Ein Effizientes Byzantinisches fehlertolerantes Übereinstimmungsprotokoll für verteilte drahtlose Echtzeitsysteme

Dissertation

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

durch die Fakultät für Wirtschaftswissenschaften der

Universität Duisburg-Essen
Campus Essen

vorgelegt von

Dipl. Inf. Omar Bousbiba, Beni Bouyahie

Essen 2016

Tag der mündlichen Prüfung:	20.12.2016
Erstgutachter:	Prof. Dr. Klaus Echtle
Zweitgutachter:	Prof. Dr. Tobias Hoßfeld

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich während meiner Zeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Verlässlichkeit von Rechensystemen unterstützt haben.

Ganz besonders bedanken möchte ich mich bei Prof. Dr. Klaus Echtle, meinem Doktorvater, für die wertvollen fachlichen Hinweise, zahlreichen Gespräche auf intellektueller und persönlicher Ebene, sowie für die Unterstützung in allen Phasen dieser Arbeit. Er brachte mir sehr viel Geduld entgegen und ohne sein unermüdliches Korrekturlesen wäre die Arbeit nicht zu dem geworden, was sie nun ist.

Weiterhin danke ich Prof. Dr. Tobias Hoßfeld für seine Bereitschaft, das Zweitgutachten zu erstellen.

Abstract

Using distributed (real-time) systems has become an integral part of industrial applications such as medical technology, automotive engineering, aeronautics and automation engineering. Along with the progress of technological development, it can be expected that the field of distributed (real-time) systems extends to other areas of industrial applications. This is a result of continuous technological advances. Given the fact that malfunctions in a distributed system (which can compromise the reliability and safety of systems) cannot be completely avoided, fault-tolerant mechanisms have to be developed and applied. Furthermore, many safety-critical applications are hard real-time applications and subject to cost restrictions. Therefore, for the practical usability of a distributed system with real-time requirements all of the following properties can become crucial: the fault tolerance capability, the communication complexity in terms of the number of required nodes, overall communication overhead as well as the overhead caused by the message storage.

The Byzantine agreement problem has been exposed as one of the most fundamental issues to be solved. However, solving the Byzantine agreement problem in an efficient way in terms of communication complexity is still a challenging task. The following thesis deals with techniques and strategies for designing efficient fault-tolerant Byzantine agreement protocols primarily for wireless distributed real-time applications. In this paper two new solutions are presented, evaluated, and proven as correct.

In the first approach, a novel synchronous single-round-based agreement protocol – called ESSEN – is presented, which copes with f arbitrary faults (including cooperative Byzantine faults) using at least $n \geq 3 \cdot f + \max(0, f - 2)$ nodes. Moreover, this is the first approach which solves the Byzantine agreement problem in a single broadcast round independent of the number of tolerated faults.

Following this, we present a novel signature generation technique, called SigSeam, to merge several signatures into a single one, which is the topic of the second part of this thesis. This advantage opens a design space for agreement protocols with significantly reduced message overhead. Moreover, the new signature technique can also be applied to existing agreement and/or consensus protocols without affecting the fault tolerance properties of the protocol.

Within the framework of this thesis it could be shown that the proposed signature technique with merging functionality significantly improves the efficiency of agreement protocols. However, to achieve a fault coverage comparable to conventional signature techniques, SigSeam requires approximately 25 percent more information redundancy.

Altogether, the goal of improving the efficiency of agreement protocols has been achieved.

Keywords:

Byzantine Faulst, Agreement Protocols, Digital Signatures for Fault Tolerance, Distributed Systems

Zusammenfassung

Der Einsatz von verteilten (Echtzeit-) Systemen ist in vielen Bereichen der Industrie nicht mehr wegzudenken, wie etwa in der Medizintechnik, der Kraftfahrzeugtechnik, der Flugtechnik oder Automatisierungstechnik. Weiterhin kann man davon ausgehen, dass sich im Zuge der fortschreitenden Technologieentwicklung der Einsatzbereich von verteilten (Echtzeit-) Systemen auch in anderen Bereichen der Industrie weiter ausdehnen wird. Da in solchen Systemen jederzeit Fehler auftreten können, welche die Zuverlässigkeit und Sicherheit beeinträchtigen, müssen geeignete Fehlertoleranz-Verfahren entwickelt und eingesetzt werden. Ferner unterliegen viele sicherheitskritische Anwendungen harten Echtzeitanforderungen und zugleich deutlichen Kostenrestriktionen. In solchen Anwendungen spielt für die praktische Umsetzbarkeit nicht einzig die Fehlertoleranzfähigkeit eine entscheidende Rolle, sondern ebenfalls der von Fehlertoleranzverfahren verursachte Kommunikationsaufwand in Form von Nachrichten-, Knoten- und Speicheroverhead.

Das Byzantinische Übereinstimmungsproblem stellt eines der wichtigsten zu lösenden Probleme in fehlertoleranten verteilten Systemen dar. Obwohl das Byzantinische Übereinstimmungsproblem gut erforscht ist und viele Lösungen unter verschiedenen Systemmodellannahmen existieren, stellt die Entwicklung effizienter Lösungen bis heute eine anspruchsvolle Aufgabe dar, die abhängig vom Fehler- und Timing-Modell sowie von den Aufwands- und Kostengrenzen alles andere als trivial zu lösen ist.

Die vorliegende Arbeit untersucht Techniken und Strategien zur Entwicklung effizienter Übereinstimmungsprotokolle für verteilte (vorwiegend drahtlose) Echtzeitsysteme, und stellt hierzu zwei Lösungen vor.

Im ersten Lösungsansatz wird ein neuartiges rundenbasiertes Übereinstimmungsprotokoll – ESSEN genannt – vorgestellt, das für synchrone verteilte Systeme effizient erbeitet. ESSEN löst das Byzantinische Übereinstimmungsproblem in Anwesenheit von bis zu f willkürlichen Fehlern (kooperierende Byzantinische Fehler inbegriffen). Hierzu benötigt ESSEN mindesten $n \geq 3f + max(0, f - 2)$ Knoten. Außerdem stellt das Übereinstimmungsprotokoll ESSEN den ersten Lösungsansatz dar, welcher das Byzantinische Übereinstimmungsproblem unabhängig von der Anzahl der zu tolerierenden Fehler in einer Runde löst.

Obwohl ESSEN eine effiziente Lösung darstellt, lag die Vermutung nahe, dass durch den Einsatz eines geeigneten Signaturverfahrens eine weitere Verbesserung bzgl. der Kommunikationskomplexität erzielt werden kann. Folglich wurde im zweiten Teil der Arbeit ein weiterer Lösungsansatz entwickelt, mit dessen Hilfe sich die Kommunikationskomplexität von Übereinstimmungsprotokollen weiter reduzieren lässt (von ESSEN abweichende Übereinstimmungsprotokolle eingeschlossen).

Im zweiten Lösungsansatz wurde zur Verbesserung der Kommunikationskomplexität von ESSEN ein neuartiges Verfahren zur Erzeugung und Prüfung von Signaturen (kurz: Signaturverfahren) – SigSeam genannt – vorgestellt, welches mehrere Signaturen zu einer einzigen Signatur zusammenfasst, ohne die Nachrichtengröße hierdurch zu verändern. Im Rahmen der Arbeit konnte gezeigt werden, dass das Signaturverfahren SigSeam in der Lage ist, die Kommunikationskomplexität von Übereinstimmungsprotokollen signifikant zu reduzieren. Dies betrifft sowohl die Nachrichtenlänge wie auch die Nachrichtenanzahl, die beide reduziert werden können. Allerdings benötigt SigSeam im Vergleich zu herkömmlichen Signaturverfahren für eine einzelne Signatur eine um ca. 25 Prozent höhere Informationsredundanz, wenn eine gleich gute Fehlerfassung wie bei diesen erzielt werden soll.

Insgesamt konnte mit den beiden Lösungen ESSEN und SigSeam das Ziel der Effizienzsteigerung von Übereinstimmungsprotokollen für verteilte (Echtzeit-) Systeme erreicht werden. Weiterhin konnte gezeigt werden, dass das Prinzip der Signaturverschmelzung zur Reduzierung der Kommunikationskomplexität prinzipiell auf einen Großteil der existierenden Übereinstimmungsprotokolle angewendet werden kann.

Schlagwörter:

Byzantinische Fehler, Übereinstimmungsprotokolle, digitale Signaturen im Bereich Fehlertoleranz, verteilte Systeme

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

„Nichts ist so beständig wie der Wechsel.“ – Heraklit von Ephesus (etwa 540 - 480 v. Chr.)

1.1 Motivation

In unserem alltäglichen Leben begegnen wir verteilten Systemen in allen möglichen Formen, sei es das ESP-System in einem Auto, das Führungssystem der Bahn, mit der täglich Güter und Personen transportiert werden, oder das Internet, mit dem Millionen von Menschen mittels sozialer Netzwerke (Facebook, Xing und Co) Informationen untereinander austauschen sowie Beziehungsnetze bilden.

Ein verteiltes System („distributed System“) lässt sich beschreiben als ein Kollektiv von autonomen Rechnern (Knoten), die über Nachrichtenaustausch miteinander interagieren. Im Gegensatz zu zentralisierten Systemen gibt es in einem verteilten System keinen „Master“-Knoten, der über die volle Kontrolle aller Komponenten im System verfügt (Tanenbaum und van Steen 2008). Außerdem ist der Einsatz von verteiltem System oft durch die physikalische Verteilung der Knoten bereits vorgegeben, wie es beispielsweise beim Auto der Fall ist (Abbildung 1).

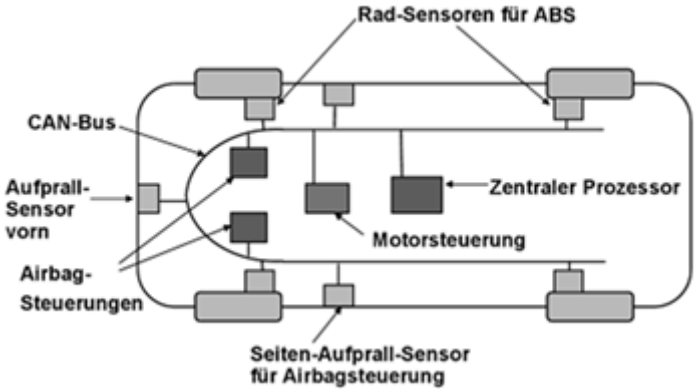


Abbildung 1: Verteiltes System am Beispiel eines Autos [S.98, (Walter Lange)]

Wie in Abbildung 1 vereinfacht skizziert (die Elektronik und die zur Kommunikation realisierten Verbindungen im Automobil sind viel komplexer als das Bild suggeriert), werden die einzelnen Funktionen (elektronische Anwendungen) im Auto von räumlich verteilten elektronischen Kontrolleinheiten (auch ECUs genannt), Sensoren und Aktuatoren erbracht, die über ein gemeinsames Kommunikationsmedium verbunden sind. Der elektronische Notfallbremsassistent ist ein Beispiel für eine Anwendung, die verteilt realisiert wird (auch

verteilte Anwendung genannt). Erkennt das System eine Gefahrensituation, so werden anhängig von der Situation entsprechende Maßnahmen eingeleitet, wobei der Fahrer als Teil des Systems involviert ist (z.B. Reaktion auf eine Auffahrwarnung). Gelten beim verteilten System zusätzlich starke zeitliche Anforderungen (z.B. wenn die Reaktion auf ein Ereignis bestimmten zeitlichen Bedingungen unterliegt), spricht man von einem verteilten Echtzeitsystem (Wörn). Wenn ein verteiltes System zur (Teil-) Automatisierung eines technischen Systems eingesetzt wird (wie in dem skizzierten Beispiel eines Fahrzeugs), handelt es sich fast immer um ein verteiltes Echtzeitsystem.

In der Literatur findet man unter dem Begriff Echtzeitsystem die folgende Definition: Ein Echtzeitsystem ist ein *Computersystem, dessen Korrektheit nicht nur vom logischen Ergebnis der Berechnung abhängt, sondern auch von dem physischen Augenblick, zu dem diese Ergebnisse erstellt werden.* (Koptez 1997)

In vielen Zweigen der Industrie haben (verteilte) Echtzeitsysteme eine bedeutende Funktion eingenommen, wie etwa in der Medizintechnik, der Kraftfahrzeugtechnik, der Flugtechnik oder Automatisierungstechnik. Weiterhin kann man davon ausgehen, dass sich im Zuge der fortschreitenden Technologieentwicklung, insbesondere bezüglich der Miniaturisierung moderner Hardware („Many/Multi – Core“) der Einsatzbereich von verteilten (Echtzeit-) Systemen auch in anderen Bereichen der Industrie weiter ausdehnt.

In einem verteilten System kann der Datenaustausch zur Koordination der Knoten, die über ein Netz (bzw. Netzwerk) miteinander verbunden sind, entweder drahtgebunden oder drahtlos erfolgen. Bei einem drahtlosen verteilten Systeme dient als Übertragungsmedium Luft/Vakuum zur Kommunikation der Knoten, während bei drahtgebundenen verteilten Systemen ein von Umwelteinflüssen abgeschirmter elektrischer-/Lichtwellen-Leiter zum Einsatz kommt. Beide Realisierungsformen haben sowohl Vorteile als auch Nachteile, auf die im Folgenden zum Teil näher eingegangen wird. Zunächst wird der Übergang von drahtgebunden zu drahtlosen Netzen motiviert.

In manchen Zweigen der Industrie lassen sich drahtgebundene verteilte Systeme nur mit einem unverhältnismäßigen hohen Kostenaufwand realisieren, wie etwa in der Luftfahrtindustrie, um nur ein Beispiel zu nennen (siehe auch (Chae et al. 2012)). Im Fall der Luftfahrindustrie gibt es bereits heute Bestrebungen, die in einem Flugzeug verbauten drahtgebundenen Komponenten durch drahtlose zu ersetzen (Elrahim et al.; Ghajari et al.; Jalali et

al.; Razi und Abedi; Elgezabal Gomez; Dinh-Khanh Dang et al.). Drahtlose Netze weisen Vor- und Nacheile auf. Im Gegensatz zu drahtgebundenen Lösungen mit einer teilweisen hohen Verkabelungskomplexität, sind sie diesbezüglich unkompliziert. Verkabelungskosten, Probleme der Leitungsführung und das Gewicht der Verkabelung, das insbesondere im Flugzeug eine große Rolle spielt, werden vermieden. Andererseits basieren kabellose Lösungen auf dem Austausch von Funksignalen, die durch störende Sender und Quellen von elektromagnetischen Wellen beeinträchtigt werden können. Jedoch lässt sich die Beeinträchtigung mittels Einsatz mehrerer Hochfrequenz-Kanäle (HF-Kanäle) signifikant mildern. Beispielsweise kann zur Realisierung der Kommunikation zwischen zwei oder mehreren Knoten das Kanalsprungverfahren verwendet werden. Eine andere Möglichkeit wäre, die zur Verfügung stehenden Kanäle (gemäß 2,4-GHz-Standard IEEE 802.15.4 stehen bis zu 15 Kanäle zu Verfügung) zur redundanten Übertragung von Information einzusetzen (wie in Abbildung 2 vereinfacht skizziert dargestellt).

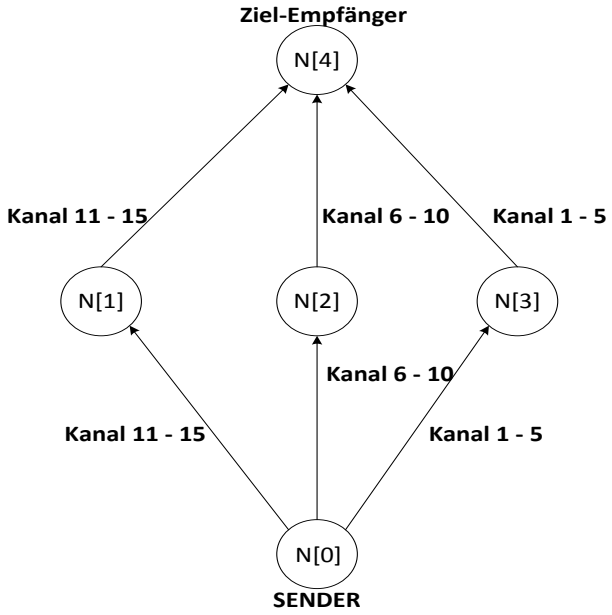


Abbildung 2: Kommunikation über redundante Kanäle in einem drahtlosen verteilten System mit der Knotenmenge N = { N[0], N[1], N[2], N[3], N[4]}.

Wenn die Knoten eines verteilten Systems zur Überwachung von physikalischen Messwerten eingesetzt werden (und ggf. über Aktuatoren verfügen, um ihre Umgebung zu beeinflussen), spricht man in der Regel von einem (drahtlosen bzw. drahtgebundenen) Sensornetz bzw.

Sensornetzwerk. Im Allgemeinen setzen sich solche Netze aus einer Vielzahl von Knoten zusammen, die über eine geeignete Topologie (wie z. B. Mesh -/Stern-/Baum-Topologie, etc.) miteinander verbunden sind. Sensornetze werden sowohl für militärische als auch für industrielle Zwecke eingesetzt (Überwachung von Bauwerken, Objektverfolgung, Maschinensteuerung, etc.). Ihr Einsatzgebiet umfasst sowohl sicherheitskritische Anwendungen (Frühwarnsysteme, Gebäudesicherheit, Erdbebenüberwachung, Langzeit-Brückenüberwachung, etc.) als auch nicht sicherheitskritische Anwendungen (Warenüberwachung, Klimatisierung von Gebäuden, etc.).

Im sicherheitskritischen Kontext ist es oft essentiell, verteilte Echtzeit-Systeme (sowohl drahtlose als auch drahtgebundene) so zu konzipieren, dass sie einen ausreichenden Grad an Resistenz gegenüber Fehlern zeigen. So darf beispielsweise ein Fehler, der irgendwo im System auftritt und als Folge manche der Knoten in einen fehlerhaften Zustand überführt, nicht zu einem Ausfall des Gesamtsystems führen („**Single-Point-of-Failure**“). Die gleichen Überlegungen kann man auch für Mehrfachfehler („**Multiple-Point-of-Failure**“) anstellen.

Je nachdem, wie kritisch die Anwendung ist, wird vom System im Fehlerfall entweder verlangt, dass es in einen sicheren Zustand übergeht („**Fail-Safe**“-Verhalten) oder seine Funktionalität auch in Anwesenheit von Fehlern weiterhin gewährleistet („**Fail-Operational**“-Verhalten, wie z.B. Monitoring und Tracking von Objekten im Bereich von Sensornetzen).

Viele (meist echtzeitkritische) Anwendungen erfordern vom System **Fail-Operational** Verhalten. Nehmen wir als Beispiel das **Fault-Tolerant Drive-by-Wire** System (Anwar 2012). Es muss seine Funktionalität jederzeit bereitstellen. Im Falle eines Fehlers müssen für einen sicheren Betrieb mindestens zwei der vier Bremsen aktiviert werden (diagonal vorn rechts und hinten links, oder aber vorn links und hinten rechts). Andernfalls riskiert man einen gefährlichen Zustand des Fahrzeugs aufgrund unzureichender Bremswirkung. Folglich muss eine Entscheidung bzgl. der zu deaktivierenden Bremsenpaare (in einer nicht blockierenden Weise) getroffen werden. Damit das System diese Eigenschaft besitzt, müssen geeignete Sicherungsmaßnahmen eingeleitet werden, wie etwa durch den Einsatz von Techniken zur Fehlertoleranz.

Mit Hilfe der Fehlertoleranz lassen sich verschiedene Zuverlässigkeitseigenschaften des verteilten Systems verbessern, sowohl für **Fail-Safe-** als auch für **Fail-Operational-**

Anwendungen. Für das genannte Beispiel der Fahrzeugbremse müsste das Fehlertoleranz-Verfahren sicherstellen, dass fehlerfreie Knoten stets eine übereinstimmende Sicht auf die Komponenten des sicherheitskritischen **Brake-by-Wire**-Systems gewinnen (hinsichtlich ausgefallener Knoten und den zu passivierenden Knoten). Andernfalls würde die Gefahr bestehen, dass das System in einem gefährlichen Zustand übergeht. Eine Möglichkeit, um die geforderte Konsistenz zu erreichen, ist der Einsatz von Protokollen, die das beschriebene Übereinstimmungsproblem lösen.

Wenn das Problem für beliebiges Verhalten fehlerhafter Knoten gelöst werden soll, spricht man von Byzantinischer Übereinstimmung (in Anlehnung an die aus der Literatur bekannten Byzantinischen Generäle, (Lamport et al. 1982)). Das Byzantinische Übereinstimmungsproblem ist ein gut erforschtes Problem mit vielen Lösungen im Bereich verteilter Rechensysteme. Nichtsdestotrotz ist die Entwicklung effizienter Lösungen bis heute Gegenstand der Forschung mit weitreichendem Einfluss auf die in Wirtschaft und Industrie eingesetzten Systeme (Jeffery und Figueiredo; Khosravi und Kavian 2012; Young und Boutaba 2011; Coelho et al.; Liu; Capella et al.; Guerrero et al. 2013; Qu et al. 2012; Abdelhakim et al.). Daher beschäftigt sich die vorliegende Dissertation mit dem Design eines effizienten Übereinstimmungsprotokolls sowie mit dem Design eines dafür geeigneten effizienten Signaturverfahrens. Sie liefert hierzu die folgenden Beiträge:

1. **ESSEN (Efficient Single Round Signature-Protected Message Exchange Agreement Protocol for Wireless Distributed Networks),** ist ein Byzantinischer Übereinstimmungsprotokoll für drahtlose verteilte Systeme bzw. Netze (Bousbiba 2015a): Es löst das Byzantinische Übereinstimmungsproblem für synchrone verteilte Systeme, bei denen sowohl Knotenfehler als auch Kommunikationsfehler in Form von Nachrichtenausfällen auftreten können. Die Anzahl der tolerierbaren Nachrichtenausfälle ist in ESSEN durch das Unmöglichkeitsergebnis von Santoro und Widmayer, das für alle Protokolle gilt, nach oben beschränkt (Santoro und Widmayer 2007, 1989). Um eine Übereinstimmung gemäß (Lamport et al. 1982) zu erzielen, muss in ESSEN jeder Knoten unabhängig von der Anzahl der zu tolerierenden Fehlern nicht mehr als eine Nachricht übertragen. Dies macht das Protokoll besonders attraktiv für Anwendungen in Echtzeitumgebungen, bei denen neben der Erzielung einer Einigung zwischen den fehlerfreien Knoten im System auch der Gesamtaufwand (Kommunikationsoverhead, Speicheroverhead und/oder Knotenoverhead) sowie der Zeitbedarf für

die Protokollausführung eine entscheidende Rolle spielen. Das Übereinstimmungsprotokoll ESSEN löst das Übereinstimmungsproblem für Single-Hop-Netze. Dies bedeutet, dass die Knoten des verteilten Systems ein vollvermaschtes Netz bilden, bei dem jeder Knoten jeden anderen Knoten über einen direkten Kommunikationsweg (ohne zwischenliegende Knoten) erreichen kann (für eine genauere Definition siehe (Kurose und Ross 2014)). Lösungen für Multi-Hop-Netze werden in dieser Arbeit nicht betrachtet. Der Grund hierfür ist, dass im Allgemeinen jede Lösung, die das Übereinstimmungsproblem für Single-Hop-Netze löst, durch den Einsatz eines geeigneten Routing-Protokolls (siehe z.B. (Awerbuch et al.)) auf Multi-Hop-Netze erweitert werden kann.

2. **Ein neuartiges Signaturverfahren zur Reduzierung des Kommunikationsoverheads in Übereinstimmungsprotokollen für synchrone und asynchrone verteilte Systeme** (siehe ggf. Kapitel 7 bzw. (Bousbiba 2015b)). Bei der Entwicklung neuartiger Übereinstimmungsprotokolle ist man im Allgemeinen sowohl an der Reduktion der Kommunikationskomplexität als auch an der Maximierung der Fehlertoleranzfähigkeit interessiert. Daher ist es eher selten, dass neuere Publikationen zu Übereinstimmungsprotokollen auf das Mittel der digitalen Signatur verzichten. Der Vorteil von Signaturen liegt klar auf der Hand: Durch Einsatz von Signaturen können sowohl die Anzahl der redundanten Knoten als auch die Anzahl der zu übertragenen Nachrichten sinken, was in der Praxis mehr als wünschenswert ist. Die Vorteile von Signaturen gehen aus den folgenden Quellen deutlich hervor (Echtle 1999; Leu 1994; Saini und Singh; Kotla et al.; Tulone 2004). Es sei betont, dass es sich in diesem Kontext generell um Fehlertoleranz-Signaturen handelt, die Maskerade-Fehler (falsche Ende-zu-Ende-Sender-Empfänger-Beziehungen) aufdecken, nicht aber intelligenten Angriffen durch Menschen. In diesem Sinne handelt es sich um kryptographisch schwache Signaturen. Wegen des großen Potentials zur Aufwandsreduzierung ist die Entwicklung eines neuartigen, an das Protokoll angepassten Signaturverfahrens, SIGSEAM genannt (Bousbiba 2015b); der nächste logische Schritt, wenn man an der Reduktion der Kommunikationskomplexität und der redundanten Mittel interessiert ist. Im Vergleich zu den existierenden Verfahren bietet SIGSEAM die Möglichkeit, mehrere Signaturen zu einer Signatur zu „verschmelzen“, wenn der gleiche Nachrichteninhalt von verschiedenen Knoten signiert worden ist. Der Nachrichteninhalt kann dann mit der verschmolzenen Signatur an andere Knoten weitergeleitet werden. Die können dann durch einen speziellen Signaturtest prüfen, dass tatsächlich alle

Knoten, deren Signatur in die verschmolzene Signatur eingeflossen ist, den Nachrichteninhalt signiert haben. Nicht nur das in dieser Arbeit präsentierte Protokoll ESSEN, sondern auch viele andere Byzantinische Übereinstimmungsprotokolle können durch den Einsatz des neuartigen Signaturverfahrens einen verbesserten Gesamtaufwand erzielen.

