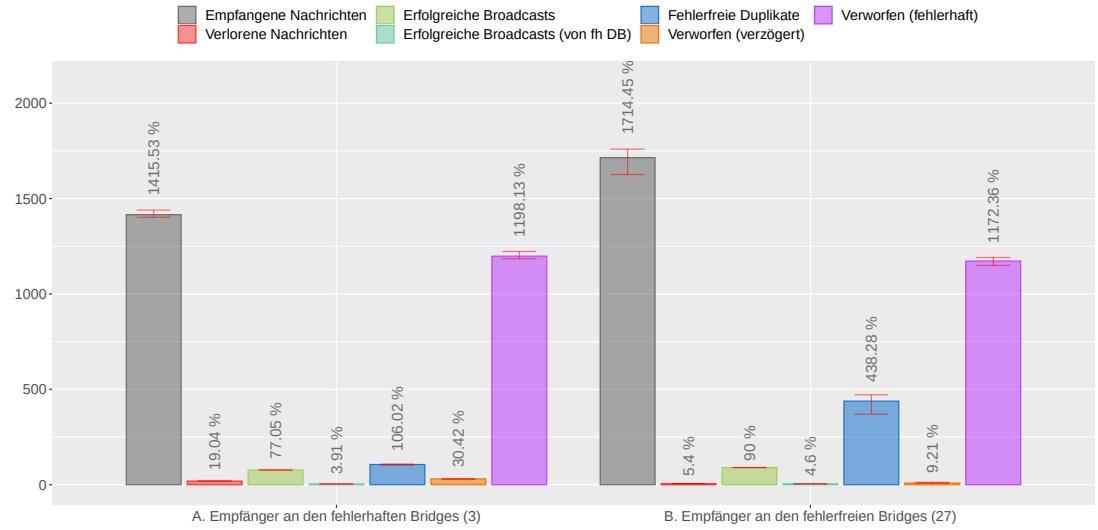



Abbildung 7.27.: Simulationsergebnisse von Simulationen mit beliebigen Fehlerinjektionen auf der Topologie `multiring_3×10`, bei niedrigen Senderaten.

und der damit einhergehenden Überlastung des Netzwerks, kann nicht durch einfache Mittel gelöst werden. Die sicherste Möglichkeit dem entgegenzuwirken, wäre die Implementierung der selben Fehlererkennungsmechanismen der Empfänger in den Bridges. Dies würde die Komplexität der Aufgaben einer Bridge deutlich erhöhen, was einerseits bei jeder Weiterleitung einer Nachricht zusätzliche Zeit kosten würde und andererseits dem ursprünglich formulierten Ziel entgegenstünde, möglichst geringe Anpassungen an den Funktionalitäten der Bridges zu fordern. Doch selbst in diesem Fall hätte man keine 100%-ige Sicherheit, sondern ist stets an die zu Grunde gelegten Sicherheitsmaßnahmen (CRC, Signatur) und die damit verbundenen Fehlerraten, welche sich durch geringen Zusatzaufwand (z.B. längere Signaturen) stets enorm verbessern lassen, gebunden. Des Weiteren ist zu beachten, dass die gewählten Fehlerwahrscheinlichkeiten in den Simulationsbedingungen unrealistisch hoch gewählt wurden und folglich das Problem in der Realität an Wichtigkeit verlieren sollte. Nachfolgender Abschnitt untersucht abschließend das Protokoll unter realistischen Parametern.

## 7.5. Simulationen mit realen Fehlerwahrscheinlichkeiten

Während in den vorangehenden Abschnitten Simulationen durchgeführt wurden, um gezielt Fehler und allgemeinere Probleme in der Implementierung, der Protokollspezifikation oder der Anwendungsumgebung zu finden und Grenzen in der Anwendung zu identifizieren, beschäftigt sich dieser Abschnitt mit der Anwendbarkeit des Protokolls, insbesondere

bei höherer Fehlertoleranzannahme, in der Realität. Besonderes Interesse kommt der Frage zu, inwieweit die Probleme der Netzüberlastung unter realen Bedingungen Bestand haben.

Zunächst sei angemerkt, dass Protokolle mit einem gegebenen Fehlertoleranzgrad $f \geq 1$ in realen Umgebungen nur fehlerfrei funktionieren, so lange die Fehlerannahme nicht verletzt wird, d.h. wenn nicht mehr als $f$ Komponenten einzelne Phasen des Betriebs, z.B. einen Rundspruch, fehlerhaft beeinflussen. In der Praxis kann dies theoretisch niemals vollständig ausgeschlossen werden, sondern lediglich mit sehr geringen Wahrscheinlichkeiten praktisch ausgeschlossen werden. Die analoge Argumentation gilt für den Fall, dass Fehler derart den Ablauf beeinflussen, dass eine Zusatznachrichtenlast auf dem Netzwerk entsteht, so dass durch die Grenzen physikalischer Verbindungsgeschwindigkeiten ein Stau entsteht. Abbildung 7.28 skizziert, dass eine Erhöhung der Datenströme durch einzelne Fehler unproblematisch ist. Sollten diese aber höher frequentiert auftreten, so dass die Einflüsse zeitgleich auftreten und sich gegenseitig verstärken, dann kann dies zu einer Überlast und somit zu Nachrichtenverlusten führen.

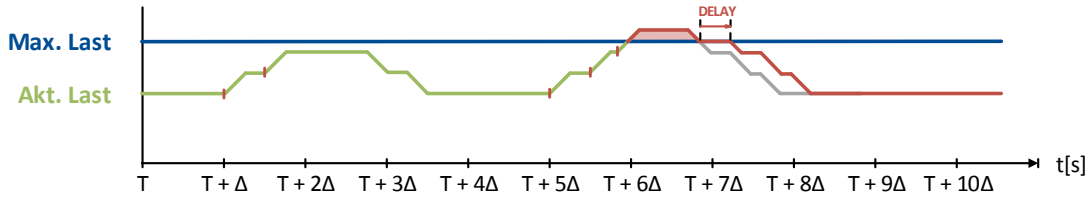



Abbildung 7.28.: Visualisierung von zusätzlichen Nachrichtenlasten auf dem Netzwerk durch Erhöhung von Strömen durch Nachrichtenverfälschung. Der zweite Anstieg stellt eine Erhöhung der Last über die maximal erlaubte Last dar, welche vom Netzwerk nicht mehr ohne zusätzliche Verzögerung oder Nachrichtenverluste durch Pufferüberläufe übertragen werden kann.

Bei kleinen Fehlerwahrscheinlichkeiten ist dieses Problem jedoch zu vernachlässigen. Diese Behauptung wird in einer letzten Reihe von Simulationen auf der Topologie `multiring_3×10`, auf welcher zuvor die Probleme der Überlastung der Verbindungen zu beobachten waren, mit möglichst realen Fehlerwahrscheinlichkeiten überprüft.

Fehlerraten realer Hardwarekomponenten bzw. einzelner Bauteile werden im Allgemeinen nicht offen angegeben. Es existieren jedoch eine Reihe von Publikationen und Arbeiten, welche sich mit dem Thema befassen. Zuverlässigkeitskenngrößen wie die Mean Operating Time to Failure (MTTF), Mean Time between Failures (MTBF) oder die Mean Time To Repair (MTTR) werden in einigen Publikationen untersucht, aus welchen man reale Fehlerraten für realistische Simulationen approximieren kann [Kra09; CP13]. Scheer und Dolezilek haben Zuverlässigkeitsberechnungen für Ethernet Netzwerke sowie einzelnen Komponenten durchgeführt [SD02]. Dabei wurden in einer Reihe von

Ethernet Komponenten sowohl Ethernet Switches als auch Ethernet Router untersucht. Letztere weisen dabei die höchste stationäre Nichtverfügbarkeit unter den verglichenen Komponenten in Höhe von

$$p = 577 \cdot 10^{-6} \tag{7.33}$$

auf, wobei eine MTTR von 48 Stunden angenommen wurde. Da die Nichtverfügbarkeit die Wahrscheinlichkeit angibt, dass eine Komponente zu beliebigem Zeitpunkt ihre Funktion nicht erfüllen kann, entspricht diese Wahrscheinlichkeit genau der Fehlerwahrscheinlichkeit für weiterzuleitende Nachrichten, so dass die Simulation mit dem Wert aus (7.33) durchgeführt wird. Um das Problem aus Abbildung 7.28 möglichst unwahrscheinlich zu machen, werden Senderaten derart gewählt, dass die gesamte Last auf dem Netzwerk stets einen Puffer nach oben hat, um keine Überlast zu erzeugen. Es wäre denkbar, analytisch an das Problem heranzugehen und eine maximal erlaubte Senderate zu bestimmen, dies erfordert jedoch stets Annahmen bzw. Restriktionen die getroffen werden müssen, um nicht mit einer endlos großen Menge von Einflüssen und Parametern arbeiten zu müssen. Zu möglichen Parametern zählen

- Anzahl der fehlerhaft erzeugten Duplikate
- Ungünstige Kombinationen von Fehlern (Duplizierung + Bitflip kann von Bridges nicht als Duplikat erkannt werden)
- Aktuelle Last des Netzwerks (kann durch Verzögerungsfehler nicht abgeschätzt werden)
- Babbling Component als zusätzlicher Störfaktor

wobei die Liste noch weitergeführt werden kann. Aus diesem Grund wurde der analytische Ansatz nicht weiter verfolgt.

Die Simulation aus Abbildung 7.26 wurde mit der Fehlerwahrscheinlichkeit aus (7.33) wiederholt, wobei die Senderaten so eingestellt wurden, dass das Netz im fehlerfreien Fall zu 100%, zu 90% sowie zu 75% ausgelastet ist.

Bereits bei den höchst möglichen und damit gleichen Senderaten wie in Abbildung 7.26, ist durch die Anpassung der Fehlerwahrscheinlichkeiten auf realistische Werte in Abbildung 7.29 eine deutliche Verbesserung der erfolgreichen Broadcasts von 10% auf über 77% zu beobachten.

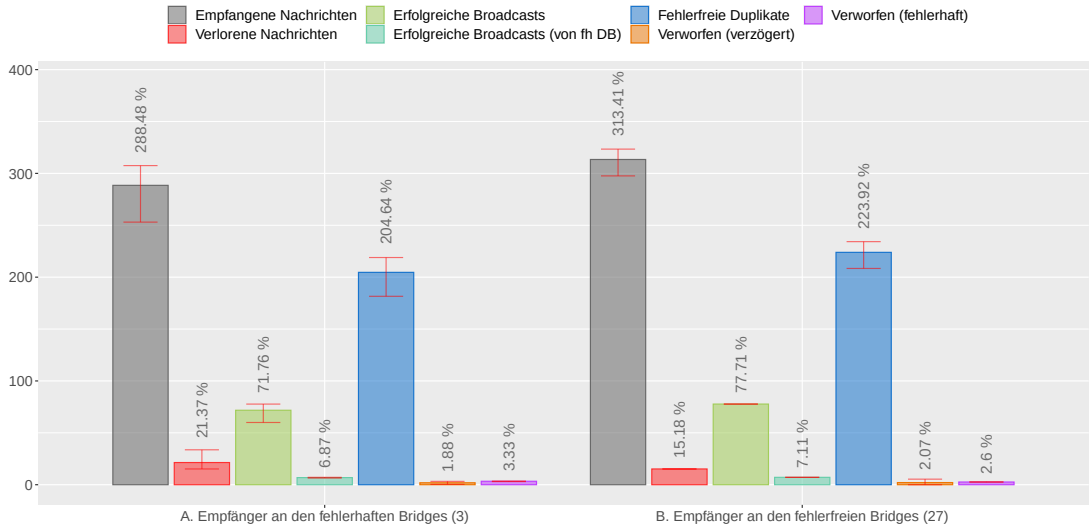



Abbildung 7.29.: Simulationsergebnisse von Simulationen mit beliebigen Fehlerinjektionen auf der Topologie `multiring_3×10`, bei 100% der maximalen Senderaten und einer Fehlereintrittswahrscheinlichkeit von $p = 577 \cdot 10^{-6}$.

Nach der Reduktion der Senderaten auf 90% der maximal erlaubten Senderaten, sind in Abbildung 7.30 nur noch weniger als 1% nicht erfolgreicher Rundsprüche zu beobachten.

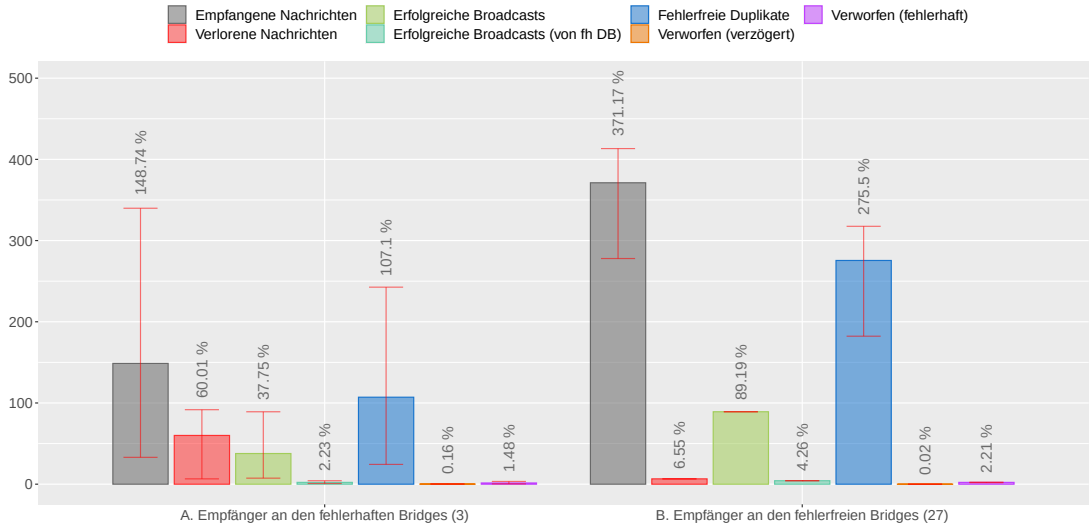



Abbildung 7.30.: Simulationsergebnisse von Simulationen mit beliebigen Fehlerinjektionen auf der Topologie `multiring_3×10`, bei 90% der maximalen Senderaten und einer Fehlereintrittswahrscheinlichkeit von $p = 577 \cdot 10^{-6}$.

Eine schrittweise Reduktion der Senderaten auf 75% der maximalen Senderaten erzielte in Abbildung 7.31 schließlich Resultate, bei welchen keine Verluste durch Überlast zu beobachten sind und insgesamt alle zu tolerierenden Fehler nachweislich toleriert werden. Es werden also alle Rundsprüche von Sendern, welche an fehlerfreie Bridges angeschlossen sind, in allen Empfängern an fehlerfreien Bridges korrekt empfangen.