In dieser Dissertation werden anhand zweier Protokolle

- Moniz, Henrique: Turquois (Moniz et al. 2013)
- Bousbiba: ESSEN (Bousbiba 2015a)

die Trade-offs des neuen Signaturverfahrens veranschaulicht.

1.2 Aufbau der Arbeit

Die vorliegende Dissertation ist wie folgt gegliedert:

Kapitel 2 vermittelt sämtliche Grundlagen, die für das Verständnis der Arbeit als notwendig erachtet werden.

Kapitel 3 zeigt den Stand der Technik im Bereich der Übereinstimmungsprotokolle.

Kapitel 4 stellt das neue byzantinische Übereinstimmungsprotokoll ESSEN vor und vergleicht es mit anderen existierenden Lösungen. Außerdem wird das dem Protokoll zugrundeliegende Systemmodell formal gefasst, so dass Korrektheitsnachweise durchgeführt werden können: Verifikation und Validierung in Kapitel 5 sowie ein formaler Korrektheitsbeweis in Kapitel 6.

Kapitel 7 stellt das neue Signaturverfahren SIGSEAM vor. Neben einer kurzen Motivation und der Vorstellung der Idee wird das Konzept der Signaturverschmelzung präsentiert. Im Anschluss erfolgt die Beschreibung des Fehlermodells (das auch als Basis für die später durchgeführte Fehlerinjektion dient) mit den dazugehörigen Annahmen und Vereinfachungen, zu denen jeweils eine Begründung gegeben wird. Das Kapitel 7 schließt mit der Auswertung der Fehlerabdeckung. Außerdem wird das neue Signaturverfahren einem anderen Signaturverfahren (siehe (Echtle 1999)) gegenübergestellt. Bei der Gegenüberstellung werden sowohl die Vorteile als auch die Nachteile des neuen Signaturverfahrens SIGSEAM skizziert.

Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick in Kapitel 8.

2 Übereinstimmungsprotokolle in verteilten Systemen

„Das Wesen der Dinge hat die Angewohnheit, sich zu verbergen.“ – Heraklit von Ephesus (etwa 540 - 480 v. Chr.)

„Ein verteiltes System ist ein System, mit dem ich nicht arbeiten kann, weil irgendein Rechner abgestürzt ist, von dem ich nicht einmal weiß, dass es ihn überhaupt gibt“– Leslie Lamport.

2.1 Knotenfehler und Kanalfehler

Seit den Anfängen der 90-er Jahre hat der Einsatz verteilter Systeme bzw. verteiltes Rechnen stetig zugenommen und ist heute in unserer elektronischen Gesellschaft nicht mehr wegzudenken. Im Banksystem automatisieren sie (und dem zufolge vereinfachen sie) die Finanztransaktionen. In Flugzeugen kommen sie als Kontrollsystem für Flugsteuerung und Flugnavigation zum Einsatz, wie etwa das „Flight Management and Guidance System“ (FMGS), und im Automobilbereich vernetzen sie die eingebauten Steuergeräte („ECUs“), um nur ein paar Einsatzgebiete zu nennen.

Gemäß (Tanenbaum und van Steen 2008) gilt für verteilte Systeme die folgende Definition: „*Als Verteiltes System wird ein System bezeichnet, bei dem sich die Hardware- und Softwarekomponenten auf vernetzten Rechnern befinden und nur über den Austausch von Nachrichten kommunizieren und ihre Aktionen koordinieren.“*

Das Finden einer Übereinstimmung in verteilten Systemen stellt eine spezielle Form der Koordination dar, bei der eine Menge von autonomen Rechnern (auch Knoten genannt) eine Einigung auf einen bzw. mehrere Werte erzielen sollen. Beispielsweise können bei kooperierenden mobilen Robotern mit Hilfe von Übereinstimmungsverfahren Aufgaben und Gebiete des Arbeitsraums zwischen den Robotern aufgeteilt werden, wodurch das System als Ganzes schneller, skalierbarer und robuster wird (Qu et al. 2012; Agmon und Peleg 2006; Santoro 2006). Sofern keine Fehler im verteilten System auftreten, ist das Problem vergleichsweise trivial lösbar: Die beteiligten Knoten senden sich jeweils ihre lokale Sicht zu einem bestimmten Zeitpunkt gegenseitig zu. Nach dem Austausch der Sichten verfügen alle Knoten über den gleichen Vektor von Sichten und können daraus nach dem gleichen deterministischen Algorithmus die gleiche gemeinsame Sicht ableiten. Treten jedoch Fehler im verteilten System auf, deren Ursprung von irgendeiner internen oder externen Quelle herrührt,

so wird das Lösen des Übereinstimmungsproblems alles andere als einfach. Das Übereinstimmungsproblem wird durch folgende Fehler erschwert:

Kanalfehler. Ein Kanal ist fehlerhaft, d. h. die zu übertragenden Nachrichten können verloren gehen bzw. inhaltlich verfälscht werden (siehe Näheres in Abschnitt 2.2.2). Wenn ein Broadcast-Kanal eine Nachricht an mehrere Empfänger senden soll, kann dies im Fehlerfall nur unvollständig erfolgen, so dass nur eine echte Teilmenge der Empfänger die Nachricht erhält (Dies ist ein Byzantinischer Fehler). Analog könnten sich Inhaltsverfälschungen auf unterschiedliche Empfänger in verschiedener Weise auswirken.

Knotenfehler. Ein Knoten wird durch interne oder externe Einflüsse (z.B. Hitze, EMV, radioaktive Einstrahlungen, etc.) in einen fehlerhaften Zustand überführt. Je nach Fehlerzustand und Fehlfunktion können fehlerhafte Knoten einen negativen Einfluss auf den Übereinstimmungsprozess ausüben (z.B. verhindern, dass die Knoten sich auf einen gemeinsamen Wert einigen). Beispielsweise kann er zu einem „babbling idiot“ (kurz: Babbling bzw. Babbling-Fehler) (Temple 1998) werden. Bei einem Babbling-Fehler sendet der betreffende Knoten zu beliebigen Zeitpunkten unkontrolliert Nachrichten zu einem oder mehreren Empfängern. Dies wiederum kann zu Kollisionen und Verwechslungen mit anderen Nachrichten führen, sofern für die Kommunikation ein gemeinsamer Kanal („**shared medium**“) verwendet wird. Auch Knoten können Byzantinische Fehler verursachen, indem sie eine Multicast-Nachricht nur an eine Teilmenge der Empfänger senden oder sogar den Empfängern mit unterschiedliche Nachrichteninhalte zukommen lassen. Jede Fehlinformation kann einen Empfänger von der korrekten Entscheidung abbringen. Bei Punkt-zu-Punkt-Verbindungen lässt sich im einfachsten Fall die Verbindung zum fehlerhaften Knoten deaktivieren, was im Falle von drahtlosen Netzen nicht gegeben ist. Demzufolge müssen Maßnahmen ergriffen werden, um die Zuverlässigkeit des Systems gegenüber solchen Angriffen nicht zu verschlechtern. Um auch im Fehlerfall eine Übereinstimmung unter den fehlerfreien Knoten sicherzustellen, sind Fehlertoleranzverfahren anzuwenden. Dies bedeutet, dass fehlertolerante Übereinstimmungsprotokolle zu entwerfen sind.

f_N bezeichnet die Anzahl der zu tolerierenden Knotenfehler und f_B die Anzahl der zu tolerierenden Blockierungsfehler (siehe ggf. Abschnitt 2.3.7).

Unter dem Begriff Zuverlässigkeit versteht man „*die Fähigkeit eines Systems, während einer vorgegebenen Zeitdauer bei zulässigen Betriebsbedingungen die spezifizierte Funktion zu erbringen*“ (Echtle 1990a), wobei die zulässigen Betriebsbedingungen durch das Systemmodell spezifiziert werden (siehe Abschnitt 2.2). Ein Systemmodell ist (vereinfacht ausgedrückt) eine vereinfachende Abbildung der Realität, auf die sowohl Probleme abgebildet als auch Lösungsstrategien angewendet werden können. Ein Systemmodell kann abhängig vom zu untersuchenden Problem das zugrundeliegende reale System sehr abstrakt (durch weglassen von Details) oder sehr fein granular wiedergeben.

Unter den Begriff Fehlertoleranz versteht man „*die Fähigkeit eines Systems, auch mit einer begrenzten Anzahl fehlerhafter Komponenten seine spezifizierte Funktion zu erfüllen*“ (Echtle 1990a).

2.2 Das Übereinstimmungsproblem

2.2.1 Problemstellung

Im Allgemeinen geht man beim Übereinstimmungsproblem davon aus, dass das verteilte System aus einer bekannten Knotenmenge N = {N[0], N[1], …, N[n – 1]} zusammengesetzt ist, in der Knoten ausschließlich über Nachrichtenaustausch eine Einigung auf einen Wert oder eine Liste von Werten erzielen müssen. Dieses Einigen auf einen Wert erfolgt ohne Mitwirkung einer zentralen Instanz. Um Übereinstimmung zu erzielen bzw. das Übereinstimmungsproblem zu lösen, müssen Nachrichten nicht an eine zentrale Instanz übertragen werden. Weiterhin nimmt man an, dass der Parameter f_N, die maximale Anzahl der fehlerhaften Knoten, im verteilten System bekannt ist. Ein Knoten wird genau dann als fehlerfrei eingestuft wird, wenn er sich gemäß Protokollspezifikation verhält. In allen anderen Fällen gilt ein Knoten als fehlerhaft.

Lösungen des Übereinstimmungsproblems setzen sich aus drei Schritten zusammen. Bevor die einzelnen Schritte im Detail besprochen werden, werden zunächst die Begriffe **Runde** und **Phase** näher definiert.

Definition 2.1 (Runde)
Eine **Runde** entspricht einem Zeitintervall, innerhalb dessen jeder Knoten genau eine Nachricht zu allen anderen Knoten senden kann. Zusätzlich können Knoten innerhalb einer Runde Nachrichten von anderen Knoten empfangen und interne Berechnungen durchführen, die vom momentanen Zustand des Knotens abhängen. In der Regel (d.h. für sehr viele Übereinstimmungs-Protokolle) gilt für die Anzahl der notwendigen Runden der Wert $\boldsymbol{f_N + 1}$.

Definition 2.2 (Phase)
Wenn alle Nachrichten einer **Runde** nebenläufig gesendet werden, dann wird anstatt **Runde** auch der Begriff **Phase** gebraucht. In der Regel (d.h. für sehr viele Übereinstimmungs-Protokolle) gilt für die Anzahl der notwendigen Phasen meist $\boldsymbol{f_N + 1}$.

*Beispiel zu **Definition 2.2** (Phase)*
Wenn ein System beispielsweise aus 5 Knoten besteht, die alle innerhalb einer Runde eine Broadcast-Nachricht senden sollen, dann besteht diese Runde aus 5 Phasen.

Im ersten Schritt werden die Elemente des Konsistenzvektors, über den eine Einigung erzielt werden soll, von den sogenannten Quellknoten als „Wertvorschlag“ an die beteiligten Knoten verteilt. Je nach Problemklasse (siehe Beschreibung zu ICP, CP und BAP auf nachfolgender Seite) besteht der Konsistenzvektor aus einem oder mehreren Elementen. Im erstgenannten Fall gibt es in der Regel genau einen Quellknoten pro Vektorelement. Im zweiten Schritt, der den Hauptteil der Protokollausführung bildet, werden mittels Nachrichtenaustausch die Elemente in einer oder mehreren Runden zwischen den Knoten ausgetauscht.

Im dritten und letzten Schritt terminieren die Knoten je nach Problemklasse mit einem einelementigen Konsistenzvektor (enthält genau einen Endwert) bzw. mit einem Vektor von Endwerten, wenn alle fehlerfreien Knoten sich auf diesen geeinigt haben. Eine Einigung muss die folgenden beiden Bedingungen erfüllen (auch als interaktive Konsistenz bzw. „interactive consistency“ IC bekannt) (Dolev und Strong 1983):

- **IC1**: Alle fehlerfreien Knoten terminieren bzw. einigen sich auf dem gleichen Konsistenzvektor (Endwert bzw. Vektor von Endwerten).
- **IC2**: Wenn der Quellknoten fehlerfrei ist, dann enthält der Konsistenzvektor, auf den sich die fehlerfreien Knoten einigen, den Wert des Quellknotens als Vektorelement.

Eine der wichtigsten Kenngrößen in Übereinstimmungsprotokollen, die auch die Zuverlässigkeit des Systems beeinflusst, ist die Anzahl $f = f_N + f_B$ der zu tolerierenden Komponentenfehler (Knoten- und Kanalfehler bzw. knoten- und kanalbedingte Nachrichtenausfälle). Der Parameter f_B gibt die Anzahl der fehlerfreien Sendeknoten an, deren Nachrichtenübertragung während einer Runde durch den Kanal beeinträchtigt wird (weitere Informationen hierzu siehe in Abschnitt 2.3.7). Eine Beeinträchtigung liegt genau dann vor, wenn die Nachricht des fehlerfreien Sendeknotens von mindestens einem fehlerfreien Empfängerknoten nicht unverfälscht empfangen wird (siehe Abbildung 3).

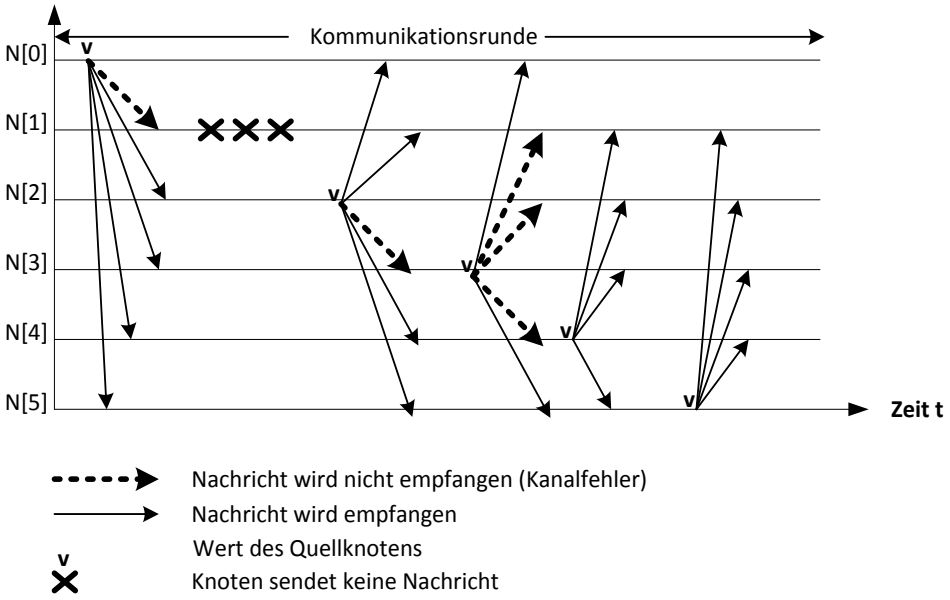


Abbildung 3: Übereinstimmungsproblem im Falle eines fehlhaften Kanals mit N = {N[0], N[1], N[2], N[3], N[4] , N[5]}. Hier gilt $f_B = 3$.

Übereinstimmungsprotokolle sind so konzipiert, dass mit dem Parameter f auch der Grad der Redundanz, d. h. die Mindestanzahl der teilnehmenden Knoten und gegebenenfalls die Anzahl der redundanten Verbindungen festgelegt wird. Wenn zu viele Fehler auftreten und f überschritten wird, werden in der Regel die beiden Bedingungen IC1 und IC2 verletzt.

Abhängig davon, aus wie vielen Elementen der Konsistenzvektor besteht, unterscheidet man beim Übereinstimmungsproblem zwischen den folgenden Problemklassen:

- **ICP** – Interactive Consistency Problem (Problem der interaktiven Konsistenz): Alle fehlerfreien Knoten müssen sich auf einen gemeinsamen Konsistenzvektor einigen, der

mindestens zwei Vektor-Elemente enthält (pro Quellknoten genau 1 Vektorelement, das den Anfangswert, d.h. die „anfängliche Sicht“ des jeweiligen Knotens repräsentiert).

- **BAP** – Byzantine Agreement Problem (Problem der Byzantinischen Übereinstimmung): Alle fehlerfreien Knoten müssen sich auf einen einelementigen Konsistenzvektor einigen, welcher von einem vorher festgelegten Knoten, dem Quellknoten, verteilt wird. Die Wahl des Quellknotens im System kann sich für jede Ausführung des Übereinstimmungsprotokolls ändern (muss es aber nicht).
- **CP** – Consensus Problem (Konsens-Problem): Alle Knoten müssen sich auf einen einzigen gemeinsamen Endwert (einelementiger Konsistenzvektor) einigen, wobei jeder Knoten als Quellknoten im System agiert. Im Vergleich zu ICP lässt sich das Problem ohne Bildung eines Konsistenzvektors lösen.
- **k-Konsens**: Beim k-Konsens müssen nicht alle fehlerfreien Knoten einen gemeinsamen Konsens erzielen. Im Vergleich zu CP reicht es aus, wenn von den n Knoten eine Mehrheit (mindestens $k > \lceil n/2 \rceil$ und $k > f_N$) der (fehlerfreien) Knoten sich auf einen einelementigen Konsistenzvektor einigen. Somit stellt der k-Konsens eine abgeschwächte Variante des CPs dar.

Das Konsens-Problem zählt zu den einfachsten zu lösenden Problemklassen. Das Problem der Byzantinischen Übereinstimmung ist von der Komplexität identisch zum Problem der interaktiven Konsistenz, so dass ICP unter Verwendung eines BAP-Protokolls gelöst werden kann (indem die BAP-Protokolle für jeden Quellknoten nebenläufig zueinander ausgeführt werden). Daher reicht es im Allgemeinen aus, eine Lösung für BAP zu entwickeln.

Das Übereinstimmungsproblem betrifft n Knoten N[0], … , N[n – 1]. Von diesen Knoten bildet eine Teilmenge von q Knoten Q[0], … , Q[q – 1] die Menge der Quellknoten, wobei {Q[0], … , Q[q – 1]} ⊆ { N[0], … , N[n – 1]} und 1 ≤ q ≤ n und eine injektive Abbildung $\hat{q}$: {0, … , q – 1} → {0, … , n – 1} jedem Quellknoten-Index den entsprechenden Knoten-Index zuordnet. Jeder Quellknoten Q[i] verfügt über einen Anfangswert A[i], der seine Sicht repräsentiert und den er in die Ausführung des Übereinstimmungsprotokolls einbringt. Ein Anfangswert ist ein Element eines gegebenen Wertebereichs W, das heißt A[i] ∈ W. Nach Ausführung des Übereinstimmungsprotokolls soll jeder der Knoten N[i], falls er fehlerfrei ist, über ein Ergebnis K[i] verfügen, das in der Regel ein Konsistenzvektor ist. Für die genannten Problemklassen gelten die folgenden Beziehungen zwischen den Parametern:

- Für ICP: Alle Knoten sind Quellknoten, das heißt q = n. Das Ergebnis K[i] eines fehlerfreien Knotens N[i] ist ein Konsistenzvektor K[i] = (K[i][0], … , K[i][n – 1]), wobei jedes Vektorelement K[i][j] den Anfangswert A[j] eines Knotens repräsentiert, D. h. K[i][j] = A[j], falls der entsprechende Quellknoten N[j] fehlerfrei ist (um IC2 zu erfüllen). Für zwei fehlerfreie Knoten N[i] und N[j] gilt die Übereinstimmung K[i] = K[j] (um IC1 zu erfüllen).
- Für BAP: Nur ein Knoten Q[0] ist Quellknoten, das heißt q = 1. Das Ergebnis K[i] eines fehlerfreien Knotens N[i] ist der Anfangswert A[N[0]] des Quellknotens Q[0], das heißt K[0] = A[N[0]], falls dieser fehlerfrei ist (um IC2 zu erfüllen). Für zwei fehlerfreie Knoten N[i] und N[j] gilt die Übereinstimmung K[i] = K[j] (um IC1 zu erfüllen).
- Für CP: Alle Knoten sind Quellknoten, das heißt q = n. Das Ergebnis K[i] eines fehlerfreien Knotens N[i] ist ein Element des Funktionswerts C(A[0], … , A[n – 1]), d. h. K[i] $\in$ C(A[0], … , A[n – 1]), einer vorgegebenen Funktion C : $W^n \rightarrow 2^W$, wobei der Anfangswert eines fehlerhaften Knoten entsprechend der nachfolgenden Beschreibung gebildet wird. Da C in die Potenzmenge von W abbildet, besteht oft ein Spielraum, welches Ergebnis ausgewählt wird. Jedoch müssen sich alle fehlerfreien Knoten diesbezüglich gleich entscheiden: Für N[i] und N[j] muss K[i] = K[j] gelten (um IC1 zu erfüllen). Je nach Funktion C gilt nur eine abgeschwächte Form von IC2 (fehlerfreie Knoten üben einen Einfluss auf den Funktionswert aus, aber der Funktionswert ist nicht unbedingt gleich ihrem Anfangswert). Ein Beispiel für C ist C(A[0], … , A[n – 1]) = [Min(A[i],…, A[j]), Max(A[i],…, A[j])], wobei Min(A[i],…, A[j]) der kleinste und Max(A[i],…, A[j]) der größte Anfangswert ist, der von einem fehlerfreien Quellknoten stammt. C besagt in diesem Beispiel, dass der Endwert im Bereich der fehlerfreien Anfangswerte liegen muss.

Je nach Fehlfunktion eines fehlerhaften Quellknotens Q[x] ist es möglich, dass sich die fehlerfreien Knoten im Laufe der Protokollausführung entweder auf einen Anfangswert A[N[x]] dieses fehlerhaften Quellknotens Q[x] einigen oder dafür einen Ersatzwert (Default-Wert) D einsetzen (insbesondere wenn Q[x] überhaupt nicht sendet), das heißt A[N[x]] = D, wobei D eine globale Konstante ist. Natürlich müssen alle fehlerfreien Knoten diesbezüglich die gleiche Entscheidung treffen, was letztlich eine hohe Anforderung an das Übereinstimmungsprotokoll darstellt. Auch wenn ein fehlerhafter Quellknoten Q verschiedene

Anfangswerte aussendet, müssen sich alle fehlerfreien Knoten auf einen einelementigen Konsistenzvektor einigen.

Die drei genannten Übereinstimmungs-Varianten ICP, BAP und CP lassen sich ordnen:

- Wenn eine Lösung für BAP existiert, dann lässt sich damit ICP lösen, indem für jeden Quellknoten nebenläufig BAP ausgeführt wird und der Konsistenzvektor aus den BAP-Ergebnissen zusammengesetzt wird.
- Wenn eine Lösung für ICP existiert, dann lässt sich damit CP lösen, indem ein Konsistenzvektor gebildet wird und jeder Knoten die Funktion C lokal auf diesen anwendet, um sein Ergebnis zu erhalten.

Grundsätzlich kommt es also darauf an, ein Übereinstimmungsprotokoll zu finden, das BAP löst.

2.2.2 Unmöglichkeitsaussagen zum Übereinstimmungsproblem

Das Übereinstimmungsproblem stellt ein fundamentales Problem in verteilten Systemen dar. Auch wenn seine Definition auf den ersten Blick einfach erscheinen mag, ist die Lösung alles andere als trivial. Um das Problem der Übereinstimmung in verteilten Systemen zu lösen, gibt es sowohl deterministische (wie etwa „Signed Message“ (Lamport et al. 1982)) als auch probabilistische Lösungsansätze (wie etwa Turquois (Moniz et al. 2013)). Die Komplexität der Lösung, falls es eine solche gibt (Santoro und Widmayer 2007), wird maßgeblich durch das Fehlermodell und das Timing-Modell bestimmt. Zusammen mit der Systemstruktur (siehe ggf. Abschnitt 2.3) bilden sie das Systemmodell. Das Timing-Modell beschreibt sowohl Annahmen über das Zeitverhalten von Knoten als auch Anforderungen an das Zeitverhalten des Übereinstimmungsprotokolls insgesamt.

Eine Lösung des Übereinstimmungsproblems erfüllt die genannten Anforderungen (siehe Abschnitt 2.2), wenn im betrachteten Systemmodell kein Fehlerszenario existiert, das zu einer Verletzung der beiden Bedingungen IC1 oder IC2 führt. Die Komplexität einer Lösung für das Übereinstimmungsproblem wird wesentlich durch zwei Aspekte beeinflusst, nämlich:

- Eigenschaften des Systemmodells (wie bereits erwähnt):
  - Welches Fehlermodell liegt zugrunde?

  - Welchen Timing-Model liegt zugrunde?
- Wie effizient soll/muss die Lösung in Bezug auf Kommunikations-, Speicher- und Knoten-Overhead sein?
  - geforderte Reduzierung der Anzahl der übertragenen Nachrichten
  - geforderte Reduzierung der Anzahl der in einem Knoten während der Protokollausführung zu speichernden Nachrichten
  - geforderte Reduzierung der Anzahl der redundanten Knoten

Ob eine deterministische Lösung (Fischer et al. 1985b; Dolev et al. 1987a) zum Übereinstimmungsproblem existiert, hängt ganz davon ab, welche Anforderungen durch das Systemmodell formuliert worden sind. So hat bereits (Fischer et al. 1985a) gezeigt, dass keine deterministische Lösung zum CP in einem asynchronen System existiert – selbst dann nicht, wenn nur ein Knoten durch die primitive und leicht zu erkennende Fehlfunktion des Anhaltens ausfällt.

Im folgenden Abschnitt werden wir das Systemmodell von Santoro und Widmayer (Santoro und Widmayer 1989, 2007) wiedergeben, da dieses Modell maßgeblichen Einfluss auf das Systemmodell der vorliegenden Arbeit hat.

2.2.3 Das Systemmodell von Santoro-Widmayer

Das Timing-Modell bezieht sich auf ein synchrones verteiltes System und das Kanalfehler-Modell geht von einem unzuverlässigen Kanal aus. Das System besteht aus n Knoten (die jeweils einen Prozess darstellen) N = {N[0], N[2],…, N[n – 1]}, die über Nachrichtenaustausch einen k-Konsens ($k > \lceil n/2 \rceil$, siehe S.14) über einen binären Wert A ϵ {0,1} erzielen sollen. Der Unmöglichkeitsbeweis sagt aus, dass es keinen endlichen deterministischen Algorithmus gibt, wenn in einem System bestehend aus n fehlerfreien Knoten innerhalb einer Phase mehr als n – 2 Übertragungsfehler (Punkt-zu-Punkt-Nachrichten) auftreten, wobei innerhalb einer Phase die Knoten ihre Nachrichten an jeden anderen Knoten versenden (insgesamt gibt es n · (n – 1) Nachrichtenübertragungen bzw. Punkt-zu-Punkt-Nachrichten pro Phase). Der Unmöglichkeitsbeweis von Santoro-Widmayer besagt weiter, dass das Übereinstimmungsproblem für synchrone verteilte Systeme nicht lösbar ist, wenn die Anzahl der Übertragungsfehler je nach Fehlerannahme eine bestimmte Grenze überschreitet (Santoro und Widmayer 2007). Das Besondere an dem Modell von

Santoro-Widmayer, das Kommunikationsfehlermodell genannt wird (Santoro und Widmayer 1989), ist darin zu sehen, dass im Gegensatz zu dem Komponentenfehlermodell (Warns 2010), die dynamisch auftretenden Nachrichtenfehler gezählt werden. Dies weicht von der üblichen Betrachtung ab, wo nur die fehlerhaften Knoten gezählt werden, die sich beliebig oft falsch verhalten können. Anders ausgedrückt: die Einzelfehlerbereiche (Echtle 1990b), die einen Fehler aufweisen können, können sich in Santoro-Widmayer-Modell von einem Zeitpunkt bzw. Zeitintervall zum anderen sprunghaft ändern (siehe auch 2.3.5). Außerdem werden die Fehler nicht auf einen fehlerhaften sendenden bzw. fehlerhaft empfangenden Knoten zurückgeführt. Vielmehr wird die Fehlerursache im Kanal selbst angenommen.

Viele Arbeiten, die sich mit dem Übereinstimmungsproblem in verteilten Systemen beschäftigt haben, gehen von der Annahme aus, dass sich die Einzelfehlerbereiche, die einen Fehler aufweisen, während der Ausführung eines Übereinstimmungsprotokolls nicht ändern. Die Einzelfehlerbereiche, die einen Fehler aufweisen, sind statisch. Diese Art von Fehlermodell wird auch als Komponentenfehlermodell („**component failure model**“) bezeichnet (Warns 2010). Man beachte, dass das Komponentenfehlermodell auch den Fall abdeckt, dass ein Fehler erst während der Protokollausführung auftritt. In diesem Fall zählt der betreffende Einzelfehlerbereich während der gesamten Protokollausführung als fehlerhaft. Ohnehin darf sich jeder fehlerhafte Einzelfehlerbereich jederzeit (zufällig) korrekt verhalten. Im Gegensatz zum Komponentenfehlermodell nach (Warns 2010) schließt das Fehlermodell nach (Buhrman et al.; Baron et al.) nicht aus, dass sich die Anzahl der zu tolerierenden Einzelfehlerbreiche während der gesamten Protokollausführung ändern kann. D. h. im Falle einer f_B-Fehlerannahme können nach dem Fehlermodell von (Buhrman et al.; Baron et al.) während der gesamten Protokollausführung in der Summe mehr als f Einzelfehlerbereiche ein fehlerhaftes Verhalten ausweisen, jedoch dürfen zu einem bestimmten Zeitpunkt bzw. in einem bestimmten Zeitintervall (siehe ggf. (Buhrman et al. 1995)) höchstens f Einzelfehlerbereiche gleichzeitig einen Fehler aufweisen.

Im Rahmen dieser Arbeit wird ein hybrider Ansatz verwendet, welcher ein ähnliches (dynamisches) Modell wie Santoro und Widmayer verwendet (siehe auch Abschnitt 2.3.5). Ein wesentlicher Unterschied besteht allerdings darin, dass nicht alle Fehler auf einen fehlerhaften Kommunikationskanal zurückgeführt werden. Stattdessen wird angenommen, dass sowohl Knotenfehler als auch Nachrichtenausfälle (der Zeitpunkt und die betroffenen Knoten sind unbekannt) auftreten können und ihre Anzahl nach oben beschränkt ist.

Gemäß dem Unmöglichkeitsergebnis von Santoro und Widmayer lässt sich schlussfolgern, dass bei der Entwicklung von Übereinstimmungsprotokollen stets eine Lösung für den Fehlertyp Babbling (Temple 1998) gefunden werden muss, sofern deterministische Lösung verfolgt werden. Beim Fehlertyp Babbling wird die Kommunikation zwischen fehlerfreien Knoten von mindestens einem fehlerhaften Knoten gestört. Die Störung kann Beispielsweise durch das Senden von Nachrichten zu willkürlichen nicht synchronisierten Zeitpunkten hervorgerufen werden (was zu Nachrichtenausfällen durch Nachrichtenkollision führen kann). Im Allgemeinen sind weder die Dauer noch die Frequenz von Babbling-Fehlern bekannt. Außerdem muss man davon ausgehen, dass die zusätzlichen Nachrichten und ggf. die Nachrichtenausfälle von den Knoten nicht in global konsistenter Weise von den Knoten erkannt werden. Vielmehr gewinnt jeder Knoten seine eigene Sicht durch die von ihm empfangenen Nachrichten. Daher ist das Babbling-Problem ohne Gegenmaßnahmen ein für Übereinstimmungsprotokolle nicht lösbares Problem (Es gibt keine deterministische Lösung). Stattdessen muss Babbling durch spezielle Maßnahmen verhindert oder zumindest eingeschränkt werden. Letztlich muss sich ein Kommunikationskanal „gegen Babbling-Sender zur Wehr setzen“.

2.2.4 Babbling-Fehler und Gegenmaßnahmen

Beim Erstellen des Fehlermodells gibt es zwei Möglichkeiten, wie man mit Babbling-Fehlern umgeht. Ein einfacher, aber für sicherheitskritische Anwendungen nicht akzeptabler Weg wäre das Ausschließen von Babbling-Fehlern. In diesem Fall sollte mindestens ein Nachweis erbracht werden, der zeigt, dass die Auftrittswahrscheinlichkeit eines solchen Fehlers für die Anwendung vernachlässigbar gering ist oder aufgrund der Topologie des Kommunikationssystems vernachlässigt werden kann. Beispielsweise ist ein solcher Fehler vernachlässigbar, wenn die Kommunikation über Punkt-zu-Punkt („**point-to-point**“) statt über ein „**shared medium**“ erfolgt. In diesem Fall kann ein Babbling-Knoten die Kommunikation zwischen fehlerfreien Knoten nicht beeinträchtigen.

Eine zweite bessere Alternative ist der Einsatz so genannter Busguardians, der im Fall von „shared medium“ einsetzbar ist. Der Busguardian realisiert einen Regelmechanismus, der den Zugriff der Knoten in einem verteilten System auf den geteilten Kanal (Übertragungsmedium) regelt (Rausch 2008). Dem Busguardian selbst werden keine Mechanismen für den Zugriff auf dem Kanal bereitgestellt. D. h. ein Busguardian verfügt nur über die Möglichkeit, den

Zugriff eines Knotens auf den gemeinsamen Kanal zu unterbinden bzw. diesen freizugeben (Rausch 2008).

Durch den Einsatz eines Busguardians verhindert man, dass ein Knoten zu unzulässigen Zeitpunkten eine Nachricht sendet (was zu Kollisionen auf dem gemeinsamen Kanal führen könnte) oder durch eine Überflutung mit Nachrichten den gemeinsamen Kanal lahmlegt. Durch den Einsatz von Busguardians wird das Übereinstimmungsproblem in verteilten Systemen wieder lösbar, sofern das Kanalfehler-Modell nicht im Widerspruch zu den Unmöglichkeitsergebnis von Santoro und Widmayer steht.

Einfehlerannahme. Es sei der Vollständigkeitshalbe erwähnt, dass dem Busguardian-Ansatz üblicherweise eine „Einfehlerannahme“ zugrunde liegt. Man nimmt an, dass zu keinen Zeitpunkt ein Zustand im System existiert (bzw. die Wahrscheinlichkeit dafür vernachlässigbar gering ist), bei dem beide Komponenten (der Busguardian und der vom Busguardian überwachte Knoten) gleichzeitig fehlerhaft sind. Im Extremfall könnte man mehrere Busguardians pro Knoten verwenden, um auch Mehrfachfehler zu tolerieren.

2.3 Systemmodell

Ein Systemmodell ist nützlich, wenn es das jeweils betrachtete Problem und die Umgebung, in der das System agiert, adäquat wiedergibt. Das Systemmodell umfasst die drei Teilaspekte Timing, Fehler und Systemstruktur (sprich: die Knoten, die Kommunikationskanäle bzw. Kommunikationstopologie, usw.). Das Timing-Modell beschreibt das zeitliche Verhalten von Vorgängen, die im System ablaufen, wie etwa die Übertragungs einer Nachricht (Nachrichtenübertragungsdauer) oder der Zeitpunkt, wann eine Nachrichtübertragung erwartet wird. Das Fehlermodell beschreibt die Fehler, die im System auftreten können bzw. berücksichtigt werden müssen, weil sie zu tolerierenden sind. Die Systemstruktur beschreibt die Gesamtheit der Komponenten eines Systems sowie deren Beziehungen untereinander.

Da sowohl das Fehlermodell als auch das Timing-Modell einen wesentlichen Beitrag zum Verständnis der vorliegenden Arbeit liefern, werden im Folgenden beide Modell definiert und soweit erläutert, wie es zum Verständnis notwendig ist.

2.3.1 Timing-Model

Das Timing-Modell legt die für die Kommunikation wichtigen zeitlichen Kenngrößen bzw. obere Schranken dafür fest, wie etwa die Ausführungsdauer t_M eines Nachrichtentransfers oder die Dauer t_R der Reaktion auf eine empfangene Nachricht.

Zur Charakterisierung verteilter Echtzeitsysteme ist es auch wichtig, dass das Timing-Modell Informationen über das zeitliche Verhalten der Knoten und Kommunikationsverbindungen liefert, wobei insbesondere die Art der Synchronität genannt werden muss. Das Timing-Modell unterscheidet diesbezüglich synchrone, teilsynchrone und asynchrone Kommunikation. Wenn alle Kommunikationen auf die gleiche Weise durchgeführt werden, spricht man auch von einem synchronen, teilsynchronen bzw. asynchronen System.

Im Folgenden wird für jeden dieser drei Fälle eine kurze Erläuterung gegeben.

2.3.2 Synchrone Kommunikation

Bei synchroner Kommunikation sind die Übertragungsrate und die Verarbeitungsgeschwindigkeit der teilnehmenden Knoten bekannt, D. h. es gibt eine bekannte obere Schranke, die von keinem teilnehmenden Knoten verletzt wird, sofern er fehlerfrei ist. Nach (Cachin et al. 2011) muss ein synchrones (verteiltes) System die folgenden Eigenschaften erfüllen:

- Jeder Rechenschritt eines Knoten erfolgt in endlicher Zeit mit bekannter oberer Zeitschranke.
- Die Nachrichtenübertragungszeit ist durch eine bekannte obere Schranke begrenzt.
- Jeder Knoten verfügt über eine lokale Uhr mit einer Drift, die durch bekannte Schranken nach oben und unten begrenzt ist („**real-time**“). Dabei besagt die Drift einer Uhr wie stark die Ableitung der Uhrzeit nach der Echtzeit von dem Idealwert 1 Uhrzeitsekunde/Echtzeitsekunde abweicht.

Diese Eigenschaften, die ein synchrones verteiltes System erfüllt, können dazu verwendet werden, um Fehler frühzeitig zu erkennen bzw. Fehlertypen auszuschließen. Beispielsweise lässt sich mittels eines Timeouts feststellen, ob ein Knoten ausgefallen ist. Durch den Einsatz geeigneter Mechanismen lässt sich sicherstellen, dass ein Knoten nur zu festgelegten Zeitpunkten eine Sendeaktion ausführt, was im Falle von drahtloser Kommunikation in

Kombination mit Busguardian-Mechanismen eine kollisionsfreie Nachrichtenübertragung erlaubt, auch wenn einzelne Knoten im System fehlerhaft agieren.

In einem synchronen System ist es möglich, den gemeinsamen Kanal unter den Knoten aufzuteilen, so dass jeder Knoten den Zeitpunkt kennt, wann eine Sendeoperation zu erfolgen hat bzw. ein Nachrichtenempfang zu erwarten ist.

2.3.3 Asynchrone Kommunikation

Bei einem asynchronen verteilten System fehlen jegliche Annahmen über das Zeitverhalten von Knoten und Verbindungen. Es liegen somit keine Informationen über die Verarbeitungszeit und Nachrichtenübertragungszeit vor. Außerdem wird keine Annahme getroffen, ob Knoten Zugriff auf eine lokale bzw. globale Uhr besitzen. Falls überhaupt Uhren Verwendung finden, ist deren Drift unbekannt.

Die Konsequenz ist, dass das Übereinstimmungsproblem für BAP, ICP, CP nicht lösbar ist, genauer gesagt, gibt es keine deterministische Lösung (siehe Unmöglichkeitsbeweis von (Fischer et al. 1985a; Santoro und Widmayer 2007)). Der Grund hierfür ist, dass es in einem asynchronen System unmöglich ist, zwischen einen langsamen und einem ausgefallenen Knoten zu unterscheiden. Im Fall von BA (analog zu ICP) würde dies bedeuten, dass die Knoten bei einem ausgefallenen Quellknoten nicht terminieren können, außer man hat Zugriff auf sogenannte Ausfalldetektoren (siehe S.23, „Ausfalldetektoren“).

Ausnahmen bilden Übereinstimmungsprobleme, bei denen das CP gelöst werden soll. Obwohl (Fischer et al. 1985a) mit ihrem Beweis gezeigt haben, dass das CP in asynchronen System nicht lösbar ist, gibt es dennoch „Lösungswege“, wie man diesen Beweis umgehen kann. Diese Ansätze, die zusätzliche Annahmen benötigen oder ein eingeschränktes Terminierungsverhalten aufweisen, werden im Folgenden näher erläutert.

Randomisierung

Eine weitere Methode benutzt Randomisierung. Bei diesem Ansatz haben die Knoten Zugriff auf eine Münzwurf-Funktion, die einen Zufallswert mit einer bestimmten Verteilung als Ergebnis zurückliefert. Das soll verhindern, dass die Knoten in einem dauerhaft nicht terminierenden Zustand z verharren. Die Grundidee der Wirkungsweise ist die folgende: Sobald im Laufe der Protokollausführung ein solcher nicht terminierender Zustand z erreicht wird,

treffen die Knoten lokal mit Hilfe einer Münzwurf-Funktion eine indeterministische Entscheidung bzgl. des Konsistenzvektors, um den Zustand z verlassen. Anschließend wird ein neuer Durchlauf initiiert. Die Wahrscheinlichkeit, dass die Knoten wieder in einen solchen Zustand z gelangen, ist kleiner als 1. Damit nimmt die Wahrscheinlichkeit, dass jede Runde im Zustand z endet, mit steigender Rundenzahl exponentiell ab und konvergiert gegen den Wert 0 (siehe (Fischer et al. 1985a)).

Ausfalldetektoren

Die Annahme von Ausfalldetektoren besagt, dass jeder Knoten über einen Ausfalldetektor verfügt, d. h. über einen Mechanismen zum Aufspüren von ausgefallenen Knoten. Im Falle eines Knotenausfalls informiert der Ausfalldetektor alle anderen Knoten im System. Der Zugriff auf den Ausfalldetektor ist aus Sicht des jeweiligen Knotens zu jeder Zeit möglich. Bei jedem Zugriff liefert der Ausfalldetektor als Ergebnis eine Liste (potentiell) ausgefallener Komponenten zurück (genaueres siehe (Chandra und Toueg 1996)). Wenn (unrealisitischerweise) man annimmt, dass ein Ausfalldetektor jeden Ausfall sofort erkennt, ist die Lösung des Übereinstimmungsproblems sehr einfach. Wie man dagegen das CP auch mit unzuverlässigen Ausfalldetektoren lösen kann, wurde erstmalig von (Chandra und Toueg 1996) vorgestellt.

Annahmen über das Zeitverhalten

Ein Ansatz den Unmöglichkeitsbeweis zu umgehen, besteht darin, das Timing-Modell des Systems an bestimmten Stellen zu ändern bzw. zu erweitern. (Dolev et al. 1987b) zeigten in ihrer Arbeit, dass durch Hinzufügen eingeschränkter Synchronisationsannahmen bzgl. des Nachrichtenaustauschs das CP wieder lösbar ist. Jedoch wird durch die Änderung am Timing-Modell (streng genommen) eine Lösung für ein teilsynchrones, nicht aber für ein asynchrones System geliefert.

2.3.4 Teilsynchrone Kommunikation

Man spricht von teilsynchroner Kommunikation, wenn ein Zeitabschnitt existiert, in dem sich das System wie ein synchrones System verhält. Es wird also eine Annahme getroffen, die besagt, dass das System in irgendeinem Zeitintervall synchron arbeitet, ohne aber das Zeitintervall genau nennen zu müssen. Folglich gibt es in einem teilsynchronen System durchaus Zeitabschnitte, in denen sich das System asynchron verhält bzw. die Annahmen über die Verarbeitungsgeschwindigkeit und/oder Nachrichtenübertragungsdauern nicht erfüllt sind.

Asynchronität kann durch eine Netzwerk-Überlastung oder Speicherknappheit der Knoten ausgelöst werden. Ein hohes Nachrichtenaufkommen kann beispielsweise in einem Knoten zum Überlauf von Pufferspeicher führen, so dass Nachrichten verloren gehen, die wiederum neu übertragen werden müssen.

Ein teilsynchrones System liegt vor, wenn mindestens einer der folgenden Punkte erfüllt ist:

- Die Verarbeitungsgeschwindigkeit der Knoten im System und die Nachrichtenübertragungsdauer sind nach oben beschränkt, aber die Schranke ist unbekannt.
- Die oberen Schranken sind bekannt, aber werden nur zeitweise erfüllt.

2.3.5 Fehlermodell

In einem Fehlermodell werden die möglichen Fehlertypen, die in einem System auftreten können, definiert (Lynch 2010; Cachin et al. 2011). Ein Fehler tritt im System genau dann auf, wenn eine Komponente (Knoten, Kanal, etc.) des Systems eine Abweichung vom korrekten Verhalten aufweist. Neben der Angabe, welche Komponenten eines Systems durch Fehler betroffen sind, muss zusätzlich auch die Fehlfunktion beschrieben werden. Somit dient das Fehlermodell der Klassifikation von Fehlern und gibt „*... die Strukturierung eines Systems in Komponenten und die Fehlermöglichkeiten der einzelnen Komponenten zumindest qualitativ an (soweit bekannt auch quantitativ).*“ (Echtle 1990b). Fehler können im System als transiente, intermittierende oder permanente Fehler in Erscheinung treten (Echtle 1990b). Darüber hinaus können sie sich zu verschiedenen Arten des Fehlverhaltens der betroffenen Komponenten manifestieren (siehe FM2 bis FM8 in diesem Abschnitt).

Im Folgenden werden die Begriffe transiente, intermittierende und permanente Fehler näher erläutert:

Transienter Fehler. Ein transienter Fehler ist ein Fehler mit einer geringen Wirkdauer. Sein Auftreten lässt sich durch eine Wahrscheinlichkeitsverteilung stochastisch beschreiben. Ausgelöst wird dieser Fehler in der Regel durch Umwelteinflüsse – wie etwa radioaktive Einstrahlungen, Temperaturschwankungen, elektrostatische Entladungen, Fluktuationen der Stromversorgung, etc. Solche Fehler manifestieren sich beispielsweise als Bitkipper in Speichern und in Daten, die über den Kanal übertragen werden, sowie in Registern und Flipflops der Central Processing Unit (CPU), etc. Trotz der geringen Wirkdauer können sie

zum Absturz des kompletten Systems führen, wenn keine Fehlertoleranzverfahren angewandt werden.

Intermittierende Fehler. Sie treten im Gegensatz zu transienten Fehler in irregulären Zeitintervallen auf. Hervorgerufen werden sie durch instabile Soft- und Hardware oder latente Fehler – wie etwa (nicht erkannte) Bit-Fehler in einer Speicherverwaltungseinheit (Memory Management Unit, kurz: MMU).

Permanente Fehler. Intermittierende Fehler können einen Permanenten Fehler hervorrufen, wenn etwa bei Erschütterung ein Kurzschluss verursacht wird, der wiederum zu einem hohen Stromfluss in den Schaltungen führt, die wiederum Schaltkreise irreversibel zerstört. Ein permanenter Fehler verweilt nach seinem Auftreten mit unbegrenzter Zeitdauer im System, bis geeignete fehlerbereinigende Mahnahmen ergriffen werden (Echtle 1990b).

Wenn ein fehlerfreier Betrieb nicht immer gewährleistet ist, so ist es wichtig zu untersuchen, welcher Art von Fehlfunktionen das System bzw. Teile davon ausgesetzt sind. Es gibt zahlreiche Fehlerarten, die sich unterschiedlich auswirken. Das Fehlermodell muss daher auch die Arten der Fehler spezifizieren, die während der Betriebszeit des Systems auftreten können.

Bei der Entwicklung von Übereinstimmungsprotokollen werden in der Regel die folgenden beiden Annahmen getroffen:

FM0 – f_N Knotenfehler. Ein fehlertolerantes Protokoll hat ein Robustheitsgrad f_N (das Protokoll wird als f_N-robust bezeichnet), wenn ein korrekter Betrieb des verteilten Rechensystems in Anwesenheit bis zu f_N Knotenfehler garantieren kann.

FM1 – Signaturen sind fälschungssicher. Übereinstimmungsprotokolle, die zur Absicherung der Kommunikation digitale Signaturen einsetzen, unterliegen der Annahme, dass Fälschungen von Signaturen von fehlerfreien Knoten stets aufgedeckt werden können. Unter der Fälschung einer Signatur versteht man,

- die Veränderung einer Signatur
- die Veränderung des signierten Inhalts
- das Signieren durch einen anderen Knoten als derjenige, der signieren soll

- eine Abweichung von der korrekten Signaturreihenfolge bei mehrfach signierten Nachrichten (Anmerkung: Bei vielen fehlertoleranten Protokollen spielt nur die Anzahl, aber nicht die Reihenfolge der Signaturen eine Rolle).

2.3.6 Das Knoten – Fehlermodell

Die Komplexität einer Lösung zum Übereinstimmungsproblem wird maßgeblich durch die zu tolerierenden Knotenfehler bestimmt, ihre Existenz hingegen durch das Kanalfehler- und Timing-Modell (siehe Unmöglichkeitsergebnis von (Fischer et al. 1985a; Santoro und Widmayer 2007)). Bei Knotenfehlern unterscheidet man die folgenden Fehlerklassen (Cachin et al. 2011):

FM2 – Knotenausfall („crash“). Ein Knoten fällt durch Crash aus, wenn er ab dem Zeitpunkt des Fehlereignisses den Betrieb einstellt („abstürzt“). Nach dem Crash werden Dienste weder bereitstellt noch angefordert. In diesem fehlerhaften Zustand ist der Knoten demzufolge weder in der Lage Nachrichten zu senden noch welche zu empfangen.

FM3 – Unterlassungsausfall („omission“). Ein Unterlassungsausfall tritt auf, wenn der betroffene Knoten in manchen Zeitintervallen weder Nachrichten senden noch empfangen kann. In allen anderen Zeitintervallen tritt der Fehler nicht auf und der Knoten arbeitet korrekt.

FM4 Timing-Fehler. Timing-Fehler sind nur für synchrone verteilte Systeme und Echtzeitsystem relevant. Bei diesem Fehlertyp wird mindestens eine der Synchronisationsannahmen des Systems verletzt, wie z.B. die Nachrichtenübertragungsdauer (oder die lokale Verarbeitungszeit oder die Drift einer Uhr, etc.). Ein Timing-Fehler eines Knotens kann zu Nachrichtenausfällen im gesamten System führen.

FM5 – Gutartige Byzantinische Fehler. Es wird nicht garantiert, dass eine von einem fehlerhaften Knoten abgesandte Nachricht von irgendeinem Knoten empfangen wird. D. h. die Nachricht kann von allen Empfängern, von einer Teilmenge der Empfänger oder von keinem Empfängerknoten empfangen werden.

FM6 – Bösartige byzantinische Fehler. Im Vergleich zu gutartigen byzantinischen Fehlern ist ein bösartig byzantinisch fehlerhafter Knoten in der Lage, eine Nachricht so zu senden, dass sie von verschiedenen Empfängern mit unterschiedlichen Inhalten empfangen

wird. Hinzu kommt die bei gutartigen byzantinischen Fehlern beschriebene Möglichkeit, dass manche Empfänger die Nachricht nicht erhalten. Man kann einen bösartig byzantinischen Fehler als eine Kombination aus mehrfacher Nachrichtenübertragung, Unterlassungsausfällen und Inhaltsverfälschungen beschreiben.