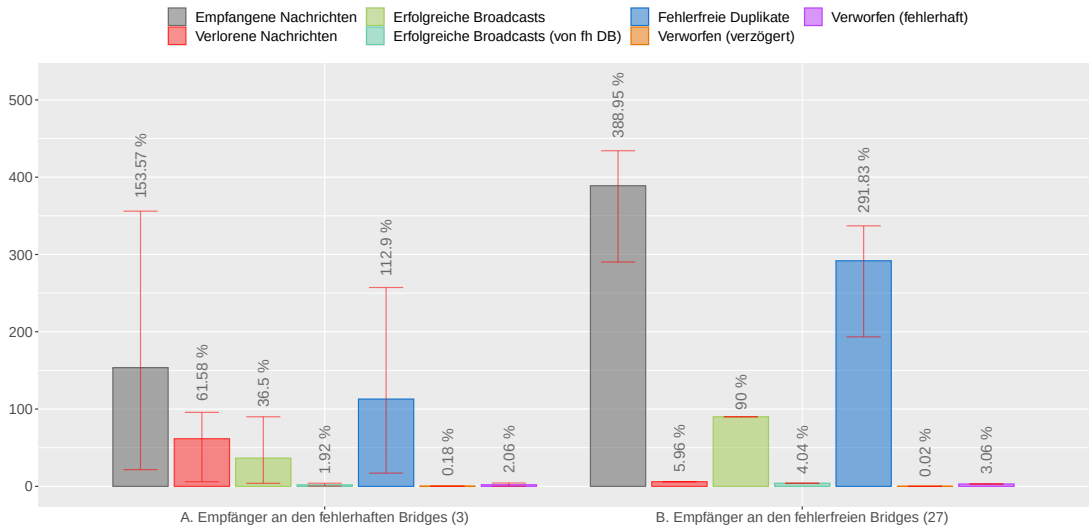



Abbildung 7.31.: Simulationsergebnisse von Simulationen mit beliebigen Fehlerinjektionen auf der Topologie `multiring_3×10`, bei 75% der maximalen Senderaten und einer Fehlereintrittswahrscheinlichkeit von $p = 577 \cdot 10^{-6}$.

## 7.6. Zusammenfassung

Eine rein theoretische Betrachtung, Entwicklung und Untersuchung sowohl von FABAN als auch von ExFABAN ist ein essentieller Schritt in der Entwicklung eines Protokolls, ist aber für die praktische Anwendung aufgrund möglicher unvorhersehbarer Komplikationen und Probleme nicht ausreichend. Um diesen vorzubeugen wurde in diesem Kapitel ExFABAN als Simulation in dem Framework OMNeT++ implementiert, um simulativ mögliche Probleme in der realen Applikation des Protokolls zu lokalisieren und Lösungen zu untersuchen.

Dabei wurde ein normales Ethernet-Netzwerk zugrunde gelegt, welches nicht die notwendige Voraussetzung der garantierten pro-Hop-Latenzen erfüllt, um zu untersuchen, inwieweit auf möglichst simpler, handelsüblicher Hardware das Protokoll implementiert werden kann. Um die notwendigen oberen Schranken der pro-Hop-Latenzen zu erhalten, wurde der Ansatz verfolgt, Senderaten zu beschränken und maximale Verzögerungen

vorherzusagen. Das Ergebnis dieses Ansatzes war überraschend gut, da die Vorhersagen nicht nur im fehlerfreien Fall, sondern auch bei den meisten Fehlerinjektionsarten zutreffend waren. Probleme, bei denen unteilbare Rundsprüche nicht bei allen fehlerfreien Empfängern ankamen, waren stets auf überlastbedingte Verzögerungen zurückzuführen und hatten nie eine andere Ursache.

Das erste durch Simulationen lokalisierte Problem war das Problem der fehlerhaften Reflektion durch fehlerhafte Weiterleitungen. Dabei wurde eine Nachricht einer Welle als Nachricht einer anderen Welle behandelt und führte zu einer Vervielfachung von einzelnen Wellen und somit zu einer Überlastung des Netzes. Dieses Problem konnte erfolgreich gelöst werden durch die Erweiterung des Nachrichtenformats um die Information der initiierenden Checking Bridge und der damit verbundenen Erweiterung der Routing-Lookup-Tabellen.

Insbesondere bei der Mehrfachfehlerannahme hat sich gezeigt, dass eine Duplikaterkennung in Bridges zwingend notwendig ist, um mehrfache Initiierung von Wellen zu verhindern. Ungünstige Nachrichtenverfälschungen können dennoch dazu führen, dass verfälschte Duplikate nicht als solche erkannt werden, in Folge dessen mehrfach Wellen initiiert werden und dadurch das Netz überlastet wird. Eine abschließende Untersuchung der Anwendbarkeit des Protokolls unter realen Fehlerraten hat gezeigt, dass diese Probleme vernachlässigbar sind bzw. die maximale Senderate eines Senders gezielt verringert werden kann, um eine beliebig kleine Wahrscheinlichkeit für den Eintritt des Problems zu erhalten.

Kapitel 8

# Entwicklung eines Prototyp-Netzwerks auf realer verteilter Hardware

Die Verifikation von theoretischen Neuentwicklungen mittels Simulationen, so wie in vorangehendem Kapitel detailliert im Bezug auf FABAN thematisiert, ist eine häufige und auch sehr gute Methode der zusätzlichen Untersuchung und Detektion von Problemen, welche in der Theorie oft nur schwer erkannt werden können. In dieser Arbeit, konnten mit Simulationen wichtige Probleme erkannt und behandelt werden, so dass bereits eine annähernd vollständige Auseinandersetzung mit FABAN und ExFABAN vorliegt. Dessen ungeachtet bringen Simulationen einen gewissen Grad der Abstraktion mit sich, durch welchen mögliche Probleme versteckt werden könnten. Aus diesem Grund wurde neben einer Simulation auch an einer prototypischen Hardwareimplementierung gearbeitet, welche in diesem Kapitel als Abschluss dieser Arbeit thematisiert wird. Im Fokus steht dabei die Entwicklung eines Prototyp-Systems und die Validierung, die zeigen soll, dass das Konzept der Implementierung des Protokolls, wie es theoretisch entwickelt und mit Hilfe von Simulationen ergänzt wurde, ohne weitere Änderungen auf reale Hardware übertragbar ist.

Im ersten Abschnitt dieses Kapitel wird zunächst die Wahl der Hardware auf den Einzelplatinencomputer mit integriertem Switch, den BananaPi R1 (BPi R1), begründet. Im darauffolgenden Abschnitt wird das Konzept der Implementierung des grundlegenden Kommunikationssystems zwischen den einzelnen BPi R1s sowie einem zentralen Kontrollrechner vorgestellt und beschrieben. Zusätzlich wird eine Schnittstelle vorgestellt, die es ermöglicht, beliebige, eigenständige Protokolle zu implementieren, welche nahtlos in das grundlegende Kommunikationssystem eingebettet werden können.

Im dritten Abschnitt werden Implementierungen erweiterter Funktionen beleuchtet.

Dabei handelt es sich sowohl um für den Betrieb notwendige Funktionen wie einer funktionierenden Uhrensynchronisation, einer Topologieerkennung um Plug and Play Eigenschaften zu ermöglichen und Mechanismen für die FABAN-spezifische Konfiguration einzelner Komponenten sowie um testbedingte Mechanismen wie der Injektion von Fehlern und der Erstellung von Statistiken.

Abschließend wird das implementierte System evaluiert, das heißt es werden (technische) Grenzen des Systems, sowie die Zuverlässigkeit des Sendens von FABAN-Rundsprüchen im fehlerfreien Betrieb untersucht und der qualitative Mehrwert der vorliegenden Hardwareimplementierung im Vergleich zur Simulation reflektiert.

## 8.1. Die Wahl der Hardware: BananaPi R1

Die Auswahl an Hardware für eine reale Implementierung von Bridges, welche FABAN ausführen und zu Bridge-Connected-Networks vernetzt werden sollen, ist groß. Verschiedene Faktoren spielen dabei wichtige Rollen: Performanz, Nutzbarkeit, Erweiterbarkeit sowie Kosten.

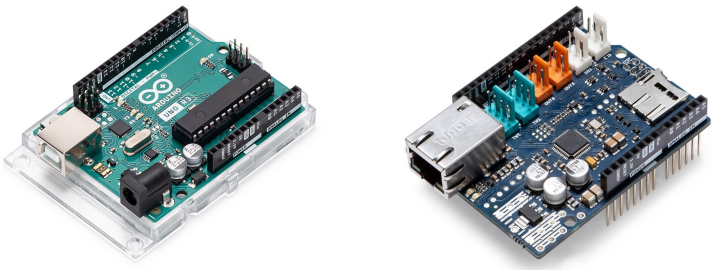


Abbildung 8.1.: Fotos eines Arduino Uno (links) sowie eines Arduino Ethernet Shields (rechts) [Ard21]

Mikrocontroller Boards, wie zum Beispiel die Arduino Plattform, sind als ein erstes wichtiges Beispiel zu nennen. Arduino Boards sind mit Preisen im niedrigen zweistelligem Bereich extrem kostengünstig, sind quelloffen und sind über exzellente Software angenehm zu programmieren. Des Weiteren sind einzelne Boards über so genannte Arduino Shields erweiterbar, so dass in dem hier behandelten konkreten Anwendungsfall ein Ethernet-verbundenes Netz von Arduino Mikrocontrollern in Kombination mit Arduino Ethernet Shields - siehe Abbildung 8.1 - denkbar wäre [Ard21]. Auf der Seite der Nachteile stehen die geringe Prozessorgeschwindigkeit, der streng limitierte SRAM sowie insbesondere die langsame Ethernet Verbindungsgeschwindigkeit, welche primär durch die Verbindungsschnittstelle SPI zum Arduino, sowie die geringe Prozessorgeschwindigkeit auf Geschwindigkeiten im $kBit/s$ Bereich beschränkt wird.

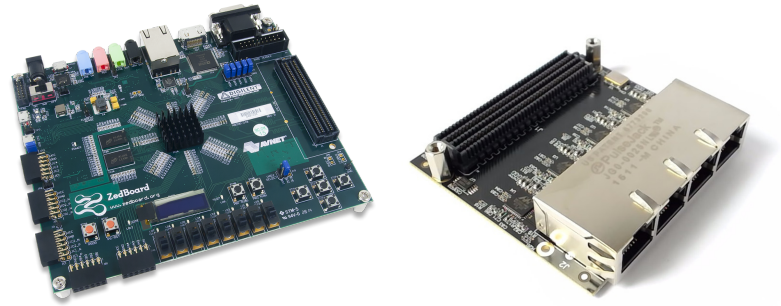


Abbildung 8.2.: Fotos eines Einsteiger-FPGAs (links) sowie einem Erweiterungsmodul mit 4 Gigabit Ethernet Ports (rechts) [Xil21; Eth21]

Im Gegensatz zu langsamen Mikrocontrollern, gehören FPGA-basierte Ansätze zu solchen Lösungen, welche die höchste Performanz erzielen können. Die Kosten für Field Programmable Gate Arrays (FPGAs) reichen dabei von mehreren hundert bis hin zu mehreren Tausend Euro, wobei mit dem steigenden Preis auch die Qualität bzw. der Funktionsumfang steigt. Neben dem sehr hohen Preis haben FPGAs auch den Nachteil deutlich höherer Implementierungskomplexität sowie geringer Vergleichbarkeit zu handelsüblichen Routern, für welche FABAN konzipiert wurde.

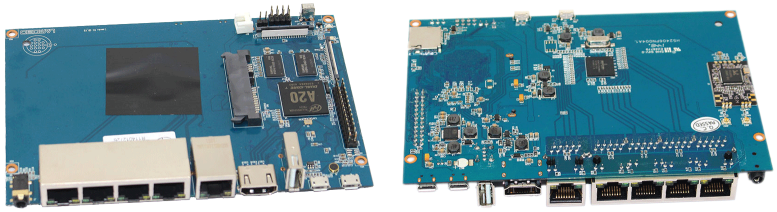


Abbildung 8.3.: Fotos der Vor- und Rückseite des BananaPi R1 Mikrocontrollers mit integriertem Switch [Sin21]

Als dritte Option wurden bei der Suche nach geeigneter Hardware sowohl handelsübliche als auch experimentelle Router in Betracht gezogen. Mit alternativen Firmwares für Router, wie zum Beispiel das Linux-basierte *OpenWRT*, lassen sich an handelsüblichen Routern diverser Hersteller umfangreiche Änderungen durchführen. Der Einzelplatinencomputer BPi R1 ist eine Kombination eines mit einem *Raspberry Pi* vergleichbaren Einzelplatinencomputer und einem Switch und wurde als Basishardware endgültig ausgewählt. Der BPi R1 verfügt über einen $1GHz$ Dualcore Prozessor, $1GB$ DDR3 Speicher und einen integrierten 5-Port Switch, bestehend aus vier LAN und einem WAN Port. Der integrierte Switch ist ein BCM3125 mit zwei RGMII GbE Ports, dessen Datenverkehr über MDIO Konfigurationen mittels VLANs getrennt werden kann. Als Betriebssystem wird das Debian basierte Betriebssystem *Rasbian* verwendet. Der WAN Port wird für

Steuerungs-, Monitoring- sowie Debuggingzwecke verwendet, während die LAN Ports für Direktverbindungen zwischen einzelnen BananaPi R1 verwendet werden.

## 8.2. Implementierungskonzept des grundlegenden Kommunikationssystems

Bei der Umsetzung der Protokolllogik auf den BPi R1s wurde ein möglichst modularer Ansatz verfolgt, anstatt die Logik in einem einzigen Programm zu bündeln. Die Separation eines Programms in unterschiedliche, unabhängige und über Schnittstellen kommunizierende Teilprogramme ermöglicht dabei die gemeinsame Verwendung von zentralen Funktionen, wie zum Beispiel das Senden von Steuernachrichten, die Nutzung von Ausgaben oder die Bereitstellung gemeinsamer Grundgerüste für die Protokollentwicklung. Das nachfolgende Unterkapitel stellt das Konzept des resultierenden, modularen Grundsystems vor, ohne auf Implementierungsdetails einzugehen.

### 8.2.1. Modulares Grundsystem

Die nachfolgende Abbildung 8.4 stellt den Aufbau des final implementierten modularen Grundsystems dar. Beschreibungen nachfolgender Unterabschnitte nehmen stets Bezug auf diese Abbildung.

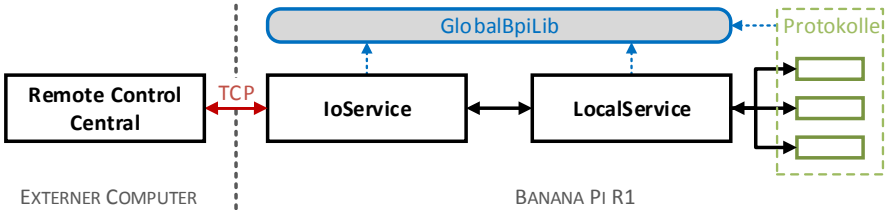



Abbildung 8.4.: Designkonzept der implementierten Softwarearchitektur auf den BPi R1s sowie einem externen Steuer- und Monitoringrechner. Durchgehende Pfeile symbolisieren Kommunikationen zwischen einzelnen Prozessen innerhalb eines BPi R1s bzw. zum externen Rechner. Gestrichelte Pfeile symbolisieren Abhängigkeiten zur gemeinsam genutzten Bibliothek *GlobalBpiLib*.

#### GlobalBpiLib

Von mehreren Teilkomponenten des gesamten modularen Systems gebrauchte Funktionen werden in der C++-Bibliothek *GlobalBpiLib* implementiert. Dazu zählen Funktionen zur Kommunikation über Prozess-Pipes, vereinheitlichten Ausgaben, für Zugriffe auf Konfigurationsdateien oder die Bereitstellung von Uhren und zugehöriger Operationen.

Zusätzlich enthält die *GlobalBpiLib* Klassen, welche Basisimplementierungen beliebiger Protokolle realisieren und über Vererbung genutzt werden können. Details folgen in Abschnitt 8.2.1.