FM7 – Bösartige Kooperation. Fehlerhafte Knoten sind in der Lage, miteinander zu kooperieren (z.B., durch Austausch ihrer privaten Schlüssel in signaturgeschützten Übereinstimmungsprotokollen) mit dem Ziel, die beiden Bedingungen IC1 und IC2 zu verletzen.

FM8 – Beliebiges Verhalten. Ein Knoten verhält sich beliebig, wenn alle oben aufgeführten Fehlverhalten zu jedem Zeitpunkt möglich sind.

2.3.7 Das Kanal – Fehlermodell

Ähnlich wie bei den Knotenfehlern kann der Kanal in einen fehlerhaften Zustand übergehen. Man unterscheidet hier die folgenden Fehlfunktionen:

- Zuverlässiger Kanal

  In einem zuverlässigen Kanal werden sämtliche übertragenen Nachrichten von allen Empfängerknoten korrekt empfangen. Ein zuverlässiger Kanal kann definiert werden als ein Kanal, welcher die folgenden Eigenschaften aufweist:

  - Verlustfrei: Wenn ein Knoten N[i] eine Nachricht M[i][j][1] an einen Knoten N[j] sendet, und sowohl N[i] als auch N[j] fehlerfrei sind, dann wird die Nachricht von Knoten N[j] garantiert empfangen.
  - Duplikatfreie Nachrichten: Jede gesendete Nachricht wird vom Empfängerknoten genau einmal empfangen. Der fehlerbedingt wiederholte Empfang eines Duplikats ist ausgeschlossen. Anmerkung: Diese Aussage ist unabhängig davon zu sehen, dass ein Ende-zu-Ende-Kommunikationsprotokoll durch negative Quittierung dafür sorgen kann, dass eine fehlerhaft übertragene Nachricht nochmals übertragen wird. Im Sinne dieser Definition zählt die nochmalige Übertragung als eine neue Nachricht.

[1] Die von Knoten N[i] gesendet und von Knoten N[j] empfangene Nachricht.

- Keine Nachrichtengenerierung: Wenn eine Nachricht M[i][j] von einem Knoten N[j] empfangen wird, dann muss die Nachricht vorher von einem Knoten N[i] gesendet worden sein.

Von einem zuverlässigen Kanal sprechen wir auch dann, wenn bei Einsatz von redundanten Kanälen zu jedem Zeitpunk nur eine echte Teilmenge der Kanäle gleichzeig fehlerhaft ist. D. .h es gibt mindestens einen Kanal, über den die fehlerfreie Knoten senden können.

- **FM9** – Gutartig Byzantinischer Kanal
  - **FMO9** – Nachrichtenausfall („**omission**“). Bei einem Omission-Fehler wird die Nachricht des Senders von mindestens einem Empfängerknoten nicht empfangen. Folglich garantiert ein gutartig Byzantinischer Kanal keine zuverlässige Verbindung, D. h. Nachrichten, die über einen solchen Kanal übertragen werden, können zu einem beliebigen (unbekannten) Zeitpunkt verloren gehen. Demzufolge ist die Menge der Empfängerknoten, welche die Nachricht tatsächlich erhalten, zu keinem Zeitpunkt bekannt. Dieser Fehler ist in der Regel genauso gravierend wie der Babbling-Fehler. Aus der obigen Beschreibung und laut **FM5** lässt sich implizieren, dass im Falle von FMO9 die von einem fehlerfreien Knoten ausgesandten Nachrichten bedingt durch den Kanalfehler ein ähnliches Verhalten aufweisen wie im Fall eines gutartig Byzantinisch fehlerhaften Senderknotens. Im Folgenden bezeichnen wir Knoten bei Anwesenheit von FMO9 als „virtuell Byzantinisch fehlerhaft“ (kurz: **VB5**).
  - **FMO10** – vollständiger Kanalfehler („**silent**“): Bei einem vollständigen Kanalfehler, hat der betreffende Knoten keine Möglichkeit zu senden, selbst dann nicht, wenn er fehlerfrei ist, weil alle Kanäle fehlerhaft sind und die Nachricht des Knotens nicht korrekt übertragen. Gegenüber den anderen Knoten wird er ein ähnliches Verhalten aufweisen wie ein gutartiger Byzantinischer Knoten-Fehler. Die Kombination von fehlerfreiem Knoten und „vollständigem Kanalfehler“ bezeichnen wird mit „Blockierungfehler“

Das von einem Kanal gemäß **FMO9** und **FMO10** verursachte Byzantinische Verteilungsproblem ist schwierig, wenn nicht unmöglich zu lösen (siehe auch (Santoro und Widmayer 2007)). Der Grund hierfür ist, dass sämtliche Lösungen zum Übereinstimmungsproblem der

f-Fehlerannahme zugrunde liegen, der virtuell Byzantinische Knotenfehler bzw. Blockierungfehler aber alle Knoten betrifft.

Im Falle eines Byzantinisch fehlerhaften Kanals kann es durchaus vorkommen, dass ein Übereinstimmungsprotokoll mehr als f_N Fehler tolerieren muss, da neben den f_N fehlerhaften Knoten noch eine Anzahl virtuell Byzantinisch fehlerhafter Knoten zu tolerieren sind (Bedingt durch den fehlerhaften Kanal besteht aus Sicht der Empfängerknoten ein gutartig Byzantinischer Fehler). Die Konsequenz davon ist, dass im Falle eines gutartig Byzantinischen Kanals das Übereinstimmungsproblem nur gelöst werden kann, wenn entweder die Nachrichtenausfälle stets von allen fehlerfreien Knoten erkennbar sind (siehe z.B. (Khosravi und Kavian 2012), auch dafür benötigt man eine Art Übereinstimmungsprotokoll) oder die maximale Anzahl der Fehler pro Kommunikationszyklus bekannt ist.

Der Vollständigkeit halber sei erwähnt, dass für CP eine asynchrone Variante existiert (Moniz et al. 2013), welche das Santoro-Widmayer-Unmöglichkeitsergebnis umgeht (siehe (Moniz et al. 2013)). Dieser Ansatz lässt sich wie folgt charakterisieren:

1. Zur Lösung des Problems wird Randomisierung genutzt. Dadurch steigt die Wahrscheinlichkeit, einen Konsens mit einer zunehmenden Anzahl von Durchläufen zu erzielen, gegen den Wert 1. Folglich handelt es sich hierbei um eine indeterministische Lösung des **CP**s (siehe auch Abschnitt 2.3.3).
2. Das Protokoll sendet in periodisch wiederkehrenden Zeitabschnitten (durch den Einsatz einer lokalen Uhr) die zuletzt übertragene Nachrichten wiederholt an alle teilnehmenden Knoten. Somit sinkt mit einer zunehmenden Anzahl von wiederholten Sendeoperationen der (letzten übertragenen) Nachricht die Wahrscheinlichkeit, dass kein Knoten die Nachricht empfängt, gegen den Wert 0.
3. Das Protokoll beschreibt die Lösung für den einmaligen Konsens. Dies bedeutet: Entscheidet sich ein Knoten für einen einelementigen Konsistenzvektor (Endwert), dann darf er zwar seine Entscheidung nicht mehr ändern, jedoch muss er weiter an der Protokollausführung teilnehmen, damit die restlichen fehlerfreien Knoten noch Nachrichten empfangen und damit auch zu einer Entscheidung kommen können.

Als nächstes wird das Fehlermodell **FM10** von Santoro und Widmayer erläutert (mit den dazugehörigen Fehlerarten):

- **FM10** – Unzuverlässiger Kanal nach (Santoro und Widmayer 2007)
  - **FMO10** – Nachrichtenausfall („**omission**“). Ähnlich zu **FMO10** bzw. **VB5**.
  - **FMC10** – Verfälschung („**corruption**“). Bei corruption wird die Nachricht des Senders zwar von allen Zielempfängern empfangen, jedoch stimmt der Inhalt nicht mit dem der ursprünglich übertragenen Nachricht überein.
  - **FMA10** – Einfügefehler (**addition**). Zusätzliche Nachrichten treten auf, wenn eine Nachricht empfangen wurde, obwohl kein Sender diese übertragen hat.
- Lösungen zum Übereinstimmungsproblem, denen das Fehlermodell **FM10** zugrunde liegt, sind in der Lage, bis zu f dynamische Fehler (siehe ggf. Abschnitt 2.2.3) zu tolerieren, und somit auch bis zu f „virtuelle“ Byzantinische Knotenfehler. Jedoch geht dieses Modell von der Annahme aus, dass sich die f Fehler nur von Runde zu Runde ändern dürfen. Übereinstimmungsprotokolle, die zur Lösung des Übereinstimmungsproblems mehr als eine Runde benötigen, müssten daher in der Lage sein, dynamische (Knoten-) Fehler zu tolerieren.
- Übereinstimmungsprotokolle (wie etwa (Bousbiba 2015a)), die eine Übereinstimmung in einer Runde erreichen, können auf solche Fehlermodelle ohne weiteres angewendet werden.

3 Übereinstimmungsprotokolle in drahtlosen verteilten Echtzeitsystemen

„It [the computer] is approachable only through complex jargon that has nothing to do with the tasks for which which people actually use computers. The state of the art is perhaps analogous to the period when scribes had to know as much about making ink or baking clay as they did about writing.“ – Mark Weiser

3.1 Stand der Technik

Das Byzantinische Übereinstimmungsproblem ist ein gut erforschtes Problem mit vielen Lösungsbeiträgen zu verteilten (Rechen-) Systemen. Das fundamentale Problem, wie man eine Einigung zwischen den Knoten eines verteilten Systems unter verschiedenen Systemannahmen (synchron, asynchron oder teilsynchron) erzielt, kann als gelöst erachtet werden bzw. dessen Unlösbarkeit für verschiedene Systemannahmen als erwiesen gelten (siehe ggf. (Santoro 2006; Fischer et al. 1985a)). Nunmehr ist man daran interessiert, das Byzantinische Übereinstimmungsproblem effizient bzgl. der Kommunikationskomplexität zu lösen.

Die Entwicklung effizienter Lösungsstrategien hat auch einen praktischen Hintergrund, nämlich den wachsenden Einsatz von verteilten Echtzeitsystemen in Militär- und Zivilanwendungen (Flugzeugbau, Katastrophenüberwachung, Einsatz von autonomen mobilen Robotern, etc.). Dort müssen Übereinstimmungsprotokolle, wenn Sie zum Einsatz kommen sollen, effizient bzgl. der Kommunikationskomplexität sein. Vor allem der Nachrichten- und Speicheroverhead sollte möglichst gering sein.

Im Folgenden wird ein Überblick zu Forschungsarbeiten gegeben, die im direkten oder indirekten Zusammenhang mit der vorliegenden Arbeit stehen. Es werden die jeweiligen Lösungsansätze und der durch sie hervorgerufene Aufwand beschrieben sowie mögliche Nachteile diskutiert.

Eine der ersten Arbeiten nach (Lamport et al. 1982), die sich mit dem Design effizienter Übereinstimmungsprotokolle auseinander gesetzt haben, waren (Dolev und Strong 1982). Dolev und Strong zeigten, dass zur Lösung des Byzantinischen Übereinstimmungsproblems für ein synchrones vollvermaschtes Netz kein exponentieller Nachrichtenaufwand erforderlich

ist. Ferner zeigten Sie, dass zur Lösung des Byzantinischen Übereinstimmungsproblems mindestens $f + 1$ Phasen benötigt werden. Gemäß (Dolev und Strong 1982; Halldórsson und Dolev) darf innerhalb einer Phase jeder Knoten senden und alle Nachrichten einer Phase werden nebenläufig übertragen. Dies entspricht auch der Definition 2.2 in Abschnitt 2. Der Kompromiss zwischen Nachrichtenoverhead und Anzahl der notwendigen Kommunikationsrunden (siehe Definition 2.1 in Abschnitt 2) zur Lösung des byzantinischen Übereinstimmungsproblems wurde als offenes Problem vermerkt, das es noch zu lösen gilt. Diese Frage wurde teilweise von (Garay und Moses 1998) beantwortet. Sie stellten unter denselben Fehlerannahmen wie in (Dolev und Strong 1982) ein Übereinstimmungsprotokoll vor, welches das Byzantinische Übereinstimmungsproblem in $f + 1$ Phasen mit polynomiellem Kommunikationsaufwand und Fehlertoleranzgrad $f < n/3$ löst. Weiterhin gibt es einen Beweis, (Lamport et al. 1982; Dolev und Strong 1982) der zeigt, dass zur Lösung des Byzantinischen Übereinstimmungsproblems bei f tatsächlich auftretenden Knotenfehlern mit „willkürlich“ fehlerhaftem Verhalten mindesten $f + 1$ Phasen benötigt werden.

Eine der ersten Arbeiten, die sich mit dem praktischen Einsatz von Übereinstimmungsprotokollen im Bereich Automotive auseinander gesetzt haben, waren (Jochim und Forest 2010). Sie stellten eine effiziente Variante des SM-Protokolls (Lamport et al. 1982) vor, welches im Vergleich zu SM (Lamport et al. 1982) einen niedrigeren Nachrichten- und Speicheroverhead zur Lösung des Byzantinischen Übereinstimmungsprotokolls benötigt. Um den Nachrichten- und Speicheroverhead zur reduzieren, wurde das ursprüngliche Fehlermodell leicht modifiziert. Folgende Modifikationen wurden vorgenommen:

- Jeder Knoten sendet höchstens zwei Nachrichten (fehlerhafte Knoten im begriffen)
- Kooperation zwischen fehlerhaften Knoten wird ausgeschlossen.

Um den Speicheraufwand zu minieren, wurde die folgende Strategie verwendet:

- Es wird stets die Nachricht mit den meisten Signaturen gespeichert und weitergeleitet.

Dadurch ist es ihnen gelungen, den Speicheraufwand (nicht aber die Anzahl der benötigten Runden zur Lösung des Byzantinischen Übereinstimmungsproblems) auf eine Nachricht pro Knoten zu reduzieren, was im Vergleich zum ursprünglichen Protokoll eine Verbesserung darstellt, obgleich das Modell hierzu verändert bzw. vereinfacht werden musste.

Insgesamt lässt sich Folgendes zusammenfassen: Um die Kommunikationskomplexität zur Lösung des Byzantinischen Übereinstimmungsproblems weiter zur reduzieren, ist es notwendig (siehe Ergebnis von (Garay und Moses 1998; Jochim und Forest 2010)), Lösungsstrategien zu entwickeln, die es erlauben, eine Übereinstimmung in weniger als $f + 1$ Runden zu erzielen.

Eine Möglichkeit, die durch $f + 1$ gegebene Schranke zu überwinden, besteht darin, Protokolle zu entwickeln, die eine Übereinstimmung in probabilistischer Weise erzielen (auch als „**randomized agreement“** bekannt, (Dolev und Strong 1982). Der Nachteil ist, dass dieser Ansatz nur zur Lösung des binären Konsensproblems herangezogen werden können, d.h. des Konsensproblems über eine 0/1-Entscheidung. Der Einsatz eines „schwachen“ globalen Münzwurf-Mechanismus (siehe ggf. (Malcolm und Strong 1985; Rabin)) stellt eine von mehreren Möglichkeiten dar, wie man das binäre Konsensproblem in O(1) Runden löst. Ein Lösung hierzu wurde von (Michael Ben-Or 2005) vorgeschlagen.

(Hsieh und Chiang 2011) konnten erstmalig zeigen, dass das Byzantinische Übereinstimmungsproblem auch ohne Einsatz von Randomisierung in O(1) Runden lösbar ist. Der Nachteil ihrer Lösung (die im Folgenden näher erläutert wird) ist zum einen, dass ihr Fehlermodell Omission-Fehler vollständig ausschließt und zum anderen einen hohen Speicheroverhead benötigt. Um das Byzantinischen Übereinstimmungsproblems in O(1) Runden zu lösen, verwenden (Hsieh und Chiang 2011) eine Art lokale Fehlerdiagnose, die im Protokoll am Ende der dritten Runde ausgeführt wird. Sie soll sicherstellen, dass die fehlerfreien Knoten ihre Entscheidung bzgl. des Endwerts auf Basis der fehlerfreien Knoten treffen (siehe Punkt 1 „$\underline{v(sx)}$“. und 2 „$\underline{\#maj3}$“ weiter unten auf Seite 35). Die während der Runden empfangenen Nachrichten werden vom jeweiligen Knoten in einer baumartigen Datenstruktur (siehe Abbildung 4) lokal verwaltet.

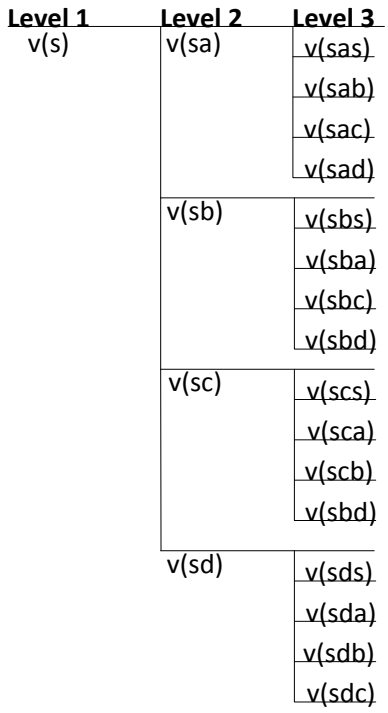


Abbildung 4: Baumartige Datenstruktur. v(s) steht für die Nachricht bzw. den Nachrichteninhalt, der vom Quellknoten s übertragen wurde (siehe mehr hierzu (Hsieh und Chiang 2011)).

Die Bezeichnungen in Abbildung 4 sind wie folgt zu lesen:

- Der Bezeichner v(s) steht für die Nachricht, die der Quellknoten s in Runde 1 übertragen hat. Jeder Empfänger hinterlegt die empfange Nachricht v(s) in Level 1 des Baumes ab.
- Der Bezeichner v(sx) mit x ϵ {a, b, c, d} steht für die Nachricht, die der Knoten x in Runde 1 empfangen und in Runde 2 weitergeleitet hat. Jeder Empfänger hinterlegt die empfangene Nachricht v(sx) in Level 2 des Baumes ab.
- In Runde 3 sendet jeder Knoten y mit y ϵ {a, b, c, d} seine in Level 2 hinterlegten Nachrichten v(sx) mit x ϵ {a, b, c, d} an alle Knoten weiter (siehe (Hsieh und Chiang 2011)). Somit steht der Bezeichner v(sxy) für die Nachricht, die der Knoten y in Runde 2 empfangen und in Runde 3 weitergeleitet hat. Jeder Empfänger hinterlegt die empfangene Nachricht v(sxy) in Level 3 des Baumes ab.

Fehlerdiagnose: In (Hsieh und Chiang 2011) entscheidet jeder fehlerfreie Knoten y (am Ende der Runde 3) anhand der beiden folgenden Bedingungen, ob ein Knoten x (mit x ≠ y) fehlerhaft ist oder nicht:

1. $v(sx) = maj3(sx)$. Zunächst prüft ein Knoten y ≠ x in seinem baumartig verwalteten Nachrichtenspeicher, ob die direkten Nachfolger von Knoten v(sx) (Hier: v(sas), v(sab), v(sac), v(sad)) mehrheitlich mittels „**Majority-Voting**“ mit dem in Knoten sx hinterlegten Wert übereinstimmen (siehe Abbildung 4),
2. $\#maj3(sx) \geq \left(n - \left\lfloor\frac{n-1}{3}\right\rfloor - 1\right)$. Mindestens $\left(n - \left\lfloor\frac{n-1}{3}\right\rfloor - 1\right)$ Nachfolgerknoten von sx weisen denselben Inhalt auf, was durch eine Mehrheitsprüfung festgestellt wird.

Nur wenn beide Bedingungen 1. und 2. erfüllt sind, wird der betroffene Knoten x von Knoten y als zuverlässig eingestuft. In allen anderen Fällen ignoriert Knoten y die Nachrichten von Knoten x (für mehr Details siehe (Hsieh und Chiang 2011)). Zum Schluss wird mittels Mehrheitsentscheid der Endwert bestimmt.

In (Khosravi und Kavian 2012) wurde ebenfalls ein Protokoll vorgestellt, welches das Byzantinische Übereinstimmungsprobleme ähnlich zu (Hsieh und Chiang 2011) in drei Runden löst. (Khosravi und Kavian 2012) verwendet zur Lösung des Byzantinischen Übereinstimmungsproblems dasselbe Fehlermodell wie in (Hsieh und Chiang 2011). Anders als in (Hsieh und Chiang 2011) kombinieren sie aber die lokale Fehlerdiagnose mit dem Versenden von Veto-Nachrichten. Außerdem verwenden (Khosravi und Kavian 2012) im Gegensatz zu (Hsieh und Chiang 2011) die Fehlerdiagnose ausschließlich zur Diagnose des Fehlerzustands (fehlerfrei / fehlerhaft) des Quellknotens (für mehr Details siehe (Khosravi und Kavian 2012)). Falls ein Knoten mittels der Fehlerdiagnose den Quellknoten als fehlerhaft einstuft, so sendet dieser in Runde 3 eine Veto-Nachricht. Abhängig von der Anzahl der empfangenen Veto-Nachrichten, entscheidet sich ein fehlerfreier Knoten entweder gegen oder für den Wert des Quellknotens als Endwert (für mehr Details siehe ggf. (Khosravi und Kavian 2012)).

Insgesamt konnte (Khosravi und Kavian 2012) mit den beiden kombinierten Techniken (lokale Fehlerdiagnose und Veto-Nachrichten) den Speicheraufwand im Vergleich zu (Hsieh und Chiang 2011) signifikant reduzieren. Nichtsdestotrotz bleibt die Frage offen, ob das Byzantinische Übereinstimmungsproblem gemäß Fehler bzw. Systemmodell (Lamport et al. 1982) in einer konstanten Rundenanzahl bzw. in weniger als $f + 1$ Runden lösbar ist. Hier

scheint die Lücke aus Sicht der Forschung noch nicht geschlossen worden zu sein. Eine Antwort auf diese Frage wird im nachfolgenden Kapitel gegeben.

3.2 Zusammenfassung

Im Kern haben bis dato alle Veröffentlichung zum Übereinstimmungsproblem eines gemeinsam: Sie benötigen einen (abhängig vom Fehlermodell) hohen Kommunikations-, Knoten- und/oder Speicheroverhead, gleichwohl viele Arbeiten den Fokus auf die Reduktion der Kommunikationskomplexität gelegt haben. Der Grund hierfür resultiert aus der simplen Tatsache, dass die Kommunikationskomplexität vom zugrundeliegenden Fehlermodell stark abhängt. So verursachen in der Regel Protokolle, die einem einfachen Fehlermodell zugrunde liegen (wie etwa in (Khosravi und Kavian 2012)) einen geringeren Aufwand (bzgl. der Kommunikationskomplexität) als solche, bei denen das Fehlermodell keine einschränkenden Annahmen bzgl. der zu tolerierenden Fehlerart trifft (siehe Abschnitt 2.1.2 und ggf. (Lamport et al. 1982)). Daher bleibt es weiterhin Gegenstand der Forschung, möglichst effiziente Lösungsstrategien zum Byzantinischen Übereinstimmungsproblem zu entwickeln.

Insgesamt lässt sich festhalten: Das Verfahren der lokalen Diagnose (siehe (Khosravi und Kavian 2012; Hsieh und Chiang 2011)), das Prinzip des Votings (Khosravi und Kavian 2012) und die Technik zur Speicherreduzierung (Jochim und Forest 2010) scheinen vielversprechende Konzepte zu sein, die zur Beantwortung der Forschungsfrage genutzt werden können.

4 ESSEN – Ein neues Byzantinisches Übereinstimmungsprotokoll

„Wer vor großen Fehlern Angst hat, hat auch nur Mut zu kleinen Erfolgen.“ – Hans-Jürgen Quadbeck-Seeger.

In diesem Kapitel wird das neue Byzantinische Übereinstimmungsprotokoll ESSEN („An **E**fficient **S**ingle Round **S**ignature Protected Message **E**xchange Agreement Protocol for Wireless Distributed **N**etworks“) vorgestellt, welches das byzantinische Übereinstimmungsproblem für synchrone drahtlose verteilte Systeme, die sich aus n Knoten zusammensetzen und von denen bis zu f_N Knoten ein beliebiges Fehlerverhalten aufweisen können, effizient löst. Zur Lösung des Byzantinischen Übereinstimmungsproblems benötigt ESSEN mindestens einen Kanal, über den die Knoten miteinander Nachrichten austauschen können. Stehen mehr als ein Kanal für die Kommunikation zu Verfügung, dann werden diese vom Protokoll ESSEN ausschließlich zum Zweck der Redundanz eingesetzt. Dies bedeutet: Bei mehrfachen Kanälen wird die Protokollausführung nur dann durch fehlerhafte Kanäle beeinträchtig, wenn ein Blockierungsfehler auftritt (siehe Definition von „Blockierungsfehler“ in Abschnitt 2.3.7). Wenn nur ein Kanal fehlerfrei ist, wird ESSEN nicht beeinträchtig.

Die nachfolgenden Abschnitte sind wie folgt angeordnet: Abschnitt 4.2 bis Abschnitt 4.8 liefern das Grundkonzept und Details zum Übereinstimmungsprotokoll ESSEN. Dazu zählen unter anderem das Systemmodel sowie das dazugehörigen Fehlermodell. Abschnitt 4.9 beschreibt den Algorithmus von ESSEN. In Abschnitt 4.10 wird die Kommunikationskomplexität von ESSEN anderen effizienten Übereinstimmungsprotokollen gegenübergestellt. Zum Schluss wird in Abschnitt 4.11 ein Resümee gezogen.

Die Ergebnisse des Verifikations- (mittels UPPAAL) und Validierungsprozesses (mittels Simulation) werden in Kapitel 5 präsentieren. Der formale Korrektheitsbeweis des Protokolls für beliebige $f > 0$ wird in Kapitel 6 erbracht.

4.1 Idee

Die Grundidee des in dieser Arbeit entwickelten Übereinstimmungsprotokolls besteht darin, das Übereinstimmungsproblem – unabhängig von der Anzahl der zu tolerierenden Fehler – stets in einer Runde zu lösen, ohne jedoch einen höheren Mehraufwand bzgl. der einzusetzenden Ressourcen (redundante Komponenten, Speicher- und Kommunikationsoverhead)

zu verursachen. Hierzu wurden zwei bereits existierende Konzepte grundlegend modifiziert und weiter entwickelt, um auf diese Weise eine hohe Effizienz zu erreichen. Es handelt sich um die beiden folgenden Konzepte:

1. der Einsatz von Veto-Nachrichten. Mit Hilfe solcher (signierten) Nachrichten erhalten die Knoten die Möglichkeit, Einfluss auf den Inhalt des Konsistenzvektors (also des Endwerts) zu nehmen.
2. Der Einsatz einer Reinigungsfunktion zur Bestimmung des Endwerts: Die Reinigungsfunktion stellt sicher, dass die Entscheidungen, die von den Knoten getroffen werden, ausschließlich auf Basis der nicht als fehlerhaft detektierten Knoten getroffen wird. Wie diese Funktion im Protokoll ESSEN umgesetzt worden ist, soll an späterer Stelle (siehe Seite 57) genauer erläutert werden.

Beide Konzepte haben bereits gezeigt (siehe (Khosravi und Kavian 2012)), dass sie grundsätzlich zur Entwicklung effizienter Übereinstimmungsprotokolle verwendet werden können. Die Schwierigkeit besteht darin, eine Lösung zu entwickeln, die ohne Vereinfachung des Fehlermodells (anders als in (Khosravi und Kavian 2012; Hsieh und Chiang 2011)) das Byzantinische Übereinstimmungsproblem effizient in O(1) Runden (siehe Definition 2.1) löst. Hierzu reicht es nicht aus, die beiden genannten Konzepte geschickt miteinander zu kombinieren, sondern es müssen vielmehr neue Lösungen erarbeitet werden, welche diese Konzepte als Kernidee enthalten. Wie die neue Lösung gestaltet wird, zeigt Abschnitt 4.9.

Ein Übereinstimmungsprotokoll, welches unabhängig von der Anzahl der zu tolerierenden Knotenfehler in einer konstanten Anzahl an Runden terminiert, gilt außerdem als besonders praxistauglich. Denn viele (verteilte) Echtzeitanwendungen, die fehlertolerant entwickelt werden müssen, und bei denen das Übereinstimmungsproblem vorliegt, setzen statt Übereinstimmungsprotokollen andere Fehlertoleranz-Verfahren ein (z. B. mehrfach redundante Auslegung von Komponenten kombiniert mit einem Mehrheitsentscheid). Der Grund hierfür liegt oft darin, dass Übereinstimmungsprotokolle einen bzgl. der Anwendung zu hohen Kommunikationsaufwand verursachen. Byzantinische Übereinstimmungsprotokolle (auch wenn ihre Lösung primär für synchrone verteilte System entwickelt ist) werden oftmals wegen ihres hohen Kommunikationsaufwands oder vereinfachten Fehlermodells für (verteilte) Echtzeitanwendungen als nicht anwendbar angesehen – eine Einschätzung, die sich mit der Entwicklung von effizienteren Protokollen ändern dürfte.

4.2 Überblick über das Protokoll ESSEN

Das hier präsentierte Übereinstimmungsprotokoll ESSEN zeigt einen neuartigen Lösungsweg, wie man das Byzantinische Übereinstimmungsproblem für synchrone verteilte Systeme, unabhängig von der Anzahl der zu tolerierenden Fehlern f_N, in einer konstanten Anzahl an Runden löst.

Zur Lösung des Byzantinischen Übereinstimmungsproblems in Anwesenheit von allen Fehlerarten bis hin zu kooperierenden Byzantinischen Fehlern (siehe FM7) bei strikter Einhaltung von IC1 und IC2 (Lamport et al. 1982) benötigt das Protokoll ESSEN nur eine einzige Runde. Der Begriff Runde bezeichnet (wie in Kapitel 2 definiert) einen Zeitabschnitt, in dem jeder Knoten die Möglichkeit bekommt, eine Nachricht an alle teilnehmenden Knoten zu übertragen. Folglich sendet innerhalb einer Runde (siehe ggf. (Santoro 2006)) jeder fehlerfreie Knoten nicht mehr als eine Nachricht und die Phasenanzahl in dieser Runde ist nicht größer als die Knotenanzahl. Die Begrenzung auf höchstens eine Nachricht gilt für die fehlerfreien Knoten, nicht aber für fehlerhafte Knoten. Man kann nicht verhindern, dass diese abhängig vom Fehlerzustand keine, eine oder mehr als eine Nachricht an alle oder nur an eine Teilmenge (leere Menge miteingeschlossen) der teilnehmenden Knoten übertragen (siehe Abbildung 4 und Abbildung 3). Busguardians können nur sicherstellen, dass für jeden Knoten eine Phase bleibt, in der die von ihm gesendeten Nachrichten nicht durch Nachrichten von fehlerhaften Knoten gestört werden.

Zur Tolerierung von f willkürlichen Fehlern benötigt ESSEN mindestens

$$n \geq 3f + max(0, f - 2) \qquad 4.1$$

Knoten. Diese Formel wird in Abschnitt 4.7 näher beleuchtet.

Bezüglich des Kommunikationsaufwandes unterscheidet sich die Runde von dem Verfahren gemäß (Santoro und Widmayer 2007) kaum, d. h. in beiden Fällen ist der Aufwand aus Sicht der Knoten identisch (Anzahl der zu prüfenden Nachrichten, etc.). Der wesentliche Unterschied besteht allerdings darin, dass bei einer Runde des Protokolls ESSEN die Sendereihenfolge unter den Knoten bereits im Vorfeld festgelegt wird, was in verteilten synchronen (Echtzeit-) Systemen durchaus üblich ist. Bei anderen Protokollen ergibt sich die Sendereihenfolge dagegen dynamisch (Khosravi und Kavian 2012; Hsieh und Chiang 2011) oder gar indeterministisch (Moniz et al. 2013), was im Folgenden als „dynamische Runde“ genannt

wird (im Gegensatz zur „statischen Runde“ in ESSEN). Die Phasen einer statischen Runde werden Slots („Zeitschlitze“) genannt. Die Festlegung in der statischen Runde bringt entscheidende Vorteile mit sich:

- Mechanismen zur Kollisionsvermeidung müssen für den fehlerfreien Fall nicht bereitgestellt werden, da die Sende- und Empfangszeitpunkte jedem teilnehmenden Knoten vorab bekannt sind.
- Der zeitliche Vorteil, den man bei einer dynamsiche Runde erhält, ist gegenüber der statischen Runde minimal, da die folgenden Aktionen stets vorhanden sind:
    1. Jede empfangene (und demzufolge abzuspeichernde) Nachricht muss in den Speicher geschrieben werden.
    2. Die Signaturchecks müssen für jede empfangene bzw. gespeicherte Nachricht einzeln ausgeführt werden.
    3. Die nächste Phase (siehe Definition 2.2) beginnt erst, nachdem sämtliche gespeicherten Nachrichten aus der aktuellen Phase bearbeitet worden sind (siehe Aktion 1 und 2), da das Sendeverhalten eines Knotens in der nächsten Phase in der Regel von den zuvor empfangenen Nachrichten abhängt.
- Entscheidung können früher getroffen werden. Beispielsweise kann ein Knoten die zuvor empfangenen Nachrichten dazu verwenden, um seine Entscheidung zu treffen, welcher Inhalt in der nächsten Nachricht gesendet werden soll (wie es im Protokoll ESSEN der Fall ist). Mit anderen Worten: Die Entscheidungen werden von Slot zu Slot und nicht erst von Runde zu Runde getroffen.
- In statischen Runden können Busguardians in viel einfacherer Weise eingesetzt werden, da der Slots, in denen ein Busguardian das Senden zulassen muss, allein durch den Ablauf der Zeit bestimmt sind.

Im Folgenden wird das Modell eines verteilten Systems beschrieben. Wir betrachten ein voll synchronisiertes System (hierzu später mehr), welches aus n Knoten aus der der Menge $N = \{N[1],..., N[n - 1]\}$ besteht, die alle durch mindestens einen Kommunikationskanal verbunden sind, über die sie Nachrichten untereinander austauschen können. Die Kommunikation zwischen den Knoten erfolgt innerhalb einer einzigen Runde und die Entscheidung über den Endwert erfolgt am Ende der Runde.

Die Slots einer Runde sind in Abbildung 4 dargestellt.

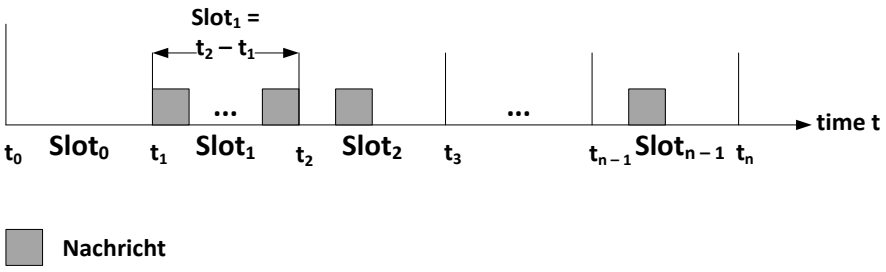


Abbildung 5: Runde, unterteilt in mehreren Slots („Zeitschlitze“) [(Bousbiba 2015a)].

Jeder Sendeknoten N[i] hat exklusiven Schreibzugriff auf einen individuell zugeordneten Slot um eine Broadcast-Transaktion auszuführen, d.h. eine Broadcast-Nachricht zu senden. In allen anderen Slots verfügt der jeweilige Sendeknoten nur über einen Lesezugriff. Folglich sind die individuellen zugeordneten Slots gegen falschen Zugriff und somit auch gegen Babbling-Fehler geschützt. Was zu den folgenden Implikationen führt:

1. **Uhrensynchronisation**

   Unter „voll synchronisiert“ verstehen wir, dass sowohl die Verarbeitungszeit der jeweiligen Knoten als auch die vom Kanal verursachte Kommunikationsverzögerung nach oben beschränkt und die Schranken bekannt sind. Außerdem wird angenommen, dass eine globale Sicht auf die Zeit im System existiert, die durch ein fehlertolerantes Uhrensynchronisationsverfahren mit beschränkter Ungenauigkeit durchgeführt wird und auch dafür die Schranke bekannt ist (z. B. solche Verfahren wie in (Welch und Lynch 1988; Wang und He; Lasassmeh und Conrad; Fan et al. 2004)).

2. **Slot-Zugriff Mechanismus**

   Wie bereits in Kapitel 2 zuvor erwähnt, ist das Übereinstimmungsproblem nicht lösbar, wenn die Kommunikation zwischen den fehlerfreien Knoten nicht vor Babbling-Fehlern geschützt ist (Santoro und Widmayer 2007), sofern man an deterministischen Lösungsansätzen interessiert ist. Folglich müssen die individuell zugeordneten Slots gegen unzulässigen Zugriff geschützt werden. Dieser Schutz kann durch sogenannte „Busguardians“ (Buswächter) erbracht werden (Temple 1998; Kai Wang et al., Kai Wang et al.).

   Durch den Einsatz eines Busguardian wird verhindert, dass ein fehlerhaft agierender Knoten im Stande ist, die Kommunikation zwischen fehlerfreien Knoten durch das

Senden von Nachrichten zu unvorhersehbaren Zeiten zu stören bzw. zu unterbinden (genaueres siehe (Temple 1998)). Jedoch ist dieser Schutz nur dann gewährleistet, wenn beim Einsatz eines Busguardians die 1-Fehlerannahme zu keinem Zeitpunkt verletzt wird. Die 1-Fehlerannahme besagt (wie das Wort bereits suggeriert) ferner, dass die zu betrachtenden Fehlerbereiche „Knoten“ und „zugehöriger Busguardian“ nicht zugleich fehlerhaft sind (Echtle 1990b). Folglich wird eine gemeinsame Fehlerquelle zwischen Busguardian und den zu überwachenden Knoten ausgeschlossen bzw. die Wahrscheinlichkeit hierfür wird als vernachlässigbar gering angenommen.

> Anmerkung: Zwischen den Knoten gilt diese 1-Fehlerannahme nicht. Es können bis zu f_N Knoten gleichzeitig fehlerhaft sein.

3. **Sende-/Empfangs-Slot**

   In drahtlosen verteilten Systemen nehmen wir an, dass bedingt durch den Funkkanal nur ein Teil der gesamten Slotdauer für die Nachrichtenübertragung verwendet wird, so dass innerhalb eines Slots mehr als eine Nachricht übertragen werden kann. Jedoch wird angenommen, dass diese so genannten aufgeblähten Slots nur von den fehlerhaften Knoten „genutzt“ werden, um die in Kapitel 2 beschriebenen Fehlertypen zu verursachen (siehe FM1 bis FM8 und ggf. Abbildung 4).

4.3 Die Knotengruppen in ESSEN

Das hier vorgestellte Protokoll ESSEN löst das Byzantinische Übereinstimmungsproblem, bei dem ein einzelner Wert von einem vordefinierten Knoten (den sogenannten Quellknoten bzw. „source node“) an mehrere Empfangsknoten (auch Sinkknoten bzw. „sink node“ genannt) verteilt wird. Alle fehlerfreien Sinkknoten müssen sich auf den vom Quellknoten übertragenen Wert einigen (d. h. ihn akzeptieren oder verwerfen). Zwischen dem Quellknoten und den Sinkknoten gibt es weitere m Versenderknoten („forwarder nodes“). Die Versenderknoten nehmen im Vergleich zu Sinkknoten aktiv im Übereinstimmungsprozess teil und zwar durch das Weiterleiten bzw. von Nachrichten an benachbarte (bzw. an alle teilnehmenden) Knoten.

In ESSEN werden die Versenderknoten in Basis-Versenderknoten („basic forwarder“) und erweiterte Versenderknoten („extended forwarder“) unterteilt. Die Basis-Versenderknoten dürfen nur die Nachricht weiterleiten, die sie unmittelbar von dem Quellknoten erhalten haben. In allen anderen Fällen nimmt der jeweilige Basis-Versenderknoten nur passiv an der Kommunikation teil (in diesen Fällen verhält sich der Knoten äquivalent zu einem „puren“

Sinkknoten). Die erweiterten Versenderknoten jedoch dürfen Nachrichten weiterleiten, die sie von einem Basis-Versenderknoten oder einem erweiterten Versenderknoten erhalten haben. Außerdem dürfen sie so genannte Veto-Nachrichten (hier: als Default-Nachricht bezeichnet) verteilen, welche einen Einfluss auf den Endwert ausüben, auf den sich die Sinkknoten am Ende einer Runde einigen. Bezüglich ihrer Entscheidung auf einen Endwert unterscheiden sich die (Basis bzw. erweiterten) Versenderknoten nicht von den reinen Sinkknoten. Jeder Versenderknoten ist auch ein Sinkknoten und muss demzufolge ebenfalls eine gemeinsame Einigung bzgl. des Endwerts gemäß IC1 und IC2 mit den restlichen fehlerfreien Knoten erreichen.

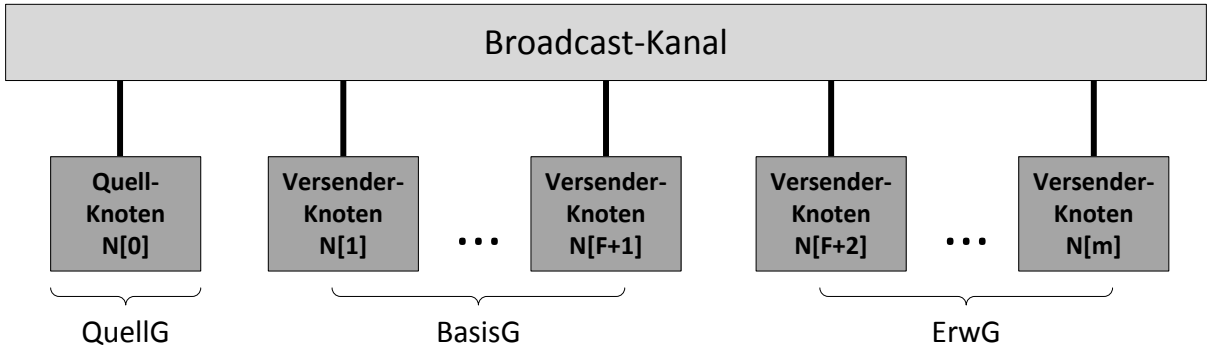


Abbildung 6: Die Knoten-Gruppen in ESSEN , die aktiv am Übereinstimmungsprozess teilnehmen.

Die n Knoten werden bezüglich ihrer Rolle im Protokoll ESSEN gemäß ihrer Funktion in Gruppen aufgeteilt (siehe auch Abbildung 6 und Abbildung 7), wobei die Knoten entsprechend ihrer Funktion, die sie in einer Protokollausführung übernehmen, indiziert sind. Wenn in einer nachfolgenden Protokollausführung beispielsweise ein anderer Quellknoten seinen Wert verteilt, dann ändert sich die Knoten-Indizierung entsprechend.

- Gruppe G = {N[0], …, N[n – 1], N[n], …, N[n + k – 1]}
  enthält die Menge aller passiven und aktiven Knoten. Nur Knoten aus dieser Menge müssen eine Übereinstimmung bezüglich des Werts des Quellknotens erzielen (akzeptieren bzw. verwerfen). Der Parameter n (siehe Formel 4.1) beschreibt die Anzahl der notwendigen Sender zur Lösung des Byzantinischen Übereinstimmungsproblems. Falls das System aus weiteren $k \geq 0$ passiven Knoten besteht, die ebenfalls eine

Übereinstimmung erzielen müssen, so ergibt sich die Gesamtzahl der Knoten zu $g = n + k$.

- Gruppe QuellG = {N[0]}
  Diese Menge besteht aus genau einem Quellknoten. Falls es jedoch mehr als einen Quellknoten gibt (wie etwa beim ICP), ändert sich die Menge entsprechend zu QuellG = {N[0],…, N[i]} mit $i \geq 1$. Jeder Knoten aus der Menge QuellG signiert und verteilt seinen Wert an alle Knoten aus der Menge G. Mit Parameter q bezeichnen wird die Kardinalität der Menge QuellG ($q = |QuellG|$). Hier wird nur der Fall $q = 1$ (BAP) betrachtet.
- Gruppe ForwG = {N[1],…, N[n – 1]}
  ist die Menge der Versenderknoten. Sie setzt sich aus den beiden Mengen BasisG und ErwG zusammen. Diese Knoten leiten abhängig vom Protokoll die Nachrichten weiter, die sie zuvor empfangen haben. Es gilt: $ForwG = BasisG \cup ErwG$.
- Gruppe BasisG = {N[1],…, N[b]}
  Die Versenderknoten in dieser Gruppe dürfen nur die Nachricht des Quellknotens kosignieren und weiterleiten. Folglich muss die zu übertragende Nachricht genau die Signatur des Quellknotens enthalten (und keine zusätzlichen Signaturen). In allen anderen Fällen nimmt der jeweilige Knoten nur passiv an der Kommunikation teil. Die Kardinalität der Menge BasisG bezeichnen wir mit b ($b = |BasisG|$).
- Gruppe ErwG = {N[b+1],…, N[b+e]}
  Die Knoten in dieser Gruppe bezeichnen wir als erweiterte Versenderknoten, da sie im Vergleich zu BasisG eine Default-Nachricht generieren und verteilen können. Ein Knoten kosigniert bzw. speichert die Nachricht des Quellknotens genau dann, wenn die Nachricht sowohl vom Quellknoten als auch von mindestens einen Knoten aus BasisG signiert wurde (mehr Details hierzu, siehe Abschnitte 4.5 und 4.6). In allen anderen Fällen wird eine Default-Nachricht (und keine weitere Nachricht) an alle Knoten aus G übertragen. Die Kardinalität der Menge ErwG bezeichnen wir mit e ($e = |ErwG|$).
- Gruppe SendG = {N[0],…, N[n – 1]}
  Alle Knoten in dieser Gruppe nehmen aktiv am Übereinstimmungsprozess teil und zwar durch das Senden einer Nachricht in ihr jeweiliges Sendezeitfenster (genau eine Nachricht pro fehlerfreien Knoten). Die Kardinalität der Menge SendG bezeichnen wir mit n. Es gilt: $SendG = QuellG \cup ForwG$.

- Gruppe PurSinkG = {N[n],…, N[n + k – 1]} mit $k \geq 0$. Für k = 0 gilt |PurSinkG| = 0.
  Die Knoten in dieser Gruppe werden als pure Sinkknoten bezeichnet, da sie stets passiv am Übereinstimmungsprozess teilnehmen und keine Nachrichten versenden. Ihre Funktion besteht ausschließlich im Empfangen und Verarbeiten von Nachrichten. Die Knoten in dieser Gruppe müssen ebenfalls eine Übereinstimmung (bzgl. des Endwerts) erzielen. Die Kardinalität der Menge PurSink ergibt sich zu g – n. Es gilt: PurSink = G \ SendG.
- Gruppe actSinkG = {N[1],…, N[n– 1]}
  alle Knoten in dieser Gruppe müssen eine Übereinstimmung bezüglich des Endwerts erzielen. Sie entspricht die Menge der Knoten, die aktiv am Übereinstimmungsprozess teilnehmen und setzt sich aus den beiden Mengen BasisG und ErwG zusammen. Die Kardinalität der Menge actSinkG ergibt sich daher zu b + e. Es gilt: actSinkG = ForwG.
- Gruppe SinkG = {N[1],… , N[n + k – 1]}
  enthält sämtliche Knoten die eine Übereinstimmung bzgl. des Endwerts erzielen müssen. Die Kardinalität der Menge SinkG ergibt sich folglich zu b + e + g – n. Es gilt: SinkG = G \ QuellG.

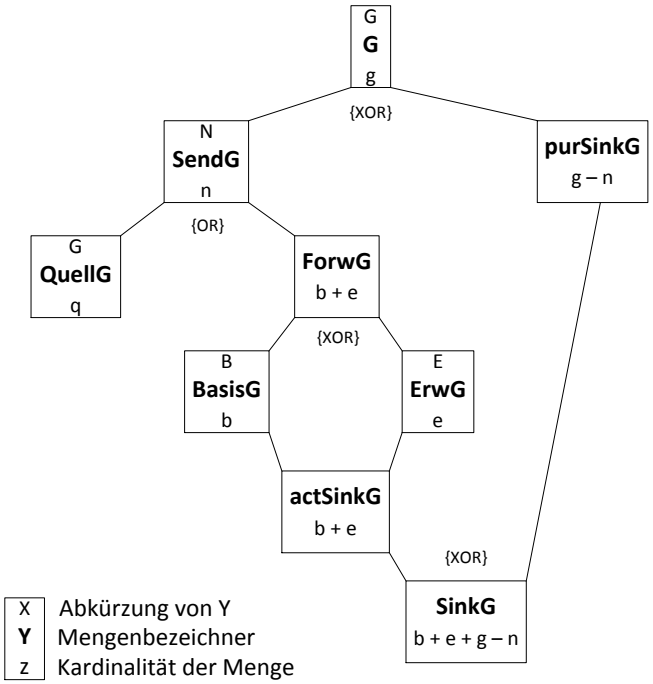


Abbildung 7: Die Darstellung der Knoten in ESSEN als Graph.

Tabelle 1: Parameterwerte b, e, n und g in Abhängigkeit der Anzahl der zu tolerierenden Fehler ***f*** (Bousbiba 2015a).

f	**b =** $f + 1$	**e =** $2(f-1) + max(0, f-2)$	**n =** $3f + max(0, f-2)$	**g =** **n + c²**
1	2	0	3	3+c
2	3	2	6	6+c
3	4	5	10	10+c
4	5	8	14	14+c

[2] c steht für einen konstanten Wert ($c \geq 0$), der die Anzahl der Knoten in der Gruppe purSink angibt.
Hinweis: Die Anzahl der Knoten in der Gruppe purSink hängt nicht von der Anzahl der zu tolerierenden Fehler f ab.

4.4 Nachrichtentypen und Schutzmechanismen gegen Verfälschungen

Das Protokoll ESSEN verwendet digitale Signaturen, um Nachrichten vor nicht erkennbaren Verfälschung und anderen Fehlern wie etwa Maskerade zu schützen. Es wird angenommen, dass jeglicher Angriff auf die Signatur stets detektierbar ist. Andernfalls müssen Lösungen verwendet werden, die auf Signaturen vollständig verzichten. Solche Ansätze benötigen jedoch zur Lösung des Übereinstimmungsproblems in der Regel einen unverhältnismäßig hohen Kommunikationsaufwand. Weiterhin wird angenommen, dass die fehlerhaften Knoten in der Lage sind, „bösartig“ miteinander zu kooperieren, z. B. indem Sie ihre privaten Schlüssel über einen geheimen Kanal untereinander austauschen. Man bezeichnet solche Fehler auch als kooperierende byzantinische Fehler.

Die verschiedenen Nachrichtentypen des Protokolls ESSEN werden im Folgenden näher erläutert.