### IoService

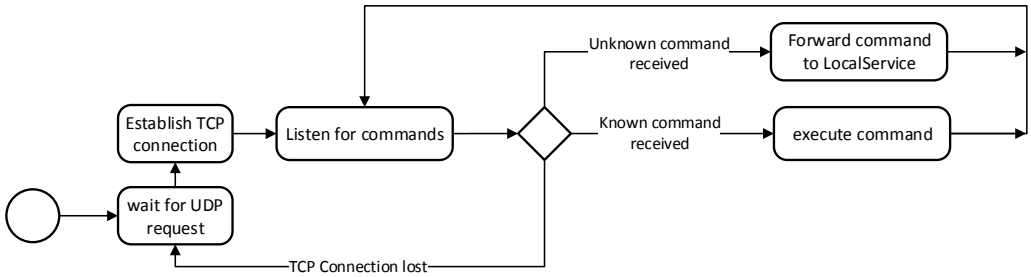



Abbildung 8.5.: Programmablaufdiagramm des *IoService*

Der *IoService* implementiert die Kommunikationsschnittstelle zwischen einem BPi R1 und einem externen Computer über eine TCP-Verbindung. Der Dienst wartet auf Verbindungsanfragen über UDP-Rundsprüche auf dem WAN-Port des BPi R1, verbindet sich mit der *Remote Control Central*, welche die Verbindungsanfrage initiiert hat und wartet anschließend auf eingehende Befehle auf der TCP-Verbindung. Der *IoService* selbst versteht eine Menge von Basisbefehlen, zu welchen die Konfiguration des Dienstes, die Kompilierung, Ausführung und Terminierung des *LocalService* sowie das Ausführen von Systembefehlen gehören. Empfangene Befehle von einer extern verbundenen *Remote Control Central*, die nicht zugeordnet werden können, werden unverändert an den *LocalService* weitergeleitet, während vom *LocalService* empfangene Nachrichten vollständig über die TCP Verbindung zur *Remote Control Central* gesendet werden. Der Hauptgrund der Separation in *IoService* sowie *LocalService* besteht darin, eine stabile Schnittstelle zu haben um jederzeit Fernzugriff zu allen BPi R1s zu ermöglichen.

### LocalService

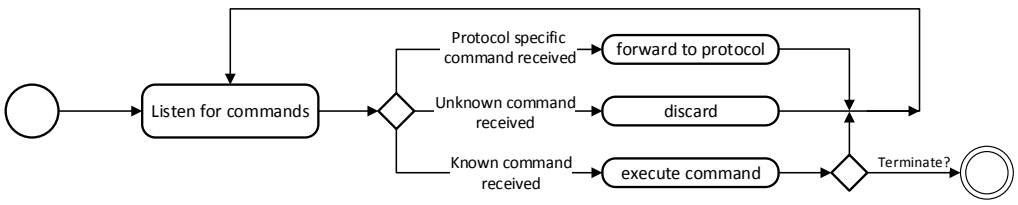



Abbildung 8.6.: Programmablaufdiagramm des *LocalService*

Der *LocalService* stellt die Steuerungsschnittstelle für Protokolle dar. Ausgeführt vom *IoService* wartet der *LocalService* auf eingehende Befehle zur Konfiguration, Kompilierung, Ausführung und Terminierung von Protokollen, sowie auf protokollspezifische Befehle, welche an entsprechende ausgeführte Protokolle weitergeleitet werden. Unbekannte Befehle werden verworfen. Nachrichten von ausgeführten Protokollen, werden an den *IoService* zur Weiterleitung an die *Remote Control Central* unverändert gesendet.

#### Nebenläufige Protokolle

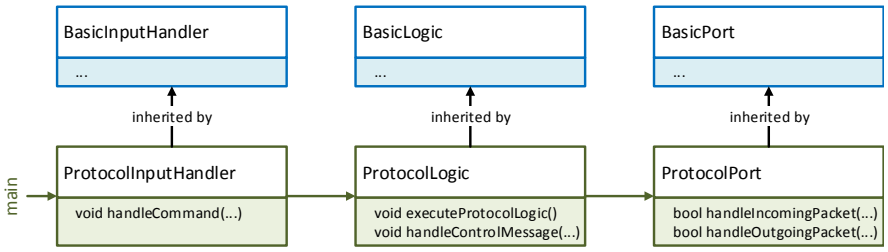



Abbildung 8.7.: Darstellung der Klassenstruktur der Protokollimplementierung. Horizontale Pfeile stellen Abhängigkeiten zwischen einzelnen Klassen dar, während vertikale Pfeile Vererbungen symbolisieren.

Protokollimplementierungen sind C++ Programme, welche über die GbE LAN Ports mit anderen BPi R1 kommunizieren. Die Kommunikation erfolgt über *RawSockets*, um Zugriff auf TCP und IP Header zu erhalten [Heu+17]. Um sowohl die Schnittstellen für die Steuerung sowie die Kommunikation über den *LocalService* zu ermöglichen, müssen alle (nebenläufig ausführbare) Protokolle grundlegende Basisfunktionen teilen, so dass deswegen grundlegende Basisimplementierungen aller für die Protokollimplementierung notwendigen Klassen in die *GlobalBpiLib* ausgelagert wurden. Zu den grundlegenden Basisfunktionen einer jeden Protokollimplementierung gehören das Empfangen von Nachrichten/Befehlen vom *LocalService* (BasicInputHandler), Empfang- und Senderoutinen für Ports (BasicPort) sowie das Puffern und Verarbeiten von Befehlen und Ethernet-Nachrichten gemäß der Protokolllogik (BasicLogic). Durch Ableitung zuvor genannter Basisklassen, können Protokolle simpel implementiert und in das Gesamtsystem nahtlos eingegliedert werden. Abbildung 8.7 zeigt eine kompakte Übersicht der Basisklassen sowie der Funktionen, welche überschrieben werden sollen/dürfen.

### 8.2.2. Zentrale Fernsteuerung

Die *Remote Control Central* ist das Resultat der Überlegungen, eine graphische Benutzerschnittstelle bereitzustellen, um parallel Mengen von BPi R1s mit Protokoll- oder Dienstaktualisierungen zu versehen, sie zu steuern, zu debuggen sowie zu überwachen.

Abbildung 8.8 zeigt ein Bildschirmfoto der in Java/JavaFX geschriebenen, plattformunabhängigen Anwendung.

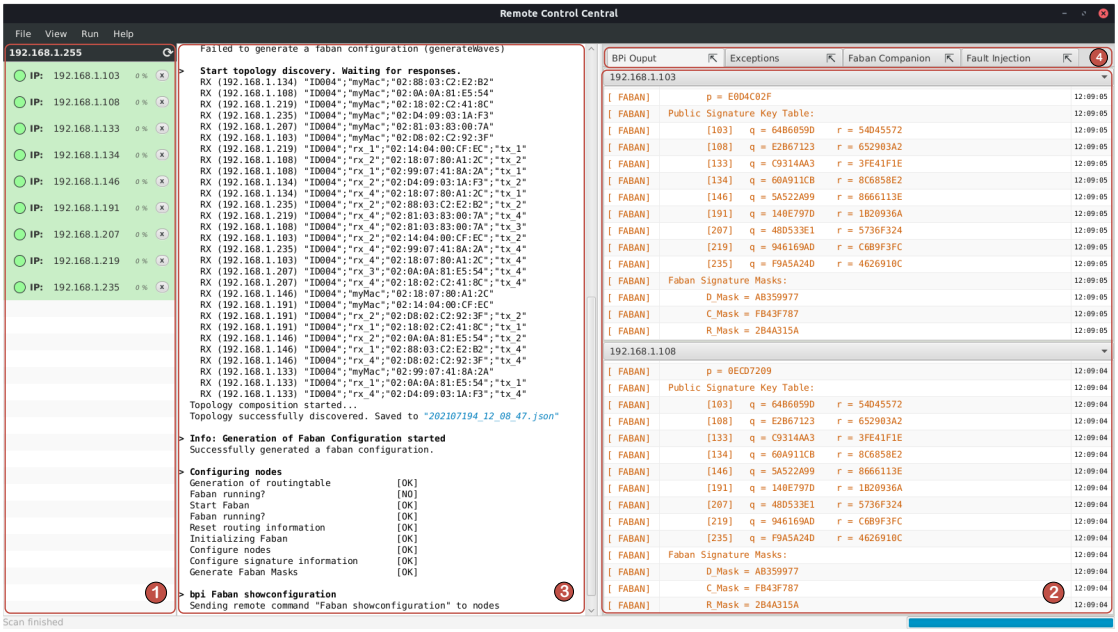



Abbildung 8.8.: Bildschirmfoto der graphischen Benutzeroberfläche der *Remote Control Central*.

Die Kernfunktionen werden nachfolgend kurz und knapp aufgezählt und beschrieben. Auf Details wird wegen wissenschaftlicher Irrelevanz nicht näher eingegangen.

1. BPi R1s mit ausgeführtem *IoService* werden in einem frei wählbarem Subnetzwerk angefragt und automatisch eine Verbindung erstellt. Verbundene Mikrocontroller werden inklusive Verbindungsstatus sowie Prozessorauslastung in der Übersicht angezeigt und können einzeln oder in beliebigen Teilmengen gesteuert werden.
2. Ausgaben verbundener BPi R1s werden individuell angezeigt, wobei Ausgaben einzelner BPi R1s bei Bedarf ausgeschaltet werden können.
3. Das universelle Terminal kann vom Benutzer zur Eingabe von Befehlen für die *Remote Control Central* selbst oder für eine Teilmenge von verbundenen BPi R1s genutzt werden.
4. Die zuvor beschriebenen Mechanismen sind vollkommen ausreichend, um Arbeit mit Fernzugriff auf BPi R1s zu ermöglichen. Bei Bedarf kann die GUI um spezielle Funktionen erweitert werden, indem neue Tabs programmiert und eingebunden werden. Im nachfolgenden Unterkapitel wird auf zwei spezielle Erweiterungen eingegangen (in Abbildung 8.8 sind der *Faban Companion* sowie der *Fault Injection* Tab zu erkennen). Zusätzlich werden erwartete sowie unerwartete Probleme im

Betrieb der *Remote Control Central* abgefangen und können im Tab *Exception* eingesehen werden.

## 8.3. Erweiterte Funktionen

Mit dem Ziel FABAN zu implementieren, zu testen und zu evaluieren, wurde das System um eine Reihe notwendiger Funktionen und Mechanismen erweitert, welche im Folgenden beschrieben werden.

### 8.3.1. Synchronisation lokaler Uhren

Eine gemeinsame Sicht auf die Zeit ist eine zentrale Voraussetzung für FABAN. Da es sich bei der hier behandelten Hardwareimplementierung um die Entwicklung eines Prototyps handelt und der Fokus selbst nicht auf der Synchronisation von Uhren liegt, war es weder eine Voraussetzung, die Uhrensynchronisation selbst fehlertolerant zu implementieren, noch war verlangt, die Genauigkeit über das benötigte Maß zu steigern. Eine steigende Genauigkeit der Uhrensynchronisation verringert im Allgemeinen die berechnete Zustellzeit von FABAN-Nachrichten, so dass diese schneller zugestellt werden können. Darüber hinaus sind keine Vorteile zu erwarten, so dass im Rahmen eines Prototyps eine quantifizierbare Genauigkeit ausreichend ist. Als Ziel wurde eine Genauigkeit im (niedrigen) $\mu s$-Bereich angestrebt, welche mit den vorhandenen Mitteln auch tatsächlich umsetzbar ist. Eine Erhöhung der Genauigkeit wäre mit externen, genaueren Uhren vorstellbar, hat aber wegen dem überschaubaren Nutzen keine weitere Relevanz. Nachfolgend eine kurze Beschreibung der beiden wichtigsten Verfahren, welche die höchste Eignung bieten.

#### Simple Clock Synchronisation

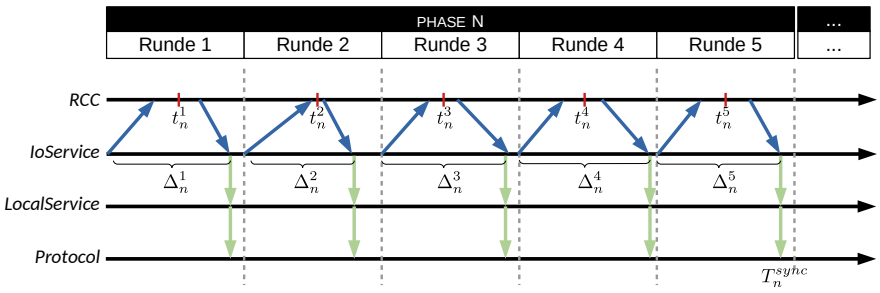



Abbildung 8.9.: Schema der Uhrensynchronisation gemäß der Eigenentwicklung *Simple Clock Synchronisation*

Als erster Ansatz wurde ein einfaches Uhrensynchronisationsprotokoll, genannt *Simple Clock Synchronisation*, entwickelt. Mit dieser Synchronisationsart stellt der *IoService* in periodischen Abständen von einer Sekunde Uhrensynchronisationsanfragen an das *Remote Control Central*, welches daraufhin die aktuelle Systemzeit des zentralen Kontrollrechners in Mikrosekunden zurücksendet (siehe Abbildung 8.9). Die Zeitintervalle zwischen einzelnen Anfragen werden als Runden bezeichnet, wobei 5 aufeinanderfolgende Runden eine Phase bilden. Der IoService berechnet am Ende jeder Phase aus den erhaltenen Zeiten $t_n^i,\ i \in \{1,\ldots,5\}$, sowie den Zeiten $\Delta_n^i,\ i \in \{1,\ldots,5\}$, zwischen Uhrensynchronisationsanfrage und Empfang der Zeit von der *Remote Control Central* einen Zeitwert $T_n^{sync}$, gemäß welchem die lokale Uhr des *IoServices*, als auch des *LocalServices* und jeglichen aktiven Protokollen angepasst wird. Hierbei wird folgende Berechnung durchgeführt:

$$T_n^{sync} := \frac{1}{3}\sum_{i=1}^{3}\left(t_n^{\sigma(i)} + \frac{\Delta_n^{\sigma(i)}}{2}\right), \tag{8.1}$$

wobei $\sigma : \{1,\ldots,n\} \to \{1,\ldots,n\}$ eine Permutation der Rundenindizes darstellt, welche die Runden aufsteigend der $\Delta_n^i$ sortiert, d.h.

$$\Delta_n^{\sigma(1)} \leq \Delta_n^{\sigma(2)} \leq \Delta_n^{\sigma(3)} \leq \Delta_n^{\sigma(4)} \leq \Delta_n^{\sigma(5)}. \tag{8.2}$$

Das Entfernen der zwei Zeitwerte, die die längste Übertragungsdauer benötigt haben, hat eine höhere Genauigkeit der Synchronisierung zur Folge, d.h. es werden „Ausreißer“ entfernt. Durch das Entfernen von Ausreißern wird die schlechte Genauigkeit der Zeitquelle auf dem Steuerrechner kompensiert, so dass eine Genauigkeit von weniger als $1\mu s$ erreicht werden kann. Neben dieser Ungenauigkeit, welche theoretisch verbessert werden kann, ist der größte Nachteil dieser Lösung, dass die Uhrensynchronisation zwingend eine aktive Verbindung zur *Remote Control Central benötigt.*

### Precision Time Protocol

Im Rahmen einer betreuten Studentenarbeit wurden weitere Uhrensynchronisationsmechanismen zur Verwendung auf den BPi R1s untersucht und evaluiert. Dabei stellte sich als wichtigste Erkenntnis heraus, dass das Precision Time Protocol (PTP) eine deutliche Verbesserung des vorherigen Ansatzes mit sich bringt [IEE08].