- Data

  Diese Nachricht enthält den Wert sowie die Signatur des Quellknotens. Alle Knoten aus SinkG müssen eine Entscheidung bzgl. dieses Werts treffen, h. h. den Wert als Endwert akzeptieren oder verwerfen. Im erstgenannten Fall muss gemäß IC1 sichergestellt sein, dass die fehlerfreien Knoten sich für denselben Inhalt entscheiden. Im zweitgenannten Fall entscheiden sich die Knoten für eine systemweit definierte globale Konstante.

- Default

  Nur die Knoten aus ErwG dürfen Default-Nachrichten erzeugen, kosignieren und übertragen. Alle anderen Knoten dürfen empfangene Default-Nachrichten speichern. Eine Default-Nachricht dient primär der Anpassung des Schwellwerts t_E („threshold“, siehe Abschnitt 4.6) für das Weiterleiten der Data-Nachricht. Der Schwellwert übt einen Einfluss auf das Sendeverhalten der Knoten aus ErwG aus. Dadurch, dass fehlerfreie Knoten aus ErwG eine Default-Nachricht senden, wenn sie Data-Nachrichten ablehnen, ist sichergestellt, dass jeder fehlerfreie Knoten aus ErwG in seinem Slot stets genau eine Nachricht sendet. Neben der Funktion der Schwellwert-Übermittlung stellt eine Default-Nachricht zusätzlich sicher, dass die Bedingungen IC1 und IC2 von keinem Knoten verletzt werden (siehe hierzu Abbildung 10).

4.5 Nachrichtenoverhead

Ohne Beschränkung der Allgemeinheit bezeichnen wir die Nachricht, die ein Knoten während der Protokollausführung empfängt, mit RxMsg.

Im Protokoll ESSEN muss jeder Knoten über genügend Speicher verfügen, um bis zu drei Nachrichten (zwei Data- und eine Default-Nachricht) speichern zu können. Die drei Speicherbereiche (jeder Bereich speichert höchstens eine Nachricht ab) werden wie folgt bezeichnet:

- Primärer Nachrichtenpuffer (kurz: PMB, steht für „primary message buffer“)
  Die Nachricht, die in diesem Puffer gespeichert wird (kurz: MsgPMB), enthält die empfangene Data-Nachricht, die die meisten Signaturen (bzw. Kosignaturen) enthält.
- Sekundärer Nachrichtenpuffer (kurz: SMB, steht für „secondary message buffer“)
  Genau wie im primären Nachrichtenpuffer wird auch hier eine Data-Nachricht (kurz: MsgSMB) gespeichert. Ein Knoten speichert eine empfangene Nachricht RxMsg in SMB genau dann, wenn die folgenden vier Bedingungen alle erfüllt sind:
  1. RxMsg enthält mindestens $f + 1$ Signaturen.
  2. RxMsg ist vom Quellknoten signiert.
  3. Sei `sig(msg)` eine Funktion, die die Menge der Knoten aus SendG angibt, welche die Nachricht msg signiert bzw. kosigniert haben. Die Mengendifferenz von `sig(RxMsg)` und `sig(MsgPMB)` liefert als Ergebnis eine nicht leere Menge:

     $$sig(RxMsg) \backslash sig(MsgPMB) \neq \emptyset \qquad 4.2$$

  4. Sei `sigCount(msg)` eine Funktion, die die Anzahl der Knoten wiedergibt, welche die Nachricht msg signiert bzw. kosigniert haben. Die Funktion `sigCount(msg)` liefert also die Kardinalität von `sig(RxMsg)`. Die Nachricht MsgSMB wird durch RxMsg genau dann ersetzt, wenn neben den drei genannten Bedingungen zusätzlich die Bedingung

     $$sigCount(RxMsg) > sigCount(MsgPMB) \qquad 4.3$$

     erfüllt ist.

Falls keine der vier Bedingung erfüllt ist, wird die empfangene Nachricht RxMsg von dem jeweiligen fehlerfreien Empfängerknoten verworfen.

- Default-Nachrichtenpuffer (kurz: DMB, steht für „default message buffer“)
  Die Default-Nachricht, die in SMB gespeichert ist, bezeichnen wir mit MsgDMB. Diese Nachricht darf nur Signaturen aus der Menge ErwG enthalten, folglich gilt:

$$sig(MsgSMB) \subseteq ErwG \; und \; sig(MsgSMB) \cap BasisG = \emptyset \qquad 4.4$$

  Eine neue empfangene Default-Nachricht (kurz: RxDef) ersetzt MsgDMB genau dann, wenn die folgenden zwei Bedingungen erfüllt sind:

  1. RxDef enthält nur Signaturen aus der Menge ErwG
  2. Es gilt:

$$sigCount(RxDef) > sigCount(MsgDMB) \qquad 4.5$$

  In allen anderen Fällen wird die empfangene RxDef Nachricht vom jeweiligen fehlerfreien Knoten verworfen.

Der Inhalt des sekundären Nachrichtenpuffers wird von den fehlerfreien Knoten niemals versendet. Er kommt allenfalls für die finale Übereinstimmungsentscheidung am Ende der Runde zum Einsatz. Im Folgenden wird zunächst das Schwellwertkonzept von ESSEN und dessen Bedeutung für die Übereinstimmung erläutert.

4.6 Das Schwellwertkonzept und seine Anwendung in ESSEN

Die Knoten aus ErwG treffen eine Entscheidung bzgl. der zu versendenden Nachricht in Abhängigkeit ihres lokalen gespeicherten Schwellwertes t_E. Abhängig vom Wert t_E entscheidet sich ein fehlerfreier Knoten bei der Auswahl der weiterzuleitenden Nachricht entweder für MsgPMB oder für MsgDMB, aber niemals für beide gleichzeitig.

Der Schwellwert wird von jedem Knoten aus ErwG lokal bestimmt, d. h. es gibt keine gemeinsame globale Sicht auf den Schwellwert. Der Schwellwert t_E eines Knoten ergibt sich aus der Anzahl der Signaturen, die die Nachricht MsgDMB (t_E = `sigCount(MsgDMB)`) enthält.

Die Entscheidung, die Data-Nachricht nicht weiterzuleiten, führt zur (Ko-)Signierung und Weiterleitung der MsgDMB Nachricht. Für die Nachricht, die als nächste empfangen wird,

wird dann die Signaturanzahl der in MsgDMB gespeicherten Nachricht zugrunde gelegt und aus dieser der neue Wert von t_E bestimmt. Bei Empfang einer Default-Nachricht, sofern sie die Bedingungen in Abschnitt 4.5 erfüllt, aktualisieren die Knoten aus ErwG ihren Schwellwert, indem sie den Parameter t_E auf den Wert `sigCount(MsgDMB)` setzen. Alle anderen Knoten (z. B. die Knoten aus BasisG) speichern nur die empfangene Default-Nachricht, sofern die empfangene Nachricht mehr Signaturen enthält als die zuvor gespeicherte Nachricht MsgDMB (siehe Abschnitt 4.5). Abhängig von der Anzahl der gespeicherten Signaturen in MsgPMB und MsgDMB entscheidet sich der Sendeknoten entweder dafür, die gespeicherte Default- oder die Data-Nachricht zu übertragen: MsgPMB wird von einem Versenderknoten aus ErwG genau dann übertragen, wenn die Ungleichung

$$sigCount(MsgPMB) > sigCount(MsgDMB) \quad (4.6)$$

erfüllt ist. In allen anderen Fällen, sendet der jeweilige Versenderknoten die MsgDMB-Nachricht.

Dabei erfüllt die Default-Nachricht aus Sicht des (fehlerfreien) Versenderknoten den folgenden Zweck:

1. Informiere alle fehlerfreien Knoten darüber, dass zum aktuellen Zeitpunkt keine gültige Data-Nachricht empfangen worden ist bzw. eine Default-Nachricht mit mehr Signaturen empfangen wurde.
2. Aktualisiere den Schwellwert t_E aller Knoten in ErwG durch Weiterleiten der Default-Nachricht an alle Knoten.
3. Speichere den neuen empfangenen Schwellwert, sofern die Bedingungen zur Speicherung der Default-Nachricht erfüllt sind.

Zusammenfassend kann man festhalten, dass das Schwellwertkonzept in ESSEN einen Schutzmechanismus darstellt, der fehlerfreie Knoten daran hindern soll, eine Data-Nachricht zu übertragen, wenn aus der Übertragung der Data-Nachricht eine Verletzung der Eigenschaften IC1 und IC2 (je nach Fehlverhalten) der Knoten resultieren kann.

4.7 Die Herleitung der Knotenzahl in ESSEN

Zur Tolerierung von f Fehlern benötigt das Protokoll ESSEN $n \geq 3f + max(0, f - 2)$ Knoten. Diese Formel wird im Folgenden hergeleitet.

Die Knoten in ForwG werden in zwei Gruppen BasisG und ErwG unterteilt. Beide Gruppen bilden eine disjunkte Menge (siehe Abschnitt 4.3). Die Knoten in der Gruppe BasisG leiten die Data-Nachricht des Quellknotens als erste weiter (jeder Knoten nacheinander). Diese Gruppe enthält mindestens $f + 1$ Knoten (siehe Fall A1 unten). Die nachfolgenden Knoten in der Gruppe ErwG leiten die Data-Nachricht des Quellknotens weiter oder senden eine Default-Nachricht (siehe Abschnitt 4.5). Auch hier gilt, dass die Knoten nacheinander senden. Um die Knotenzahl für beide Gruppen zu bestimmen, müssen die folgenden Fälle (siehe Fall A1 und Fall A2) betrachtet werden.

Fall (**A1**): Der Quellknoten N[0] ist fehlerfrei.

Um IC1 und IC2 sicherzustellen, muss die Nachricht (siehe Beschreibung zu Gruppe ErwG in Abschnitt 4.3 bzw. die Bedingung NC2 in Abschnitt 5.1) des Quellknotens von mindestens einem weiteren fehlerfreien Knoten in BasisG empfangen worden sein, da andernfalls kein Knoten aus ErwG die Data-Nachricht des Quellknotens akzeptiert. Dies ist genau dann gewährleistet, wenn die Gruppe BasisG aus mindestens $f + 1$ Knoten besteht, damit mindestens ein fehlerfreier Knoten unter ihnen ist. Da der Aufwand des Protokolls minimal sein soll, wird für BasisG die minimale Knotenanzahl $f + 1$ gewählt.

Weiterhin muss die Nachricht laut AG1 (siehe Abschnitt 5.1) insgesamt von mindestens $f + 1$ Knoten signiert worden sein. Da die Gruppe BasisG im ungünstigsten Fall f fehlerhafte Knoten enthalten kann, muss demzufolge die Gruppe ErwG aus mindestens $f - 1$ Knoten bestehen. Die geforderten $f + 1$ Signaturen werden dann vom Quellknoten, von einem Knoten aus BasisG und von $f - 1$ Knoten aus ErwG gebildet.

Fall (**A2**): Der Quellknoten N[0] ist fehlerhaft.

Die Anzahl der Knoten in ErwG für wachsende f ($f >> 1$) ist im Vergleich zu Fall A1 schwieriger zu ermitteln, da das Sendeverhalten (siehe Abschnitt 4.6) eines fehlerfreien Knotens von den zuvor empfangenen, möglicherweise fehlerhaft gesendeten Nachrichten bestimmt wird. Daher wurde im ersten Schritt die Anzahl der Knoten für $f \leq 3$ mit Hilfe eines Modellcheckers formal verifiziert (siehe Abschnitt 5.2) und für $f \leq 6$ simulativ validiert (siehe Abschnitt 5.3). Die notwendige Anzahl der Knoten in den Gruppen BasisG und ErwG für $f \leq 6$ ist in Tabelle 2 aufgelistet. Die Tabelle dient als Grundlage zur Bestimmung der Knotenzahl in ErwG, auf die im nachfolgenden näher eingegangen wird.

Tabelle 2: Parameterwerte b und e in Abhängigkeit von der Anzahl der zu tolerierenden Fehler ***f*** (Die Werte sind mit Hilfe von formaler Verifikation und simulativer Analyse ermittelt worden).

f	**Anzahl der Knoten in BasisG (b_f)**	**Anzahl der Knoten in ErwG (e_f)**	**Anzahl der Knoten in ForwG ($b_f + e_f$)**
1	$b_1 = 2$	$e_1 = 0$	2
2	$b_2 = 3$	$e_2 = 2$	5
3	$b_3 = 4$	$e_3 = 5$	9
4	$b_4 = 5$	$e_4 = 8$	13
5	$b_5 = 6$	$e_5 = 11$	17
6	$b_6 = 7$	$e_6 = 14$	21
Die Werte b_i und e_i heißen Folgenglieder mit $1 \leq i \leq f$. Sie bestimmten die notwendige Anzahl der Knoten für die Gruppen BasisG und ErwG. Der Index i in e_i bzw. b_i gibt die Anzahl f der zu tolerierenden Fehler an.			

Da die Anzahl der Knoten in ErwG (siehe Tabelle 2) von der Anzahl der zu tolerierenden Fehler f abhängt, muss es demzufolge für die Folgenglieder e_f ein Bildungsgesetzt $<e_f>$ existieren. Da das Bildungsgesetzt $<e_f>$ jedoch zunächst unbekannt ist, muss es aus den Folgegliedern e_1 bis e_6 aus Tabelle 2 hergeleitet werden (für mehr Information siehe Abschnitt 5.3). Für die Folge e_i mit $i \leq 6$ lässt sich das folgende Bildungsgesetzt herleiten:

$$< e_1 > = 0$$

$$< e_2 > = 2$$

...

$$< e_f > = \begin{cases} 2 \cdot (f-1) + max(0, f-2), & \text{für } f \geq 3 \\ 2 \cdot (f-1), & sonst \end{cases} \Rightarrow$$

$$b = \begin{cases} 2 \cdot (f-1) + max(0, f-2), & \text{für } f \geq 3 \\ 2 \cdot (f-1), & sonst \end{cases} \blacksquare$$

Interpretiert man das Bildungsgesetzt richtig, so stellt man fest, dass zur Tolerierung von f Fehlern $3f - 1 + max(0, f-2)$ redundante Knoten benötigt werden. Dass diese Mindest-Knotenanzahl hinreichend für ein fehlertolerantes Protokollverhalten ist, wird durch den formalen Korrektheitsbeweis in Kapitel 6 gezeigt.

4.8 Tolerierte Fehler

Um das Übereinstimmungsproblem (Lamport et al. 1982) in einer Runde zu lösen, werden mindestens $n \geq 3f + max(0, f - 2)$ Knoten benötigt (Quellknoten mit eingeschlossen). Davon können höchstens f_N fehlerhaften Knoten sich willkürlich fehlerhaft verhalten mit $0 \leq f_N \leq f$. Dies beinhaltet unter anderem die Kooperation mit anderen fehlerhaften Knoten, das Versenden von mehreren Nachrichten mit falschen Werten, etc. (siehe Kapitel 2).

Bezüglich des (Broadcast) Kanals wird angenommen, dass sich dieser höchstens gutartig Byzantinisch verhält – mit der Einschränkung, dass sein Einfluss auf die Kommunikation zwischen den fehlerfreien Knoten durch einen Parameter $f_B$nach oben beschränkt und der Wert von f_B bekannt ist. Hierbei drückt der Parameter f_B die maximale Anzahl der Slots aus, die durch die fehlerhaften Kanäle in der Weise beeinträchtigt werden können, dass ein korrekter Broadcast der Nachrichten in einem solchen Slot nicht mehr gegeben ist. Infolgedessen können im ungünstigsten Fall innerhalb eines Slots bis zu maximal g – 1 Nachrichtenausfälle auftreten. Dies bedeutet, dass bis zu g – 1 Empfänger eine Nachricht nicht korrekt empfangen. Folglich ergibt sich der Parameter f zu

$$f = f_N + f_B \qquad 4.7$$

wobei f_N die Anzahl der fehlerhaften Knoten und f_B die Anzahl der Blockierungsfehler wiedergibt (Formel 4.7 wurde bereits in Abschnitt 2.2.1 eingeführt).

Da die Slotausfälle zufällig auftreten, stellt sich nun die Frage, wie das Übereinstimmungsproblem im Falle eines fehlerfreien Quellknotens, dessen Sendeslot vom fehlerhaften Kanal beeinträchtigt ist, gelöst werden kann, sodass die Bedingung IC2 nicht verletzt wird?

Wenn man nicht deterministische Lösungsansätze und asynchrone Verfahren ausschließt, lautet die Antwort hierzu: Das Übereinstimmungsproblem ist im Falle eines fehlerhaften Kanals nicht lösbar, wenn bezüglich des Broadcasts der Nachricht, die vom fehlerfreien Quellknoten gesendet wird, keine Aussagen getroffen werden können (siehe Unmöglichkeitsergebnis von (Santoro und Widmayer 2007, 1989)). Der fehlerhafte Kanal könnte den fehlerfreien Quellknoten quasi vom restlichen System abschneiden. In diesem Fall würde das Sendeverhalten dem eines Byzantinisch fehlerhaften Knotens ähneln. Wenn man diesen Fall zugrunde legt und den Quellknoten als „Folgefehler“ des fehlerhaften Kanals betrachtet, verlangt das Übereinstimmungsproblem lediglich die Einhaltung der Bedingung IC1, da die

Voraussetzungen für IC2 nicht gegeben sind. Daraus lässt sich schließen, dass in diesem Fall (tatsächlich fehlerfreier Quellknoten kombiniert mit einem fehlerhaften Sendeslot) ebenfalls Fehlerszenarien existieren, die zu einer Verletzung von IC2 führen. Anders ausgedrückt: Sofern keine redundanten Kanäle oder Slots zu Verfügung gestellt werden, bilden die Knoten und die dazugehörigen Sendslots einen gemeinsamen Fehlerbereich (Echtle 1990b).

Entsprechend bezeichnet der Begriff „Slotfehler“ einen in einem Slot sich auswirkenden Fehler, der sich auf alle verfügbaren Kanälen auswirkt (siehe ggf. FM9 in Abschnitt 2.3.7). Ein Slotfehler tritt genau dann auf, wenn entweder der Knoten fehlerhaft ist, der in dem betreffenden Slot sendet, oder die Kanäle alle fehlerhaft sind und einen Blockierungsfehler verursachen. Unabhängig davon müssen fehlerfreie Knoten, bei denen ein Slotfehler vorliegt, ebenfalls Übereinstimmung (bzgl. des Endwerts) gemäß IC1 und IC2 erzielen.

Diese Problematik soll im Folgenden durch ein einfaches Beispiel skizziert werden. Bevor jedoch das Beispiel aufgeführt wird, müssen zuvor einige Notationen zur Beschreibung von Signaturen eingeführt werden, auf die im Folgenden näher eingegangen wird.

Sei v ein beliebiger Wert und N[i] ein beliebiger Knoten aus N = {N[0],…, N[n – 1]}. Dann verwenden wir die Notation $\sigma_i(v)$, um ausdrücken, dass der Wert v von einem Knoten N[i] signiert ist. Diese Vereinbarung soll ebenfalls zur Darstellung von Kosignaturen verwendet werden.

Bei der Kosignierung (auch bekannt als Mehrfachsignierung) unterscheidet man generell zwischen abhängigen und unabhängigen Kosignaturen. Als abhängige Kosignaturen bezeichnen wir solche, bei denen in die Generierung der Kosignatur die bereits zuvor erzeugte Signatur $\sigma_i(v)$ der Nachricht einfließt. Typische Vertreter solcher Signaturverfahren sind z. B. (Echtle und Kimmeskamp 2010; Echtle 1999). Zur Darstellung von abhängigen Kosignaturen verwenden wir die folgende Notation:

$$\left(\sigma_i \circ \sigma_j\right)(v) = \sigma_i\left(\sigma_j(v)\right) \; mit \; i \neq j \qquad 4.8$$

Anmerkung: Die Notation $\left(\sigma_i \circ \sigma_j\right)(v)$ bzw. $\sigma_i\left(\sigma_j(v)\right)$ beschreibt die Komposition (bzw. Verkettung) von zwei Signaturfunktionen σ_i und σ_j, bei der eine Signatur $\sigma_i(v)$ durch eine neue Kosignatur $k = \left(\sigma_i \circ \sigma_j\right)(v)$ mit gleicher Bitbreite ersetzt wird.

Signaturverfahren, die zur Kosignaturerstellung nur den zu signierenden Wert v verwenden, bezeichnen wir als unabhängige Kosignaturen. Zur Darstellung von unabhängigen Kosignaturen verwenden wir als Notation die folgende Darstellung:

$$v{:}V(i,j)\ mit\ i \neq j\ und\ V{:}D \subseteq \mathbb{N}^{n-1} \rightarrow \mathbb{N} \qquad 4.9$$

Die Funktion $V{:}D \subseteq \mathbb{N}^{n-1} \rightarrow \mathbb{N}$ wird als Verschmelzungsfunktion (siehe Kapitel 7) bezeichnet. Das Ziel der Verschmelzungsfunktion ist es, mehrere unabhängige Signaturen (hier: Signatur i und Signatur j) auf eine neue Signatur k = v:V(i, j) ϵ $\mathbb{N}$ abzubilden. Es sei angemerkt, dass die Verschmelzungsfunktion $V{:}D \subseteq \mathbb{N}^{n-1} \rightarrow \mathbb{N}$ kommutativ ist, d. h. es gilt zusätzlich $v{:}V(i,j) = v{:}V(j,i)$für beliebige i, j ϵ $\mathbb{N}$.

Anders ausgedrückt: Abhängige Signaturen/Kosignaturen werden sequentiell erzeugt und jeder Signaturwert fließt in die nachfolgende Kosignatur ein. Unabhängige Kosignaturen werden hingegen nebenläufig erzeugt und danach zu einem einzigen Signaturwert verschmolzen.

Die beiden Ausdrücke $(\sigma_i\left(\sigma_j(v)\right)$ und $v{:}V(i,j)$ stellen jeweils eine Nachricht dar, bei der ein Wert v von zwei Knoten N[i] und N[j] mit N[i] ≠ N[j] signiert wurde. Auf diese Weise lassen sich Mehrfachsignaturen für eine beliebige Knotenzahl kompakt darstellen (näheres hierzu siehe Tabelle 3).

Tabelle 3: Einfache Darstellung von Mehrfachsignaturen.

Knoten, die die Nachricht v signiert haben	$\sigma_i(v)$	$V{:}D \subseteq \mathbb{N}^n \rightarrow \mathbb{N}$
N[0]	$\sigma_0(v)$	$v{:}V(0) = \sigma_0(v)$
N[0], N[1]	$\sigma_1\left(\sigma_0(v)\right)$	v:V(0,1)
N[0], N[1], N[2]	$\sigma_2\left(\sigma_1\left(\sigma_0(v)\right)\right)$	v:V(0,1,2)
…	…	…
N[0], …, N[n – 1]	$(\sigma_{n-1} \circ \ldots \circ \sigma_0)(v)$	$v{:}V(0,\ldots,n)$

Nehmen wir an, dass der Slot des fehlerfreien Quellknotens durch den fehlerhaften Kanal beeinflusst wird. Weiter nehmen wir an, dass die Nachricht des Quellknotens (bedingt durch den fehlerhaften Kanal) von keinem der Knoten empfangen wird (siehe Abbildung 3). Da die

Nachricht des Quellknotens von keinem der Knoten weitergeleitet werden kann, entscheiden sich alle fehlerfreien Sinkknoten für die globale Konstante als Ersatzwert. Dies steht im Widerspruch zu IC2.

Es gibt drei Möglichkeiten, um dieses Problem zu umgehen:

1. Es wird angenommen, dass Fehler nur in den Knoten auftreten. Somit liegt der Kanal im Perfektionskern und es gilt $f = f_N$,
2. Dem Knoten stehen redundante Kanäle zu Verfügung. Es gibt mindestens $f_B + 1$ Kanäle,
3. Dem Quellknoten werden $f_B + 1$ redundante Slots bereitgestellt.

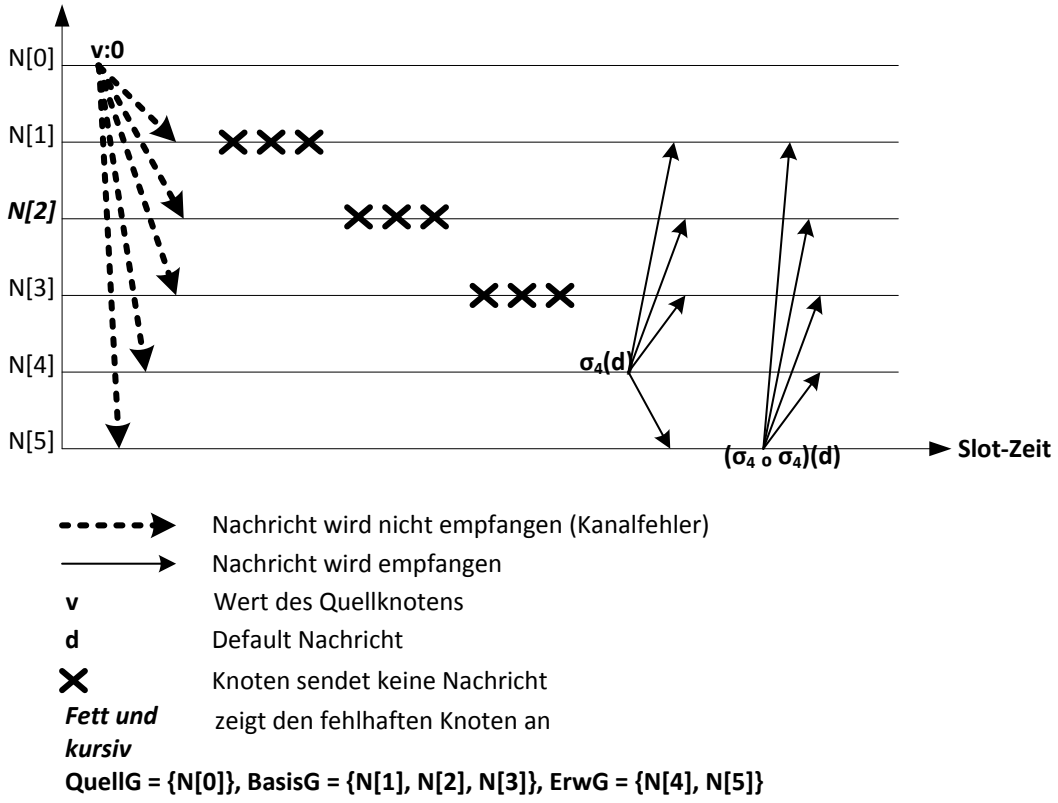


Abbildung 8: Übereinstimmungsproblem im Falle von $f_N = 1$ und $f_B = 1$.

In allen anderen Fällen lässt sich das Problem zumindest nicht deterministisch lösen. Es sei hier der Vollständigkeit halber erwähnt, dass Slotfehler ähnlich zu Omission-Fehlern zu zufälligen Zeitpunkten auftreten können. Folglich können sich die Slotfehler von Runde zu

Runde unterscheiden bzw. sich auf andere Knoten auswirken. Da, wie bereits erwähnt, die Knoten und Sendeslots einen gemeinsamen Fehlerbereich bilden, treten folglich im ungünstigsten Fall bis zu f_N Knotenfehler sowie f_B Blockierungsfehler pro Runde (siehe Definition 2.1) auf. In anderen Worten: Es müssen neben den f_N Knotenfehler zusätzlich f_B Blockierungsfehler toleriert werden, die als „Folgefehler" durch den fehlerhaften Kanal ein ähnliches Verhalten wie ein gutartiger Byzantinischer Knoten-Fehler aufweisen. Somit gibt es kein (deterministisches) Übereinstimmungsprotokoll mit Toleranzgrad f_N, welches das Übereinstimmungsproblem in Anwesenheit eines einzigen Blockierungsfehler löst (siehe obiges Beispiel). Folglich muss mindestens einer der drei genannten Vorschläge umgesetzt werden, wenn die Bedingung IC2 auch in Anwesenheit von Blockierungsfehler stets gewährleistet werden soll.

4.9 Algorithmische Beschreibung des Protokolls

Dieser Abschnitt präsentiert das Protokoll ESSEN (siehe **Algorithmus 1**), welches das Übereinstimmungsproblem unter Einhaltung der Bedingungen IC1 und IC2 gemäß (Lamport et al. 1982) stets garantiert. Der Algorithmus wird innerhalb einer Runde wie folgt ausgeführt. Zuerst sendet der Quellknoten N[0] seinen signierten Wert in Slot 0 (Zeile 13, **Algorithmus 1**). Diese Nachricht wird ausschließlich von den Knoten aus BasisG akzeptiert und gespeichert. Alle anderen Knoten (SinkG\BasisG) verwerfen die Nachricht (siehe auch Abbildung 9).

In Slot i mit $1 \leq i \leq b$ sind die Knoten aus BasisG aktiv (Zeile 14, **Algorithmus 1**). Ein Knoten N[i] aus BasisG kosigniert und leitet seine Nachricht MsgPMB an alle anderen Knoten weiter. Wenn ein Knoten N[i] keine Data-Nachricht empfangen hat, bevor sein Sendeslot beginnt, dann nimmt dieser Knoten N[i] nur passiv an der Kommunikation teil (empfängt ausschließlich Nachrichten).

In den nachfolgenden Slots j mit $b + 1 \leq j \leq n - 1$ nehmen die Knoten aus ErwG aktive an der Kommunikation teil (senden Nachrichten in ihrem jeweiligen Slot). Ein fehlerfreier Knoten N[j] entscheidet sich entweder eine Data- oder eine Default-Nachricht zu übertragen, aber niemals für beide gleichzeitig (Zeile 15 – 16, **Algorithmus 1**). Eine Data-Nachricht wird genau dann gesendet, wenn:

1. sie vom Quellknoten als erstes und von mindestens einem Versenderknoten aus BasisG signiert wurde, und

2. die folgende Ungleichung erfüllt ist:

$$sigCount(MsgPMB) > sigCount(MsgDMB) \quad 4.10$$

In allen anderen Fällen sendet der jeweilige Knoten eine signierte Default-Nachricht. Am Ende der Runde zählt jeder fehlerfreie Knoten die Anzahl der Signaturen in MsgPMB. Wenn

$$sigCount(MsgPMB) < f + 1 \quad 4.11$$

gilt, dann entscheidet sich der jeweilige Knoten für eine vordefinierte globale Konstante als Ersatzwert.

Die Bedingung für das Akzeptieren oder Verwerfen der Data-Nachricht im Falle von

$$sigCount(MsgPMB) > f \quad 4.12$$

wird im Folgenden diskutiert.

Das Protokoll ESSEN verwendet eine „Reinigungsfunktion“ (kurz: CF), die das relative Komplement der im Speicher hinterlegten Nachrichten (MsgPMB und MsgSMB) bestimmt. Die Anzahl der Signaturen der beiden Differenzmengen sig(MsgSMB) \ sig(MsgDMB) und sig(MsgPMB) \ sig(MsgDMB) werden in

$$sigCountPMB = sigCount(sig(MsgPMB) \backslash sig(MsgDMB)) \quad 4.13$$

$$sigCountSMB = sigCount(sig(MsgSMB) \backslash sig(MsgDMB)) \quad 4.14$$

gespeichert.

Nehmen wir Beispielsweise an, dass im Falle von $f_N = 3$ und $f_B = 0$ ein fehlerfreier Knoten N[i] am Ende einer Runde die folgenden Nachrichten im Speicher hinterlegt hat (siehe hierzu auch Abbildung 10):

- Die Nachricht MsgPMB ist von den folgenden Knoten kosigniert:
  N[0], N[1], N[3], N[4], N[5], N[7],
- MsgSMB enthält die Signaturen der Knoten N[0], N[2], N[5], N[6] und
- MsgDMB enthält die Signaturen der Knoten N[5], N[6], N[7], N[8], N[9].

In diesem Beispiel würde der Knoten N[i], die folgenden zwei Werte bestimmen:

$$sigCountPMB = sigCount(\{0,1,3,4,5,7\}\backslash\{5,6,7,8,9\})$$

$$\Rightarrow sigCountPMB = sigCount(\{0,1,3,4\}) = 4$$

$$sigCountSMB = sigCount(\{0,2,5,6\}\backslash\{5,6,7,8,9\})$$

$$\Rightarrow ssigCountSMB = sigCount((\{0,2\}) = 2$$

In der letzten Gleichung stellen die in geschweifter Klammer enthaltenen natürlichen Zahlen Signaturen dar, wobei jede Zahl den Index des jeweiligen Signierknotens angibt.

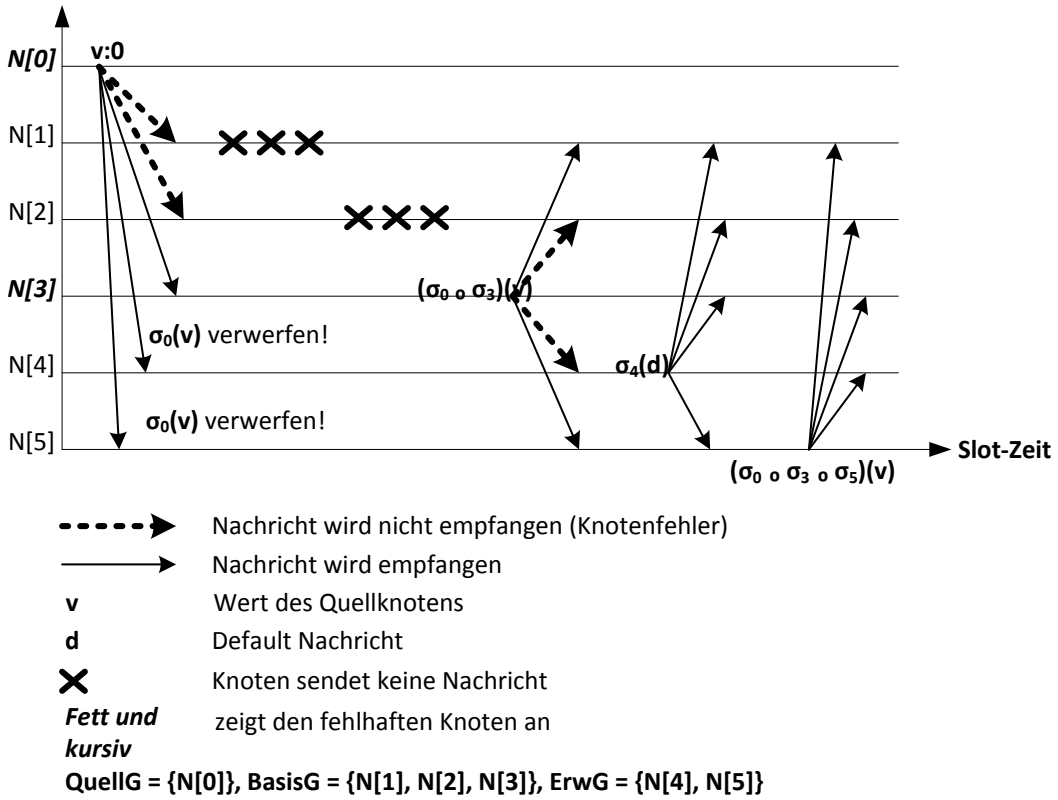


Abbildung 9: Eines von vielen möglichen Nachrichtentransferszenarien für $f_N = 2$ und $f_B = 0$.

Für $f \leq 3$ würde der Knoten N[i] die Data-Nachricht als Endwert akzeptieren und für $f > 3$ sich für die vordefinierte globale Konstante als Ersatzwert entscheiden.

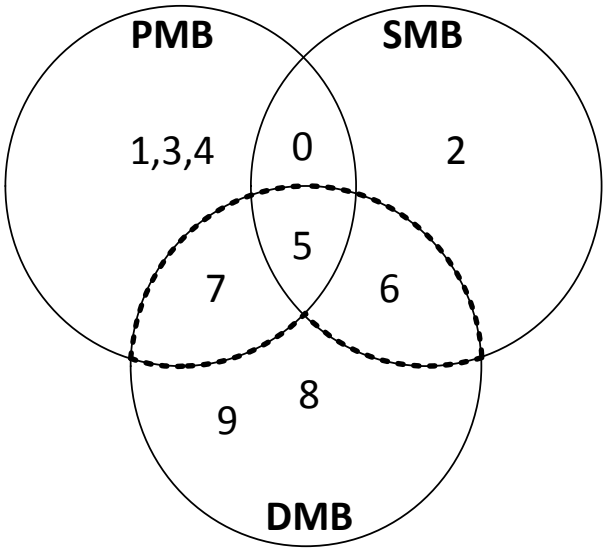


Abbildung 10: Das Konzept der „Reinigungsfunktion“ in ESSEN, dargestellt durch die Mengen der signierenden Knoten.

Die Signaturen im gestrichelten Bereich (die Menge {5, 6, 7} in Abbildung 10) enthalten die Menge der als fehlerhaft detektierten Knoten. Jeder fehlerfreie Knoten stellt sicher, dass die finale Übereinstimmungsentscheidung ausschließlich auf der Basis der fehlerfreien (oder genauer: nicht als fehlerhaft detektierten) Knoten getroffen wird, da andernfalls IC1 und IC2 nicht mehr unter allen möglichen Fehlerszenarien garantiert werden könnte.

Am Ende der Runde führt jeder fehlerfreie Knoten die folgenden sechs Schritte aus:

S1: Prüfe am Ende der Runde die in MsgPMB gespeicherte Data-Nachricht. Falls die Data-Nachricht mindestens die Signatur des Quellknotens enthält, gehe zu Schritt 2. Andernfalls gehe zu Schritt 6. (vergl. Zeile 34, **Algorithmus 1**).

S2: Bestimme die Anzahl der Signaturen in MsgPMB. Wenn die gespeicherte Nachricht mindestens $f + 1$ Signaturen enthält, gehe zu Schritt 3. Andernfalls gehe zu Schritt 6. (vergl. Zeile 34, **Algorithmus 1**).

S3: Führe die Reinigungsfunktion auf beide Nachrichten MsgPMB und MsgSMB aus, so dass nur solche Signaturen gezählt werden, die keine mit der Nachricht MsgDMB gemeinsame Signaturquellen besitzen. Anschließend gehe zu Schritt 4. (vergl. Zeile 35 – 40, **Algorithmus 1**).

S4: Enthält die MsgPMB-Nachricht nach Schritt 3 mindestens f Signaturen bzw. die MsgSMB Nachricht mindestens $f + 1$ Signaturen, gehe zu Schritt 5. Andernfalls gehe zu Schritt 6. (vergl. Zeile 41, **Algorithmus 1**).

S5: Entscheide für den Inhalt der in MsgPMB gespeicherten Nachricht als finalen Übereinstimmungsendwert (vergl. Zeile 43, **Algorithmus 1**).

S6: Entscheide für die vordefinierte globale Konstante als finalen Übereinstimmungsendwert (vergl. Zeile 43, **Algorithmus 1**).

Zusammenfassend: Ein fehlerfreier Knoten entscheidet sich für eine vordefinierte globale Konstante genau dann, wenn eine der beiden Bedingungen in Schritt 1 oder in Schritt 4 nicht erfüllt wird. In allen anderen Fällen wird der in MsgPMB lokal gespeicherte Inhalt vom jeweiligen fehlerfreien Knoten als finaler Übereinstimmungsendwert genommen.

Algorithmus 1: Byzantinischer Übreeinstimmungsalgorithmus für Knoten N[i], wobei i = 0 für den Quellknoten und i ϵ {1,..., n – 1} für ein Versenderknoten steht.

```
Globale Definition:
01# const int f; // Anzahl der zu tolerierenden willkürlichen Fehler.
02# const int n = 3(f − 1) + max(0, f − 2); // Die Gesamtzahl der Knoten.
03# const int m = n – 1;  // Gesamtzahl der Versenderknoten.
04# sigKey u0, ..., un; // globale öffentliche Schlüssel für Signaturtest
Lokale Definition:
05# const int i; // Lokaler Knotenindex. i = 0 ist der Quellknoten.
06# sigKey s; // private Signierschlüssel zur Signiergenierung.
07# Value a0 = input(); // der zu verteilende Wert, nur im Quellknoten.
08# SignedMsg a = i=0 ? a0 : null; // Primäre Data-Nachricht.
09# SignedMsg b = null; // Sekundäre Data-Nachricht.
10# SignedMsg d = newMsg(DEFAULT); // Leere Default-Nachricht.
11# int tE = 1; // Schwellwert in Gruppe ErwG: Amzahl der notwendigen Sign.
12# Value z = DEFAULT; // Finaler Wert. null repräsentiert den Defaultwert.
N[i] ϵ QuellG zu Beginn des eigenen Slots:
13# signSend(a);
N[i] ϵ BasisG zu Beginn des eigenen Slots:
14# if(sigCount(a) > 0) signSend(a);
N[i] ϵ ErwG zu Beginn des eigenen Slots:
15# if(sigCount(a) > tE) signSend(a);
16# else signSend(d);
Knoten N[i] während Slot j, für iϵ{1, ..., n – 1} und jϵ{1, ..., n – 1}
17# SignedMsg y = receive(); // Empfängt ebenfalss die eigen Nachricht
18# if (valueOf(y) != DEFAULT && sigCount(y) > sigCount(a))
```

```
19# {if (unequal(a, y))  b = null;
20#  a = y;
21# }
22# else if (valueOf(y) == DEFAULT && j > |BasisG| &&
23#          sigCount(y) > sigCount(d))
24# {d = y; t_E = max(t_E, sigCount(y));
25# }
26# else if (valueOf(y) != DEFAULT && sigCount(y) ≥ f + 1 &&
27#          sigCount(b) < sigCount(y) && equalContent(a, y)) b = y;
28# else if (valueOf(y) == DEFAULT && j > |BasisG| &&
29#          sigCount(y) > sigCountd(d))
30# {d = y; t_E = max(t_E, sigCount(y));
31# }
32# else if (valueOf(y) != DEFAULT && sigCount(y) ≥ f + 1 &&
33#          sigCount(b) < sigCount(y) && equalContent(a, y)) b = y;
```

Knoten N[i] schreibt nach dem letzten Slot j = |SendG| den finalen Endwert in die Variable z

```
34# if (a == null || sigCount(a) ≥ f + 1)
35# {for (int k = 1 to n) //Entfernt identische Signaturen (Default, Data).
36#  {if(N[k] hat d signiert)
37#   {if (N[k] hat a signiert) a = a \ {Signatur von N[k]};
38#    if (N[k] hat b signiert) b = b \ {Signatur von N[k]};
39#   }
40#  }
41#  if (sigCount(a) ≥ f || sigCount(b) ≥ f + 1) z = valueOf(a);
42# }
43# output(z); // Wenn z = valueOf(a) nicht ausgeführt wird, dann
                 entscheidet sich der Knoten für den Defaultwert.
```

Lokale Methoden:

```
44# void signSend(SignedMsg msg)const int i; ist der Quellknoten.
45# {msg = msg.sign(s); // signierte Nachricht mit Signierschlüssel s.
46#  send msg via broadcast;
47# }
48# SignedMsg receive(void)
49# {return message received from broadcast channel;
50# }
51# int sigCount(SignedMsg msg)
52# {if (msg == null || syntacticallyWrong(msg) || !validSig(msg))return 0;
53#  int cnt = 0;
54#  if (valueOf(msg) != DEFAULT) return 0;
55#  for (int i = |BasisG| + 1; to n) // nur Knoten aus ErwG.
56#  { if (msg.isSigned(u_i)) cnt++;
57#  }
58#  return cnt;
59# }
60# boolean validSig(SignedMsg msg)
61# {if (valueOf(msg) != DEFAULT)
```

```
62#  { return msg.isSigned(u0) && ∀i ∈ ℕ0 und N[i] ∈ BasisG: msg.isSigned(ui) = false;
63#  }
64#  else return ∀i∈ℕ0 mit N[i]∈SendG: msg.isSigned(ui) = true ⟹ N[i]∈ErwG
65# }
66# Value valueOf(SignedMsg msg)
67# { return value of msg without signatures;
68# }
69# boolean equalContent(SignedMsg msgA, SignedMsg msgB)
70# {return msgA != null && msgB != null && valueOf(msgA) == valueOf(msgB);
71# }
72# boolean unequalContent(SignedMsg msgA, SignedMsg msgB)
73# { return !equalContent(msgA, msgB);
74# }
```

4.10 Vergleich der Kommunikationskomplexität

In diesem Abschnitt wird das Protokoll ESSEN mit anderen wissenschaftlichen Arbeiten zu Übereinstimmungsprotokollen (Jochim und Forest 2010; Echtle; Hsieh und Chiang 2011; Khosravi und Kavian 2012) bzgl. der notwendigen Knotenzahl, des Nachrichtenspeicheroverheads sowie des gesamten Kommunikationsoverheads verglichen. Die Ergebnisse sind in Abbildung 11 bis Abbildung 13 dargestellt. Es wurden nur solche Protokolle für den Vergleich in Betracht gezogen, die eine hohe Fehlertoleranzfähigkeit aufweisen und eine sehr effizient Lösung (bzgl. der Kommunikationskomplexität) zum Übereinstimmungsprotokoll als Ziel hatten.

Außerdem wurde für den Vergleich eine angepasste Variante des Pendel-Protokolls verwendet, welche Übereinstimmung in eine Runde statt in $f + 1$ Runden erreicht (genaueres siehe (Echtle)). Im Weiteren bezeichnen wir dieses Protokoll als Pendulum*. Durch den Wegfall der f Runden kann es in Pendulum* (im Vergleich zu ursprünglichen Pendel-Protokoll) passieren, dass sich die fehlerfreien Knoten statt für einen fehlerfreien für einen fehlerhaften Wert entscheiden. Nichtsdestotrotz wird Pendulum* anstatt des in (Echtle) beschriebenen Pendel-Protokolls für den Vergleich herangezogen. Der Grund hierfür ist, dass das Protokoll ein absolutes Minimum an Nachrichtenübertragungen benötigt, um das Übereinstimmungsproblem zu lösen. Daher eignet sich Pendulum* hervorragend, um die Güte eines entwickelten bzw. bereits existierenden Übereinstimmungsprotokolls zu bestimmen, und zwar durch Gegenüberstellung des Nachrichtenoverheads mit dem (theoretisch) minimal möglichen. Außerdem wurden zusätzlich zu Pendulum* noch die folgenden (unveränderten) Protokolle für den Vergleich herangezogen:

- Reaching Agreement in a Distributed Environment in Absence of Omission Faults (kurz: AAO, (Khosravi und Kavian 2012)),
- A new Solution for the Byzantine Agreement Problem (kurz: EAC, (Hsieh und Chiang 2011)),
- An Efficient Implementation of the SM Agreement Protocol for a Time-Triggered Communication Systems (kurz: SM, (Jochim und Forest 2010)).

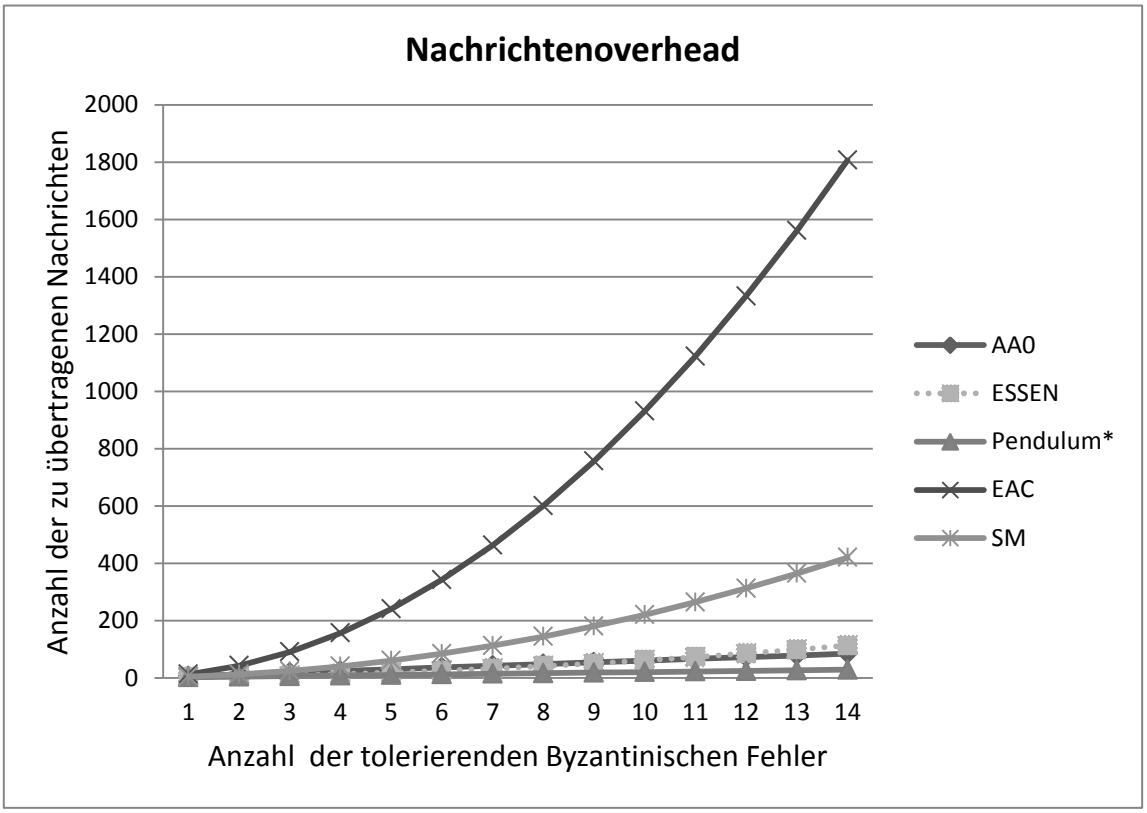


Abbildung 11: Nachrichtenoverhead zur Lösung des Übereinstimmungsproblems. (Bousbiba 2015b).

Wie in Abbildung 11 zu sehen, verursacht ESSEN für bis zu sieben Fehler einen geringen Nachrichtenoverhead. Der Nachrichtenoverhead von ESSEN bewegt sich zwischen Pendulum* und AAO. Sind mehr als sieben Fehler zu tolerieren, so benötigt AAO einen im Vergleich zu ESSEN geringeren Nachrichtenoverhead. Jedoch resultiert dieser für eine höhere Fehleranzahl geringere Nachrichtenoverhead allein aus der Tatsache, dass jegliche Art von Omission-Fehlern (wie etwa gutartig Byzantinische Fehler) im Fehlermodell vollständig

ausgeschlossen werden, was folglich zu einem einfachen Protokolldesign führt (mehr hierzu siehe (Khosravi und Kavian 2012)).

Abbildung 12 zeigt die für das Lösen des Übereinstimmungsproblems benötigte Anzahl der zu speichernden Nachrichten pro Knoten. Wie in der Abbildung zu sehen, benötigen (außer Pendulum* und ESSEN) die untersuchten Protokolle eine proportional mit der Anzahl der zu tolerierenden Fehler wachsende Anzahl von zu speichernden Nachrichten. Der Speicheraufwand beim Pendulum* und ESSEN ist stets konstant, genauer gesagt muss ein Knoten in Pendulum* genügend Speicher verfügen, um eine Nachricht und bei ESSEN um bis zu drei Nachrichten im ungünstigsten Fall speichern zu können. Folglich eignen sich beide Protokolle (bzgl. des Speicheraufwands) für Anwendungen, bei denen die eingesetzten Knoten nur Zugriff auf begrenzte Speicherressourcen verfügen, wie es etwa bei Sensornetzen (siehe z. B. (Guerrero et al. 2013; Basir und Shen; Santoro 2006)) der Fall ist.