PTP ist entwickelt worden, um in lokalen Netzwerken eine hohe Genauigkeit der Uhrensynchronisation zu erzielen. Auf den BPi R1s lässt sich die quelloffene Implementierung des PTP-Standards für Linux *ptpd2* einrichten [Har15] und kann über die *Remote Control Central* gestartet werden. Tatsächlich konnte die Genauigkeit der Uhren im Vergleich zum vorherigen Ansatz leicht verbessert werden und auf maximal ca. $\pm 10\mu s$ reduziert werden (siehe Abbildung 8.10). Neben der besseren Genauigkeit wird zusätzlich keine aktive Verbindung zur *Remote Control Central* benötigt. Auf nähere Erklärungen der Funktionsweise des Protokolls wird an dieser Stelle bewusst verzichtet.

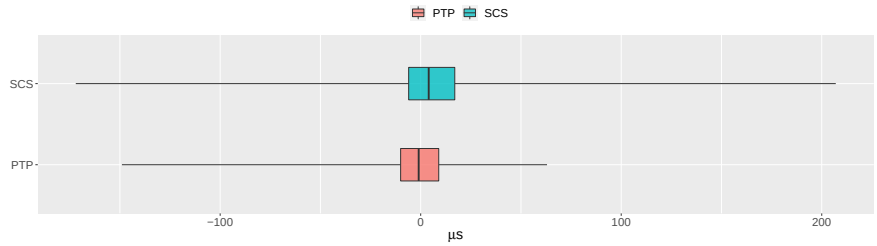



Abbildung 8.10.: Darstellung experimentell ermittelter lokaler Korrekturen der Uhren gemäß der eigen entwickelten *Simple Clock Synchronisation* sowie dem *Precision Time Protocol* in $\mu s$ in einem Beobachtungszeitraum von über 2 Stunden und über 1440 Synchronisationen in diesem Zeitintervall als Boxplot.

### 8.3.2. Topologieerkennung und FABAN-Konfiguration

Ein Ziel während der Implementierung war es, die BPi R1s beliebig miteinander verbinden zu dürfen, ohne manuelle Eingriffe in die Software oder die Konfiguration tätigen zu müssen. Für die Konfiguration von FABAN-Routings ist die Kenntnis der Netzwerktopologie notwendig, so dass aus diesem Grund ein Topologieerkennungsprotokoll, das *DiscoveryProtocol* implementiert wurde. Die komplette FABAN-Konfiguration wurde weiterhin vollständig automatisiert und in der *Remote Control Central* integriert. Abbildung 8.11 zeigt ein Bildschirmfoto der fabanspezifischen Funktionen und Anzeigen. Die Implementierung der Protokollmechanismen sind in hohem Grad redundant zu der Implementierung der Simulation in OMNeT++ und wird hier nicht weiter behandelt. Die vollständige Implementierung ist auf dem beigelegten Datenträger zu finden.

### 8.3.3. Fehlerinjektion

Im Rahmen einer weiteren studentischen Arbeit wurde das System um die Möglichkeit der Injektion von Fehlern erweitert. Dies wurde konzeptionell so umgesetzt, dass Fehler individuell in Protokolle und nicht global in die jeweiligen BPi R1s injiziert werden, damit mögliche nebenläufige Protokolle zum Debuggen oder zur Umsetzung diverser Zusatzfunktionen von Fehlern unbeeinträchtigt bleiben. Die Klassenstruktur der Protokollimplementierung aus Abbildung 8.7 wurde dabei um eine weitere Zwischenschicht von Klassen erweitert, welche Implementierungen der Fehlerinjektionsmechanismen, welche weiterzuleitende Nachrichten betreffen, enthalten und somit ebenfalls in die *GlobalBpiLib* ausgelagert sind (siehe Abbildung 8.12). Protokollimplementierungen müssen lediglich von den jeweiligen neuen Klassen mit Fehlerinjektion erben. Weitere Anpassungen sind nicht erforderlich.

Die *Remote Control Central* wurde um eine weitere UI erweitert, über welche Fehler individuell in Protokolle der BPi R1s injiziert werden können, die Nachrichten direkt

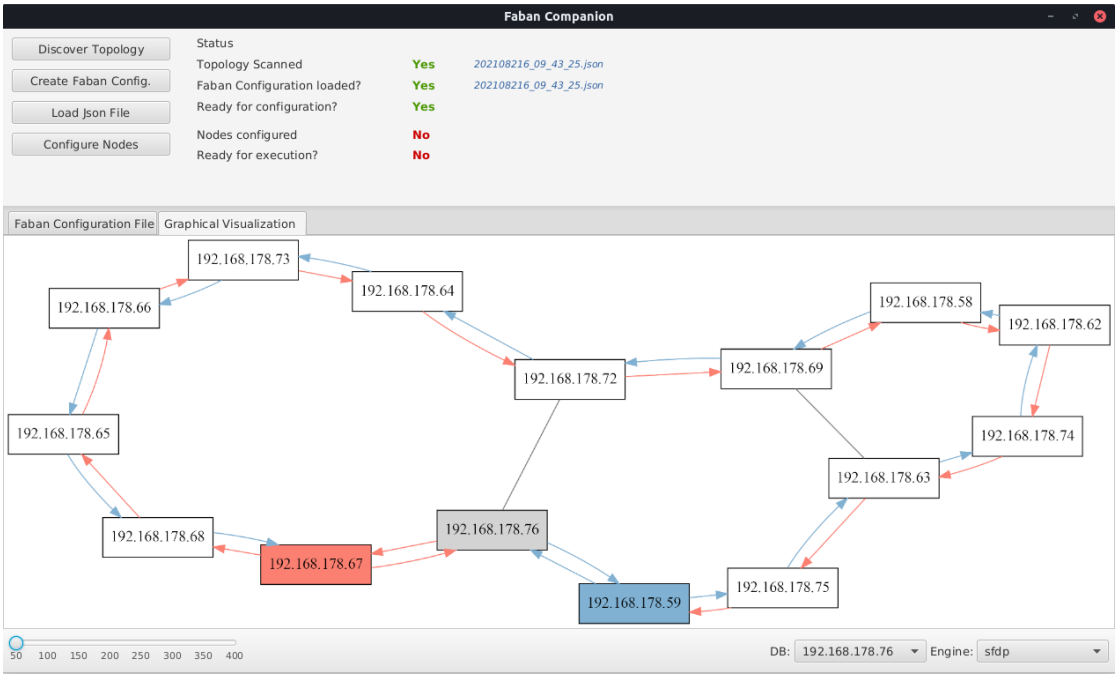



Abbildung 8.11.: Bildschirmfoto des FabanCompanion der *Remote Control Central*. Sequentiell können vom Benutzer die Topologieerkennung, die Erstellung der Faban Konfiguration für die erkannte Topologie sowie die Ausführung und Konfiguration des Faban-Protokolls auf den BPi R1s über die entsprechenden Buttons gestartet werden. Die UI enthält die Topologiebeschreibung sowie die FABAN-Routings in formaler sowie graphischer Darstellung.

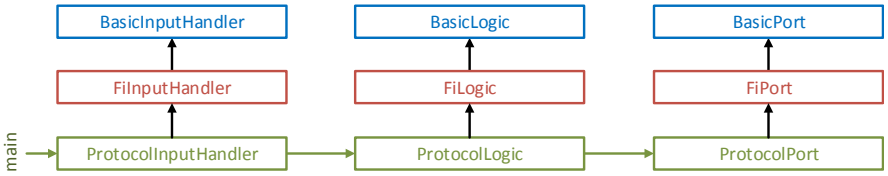



Abbildung 8.12.: Darstellung der Klassenstruktur der Protokollimplementierung mit Fehlerinjektionsmechanismen. Horizontale Pfeile stellen Abhängigkeiten zwischen einzelnen Klassen dar, während vertikale Pfeile Vererbungen symbolisieren.

betreffen. Fehlerkonfigurationen können dabei in externen Dateien gespeichert und wieder eingelesen werden. Es werden Mechanismen zur Injektion von Ausfall-, Unterlassungs-, Verzögerungs-, Bitverfälschungs- und Duplizierungsfehlern sowie fehlerhafter Weiterleitung bereitgestellt, welche wie folgt parametrisiert sind:

- Für alle Fehlerarten wird individuell mit $p_C, p_{Om}, p_{Del}, p_{BF}, p_{Dup}, p_{WF} \in \{0, 1\}$ die Fehlerwahrscheinlichkeit festgelegt. Gemäß dem jeweils eingestellten Wert wird für jede weiterzuleitende Nachricht überprüft, ob die Fehlerinjektion stattfindet.

- Für die fehlerhafte Weiterleitung kann zusätzlich die Menge der ausgehenden Ports individuell pro Bridge eingeschränkt werden, aus welchen im Fehlerfall der Port für die Weiterleitung gleichverteilt ausgewählt wird.

- Die Anzahl der Bitflips, die Anzahl der Duplikate sowie die Zeitdauer einer Verzögerung kann über Wahrscheinlichkeitsverteilungen eingestellt werden. Dabei wurden drei stetige Verteilungen (Gleichverteilung, Normalverteilung, Exponentialverteilung) implementiert. Die Normalverteilung sowie die Exponentialverteilung wurden durch untere und obere Schranken beschränkt und die Zufallszahlen bei der Anzahl der Bitflips bzw. der Duplikate in ganze Zahlen abgerundet.

Zusätzlich kann in Kombination mit jeder Fehlerart byzantinisches Verhalten aktiviert werden, so dass die Fehlerinjektion auf allen ausgehenden Ports unabhängig geschehen kann. Dies gilt für alle Fehlerarten. Abbildung 8.13 zeigt ein Bildschirmfoto der graphischen Benutzeroberfläche zur Fehlerkonfiguration.

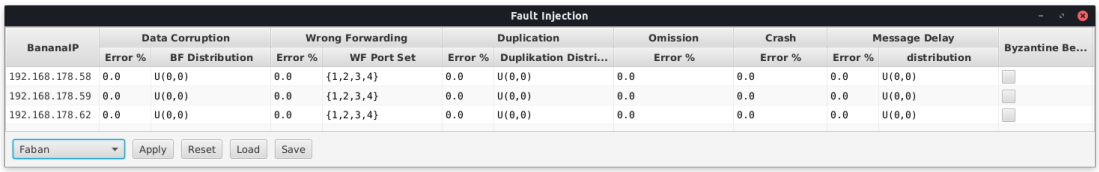



Abbildung 8.13.: Bildschirmfoto der Fehlerkonfiguration in der *Remote Control Central*

Darüber hinaus wurde das Verhalten einer Babbling Component von sowohl vollkommen willkürlichen Nachrichten als auch von spontanen, zusätzlichen FABAN-Nachrichten durch ein separates Programm mit der Bezeichnung *FaultyContentGenerator* realisiert. Der *FaultyContentGenerator* wird dabei über die *Remote Control Central* ausgeführt, konfiguriert und die eingestellte Nachrichtengenerierung gestartet, wobei folgende Konfigurationen möglich sind:

- Die zeitliche Verzögerung vor der nächsten Generierung einer Nachricht wird durch eine Normalverteilung realisiert, welche auf den positiven Bereich eingeschränkt wird. Die Parameter der Normalverteilung, also der Erwartungswert und die Varianz, können frei eingestellt werden.

- Mit $p_{FABAN} \in \{0, 1\}$ wird die Wahrscheinlichkeit festgelegt, mit der eine generierte Nachricht eine valide FABAN-Nachricht ist. Andernfalls wird eine komplett zufällige Nachricht generiert.

Die zufälligen, spontan generierten Nachrichten werden dabei auf allen ausgehenden Ports gesendet.

### 8.3.4. Statistiken

Die Erfassung von Statistiken kann beliebig detailliert implementiert werden. Im Rahmen dieser Prototyp-Entwicklung, wird das Sendeverhalten der Komponenten über das Terminal der *Remote Control Central* konfiguriert und muss explizit gestartet werden. Statistiken können jederzeit zwischen Sendevorgängen zurückgesetzt und/oder angezeigt werden und wurden während der Arbeit den aktuellen Notwendigkeiten angepasst. Eine detaillierte Statistikerfassung zum Beispiel in lokalen verteilten Datenbanken, einer zentralen Datenbank oder Logdateien war nicht erforderlich. Die gesammelten Statistiken wurden nicht zwischengespeichert, sondern über das *Remote Control Central* im laufenden Betrieb abgerufen.

## 8.4. Evaluation des Systems

Die grundlegenden Funktionen des implementierten Prototyp-Systems wurden getestet und funktionieren wie geplant. Reale Hardware bringt aber immer Limitierungen unterschiedlicher Arten mit sich, so dass es zwingend erforderlich ist, unterschiedliche Aspekte zu untersuchen, bevor Ergebnisse von Protokoll-Ausführungen legitim interpretiert werden dürfen.

### 8.4.1. Limitierungen der Sendeströme

Ein erster limitierender Parameter, der untersucht werden muss, ist die Übertragungsgeschwindigkeit zwischen zwei BPi R1s. Gemäß den technischen Spezifikationen ist die maximale Übertragungsrate limitiert auf $1 GBit/s$, als Verbindungskabel wurden CAT5e Kabel gewählt, so dass durch diese keine zusätzliche Beeinträchtigung entstehen dürfte. Die Routing-Logik findet auf der Software-Ebene statt, so dass Beeinträchtigungen durch den Prozessor, das Betriebssystem oder den Arbeitsspeicher entstehen können. Diese Einflüsse sind schwer quantifizierbar, so dass experimentell ermittelt wurde, ob die maximale Übertragungsrate von $100 MBit/s$ erreicht wird.

Der Empfang von Paketen wird auf der Software-Ebene auf den BPi R1s mit der Programmierschnittstelle *pcap* realisiert, welche unter anderem von prominenter Software wie Wireshark verwendet wird [The21]. Die Verarbeitung von Paketen kann unmittelbar bei Empfang eines Paketes erfolgen (*Immediate Mode*) oder empfangene Pakete können zwischengepuffert werden, um mögliche Paketverluste auf Kosten von Verzögerungen zu vermeiden. Der Unterschied beider *pcap*-Modi wurde experimentell ermittelt und ist in Abbildung 8.14 visualisiert.

Die Tests wurden dabei zwischen zwei BPi R1s mit minimalistischen Sende- und Empfangsroutinen durchgeführt, um Einflüsse anderer Elemente auszuschließen. Die Senderaten wurden dabei so gedrosselt, dass zwischen zwei Sendevorgängen Wartezeiten eingefügt wurden, um die gewünschten Senderaten zu erhalten. Die Schrittweite der Senderaten zwischen einzelnen Messungen betrug

- $1MBit/s$ bei Geschwindigkeiten von weniger als $10MBit/s$ und
- $10MBit/s$ bei Geschwindigkeiten zwischen $10MBit/s$ und $100MBit/s$.

Ebenfalls wurde bei den Messungen ein einseitiges Sendeverhalten sowie ein in beide Richtung stattfindendes Sendeverhalten verglichen. Es wurden bei jedem Test jeweils 1 Million Nachrichten maximaler Ethernet-Größe gesendet.

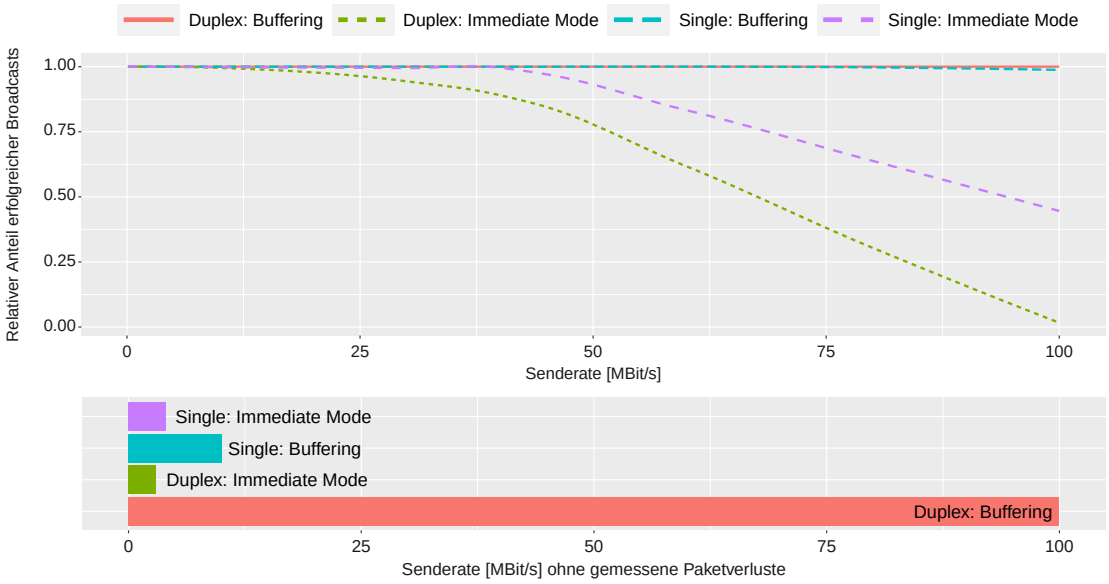



Abbildung 8.14.: Das obere Diagramm stellt die Anteile erfolgreicher Datenübertragungen in Abhängigkeit von den gedrosselten Senderaten dar. Dabei wurde sowohl einseitiges (Single) als auch beidseitiges (Duplex) Sendeverhalten, sowie ein- und ausgeschalteter *pcap* Immediate Mode miteinander verglichen (Immediate bzw. Buffering). Das untere Diagramm visualisiert die Bereiche von Senderaten, bei welchen in den jeweiligen Modi verlustfreies Senden möglich war.

Die Ergebnisse zeigen, dass bei aktiver Zwischenpufferung stets fast alle Pakete angekommen sind, während im *Immediate Mode* ein stark zunehmender Paketverlust mit steigenden Senderaten nicht zu übersehen ist. Eine vollständig zuverlässige Datenübertragung war jedoch erst bei Geschwindigkeiten unter $8MBit/s$ gewährleistet. Ein genauer Grund, warum bei einseitigem Senden zusätzlich Pakete verloren gehen, konnte nicht ermittelt werden. Die Vermutung ist aber, dass bei beidseitigem Senden so viel zusätzliche

Rechenleistung benötigt wird, dass als Nebeneffekt das Senden sowohl genügend verzögert wird als auch die Abstände zwischen Sendevorgängen unregelmäßiger werden, so dass Nachrichten unwahrscheinlicher verloren gehen. Da jedoch nicht ausgeschlossen werden kann, dass in Teilen des Netzwerks genau solches ungünstige Transferverhalten (temporär) entsteht, ist es sinnvoll, die eingehenden Datenraten der BPi R1s auf das Maximum von $8MBit/s$ festzusetzen.

## 8.4.2. Evaluation von Faban

Für die Evaluation von FABAN sind, wie bereits im Simulationskapitel gesehen, insbesondere Statistiken von Sender- und Empfängerknoten (Broadcast-Initiatoren und Broadcast-Empfänger) von Interesse. Dedizierte Sender- und Empfängerhardware wurde hierbei jedoch nicht benutzt, sondern die Logik von Sendern und Empfängern in die BPi R1s integriert, welche wiederum die Sende- und Empfangsstatistiken sammeln bzw. aktualisieren. Eine vollständige Automatisierung des Verarbeitungsprozesses von Statistiken war im Rahmen der prototypischen Entwicklung nicht notwendig und wurde nicht umgesetzt, so dass gesammelte Statistiken nach erfolgten Ausführungen abgelesen, zusammengeführt und interpretiert wurden. Folgende Sende- und Empfangsstatistiken wurden von jedem BPi R1 gesammelt:

- Initiierte FABAN-Rundsprüche
- Empfangene FABAN-Nachrichten
- Empfangene fehlerfreie FABAN-Nachrichten
- Empfangene FABAN-Duplikate
- Empfangene fehlerhafte FABAN-Nachrichten
- Kumulierte Nachrichtentransferdauer von FABAN-Nachrichten
- Maximale Nachrichtentransferdauer aller empfangenen FABAN-Nachrichten

Alle Messungen werden auf dem in Abbildung 8.11 dargestellten Netz von 15 BPi R1s durchgeführt. Es wird auch nur der Fall der Toleranz von $f = 1$ Fehlern betrachtet, da für eine sinnvolle Betrachtung höherer Fehlertoleranz nicht genügend Hardware zur Verfügung stand und dieser Fall detailliert durch Simulation in Kapitel 7 untersucht wurde.

### Verlustfreie Rundsprüche

Betrachtet man im Allgemeinen ein vollständiges Netz von $n \geq 1$ BPi R1s, so erzeugt jeder BPi R1 einen Zuwachs der Datenrate des Gesamtnetzes in Höhe seiner zweifachen

Senderate. Folglich ist es notwendig, in einem Netz von $n$ Bridges die Senderaten von FABAN-Rundsprüchen auf ein Maximum von

$$S_{Fab} \geq \frac{8}{2n} MBit/s \tag{8.3}$$

zu beschränken. Zusätzlich kann es vorkommen, dass die Datenrate nicht gleichmäßig über die Zeit verteilt ist, sondern durch die Duplikation von Nachrichten in zwei Wellen, sowie die unvorhersehbare Zusammenführung von Nachrichtenströmen unterschiedlicher Sender in den BPi R1s zur zeitweisen Erhöhung der lokalen Datenraten führt. Aus diesem Grund muss die Senderate von FABAN-Rundsprüchen geringfügig weiter reduziert werden. Konkret reicht in dem hier vorliegenden Fall mit 15 BPi R1s nach Formel 8.3 eine Senderate von $266kBit/s$ nicht aus und führt zu minimalen Paketverlusten im fehlerfreien Fall, funktioniert aber nach Reduktion auf $200kBit/s$ zuverlässig. Dies wurde mehrfach mit Sendetests mit jeweils 100.000 Rundsprüchen pro Sender verifiziert.

### Fehlertoleranzmechanismen

Die Realisierung der Mechanismen zur Toleranz von zu tolerierenden Fehlern, ist vollständig gemäß dem in Kapitel 7.2 beschriebenen Implementierungskonzept erfolgt. Mit Beachtung der zuvor spezifizierten Einschränkungen im Bezug auf die Senderaten, konnten die Beobachtungen aus den Simulationen für nicht-kritische Fehlerarten (vgl. Abbildung 7.17) reproduziert und somit die Toleranz dieser Fehler in der realen Anwendung bestätigt werden. Die als kritische Fehlerarten bezeichneten Fehler der fehlerhaften Weiterleitung, der Duplizierung sowie der Babbling Component (vgl. Abbildungen 7.22, 7.24) führten ebenfalls zu keinen neuen Problemen oder Erkenntnissen, sondern bestätigten die Beobachtungen aus der Simulation. Dabei spielt es bei allen Arten der Fehlerinjektion keine Rolle, ob BPi R1s Byzantinisches Verhalten aufweisen oder nicht. Des Weiteren waren die beobachteten Übertragungszeiten höher, je weniger Sender zeitgleich Rundsprüche initiiert haben. Analog zu den Beobachtungen in Abbildung 8.14, bei welchen Nachrichtenverluste auf einer Verbindung bei beidseitigem Senden geringer waren, wäre es auch hier damit zu erklären, dass durch geringeren Nachrichten-Gegenverkehr auf einzelnen Verbindungen, die geringe Netzwerkchip- bzw. Rechenleistung einen störenden Faktor darstellt, der zu übermäßigem Puffern und somit zu höheren Verzögerungen führt.

Im Allgemeinen konnten auch keine außergewöhnlichen Probleme im Bezug auf Verzögerungsfehler beobachtet werden. Wird eine Nachricht zu stark verzögert und überschreitet sie die zuvor berechnete Zustellzeit $t_d$, so wird die Nachricht beim Empfang verworfen. Die Berechnung der Zustellzeit ist allerdings nicht trivial und kann, wie im nachfolgenden Abschnitt im Detail betrachtet, nicht als zuverlässige Größe bestimmt werden.

### Instabile Übertragungszeiten

Die Berechnung der Zustellzeit basiert auf der Abschätzung einer maximalen Übertragungsdauer einer Nachricht zwischen zwei benachbarten BPi R1s (vgl. 7.2). Bei realer Hardware, insbesondere solcher ohne Echtzeiteigenschaften, können nicht vorhersehbare Ursachen Verzögerungen verursachen. Aus diesem Grund wurden experimentell Messungen von Übertragungszeiten ermittelt mit dem Ziel, zuverlässige pro Hop Latenzen zu bestimmen. Tabelle 8.1 gibt einen Überblick über die durchschnittlichen Übertragungszeiten von FABAN-Nachrichten. Dabei wurden jeweils 10.000 FABAN-Rundsprüche von jedem BPi R1 mit $200kBit/s$ bzw. $20kBit/s$ gesendet, wobei gleichzeitiges Senden in Gruppen von $n$ BPi R1s erfolgt und die nächste Gruppe mit dem Senden erst dann mit neuen Rundsprüchen beginnt, wenn die Rundsprüche der aktuellen Gruppe vollständig abgeschlossen sind.

| | 200kBit/s | | 20kBit/s | |
|---|---|---|---|---|
| n | $\max\Delta_t$ | $\text{avg}\Delta_t$ | $\max\Delta_t$ | $\text{avg}\Delta_t$ |
| 1 | $9240ms - 68540ms$ | $30\mu s - 4110\mu s$ | $142ms - 206ms$ | $5.74\mu s - 7.00\mu s$ |
| 3 | $221ms - 40893ms$ | $0.74\mu s - 136\mu s$ | $158ms - 215ms$ | $5.29\mu s - 7.19\mu s$ |
| 5 | $196ms - 267ms$ | $0.65\mu s - 0.89\mu s$ | $172ms - 241ms$ | $5.81\mu s - 8.04\mu s$ |
| 15 | $366ms - 451ms$ | $1.2\mu s - 1.5\mu s$ | $191ms - 256ms$ | $6.37\mu s - 8.55\mu s$ |

Tabelle 8.1.: Gemessene Übertragungszeiten von FABAN-Rundsprüchen auf einem Ringnetz mit insgesamt 15 BPi's bei Senderaten von $200kBit/s$ sowie $20kBit/s$. Bei jeder Messung sendet jeder BPi exakt 10000 Rundsprüche, wobei die Anzahl $n$ parallel sendender BPi's variiert wurde. Bei jedem BPi wurde bei jeder empfangenen Nachricht die maximale Übertragungszeit $\max\Delta_t$ sowie die kumulierte Übertragungszeit aller Nachrichten seit Beginn der jeweiligen Messung aktualisiert. Nach Abschluss aller Sendevorgänge aller $15/n$ Sendegruppen wurde jeweils aus den kumulierten Übertragungszeiten die durchschnittliche Übertragungszeit $\text{avg}\Delta_t$ ermittelt. Pro Messung liegen sowohl für $\max\Delta_t$ als auch für $\text{avg}\Delta_t$ 15 Werte vor, von welchen in dieser Tabelle der kleinste und der größte angegeben sind.

Die Ergebnisse zeigen in voller Deutlichkeit, dass keine stabilen Übertragungszeiten garantiert werden können. Bei der zuvor ermittelten, maximalen stabilen Übertragungsrate von $200kBit/s$ schwanken abhängig von der Anzahl parallel sendender BPi R1s die durchschnittlichen Übertragungszeiten stark im Bereich weniger Mikrosekunden bis zu mehreren Millisekunden. Eine maximale Übertragungsdauer eines Hops kann daraus allerdings nicht ermittelt werden, weil bei den tatsächlichen Übertragungszeiten extreme Ausreißer in Höhe von bis zu 68 Sekunden gemessen wurden. Insbesondere da die Diskrepanz zwischen maximalen Übertragungszeiten mit variierendem $n$ groß

ist, kann nicht ausgeschlossen werden, dass trotz fehlerfreiem Betrieb die tatsächlichen Übertragungszeiten in einzelnen Fällen noch höher ausfallen.

Nach Reduktion der Senderaten um den Faktor 1/10 konnten die Übertragungsraten stabilisiert werden. Sowohl die durchschnittlichen Übertragungszeiten sowie die maximal gemessenen Übertragungszeiten lagen bei allen Messungen jeweils nah beieinander, unabhängig wie viele Sender zeitgleich Last auf dem Netzwerk erzeugten.

Nichtsdestotrotz ist die starke Reduktion der Senderaten in den unteren $kBit/s$ Bereich eine eklatante Einschränkung, welche realitätsnahe Experimente unmöglich macht. Zeitkritische Anwendungen können damit, selbst auf prototypischer Entwicklungsebene nicht im Bezug auf zeitliche Untersuchungen evaluiert werden. In diesem Zusammenhang verspricht beispielsweise TSN besseres Echtzeit-Verhalten bei Ethernet, welches in zukünftigen Arbeiten bei realen Implementierungen Verbesserungen bringen dürfte [SS16].

### 8.4.3. Reflektion qualitativen Nutzens

Im Hinblick auf Performanz, aber auch im Bezug auf negative Einflüsse durch die Hardware, die physikalischen Verbindungen sowie das Betriebssystem inklusive nebenläufiger Softwaredienste, bringt die präsentierte prototypische Entwicklung auf realer Hardware deutliche Limitierungen mit sich. Die Senderaten mussten stark limitiert werden, um eine möglichst stabile Kommunikation zu ermöglichen. Des Weiteren konnten die wichtigsten Fragen im Bezug auf die Funktionalität von FABAN sowie den Fehlertoleranzeigenschaften mit Hilfe von speziell darauf ausgelegten Parametrisierungen von Simulationen bereits zufriedenstellend beantwortet werden, so dass Fehlerinjektionen auf realer Hardware aufgrund der hier notwendigen starken Limitierungen keinen nennenswerten Nutzen im Hinblick auf Fehlertoleranzeigenschaften - mit der Ausnahme der zusätzlichen Bestätigung der Ergebnisse - bringen.

Auf der anderen Seite spiegeln Simulationen nur bedingt reale Anwendungsgebiete wieder. Triviale Probleme, wie beispielsweise eine grundlegend stabile Kommunikation zwischen zwei Komponenten zu erhalten, müssen in einer Simulation nicht behandelt werden. Aus diesem Grund ist der Beitrag, den eine Implementierung auf realer Hardware liefert, von großem zusätzlichen Nutzen. Sie zeigt nämlich erstens, dass eine Anwendung in realer Umgebung möglich ist und kann zweitens genutzt werden, um Grenzen der stabilen Anwendung des Protokolls zu evaluieren. Weiterhin kann eine reale Implementierung in den unterschiedlichsten Umgebungen, deren Einflüsse von einer Simulation nur sehr schlecht abgebildet werden können, ausgeführt werden, um Auswirkungen zusätzlicher Fehlerquellen testen zu können. Zusätzliche Fehlerquellen können dabei unterschiedliche Temperaturen und Witterungsbedingungen, variierende Atmosphären oder radioaktiv

belastete Gebiete sein, um dabei nur einige wichtige Beispiele zu nennen. Sowohl bei dieser Prototyp-Implementierung als auch bei zukünftigen Implementierungen auf geeigneterer Hardware bietet es sich an, die genannten Aspekte in zukünftigen Arbeiten zu untersuchen.

## 8.5. Zusammenfassung

Die Protokollentwicklung von FABAN sowie der Erweiterung ExFABAN wurde in dieser Arbeit in den Kapiteln 5 und 6 zunächst sehr gründlich aus theoretischer Perspektive betrachtet, bevor beide Protokollvarianten in einer anschließenden Simulation in vollem Umfang realisiert und auf diversitärem Wege verifiziert wurden. In diesem abschließenden Kapitel wurde eine prototypische Entwicklung eines realen Systems beschrieben, dessen Ziel es war zu überprüfen, inwieweit die Applikation des Entwicklungskonzepts der OMNeT++-Simulation auf möglichst handelsübliche Hardware möglich ist und mit welchen möglichen Problemen dies einhergeht.

Die Wahl der Hardware ist auf Einzelplatinencomputer des Typs BananaPi R1 gefallen, bei welchen es sich um Raspberry Pi ähnliche, kostengünstige Einzelplatinencomputer handelt, welche zusätzlich einen integrierten Gigabit Ethernet Switch enthalten. Mögliche Erschwernisse durch fehlende Echtzeit-Eigenschaften der Hardwarekomponenten sowie des Betriebssystems Linux wurden bewusst in Kauf genommen und erwartet. Garantierte pro Hop Latenzen sind durch das einfache System nicht gegeben. Durch eingeschränktes Sendeverhalten sollten, wie bereits zuvor in der OMNeT++ Simulation thematisiert, Latenzen realisiert werden, die mit hoher Wahrscheinlichkeit einen vorgegebenen Wert unterschreiten.

Während die Umsetzung des Implementierungskonzepts der OMNeT++ Simulation nahezu vollständig übernommen werden konnte und (auch in der Präsenz von Fehlern) zu erwartbaren und positiven Ergebnissen führte, konnte das Problem zu sehr schwankender Übertragungszeiten nicht gelöst werden. In allen durchgeführten Experimenten sind stets wenige Nachrichten derart stark verzögert worden, dass damit keine zufriedenstellende Ableitung einer maximalen pro Hop Latenz ermittelt werden konnte.

Dennoch konnte mit der Implementierung gezeigt werden, dass die Umsetzung der Protokolllogik im Allgemeinen unproblematisch ist und auf einem System, welches die Grundvoraussetzungen für FABAN und ExFABAN erfüllt, definitiv möglich ist. Des Weiteren wurde mit der modularen Implementierung ein Basissystem geschaffen, welches auch für die prototypische Implementierung weiterer Protokolle wiederverwendet werden kann.

Kapitel 9

# Zusammenfassung und Ausblick

## 9.1. Zusammenfassung

In dieser Dissertation wurde die Arbeit an dem neuen Protokoll *Fault-Tolerant Atomic Broadcast and Agreement in Bridge-Connected Networks*, welches das Übereinstimmungsproblem durch unteilbare Rundsprüche für sicherheitskritische verteilte Systeme mit (zum Teil) harten Echtzeit-Anforderungen auf Bridge-verbundenen Netzwerken effizient löst, fortgeführt. Es ist gelungen, die informelle Idee des Protokolls, insbesondere die Anforderungen und Regeln des redundanten Routings für die Fehlerannahme $f = 1$ zu formalisieren und mathematisch vollständig zu beweisen. Eine algorithmische Lösung zur Generierung von Routings wurde entwickelt, welche feststellt, ob ein Routing für eine gegebene Netzwerktopologie erstellt werden kann und, falls möglich, effizient ein Routing mit sehr guten bis optimalen Eigenschaften bezüglich der Länge der redundanten Pfade ermittelt. Der Algorithmus wurde sowohl formal bewiesen als auch einer analytischen sowie experimentellen Untersuchung unterzogen, um die Effizienz und Korrektheit der Methode zu validieren.

Bei Betrachtung der allgemeinen Fehlerannahme $f \geq 1$ konnten konzeptionelle Schwächen, welche Einfluss auf das Ausmaß der Fehlertoleranz haben, festgestellt werden und in Folge dessen eine, vollständig zu FABAN im Fall $f = 1$ kompatible, Protokollerweiterung mit der Bezeichnung *Extended Fault Tolerant Atomic Broadcast and Agreement in Bridge Connected Networks for Multiple Faults (ExFABAN)* entwickelt werden. Dabei wurde das Routing in die zwei Phasen primäres sowie sekundäres ExFABAN-Routing separiert. Während für das primäre Routing es gelungen ist, sowohl hinreichende als auch notwendige Bedingungen zu formulieren und mathematisch zu beweisen, konnten für das sekundäre Routing hinreichende Bedingungen formuliert und bewiesen werden. Abschließend wurden

Klassen von Netzwerktopologien beschrieben, welche mit der Fehlerannahme $f$ skalierbar und für die Anwendung von ExFABAN geeignet sind.

Abschließend wurden die theoretischen Ergebnisse von ExFABAN durch ausführliche Simulationen in dem Simulationsframework OMNeT++ überprüft, die Implementierung betreffende Verbesserungen ausgearbeitet, und die theoretischen Ergebnisse im Bezug auf den Grad der Fehlertoleranz vollständig validiert. Zusätzlich wurden die gewonnenen Erkenntnisse verwendet, um eine Demonstration auf realer Hardware umzusetzen.

Zusammenfassend wurde ein Protokoll zur Erzielung von Übereinstimmung durch unteilbare Rundsprüche für die Toleranz von beliebig vielen Fehlern $f \geq 1$ vollständig beschrieben, bewiesen und untersucht. Im Vergleich zu Flaviu Cristians Protokoll [Cri+95] wird der Rundspruch nicht über Flutung realisiert, sondern über gezielte Wellen, welche sowohl hinreichendes als auch notwendiges Routing - wobei von letzterem bei Bedarf abgewichen werden darf - darstellen und somit deutliche Effizienzgewinne implizieren. Die Vorteile des entwickelten Protokolls liegen in den kostengünstigen Hardwarevoraussetzungen - Ethernet-Verbindungen und einfache (nur minimal veränderte) Bridges - und der Flexibilität der kompatiblen Netzwerktopologien. Insbesondere ist FABAN/ExFABAN in Hinblick auf zukünftige Automatisierungsszenarien auch für Ethernet-Varianten wie TSN geeignet. Des Weiteren lässt sich für einen angestrebten Fehlertoleranzgrad und den Anwendungsstandort zielgerichtet eine ExFABAN kompatible und effiziente Netzwerktopologie konstruieren.

## 9.2. Offene Fragen und Ausblick

Für die Fehlerannahme $f = 1$ konnte ein effizienter Algorithmus entwickelt werden, welcher für eine gegebene Netzwerktopologie ein gültiges FABAN-Routing ermittelt, falls es existiert. Im Einfehlerfall besteht ein FABAN-Routing aus zwei redundanten Pfaden zwischen jedem Sender und jedem Empfänger, im allgemeinen Fall $f \geq 1$ besteht ein ExFABAN aus mindestens $f + 1$ redundanten Pfaden zwischen jedem Sender und jedem Empfänger. Mit zunehmendem $f$ sinkt der Anteil aller ExFABAN-kompatibler Topologien aufgrund der Grundvoraussetzung der Existenz von $f + 1$ redundanten Pfaden zwischen allen Sender-/Empfängerpaaren deutlich, so dass in dieser Arbeit die Suche nach einem Algorithmus zur Generierung von ExFABAN-Routings nicht im Vordergrund stand, sondern die systematische Konstruktion von kompatiblen Topologien. Nichtsdestotrotz bleibt es eine interessante Frage, ob ein effizienter Algorithmus, das heißt ein Algorithmus der effizienter arbeitet als ein Brute-Force-Ansatz, existiert und eine Herausforderung, diesen dann zu entwickeln und zu beweisen.

Der zweite offene Punkt der Protokollentwicklung betrifft die Formulierung des sekundären Routings von ExFABAN. Es wurde lediglich gezeigt, dass es sich bei den hier

formulierten Bedingungen um hinreichende Bedingungen handelt. Es ist auch implizit bewiesen worden, dass es sich nicht um eine notwendige Bedingung handelt, da diese gezielt in der Klasse der mehrschichtigen Ringe, verletzt wurden. Es ist nicht gelungen, eine bessere Variante der Bedingungen zu formulieren und zu beweisen. Obwohl die Toleranz einer hohen Anzahl von Fehlern eine eher geringe Praxisrelevanz hat und die in dieser Arbeit präsentierten Topologien für Einfach-, Doppel- und Dreifachfehler eine sehr gute Lösung darstellen, bleibt es eine durchaus interessante Frage, ob die Bedingungen des sekundären Routings in einer besseren Variante formuliert und bewiesen werden können.

Während statische Systeme, wie in dieser Arbeit thematisiert, insbesondere in (Echtzeit-)Automatisierungssystemen relevant sind, ist die Frage, inwieweit man ExFABAN für zur Laufzeit dynamisch veränderbare Systeme einsetzen könnte, ein dritter Ansatz für zukünftige Forschung. Die vorgestellten Graphenklassen der mehrschichtigen Ringnetze, der vollvermaschten Netze sowie der redundanten Sterntopologien könnten hilfreiche Eigenschaften besitzen, welche dazu beitragen, dass ein lokales Routing ohne vorkonfigurierte Routing-Informationen denkbar wäre. Dadurch könnten Systeme zur Laufzeit in einem gewissen Rahmen verändert werden, ohne die Funktionalität einzuschränken.

Sowohl in der Simulation als auch in der Hardwareimplementierung wurden Nachrichten in Bridges stets empfangen, bevor die Weiterleitung startete. Ethernet unterstützt beispielsweise die Weiterleitung von Segmenten von Nachrichten, bevor der vollständige Inhalt übertragen wurde. Denkbar sind auch Realisierungen bei ExFABAN, insbesondere hinsichtlich der erforderlichen Duplikaterkennung, welche diesen Übertragungsmodus unterstützen würden. Die Machbarkeit dieser Idee müsste zukünftig untersucht werden.

FABAN sowie ExFABAN wurden im Rahmen der Simulation unter anderem im Hinblick auf Grenzen hoher Senderaten untersucht. Dabei wurden die Protokolle losgelöst von zusätzlichem Nachrichtenverkehr durch andere Anwendungen, welche auf dem selben Netzwerk kommunizieren, betrachtet. Des Weiteren wurde das Sendeverhalten von FABAN bzw. ExFABAN Rundsprüchen lediglich in Form von variierenden Sendeströmen modelliert und untersucht. Zusätzlich dazu wäre eine Performanz-Bewertung von anwendungsabhängigem Sendeverhalten durch die Modellierung von Lastprofilen realer Automatisierungssysteme von großem Interesse und eine sinnvolle Anschlussarbeit.

Zuletzt bliebe für eine zukünftige Arbeit die Implementierung des Protokolls auf echter, performanter Hardware. Eine Simulation ist kein adäquater Ersatz für eine Ausführung auf echter Hardware und auch die im Rahmen dieser Arbeit durchgeführte Implementierung auf BananaPi R1 Mikrocontrollern, auch wenn diese ebenfalls wichtige Erkenntnisse oder Bestätigungen liefert, hat ihre Nachteile (Linux als Betriebssystem, keine Echtzeiteigenschaften und geringe Übertragungsraten) und ist folglich nicht mit einer Implementierung auf geeigneterer Hardware gleichzusetzen. Es wäre vorstellbar,

FPGAs oder handelsübliche Router mit manipulierter Firmware zu verwenden.

Abschließend ist anzumerken, dass auch ohne diese angeführten zusätzlichen Arbeiten gezeigt wurde, dass sowohl FABAN als auch ExFABAN durchaus die Tauglichkeit besitzen, in Echtzeitsystemen für sicherheitskritische Automatisierungsaufgaben einen fehlertoleranten Rundspruch in effizienter Weise zu realisieren.

Anhang A

# Digitaler Anhang

Der Anhang dieser Arbeit liegt vollständig in digitaler Form auf der beigelegten CD vor. Der Inhalt ist nachfolgend skizziert:

- CD/
  - 1 Weitere Simulationsergebnisse/
    Simulationsergebnisse aller in Kapitel 7 beschriebenen Netzwerktopologien und Fehlerkonfigurationen. Die Ergebnisse liegen in Paaren von *.pdf sowie *.txt Dateien vor, wobei letztere jeweils statistische Informationen zum Konfidenzintervall enthält.
  - 2 Java/
    - Faban Topology Generator/
      Programm zur Generierung und zum Vergleichen von systematischen und randomisierten Netzwerktopologien.
    - Faban Wave Tool/
      Implementierung des Algorithmus zur Geneirung von Wellen.
    - RemoteControlCentral/
      Implementierung der Remote Control Central in Java 8/JavaFX.
  - 3 Omnet/
    - FabanSimulator/
      Faban/ExFaban Simulation in OMNeT++. Der Ordner 'simulations' enthält alle für Simulationen verwendete Netzwerktopologien sowie zugehörige Simulationskonfigurationen.
  - 4 Banana Pi R1 Protocol Emulation/
    Enthält die vollständigen C++ Codes aller Implementierungen auf den BPi R1 Mikrocontrollern.

# Literaturverzeichnis


[AH90] James Aspnes und Maurice Herlihy. „Fast Randomized Consensus Using Shared Memory“. In: *Journal of Algorithms* 11.3 (Sep. 1990), S. 441–461. ISSN: 0196-6774. DOI: 10.1016/0196-6774(90)90021-6.

[Ard21] Arduino. *Official Arduino Homepage*. 2021. URL: https://www.arduino.cc/ (besucht am 07. 07. 2021).

[Avi76] Aviziens. „Fault-Tolerant Systems“. In: *IEEE Transactions on Computers* C-25.12 (Dez. 1976), S. 1304–1312. ISSN: 0018-9340. DOI: 10.1109/TC.1976.1674598.

[Bar97] Michael Barabanov. „A Linux-Based Real-Time Operating System“. In: (1997). URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.9360.

[BD85] Oezalp Babaoglu und Rogério Drummond. „Streets of Byzantium: Network Architectures for Fast Reliable Broadcasts“. In: *IEEE Transactions on Software Engineering* 6 (1985), S. 546–554. ISSN: 0098-5589. DOI: 10.1109/TSE.1985.232247.

[Ben83] Michael Ben-Or. „Another Advantage of Free Choice (Extended Abstract): Completely Asynchronous Agreement Protocols“. In: *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*. PODC '83. New York, NY, USA: Association for Computing Machinery, Aug. 1983, S. 27–30. ISBN: 978-0-89791-110-8. DOI: 10.1145/800221.806707.

[BJ87] Kenneth P. Birman und Thomas a. Joseph. „Reliable Communication in the Presence of Failures“. In: *ACM Transactions on Computer Systems* 5.1 (1987), S. 47–76. ISSN: 0734-2071. DOI: 10.1145/7351.7478.

[Bou15a] Omar Bousbiba. „ESSEN - An Efficient Single Round Signature Protected Message Exchange Agreement Protocol for Wireless Distributed Networks“. In: *ARCS 2015 - The 28th International Conference on Architecture of Computing Systems. Proceedings*. März 2015, S. 1–8. ISBN: 978-3-8007-3657-7.

[Bou15b] Omar Bousbiba. „Reducing the Communication Complexity of Agreement Protocols By Applying A New Signature Scheme Called SIGSEAM“. In: *DEPEND 2015: The Eighth International Conference on Dependability*. Venice, Italy, 2015, S. 29–32. ISBN: 978-1-61208-429-9.

[Bou16] Omar Bousbiba. „Ein Effizientes Byzantinisches fehlertolerantes Übereinstimmungsprotokoll für verteilte drahtlose Echtzeitsysteme“. Diss. Essen: universität Duisburg-Essen, 2016. URL: https://duepublico2.uni-due.de/receive/duepublico_mods_00043533.

[Bra87] Gabriel Bracha. „Asynchronous Byzantine Agreement Protocols“. In: *Information and Computation* 75.2 (Nov. 1987), S. 130–143. ISSN: 0890-5401. DOI: 10.1016/0890-5401(87)90054-X.

[BS85] J.W. Bandler und A.E. Salama. „Fault Diagnosis of Analog Circuits“. In: *Proceedings of the IEEE* 73.8 (1985), S. 1279–1325. ISSN: 0018-9219. DOI: 10.1109/PROC.1985.13281.

[CC08] Shi-Fan Chang und Yung-Fu Chen. „Redundant Array Of Independent Disks System“. US 2008/0052459 A1. 2008.

[CC83] Vinton G. Cerf und Edward Cain. „The DoD Internet Architecture Model“. In: *Computer Networks (1976)* 7.5 (1983), S. 307–318. ISSN: 0376-5075. DOI: 10.1016/0376-5075(83)90042-9.

[CD89] Benny Chor und Cynthia Dwork. „Randomization in Byzantine Agreement“. In: *Advances in Computing Research* 5 (1989), S. 443–497.

[CM84] Jo-Mei Chang und N. F. Maxemchuk. „Reliable Broadcast Protocols“. In: *ACM Transactions on Computer Systems* 2.3 (1984). ISSN: 0734-2071. DOI: 10.1145/989.357400.

[Coo85] Eric C. Cooper. „Replicated Distributed Programs“. In: *The Tenth ACM Symposium*. 1985, S. 63–78. ISBN: 0-89791-174-1. DOI: 10.1145/323647.323635.

[Cor+09] Thomas H. Cormen u. a., Hrsg. *Introduction to Algorithms*. Third Edition. Cambridge, Massachusetts: MIT Press, 2009. ISBN: 978-0-262-03384-8.

[Cor12] Intel Corporation. *Specification: Intel® Xeon® Prozessor E5-2650*. 2012. URL: https://ark.intel.com/content/www/de/de/ark/products/64590/intel-xeon-processor-e5-2650-20m-cache-2-00-ghz-8-00-gt-s-intel-qpi.html (besucht am 14. 04. 2020).

[Cor20] Core4INET. *Core4INET - Real-Time Ethernet Protocols for INET*. 2020. URL: https://omnetpp.org/download-items/Core4INET.html (besucht am 22. 07. 2020).

[Cou+12] George F. Coulouris u. a. *Distributed Systems: Concepts and Design*. Fifth Edition. Boston, Massachusetts: Addison-Wesley, 2012. ISBN: 978-0-13-214301-1.

[CP13] Narendra D. Chauhan und Nilesh H. Pancholi. „Guidelines to Understanding to Estimate MTBF“. In: *IJSRD - International Journal for Scientific Research & Development* 1.3 (2013), S. 493–495. ISSN: 2321-0613.

[Cri+95] F. Cristian u. a. „Atomic Broadcast: From Simple Message Diffusion To Byzantine Agreement“. In: *Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years'*. 1995, S. 158–179. ISBN: 0-8186-7150-5. DOI: 10.1109/FTCSH.1995.532668.

[Cri90] Flaviu Cristian. „Synchronous Atomic Broadcast for Redundant Broadcast Channels“. In: *Real-Time Systems* 2.3 (Sep. 1990), S. 195–212. ISSN: 0922-6443. DOI: 10.1007/BF00365327.

[Cri91] Flaviu Cristian. „Understanding Fault-Tolerant Distributed Systems“. In: *Communications of the ACM* 34.2 (1991), S. 56–78. ISSN: 0001-0782. DOI: 10.1145/102792.102801.

[CS02] Xuejun Chen und Martin Simons. „A Component Framework for Dynamic Reconfiguration of Distributed Systems“. In: *International Working Conference on Component Deployment*. Springer. 2002, S. 82–96. ISBN: 978-3-540-45440-3.

[CT91] Tushar Deepak Chandra und Sam Toueg. „Unreliable Failure Detectors for Reliable Distributed Systems“. In: *Journal of the ACM* 43.2 (1991), S. 225–267. ISSN: 0004-5411. DOI: 10.1145/226643.226647.

[CZ85] David R. Cheriton und Willy Zwaenepoel. „Distributed Process Groups in the V Kernel“. In: *ACM Transactions on Computer Systems* 3.2 (Mai 1985), S. 77–107. ISSN: 0734-2071. DOI: 10.1145/214438.214439.

[Dav+07] Abhijit Davare u. a. „Period Optimization for Hard Real-Time Distributed Automotive Systems“. In: *2007 44th ACM/IEEE Design Automation Conference*. 2007, S. 278–283. ISBN: 978-1-59593-627-1. DOI: 10.1145/1278480.1278553.

[DHS86] Danny Dolev, Joseph Y. Halpern und H. Raymond Strong. „On the Possibility and Impossibility of Achieving Clock Synchronization“. In: *Journal of Computer and System Sciences* 32.2 (Apr. 1986), S. 230–250. ISSN: 0022-0000. DOI: 10.1016/0022-0000(86)90028-0.

[Di +88] Felicita Di Giandomenico u. a. „Gracefully Degradable Algorithm for Byzantine Agreement.“ In: *Comput. Syst. Sci. Eng.* 3.1 (1988), S. 32–40. ISSN: 0267-6192. DOI: 10.5555/45898.45902.

[Die06] Reinhard Diestel. *Graphentheorie*. Fifth. Springer-Verlag Heidelberg, 2006. ISBN: 3-540-21391-0.

[Dij68] Edsger W. Dijkstra. „The Structure of the “THE” Multiprogramming System“. In: *Communications of the ACM* 11.5 (1968). Hrsg. von Per Brinch Hansen, S. 341–346. ISSN: 0001-0782. DOI: 10.1145/363095.363143.

[DLS88] Cynthia Dwork, Nancy Lynch und Larry Stockmeyer. „Consensus in the Presence of Partial Synchrony“. In: *Journal of the ACM* 35.2 (1988), S. 288–323. ISSN: 0004-5411. DOI: 10.1145/42282.42283.

[DRS90] Danny Dolev, Ruediger Reischuk und H. Raymond Strong. „Early Stopping in Byzantine Agreement“. In: *Journal of the ACM* 37.4 (Okt. 1990), S. 720–741. ISSN: 0004-5411. DOI: 10.1145/96559.96565.

[DS83] D. Dolev und H. R. Strong. „Authenticated Algorithms for Byzantine Agreement“. In: *SIAM Journal on Computing* 12.4 (1983). ISSN: 0097-5397. DOI: 10.1137/0212045.

[Ech18] Klaus Echtle. „Detection of Wrongly Directed Messages by Efficient Fault-Tolerance Signatures“. In: *ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems*. Braunschweig: VDE VERLAG GMBH, Berlin, Offenbach, Apr. 2018, S. 11–20. ISBN: 978-3-8007-4559-3.

[Ech89] K. Echtle. „Distance Agreement Protocols“. In: *The Nineteenth International Symposium on Fault-Tolerant Computing. Digest of Papers*. Chicago, IL, USA, 1989. ISBN: 0-8186-1959-7. DOI: 10.1109/FTCS.1989.105565.

[Ech90] Klaus Echtle. *Fehlertoleranzverfahren*. Springer-Verlag Berlin Heidelberg, 1990. ISBN: 978-3-642-75765-5.

[Ech99] Klaus Echtle. „Avoiding Malicious Byzantine Faults by a New Signature Generation Technique“. In: *European Dependable Computing Conference*. Springer. 1999, S. 106–123. ISBN: 978-3-540-66483-3. DOI: 10.1007/3-540-48254-7_9.

[EF16] Klaus Echtle und Valentin Fitz. „FABAN - Fault-Tolerant Atomic Broadcast and Agreement in Bridge-Connected Networks“. In: *ARCS 2016; 29th International Conference on Architecture of Computing Systems*. Nürnberg: VDE VERLAG GMBH, Berlin, Offenbach, 2016, S. 1–10. ISBN: 978-3-8007-4157-1.

[EM17] Klaus Echtle und Zoha Moztarzadeh. „Fault-Tolerant Clock Synchronization in Ring-Networks“. In: *Proceedings of the Symposium on Applied Computing*. Marrakech Morocco: ACM, Apr. 2017, S. 465–468. ISBN: 978-1-4503-4486-9. DOI: 10.1145/3019612.3019864.

[EM96] K. Echtle und A. Masum. „A Multiple Bus Broadcast Protocol Resilient to Non-Cooperative Byzantine Faults“. In: *Proceedings of Annual Symposium on Fault Tolerant Computing*. Juni 1996, S. 158–167. ISBN: 0-8186-7262-5. DOI: 10.1109/FTCS.1996.534603.

[Eth21] EthernetFMC. *Ethernet FMC | Gigabit Ethernet FPGA Mezzanine Card*. 2021. URL: https://ethernetfmc.com/ (besucht am 07. 07. 2021).

[FE18] Valentin Fitz und Klaus Echtle. „General Extension of FABAN and a Topology-Based Approach to Tolerate Any Number of Faults“. In: *ARCS 2018; 31th International Conference on Architecture of Computing Systems*. Braunschweig: VDE VERLAG GMBH, Berlin, Offenbach, 2018, S. 3–10. ISBN: 978-3-8007-4559-3.

[FF97] Sally Floyd und Kevin Fall. *Router Mechanisms to Support End-to-End Congestion Control*. Techn. Ber. 1997.

[FiC20] FiCo4OMNeT. *FiCo4OMNeT - Fieldbus Communication (CAN, FlexRay)*. 2020. URL: https://omnetpp.org/download-items/FiCo4OMNeT.html (besucht am 22. 07. 2020).

[Fis83] Michael J. Fischer. „The Consensus Problem in Unreliable Distributed Systems (a Brief Survey)“. In: *Foundations of Computation Theory*. Hrsg. von Marek Karpinski. Bd. 158. Lecture Notes in Computer Science. Berlin [u.a.]: Springer, 1983, S. 127–140. ISBN: 978-3-540-12689-8. DOI: 10.1007/3-540-12689-9âĂŮ.

[FLP85] Michael J. Fischer, Nancy A. Lynch und Michael S. Paterson. „Impossibility of Distributed Consensus with One Faulty Process“. In: *Journal of the ACM* 32.2 (1985), S. 374–382. ISSN: 0004-5411. DOI: 10.1145/3149.214121.

[Fou20] The R Foundation. *R: The R Project for Statistical Computing*. 2020. URL: https://www.r-project.org/ (besucht am 22. 07. 2020).

[FP02] „Probabilistic Atomic Broadcast“. In: (2002). Hrsg. von Pascal Felber und Fernando Pedone. ISSN: 1060-9857. DOI: 10.1109/RELDIS.2002.1180186,.

[Fuc+14] Christian M Fuchs u. a. „A Fault-Tolerant Radiation-Robust Mass Storage Concept for Highly Scaled Flash Memory“. In: (2014). URL: https://ui.adsabs.harvard.edu/abs/2015ESASP.732E..13F.

[Gho14] Sukumar Ghosh. *Distributed Systems: An Algorithmic Approach*. Second Edition. Chapman & Hall/CRC Computer and Information Science Series. Boca Raton [u.a.]: Chapman & Hall/CRC, 2014. ISBN: 978-1-4987-6005-8.

[GM98] Juan A. Garay und Yoram Moses. „Fully Polynomial Byzantine Agreement for n > 3t Processors in t + 1 Rounds“. In: *SIAM Journal on Computing* 27.1 (Feb. 1998), S. 247–290. ISSN: 0097-5397. DOI: 10.1137/S0097539794265232.

[GMY95] Z. Galil, A. Mayer und Moti Yung. „Resolving Message Complexity of Byzantine Agreement and Beyond“. In: *Proceedings of IEEE 36th Annual Foundations of Computer Science*. Okt. 1995, S. 724–733. DOI: 10.1109/SFCS.1995.492674.

[Had87] Vassos Hadzilacos. „Connectivity Requirements for Byzantine Agreement under Restricted Types of Failures“. In: *Distributed Computing* 2.2 (1987), S. 95–103. ISSN: 0178-2770. DOI: 10.1007/BF01667081.

[Hal+84] Joseph Y Halpern u. a. „Fault-Tolerant Clock Synchronization“. In: *PODC '84: Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*. 1984, S. 89–102. ISBN: 0-89791-143-1. DOI: 10.1145/800222.806739.

[Har15] Evan Harris. *Precision Time Protocol Daemon - Ptpd2*. Feb. 2015. URL: https://sourceforge.net/projects/ptpd2/ (besucht am 20. 07. 2021).

[Heu+17] Jens Heuschkel u. a. „Introduction to RAW-Sockets“. In: *The Technical Reports Series of the TK Research Division, TU Darmstadt* (Mai 2017). ISSN: 18640516. URL: https://tuprints.ulb.tu-darmstadt.de/6243/1/TR-18.pdf (besucht am 14. 07. 2021).

[Hol91] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall Software Series. Englewood Cliffs and London: Prentice Hall, 1991. ISBN: 978-0-13-539925-5.

[HT94] Vassos Hadzilacos und Sam Toueg. „A Modular Approach to Fault-Tolerant Broadcasts and Related Problems“. In: Ithaca, NY, USA: Cornell University, 1994. URL: https://dl.acm.org/doi/10.5555/866693.

[IEE08] IEEE. „IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems“. In: *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*. Juli 2008, S. 1–269. ISBN: 978-0-7381-5400-8. DOI: 10.1109/IEEESTD.2008.4579760.

[IEE21] IEEE 802.1. *IEEE 802.1 Time-Sensitive Networking Task Group*. 2021. URL: https://www.ieee802.org/1/pages/tsn.html (besucht am 16. 09. 2021).

[INE20] INET. *INET Framework - INET Framework*. 2020. URL: https://inet.omnetpp.org/ (besucht am 22. 07. 2020).

[Ise93] Rolf Isermann. „Fault Diagnosis of Machines via Parameter Estimation and Knowledge Processing—Tutorial Paper“. In: *Automatica* 29.4 (1993), S. 815–835. ISSN: 0005-1098. DOI: 10.1016/0005-1098(93)90088-B.

[ITU94] ITU-T. „Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model“. In: (Juli 1994). URL: http://handle.itu.int/11.1002/1000/2820.

[JD02] Hao Jiang und Constantinos Dovrolis. „Guardian: A Router Mechanism for Extreme Overload Prevention“. In: *Scalability and Traffic Control in IP Networks II*. Bd. 4868. SPIE, Juli 2002, S. 277–288. DOI: 10.1117/12.475278.

[KC04] P. Koopman und T. Chakravarty. „Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks“. In: *International Conference on Dependable Systems and Networks, 2004*. Juni 2004, S. 145–154. ISBN: 0-7695-2052-9. DOI: 10.1109/DSN.2004.1311885.

[Kni02] J. C. Knight. „Safety Critical Systems: Challenges and Directions“. In: *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*. 2002, S. 547–550. ISBN: 1-58113-472-X.

[KO87] Hermann Kopetz und Wilhelm Ochsenreiter. „Clock Synchronization in Distributed Real-Time Systems“. In: *IEEE transactions on computers* 100.8 (1987), S. 933–940. ISSN: 0018-9340. DOI: 10.1109/TC.1987.5009516.

[Koo02] P. Koopman. „32-Bit Cyclic Redundancy Codes for Internet Applications“. In: *Proceedings International Conference on Dependable Systems and Networks*. Juni 2002, S. 459–468. ISBN: 0-7695-1101-5. DOI: 10.1109/DSN.2002.1028931.

[Kop+89] H. Kopetz u. a. „Distributed Fault-Tolerant Real-Time Systems: The Mars Approach“. In: *IEEE Micro* 9.1 (1989), S. 25–40. ISSN: 0272-1732. DOI: 10.1109/40.16792.

[Kra09] Milena Krasich. „How to Estimate and Use MTTF/MTBF Would the Real MTBF Please Stand Up?“ In: *2009 Annual Reliability and Maintainability Symposium*. Jan. 2009, S. 353–359. ISBN: 978-1-4244-2508-2. DOI: 10.1109/RAMS.2009.4914702.

[KS08] Ajay D. Kshemkalyani und Mukesh Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge: Cambridge Univ. Press, 2008. ISBN: 978-0-521-87634-6.

[LA86] T. V. Lakshman und Ashok K. Agrawala. „Efficient Decentralized Consensus Protocols“. In: *IEEE Transactions on Software Engineering* SE-12.5 (Mai 1986), S. 600–607. ISSN: 0098-5589. DOI: 10.1109/TSE.1986.6312956.

[Lam06] Leslie Lamport. „Fast Paxos“. In: *Distributed Computing* 19.2 (2006), S. 79–103. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2005-112.pdf.

[Lan+15] M Langer u. a. „MOVE-II - der zweite Kleinsatellit der Technischen Universität München“. In: (2015). URL: https://www.dglr.de/publikationen/2015/370241.pdf.

[Lar+14] Per Larsen u. a. „SoK: Automated Software Diversity“. In: *2014 IEEE Symposium on Security and Privacy*. San Jose, CA: IEEE, Mai 2014, S. 276–291. ISBN: 978-1-4799-4686-0. DOI: 10.1109/SP.2014.25.

[Law13] Averill M. Law. *Simulation Modeling and Analysis*. Fifth edition. McGraw-Hill Series in Industrial Engineering and Management Science. Dubuque: McGraw-Hill Education, 2013. ISBN: 978-0-07-340132-4.

[LM04] Leslie Lamport und Mike Massa. „Cheap Paxos". In: *International Conference on Dependable Systems and Networks, 2004*. IEEE. 2004, S. 307–314. ISBN: 0-7695-2052-9. DOI: 10.1109/DSN.2004.1311900.

[LM84] Leslie Lamport und P M Melliar-Smith. „Byzantine Clock Synchronization". In: *PODC '84: Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*. 1984, S. 68–74. ISBN: 0-89791-143-1. DOI: 10.1145/800222.806737.

[LP06] Lawrence M. Leemis und Stephen K. Park. *Discrete-Event Simulation: A First Course*. Pearson Prentice Hall, 2006. ISBN: 978-0-13-142917-8.

[LPS01] Bev Littlewood, Peter Popov und Lorenzo Strigini. „Modeling Software Design Diversity: A Review". In: *ACM Computing Surveys* 33.2 (Juni 2001), S. 177–208. ISSN: 0360-0300. DOI: 10.1145/384192.384195.

[LSP82] Leslie Lamport, Robert Shostak und Marshall Pease. „The Byzantine Generals Problem". In: *ACM Transactions on Programming Languages and Systems* 4.3 (1982), S. 382–401. ISSN: 01640925. DOI: 10.1145/357172.357176.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. [Nachdr.] The Morgan Kaufmann Series in Data Management Systems. San Francisco, California: Kaufmann, 1996. ISBN: 978-1-55860-348-6.

[Mar+10] Parisa Jalili Marandi u. a. „Ring Paxos: A High-Throughput Atomic Broadcast Protocol". In: *Proceedings of the International Conference on Dependable Systems and Networks* (2010), S. 527–536. ISSN: 1530-0889. DOI: 10.1109/DSN.2010.5544272.

[MC01] Matthew V Mahoney und Philip K Chan. „PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic". In: *Florida Institute of Technology Technical Report CS-2001-04* (2001). URL: http://hdl.handle.net/11141/94.

[Mey+20] Philipp Meyer u. a. „Network Anomaly Detection in Cars Based on Time-Sensitive Ingress Control". In: *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*. Nov. 2020, S. 1–5. ISBN: 978-1-72819-484-4. DOI: 10.1109/VTC2020-Fall49728.2020.9348746.

[MHS11] Z. Milosevic, M. Hutle und André Schiper. „On the Reduction of Atomic Broadcast to Consensus with Byzantine Faults". In: (2011). ISSN: 1060-9857. DOI: 10.1109/SRDS.2011.36.

[MMA90] P.M. Melliar-Smith, L.E. Moser und V. Agrawala. „Broadcast Protocols for Distributed Systems“. In: *IEEE Transactions on Parallel and Distributed Systems* 1.1 (Jan./1990), S. 17–25. ISSN: 1045-9219. DOI: 10.1109/71.80121.

[Moz18] Zoha Moztarzadeh. „Fault-Tolerant Clock Synchronization with Only Two Redundant Paths“. In: *Computer Safety, Reliability, and Security*. Västerås, Sweden: Springer Nature Switzerland, 2018, S. 235–249. ISBN: 978-3-319-99129-0. DOI: 10.1007/978-3-319-99130-6_16.

[Moz19] Zoha Moztarzadeh. „Fault-Tolerant Clock Synchronization in Distributed Systems with Low Topological Requirements“. Diss. Essen: Universität Duisburg-Essen, 2019. URL: https://duepublico2.uni-due.de/receive/duepublico_mods_00070894.

[MRB90] A. Majumdar, C.S. Raghavendra und M.A. Breuer. „Fault Tolerance in Linear Systolic Arrays Using Time Redundancy“. In: *IEEE Transactions on Computers* 39.2 (Feb./1990), S. 269–276. ISSN: 0018-9340. DOI: 10.1109/12.45214.

[Nee93] Roger M. Needham. „Denial of Service“. In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. CCS '93. New York, NY, USA: Association for Computing Machinery, Dez. 1993, S. 151–153. ISBN: 978-0-89791-629-5. DOI: 10.1145/168588.168607.

[Neu86] Peter Gabriel Neumann. „On Hierarchical Design of Computer Systems for Critical Applications“. In: *IEEE Transactions on Software Engineering* SE-12.9 (1986), S. 905–920. ISSN: 0098-5589. DOI: 10.1109/TSE.1986.6313046.

[Nør00] Kjetil Nørva. „An Introduction to Fault-Tolerant Systems“. In: (2000). ISSN: 0802-6394.

[NZB13] Meiyappan Nagappan, Thomas Zimmermann und Christian Bird. „Diversity in Software Engineering Research“. In: (2013), S. 466–476. DOI: 10.1145/2491411.2491415.

[ÖB09] Onur Özgün und Yaman Barlas. „Discrete vs. Continuous Simulation: When Does It Matter?“ In: *Proceedings of the 27th International Conference of The System Dynamics Society*. Albuquerque, NM, USA, 2009. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.644.21.

[Ope20a] OpenSimLtd. *OMNeT++ - Simulation Manual*. 2020. URL: https://doc.omnetpp.org/omnetpp/manual/ (besucht am 22. 09. 2020).

[Ope20b] OpenSimLtd. *OMNeT++ Discrete Event Simulator*. 2020. URL: https://omnetpp.org/ (besucht am 22. 07. 2020).

[Pap+07] Charis Papadakis u. a. „A Feedback-Based Approach to Reduce Duplicate Messages in Unstructured Peer-To-Peer Networks“. In: *Integrated Research in GRID Computing: CoreGRID Integration Workshop 2005 (Selected Papers) November 28–30, Pisa, Italy*. Hrsg. von Sergei Gorlatch und Marco Danelutto. Boston, MA: Springer US, 2007, S. 103–118. ISBN: 978-0-387-47658-2. DOI: 10.1007/978-0-387-47658-2_8.

[Par72] D. L. Parnas. „On the Criteria to Be Used in Decomposing Systems into Modules“. In: *Communications of the ACM* 15.12 (1972), S. 1053–1058. ISSN: 0001-0782. DOI: 10.1145/361598.361623.

[PBS89] Larry L. Peterson, Nick C. Buchholz und Richard D. Schlichting. „Preserving and Using Context Information in Interprocess Communication“. In: *ACM Transactions on Computer Systems* 7.3 (Aug. 1989), S. 217–246. ISSN: 0734-2071. DOI: 10.1145/65000.65001.

[PS98] Fernando Pedone und André Schiper. „Optimistic Atomic Broadcast“. In: *Distributed Computing*. Hrsg. von Gerhard Goos u. a. Bd. 1499. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, S. 318–332. ISBN: 978-3-540-65066-9. DOI: 10.1007/BFb0056492.

[RG88] Tenkasi V Ramabadran und Sunil S Gaitonde. „A Tutorial on CRC Computations“. In: *IEEE micro* 8.4 (1988), S. 62–75. ISSN: 0272-1732. DOI: 10.1109/40.7773.

[RK06] J. Ray und P. Koopman. „Efficient High Hamming Distance CRCs for Embedded Networks“. In: *International Conference on Dependable Systems and Networks (DSN'06)*. Juni 2006, S. 3–12. ISBN: 0-7695-2607-1. DOI: 10.1109/DSN.2006.30.

[Rob91] Robert Bosch GmbH. „CAN Specification“. In: (1991). URL: http://esd.cs.ucr.edu/webres/can20.pdf.

[SD02] Gary W Scheer und David J Dolezilek. „Comparing the Reliability of Ethernet Network Topologies in Substation Control and Monitoring Networks“. In: *UTC Telecom 2002* (2002).

[SF91] Ralf Seliger und Paul M Frank. „Fault-Diagnosis by Disturbance Decoupled Nonlinear Observers“. In: *[1991] Proceedings of the 30th IEEE Conference on Decision and Control*. IEEE. 1991, S. 2248–2253. ISBN: 0-7803-0450-0. DOI: 10.1109/CDC.1991.261547.

[SH94] K.G. Shin und Hagbae Kim. „A Time Redundancy Approach to TMR Failures Using Fault-State Likelihoods“. In: *IEEE Transactions on Computers* 43.10 (Oct./1994), S. 1151–1162. ISSN: 0018-9340. DOI: 10.1109/12.324541.

[She92] Robin L. Sherman. „Distributed Systems Security“. In: *Computers & Security* 11.1 (1992), S. 24–28. ISSN: 0167-4048. DOI: 10.1016/0167-4048(92)90216-E.

[Sin21] Sinovoip. *BPI-R1*. 2021. URL: http://www.banana-pi.org/r1.html (besucht am 08. 07. 2021).

[SJ08] Robert Shaw und Brendan Jackman. „An Introduction to FlexRay as an Industrial Network“. In: *2008 IEEE International Symposium on Industrial Electronics*. 2008, S. 1849–1854. ISBN: 978-1-4244-1665-3. DOI: 10.1109/ISIE.2008.4676987.

[SR04] John A Stankovic und Raj Rajkumar. „Real-Time Operating Systems“. In: *Real-Time Systems*. Bd. 28. 2004, S. 237–253. ISBN: 978-3-642-88051-3. DOI: 10.1007/978-3-642-88049-0_5.

[SS16] Johannes Specht und Soheil Samii. „Urgency-Based Scheduler for Time-Sensitive Switched Ethernet Networks“. In: *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*. 2016, S. 75–85. ISBN: 978-1-5090-2811-5. DOI: 10.1109/ECRTS.2016.27.

[ST87] T. K. Srikanth und Sam Toueg. „Optimal Clock Synchronization“. In: *J. ACM* 34.3 (Juli 1987), S. 626–645. ISSN: 0004-5411. DOI: 10.1145/28869.28876.

[Ste18] Till Steinbach. „Evaluierung in der Simulation“. In: *Ethernet-basierte Fahrzeugnetzwerkarchitekturen für zukünftige Echtzeitsysteme im Automobil*. Hrsg. von Till Steinbach. Wiesbaden: Springer Fachmedien, 2018, S. 103–220. ISBN: 978-3-658-23500-0. DOI: 10.1007/978-3-658-23500-0_4.

[The21] The Tcpdump Group. *TCPDUMP/LIBPCAP Public Repository*. 2021. URL: https://www.tcpdump.org/ (besucht am 04. 08. 2021).

[TS92] J. Turek und D. Shasha. „The Many Faces of Consensus in Distributed Systems“. In: *Computer* 25.6 (1992), S. 8–17. ISSN: 0018-9162. DOI: 10.1109/2.153253.

[TS94] Sandra R Thuel und Jay K Strosnider. „Enhancing Fault Tolerance of Real-Time Systems through Time Redundancy“. In: *Foundations of Dependable Computing*. Springer, 1994, S. 265–318. ISBN: 978-0-7923-9486-0.

[TV07] Andrew S. Tanenbaum und Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Second Edition. Always Learning. Upper Saddle River: Pearson Education, 2007. ISBN: 0-13-239227-5.

[VSN14] Antonio Virdis, Giovanni Stea und Giovanni Nardini. „SimuLTE - A Modular System-Level Simulator for LTE/LTE-A Networks Based on OMNeT++“. In: *2014 4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications (SIMULTECH)*. Aug. 2014, S. 59–70. ISBN: 978-989-758-038-3. DOI: 10.5220/0005040000590070.

[Xil21] Xilinx. *ZedBoard Zynq-7000 ARM/FPGA SoC Development Board*. 2021. URL: https://www.xilinx.com/products/boards-and-kits/1-elhabt.html (besucht am 07. 07. 2021).

[Zla15] Nikola Zlatanov. „Computer Security and Mobile Security Challenges“. In: *Tech Security Conference*. San Fransisco, CA, 2015. URL: https://www.researchgate.net/publication/298807979_Computer_Security_and_Mobile_Security_Challenges.

# Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig ohne unzulässige Hilfe Dritter verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und alle wörtlich oder inhaltlich übernommenen Stellen unter der Angabe der Quelle als solche gekennzeichnet habe.

Die Grundsätze für die Sicherung guter wissenschaftlicher Praxis an der Universität Duisburg-Essen sind beachtet worden.

Ich habe die Arbeit keiner anderen Stelle zu Prüfungszwecken vorgelegt.

_____

(Datum, Unterschrift)