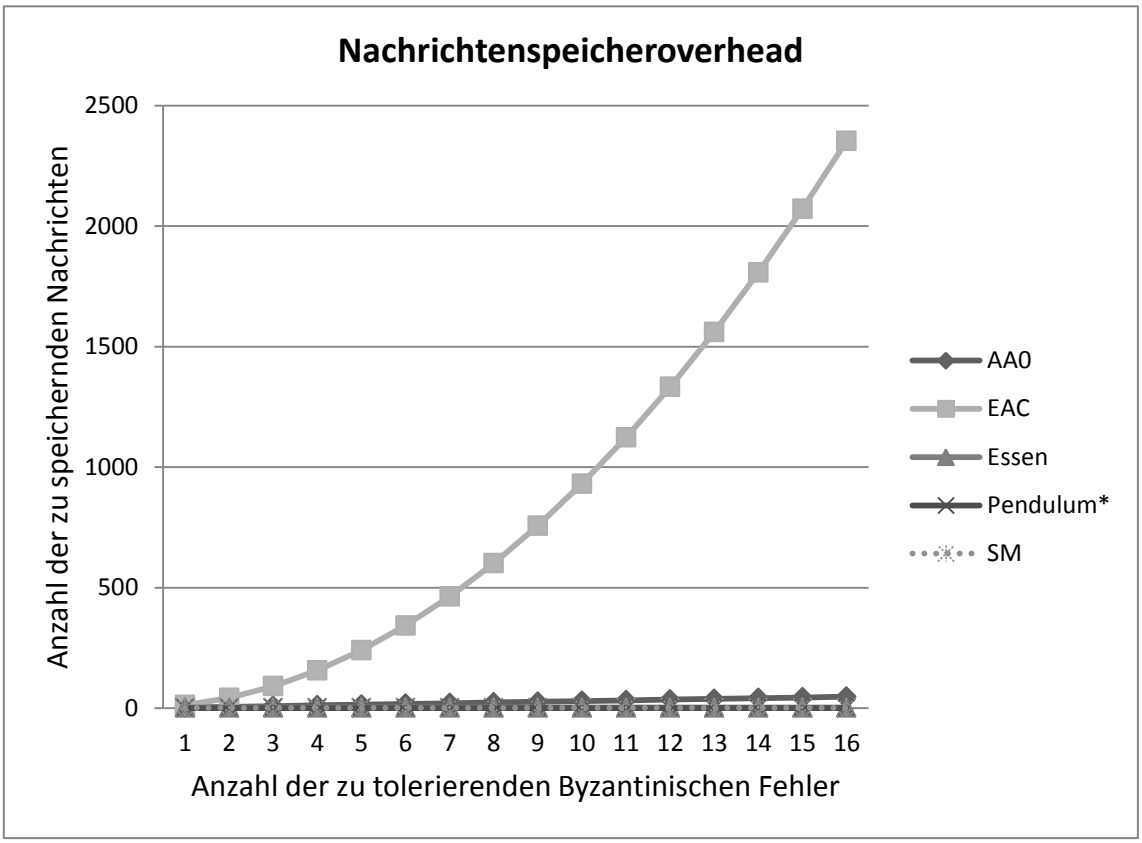


Abbildung 12: Speicheroverhead um das Übereinstimmungsproblem zu lösen (Bousbiba 2015b).

Die letzte Abbildung (Abbildung 13) zeigt für jedes Protokoll das Verhältnis zwischen der Anzahl der notwendigen redundanten Knoten und der Anzahl der zu tolerierenden Fehler f. Wie in Abbildung 13 zu sehen, benötigt ESSEN im Gegensatz zu den anderen untersuchten Protokollen eine um den Faktor f größere Knotenzahl (ungefähr $n \geq 4f$). Folglich benötigen die anderen Protokolle für ein wachsendes f (f » 3) diesbezüglich einen niedrigen Knotenoverhead.

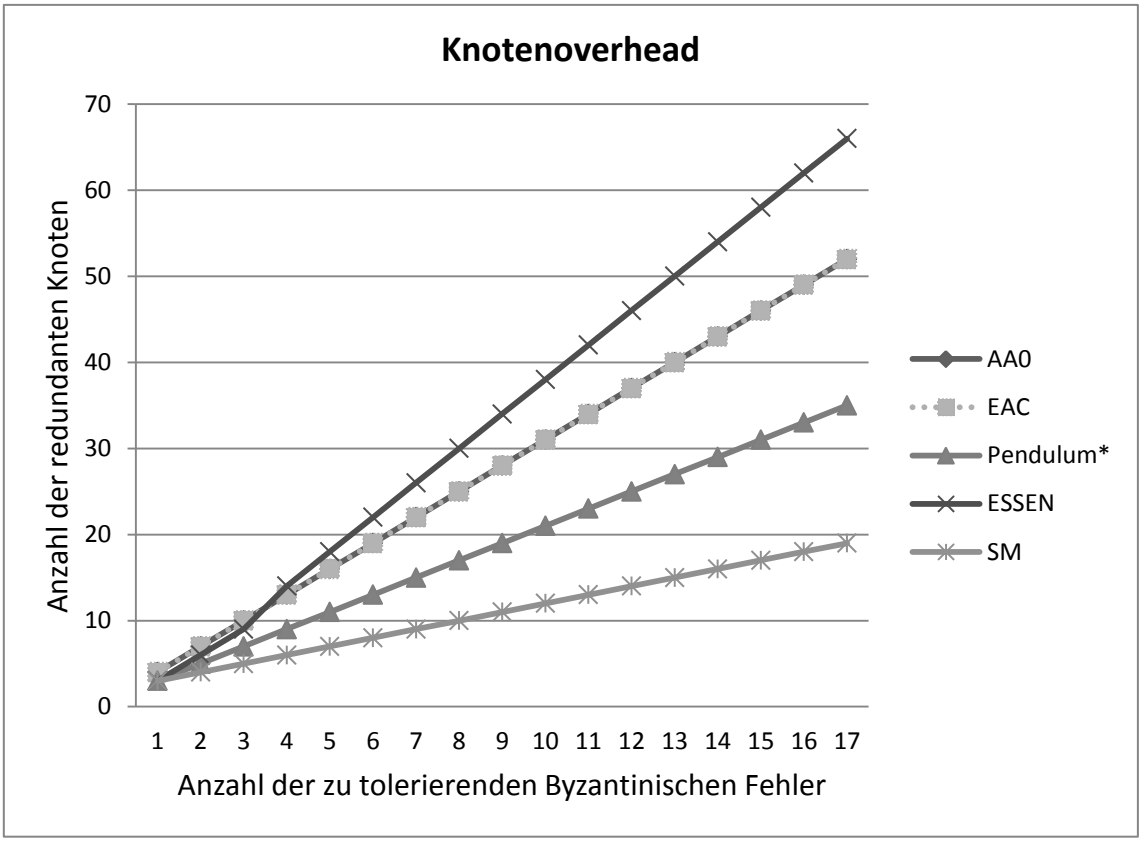


Abbildung 13: Die Anzahl der redundanten Knoten , die notwendig sind, um das Übereinstimmungsproblem zu lösen (Bousbiba 2015b).

Werden jedoch bei der Wahl eines Übereinstimmungsprotokolls sämtliche Parameter in Betracht gezogen, wie etwa die Fehlertoleranzfähigkeit, die Anzahl der maximal zu tolerierenden Fehlern, etc., dann muss ein Kompromiss zwischen Kommunikations- und Knotenoverhead gefunden werden. In dieser Hinsicht zeigt das Protokoll ESSEN im Vergleich zu den anderen untersuchten Protokollen bessere Eigenschaften.

4.11 Zusammenfassung

Die Entwicklung effizienter Lösungen zum Byzantinischen Übereinstimmungsproblem in verteilten Rechensystemen stellt immer noch eine herausfordernde Aufgabe dar, die alles andere als einfach zu lösen ist, vor allem wenn Lösungen für verschiedene System-Modelle angestrebt werden, wie etwa für asynchrone oder teilsynchrone Systeme.

Das in diesem Kapitel vorgestellte Übereinstimmungsprotokoll ESSEN löst das Byzantinische Übereinstimmungsproblem für synchrone verteilte drahtlose Netze und toleriert alle Fehlfunktionen – bis hin zu bösartig kooperierenden Byzantinische Fehlern! Die Anzahl der Runden sowie der Speicheraufwand bleibt unabhängig von der Anzahl der tolerierenden Fehler f stets konstant. Das Protokoll terminiert nach nur 1 Runde. Die Anzahl der verwendeten redundanten Knoten wächst linear mit der Anzahl der zu tolerierenden Fehler f. Für eine kleine Fehleranzahl, wie etwa $f \leq 4$, wie es in vielen praktischen Anwendungen angenommen wird, ist der von ESSEN verursachte Knotenoverhead akzeptabel.

Der Einsatz der „Reinigungsfunktion“ in ESSEN, die es so in keinem anderen Übereinstimmungsprotokoll gibt, hat einen wesentlichen Beitrag zur Reduktion des Nachrichtenoverheads geleistet. Folglich ist das Protokoll ESSEN geeignet für drahtlose verteilte Netze, bei denen die Anzahl der zu tolerierenden Fehler ($f \leq 4$) nicht zu groß ist und bei dem die Reduktion des Kommunikationsoverheads eine höhere Priorität hat als die Reduktion des Knotenoverheads (wie es z. B. in Sensornetze der Fall ist, siehe auch (Agmon und Peleg 2006; Basir und Shen)).

5 Verifikation und Simulation

“The man of science has learned to believe in justification, not by faith, but by verification.” – Thomas Huxley

“Software testing proves the existence of bugs not their absence.” – Anonymous

Bei der Überprüfung der Fehlertoleranzeigenschaften von Übereinstimmungsprotokollen in verteilten Systemen ist man daran interessiert, welche Auswirkung die fehlerhaften Knoten auf die fehlerfreien Knoten haben. Das interne Verhalten der fehlerhaften Knoten muss dagegen in der Regel nicht betrachtet werden (Echtle und Silva 1998). Daher reicht es zum Testen der Fehlertoleranzfähigkeit aus, wenn Fehlerinjektoren auf System-Ebene für diesen Zweck verwendet werden (Echtle und Silva 1998; Mei-Chen Hsueh et al. 1997). Für die Fehlerinjektion bedeutet dies, dass ausschließlich die vom Knoten ausgesandten Nachrichten zu ändern sind. Vereinfacht ausgedrückt bildet der Fehlerinjektor die „Auswirkung“ des Knotenfehlers in Form von Nachrichtenverspätungen, inhaltlichen Verfälschungen, Fehlleitung von Nachrichten etc. nach.

Für den Korrektheitsnachweis ist der Test allein nicht ausreichend. Daher muss neben dem Test zusätzlich eine formale Verifikation erfolgen. Auf der anderen Seite benötigen bzw. verursachen Theorembeweiser oft einen hohen Rechenaufwand, sodass man gezwungen ist, das zu untersuchende Verfahren bzw. Teile davon isoliert für eine kleine Anzahl an Fehlern f zu verifizieren. Daher ist es ratsam, bei der Überprüfung der Fehlertoleranzeigenschaften sowohl eine formale Verifikation als auch ein Test unter Fehlerinjektion einzusetzen (Echtle und Silva 1998).

Aus diesem Grund wurden im Rahmen dieser Arbeit zur Verifikation und Validierung des Übereinstimmungsprotokolls ESSEN sowohl eine formale Verifikation als auch ein auf Fehlerinjektion basierender Test durchgeführt. Zwecks Verifikation wurde ESSEN zunächst mit Hilfe des Modell-Checkers UPPAAL für bis zu 9 Knoten (näheres hierzu siehe Abschnitt 5.2.1) und drei Knotenfehler verifiziert, wobei die in Kapitel 2 beschriebenen Fehlverhaltensweisen nicht vollständig umgesetzt wurden (siehe die Begründung und die Nennung der betroffenen Fehler in Abschnitt 5.2.1 und 5.3.1). Mehr als drei Knotenfehler konnten wegen des Problems der Zustandsraum-Explosion mittels UPPAAL nicht untersucht werden.

Daher wurde im Anschluss daran eine implementierte Variante des UPPAAL-Models in Java umgesetzt, mit dessen Hilfe zum einen das UPPAAL-Modell validiert und zum anderen das Verhalten des Protokolls für höhere Fehleranzahlen untersucht werden konnte.

Um zu zeigen, dass das Protokoll allgemein für sämtliche $f_N \in \mathbb{N}$ Fehler stets IC1 und IC2 garantiert, wurde zusätzlich zu der Verifikation mittels UPPAAL ein formaler Korrektheitsbeweis geführt (siehe hierzu Abschnitt 5.4).

In Abschnitt 5.2 und 5.3 werden die in der Simulation und Verifikation durchgeführten Vereinfachungen erläutert und begründet.

5.1 Notationsübersicht: Bedingungen und Funktionen

Der folgende Abschnitt liefert eine Übersicht über die Notation sämtlicher Bedingungen und Operationen in ESSEN, auf die im späteren formalen Beweis (siehe Abschnitt 5.4) zurückgegriffen wird. Zusätzlich werden die unten aufgeführten Notationen vorab in Abschnitt 5.2.1 zur Argumentation der im UPPAAL-Modell durchgeführten Abstraktionen verwendet.

AG1 — Bedingung: Die in MsgPMB enthaltene Data-Nachricht enthält vor Ausführung von CF mindestens $f + 1$ Signaturen. Wenn diese Bedingung nicht erfüllt ist (¬AG1), dann entscheidet sich der Knoten für eine globale Konstante.

AG2 — Bedingung: Die in MsgPMB gespeicherte Nachricht enthält nach Ausführung von CF mindestens f Signaturen. Wenn diese Bedingung nicht erfüllt ist (¬AG2), dann entscheidet sich der Knoten für eine globale Konstante.

AG3 — Bedingung: Die in MsgSMB gespeicherte Nachricht enthält nach Ausführung von CF mindestens $f + 1$ Signaturen. Wenn diese Bedingung nicht erfüllt ist (¬AG3), dann entscheidet sich der Knoten für eine globale Konstante.

CF(M_{data}, $M_{default}$) — Operation: Gemeinsame Signaturquellen in den beiden Nachrichten $M_{data} \in$ {MsgPMB, MsgSMB} und $M_{default}$ = MsgDMB (d.h. in einer gespeicherten Data-Nachricht und der gespeicherten Default-Nachricht) werden aus M_{data} ignoriert. indem in der Signaturliste von M_{data} die entsprechenden Einträge als „ignoriert“ markiert werden, so dass sie nicht mitgezählt werden. D. h. die Funktion CF gibt die Anzahl der Knoten wieder, welche nur die Data-Nachricht M_{data} signiert haben (vergl. Zeile 36 – 38, **Algorithmus 1**).

MD1 Bedingung: Die in MsgPMB gespeicherte Data-Nachricht enthält mehr Signaturen als die in MsgDMB gespeicherte Data-Nachricht, das heißt $sigCount(MsgPMB) > sigCount(MsgDMB)$. Wenn diese Bedingung erfüllt ist, sendet ein fehlerfreier Knoten aus der Differenzmenge SendG \ QuellG seine in MsgPMB gespeicherte Data-Nachricht.

MD2 Bedingung: Die in MsgDMB gespeicherte Default-Nachricht enthält mindestens so viele Signaturen wie die in MsgPMB gespeicherte Data-Nachricht, das heißt $sigCount(MsgDMB) \geq sigCount(MsgPMB)$. Wenn diese Bedingung erfüllt ist, sendet ein fehlerfreier Knoten aus ErwG seine in MsgDMB gespeicherte Default-Nachricht.

NC1 Bedingung: Eine Data-Nachricht enthält mindestens die Signatur des Quellknotens. Wenn diese Bedingung nicht erfüllt ist (¬NC1), wird die betreffende Nachricht von jedem fehlerfreien Knoten aus BasicG verworfen.

NC2 Bedingung: Eine Data-Nachricht enthält mindestens die Signatur des Quellknotens und mindestens eine Signatur von einen Knoten aus BasisG. Wenn diese Bedingung nicht erfüllt ist (¬NC2), wird die betreffende Nachricht von keinen fehlerfreien Knoten aus ErwG akzeptiert.

NC3 Bedingung: Eine Default-Nachricht ist ausschließlich von Knoten aus ErwG signiert bzw. kosigniert. Wenn diese Bedingung erfüllt ist, wird die betreffende Default-Nachricht als valide angesehen. Andernfalls (¬NC3) wird sie von keinen fehlerfreien Knoten aus SendG akzeptiert.

SC1(RxMsg) Bedingung: Die Nachricht RxMsg enthält mindestens $f + 1$ Signaturen.

SC2(RxMsg) Bedingung: Die Nachricht RxMsg (siehe ggf. Abschnitt 4.5) enthält mehr Signaturen als die vom jeweiligen Empfangsknoten zuvor gespeicherte Nachricht MsgPMB oder MsgSMB:

```
(sigCount(RxMsg) > sigCount(MsgPMB)   ||
 sigCount(RxMsg) > sigCount(MsgDMB))
```

Wenn diese Bedingung nicht erfüllt ist (¬SC2), wird die Nachricht RxMsg verworfen.

SC3(RxDef) Bedingung: Die Nachricht RxDef (siehe ggf. Abschnitt 4.5) enthält mehr Signaturen als die zuvor gespeicherte Default-Nachricht:

```
(sigCount(RxDef) > sigCount(MsgDMB)
```

Wenn diese Bedingung nicht erfüllt ist (¬SC3), wird die Nachricht RxDef verworfen.

5.2 UPPAAL-Verifikation

Um den Zustandsraum des UPPAAL-Models (siehe Anhang B) klein zu halten, wurden bzgl. des Kanal- und des Fehlermodells die folgenden Abstraktionen vorgenommen.

1. Die Slot-Zeitpunkte werden mittels Synchronisations-Labels realisiert (siehe auch (Behrmann et al. 2004)). Es kommt kein Uhrensynchronisationsalgorithmus zum Einsatz, da dieser unnötigerweise den Zustandsraum vergrößern würde, ohne jedoch irgendwelche Aussagen über die Fehlertoleranzeigenschaft des Protokolls ESSEN zu liefern (siehe Fehlermodellannahme in Abschnitt 4.8). Stattdessen wird einfach von synchronisierten Uhren ausgegangen.
2. Die Anzahl der Kotenfehler f_N im Modell wird auf drei beschränkt (entspricht **FM0**), was den meisten praktischen (sicherheitskritischen) Anwendungen entspricht (Leitenberger 2015). In der Praxis werden sogar meist nur Einzel- und Doppelfehler angenommen.
3. Unerkannte Signaturverfälschungen werden im Modell ausgeschlossen (entspricht **FM1**), zumal beim Einsatz von Signaturen häufig die Annahme getroffen wird, dass Verfälschungen jeglicher Art stets aufgedeckt werden bzw. die Wahrscheinlichkeit einer nicht erkannten Verfälschung als vernachlässigbar gering eingeschätzt wird. Folglich genügt es im Falle von UPPAAL, die Signaturen in Form von Listeneinträgen darzustellen, wobei der Index in der Liste die Signatur des jeweiligen Knoten repräsentiert.
4. Die fehlerhaften Knoten können sich beliebig verhalten (entspricht **FM8**) mit der Einschränkung, dass Timing-Fehler (entspricht **FM4**) im Modell nicht berücksichtigt werden (Grund: siehe **FM4**). Diese Vereinfachung ist dadurch gerechtfertigt, dass für jeden Knoten ein Sendeslot statisch zugeteilt wird und ein Knoten wegen den Busguardians nicht außerhalb seines Slots senden kann.
5. Der Quellknoten sendet nur die Werte 0 oder 1, wodurch die Unterscheidung zwischen zwei beliebigen Werten repräsentiert wird. Alle anderen Knoten können zusätzlich den Wert 2 (repräsentiert die im Protokoll verwendete Default-Nachricht) senden. Dies stellt eine Einschränkung von **FM6** dar, da der mögliche Wertebereich der Daten auf die natürlichen Zahlen 0, 1 und 2 begrenzt wird. Diese Einschränkung ist als unkritisch anzusehen, da durch den Einsatz von drei verschiedenen Werten sämtliche relevante Fehlerfälle generiert werden können. Durch den Einsatz von zwei verschiedenen Datenwerten (0 und 1) lassen sich alle Fälle konstruieren, bei

denen fehlerfreie Knoten zu unterschiedlichen Sichten kommen. Durch den Einsatz des letzten Wertes (Default) werden zusätzlich die Fälle betrachtet, bei denen sich Knoten gegen den Wert des Quellknotens entscheiden. Folglich reicht es im Falle von ESSEN aus, den Wertebereich auf drei Werte zu beschränken.

6. Jeder fehlerhafte Knoten hat Zugriff auf die Signatur der anderen fehlerhaften Knoten (entspricht **FM7**). Dadurch wird die „bösartige“ Kooperation zwischen fehlerhaften Knoten ausgedrückt.
7. Jeder fehlerhafte Knoten ist in der Lage, mehr als eine Nachricht in seinem Sendeslot zu übertragen (entspricht **FM6**). Im Modell kann ein fehlerhafter Knoten bis zu maximal drei verschiedene Werte verteilen. Dies stellt zwar stellt eine Einschränkung von **FM6** dar. Die Einschränkung auf drei Werte ist jedoch, wie bereits in Punkt 3 begründet, zur Untersuchung der Fehlertoleranzeigenschaften vollkommen ausreichend.
8. Die Nachrichten, die ein fehlerhafter Knoten sendet, können von allen Knoten, von einer Teilmenge der Knoten oder von keinem Knoten empfangen werden (entspricht **FM5** und **FM6**).
9. Ist der Sender fehlerfrei, dann wird sichergestellt, dass die Nachricht von allen Knoten empfangen wird (entspricht **FM10**).

5.2.1 Zustandsraumreduzierung zum Nachweis der Fehlertoleranzfähigkeit

Um die Korrektheit des Protokoll ESSEN für $f \leq 3$ mittels UPPAAL nachzuweisen (welches unter anderem zur Herleitung des Bildungsgesetzes in Abschnitt 4.7 verwendet wurde), wurden wegen der drohenden Zustandsraumexplosion zusätzlich zu den in Abschnitt 5.2 aufgeführten Abstraktionen weitere Vereinfachungen (siehe (a) – (c) unten) am Modell vorgenommen, auf die im Folgenden näher eingegangen werden soll.

(a) Der Quellknoten gilt stets als fehlerhaft. Im Modell werden die Fälle, bei denen der Quellknoten fehlerfrei ist, nicht berücksichtigt. Daraus folgt, dass die Gruppe ErwG höchstens zwei fehlerhafte Knoten enthält.
(b) BasisG enthält höchstens einen fehlerhaften Knoten.
(c) Es werden nur die Fälle betrachtet, bei denen höchstens ein Knoten in BasisG die Nachrichten des Quellknotens sendet. In diesem Fall kann die Gruppe BasisG, die aus $f + 1$ Knoten besteht, zu einem einzigen Knoten zusammengefasst werden.

Der nachfolgende Abschnitt soll zeigen, dass die in (a) bis (c) getroffenen Annahmen hinreichend sind, um die Fehlertoleranzfähigkeit des Protokolls für $f \leq 3$ mittels UPPAAL nachzuweisen. Der Nachweis des Protokolls ESSEN für $f \leq 3$ ist insofern von Bedeutung, da er die Herleitung der notwendigen Anzahl redundanter Knoten zur Tolerierung von f Fehlern erlaubt. Die aus dem UPPAAL-Modell hergeleitete Gleichung bzw. Abhängigkeit zwischen der Anzahl redundanter Knoten und Fehlergrad f kann durch Simulation geprüft werden, mit deren Hilfe höhere Knotenfehler-Anzahlen untersucht werden können. Umso wichtiger ist es zu zeigen, dass die in (a) bis (c) getroffenen Modifikationen zur Vereinfachung keinen Einfluss auf die oben formulierte Zielsetzung ausüben (d.h. auf die Herleitung der Abhängigkeit zwischen Fehlertoleranzgrad f und Anzahl der notwendigen redundanten Knoten). Im Folgenden wird für die Fälle, die eine der Bedingungen (a), (b) oder (c) verletzen, formal bewiesen, dass für $f \leq 3$ ebenfalls Fehlertoleranz gemäß IC1 und ggf. IC2 vorliegt.

Behauptung 1. *Falls der Quellknoten fehlerfrei ist, dann ist IC1 und IC2 für $f = 3$ stets erfüllt, unabhängig davon, ob (b) und (c) gilt.*

Behauptung 1 ist komplementär zu (a). Der folgende Beweis von Behauptung 1 deckt also alle Fälle ab, in denen (a) nicht gilt.

Beweis: Da der Quellknoten fehlerfrei ist, empfängt jeder fehlerfreie Knoten in BasisG die Data-Nachricht. Folglich leiten alle fehlerfreien Knoten in BasisG diese Nachricht weiter.

Fall 1: BasisG enthält höchstens einen fehlerhaften Knoten.

Enthält BasisG höchstens einen fehlerhaften Knoten, dann sendet der letzte fehlerfreie Knoten in BasisG eine Nachricht mit mindestens $f + 1$ Signaturen. Da die Gruppe ErwG höchstens f fehlerhafte Knoten enthält, kosignieren und übertragen alle fehlerfreien Knoten aus ErwG die Data-Nachricht des Quellknotens. Laut AG2 entscheiden sich demzufolge alle Knoten gemäß IC1 und IC2 für denselben finalen Endwert.

Fall 2: BasisG enthält mindestens 2 fehlerhafte Knoten.

Dies impliziert, dass $2 \leq f \leq 3$ ist.

Fall 2.1: BasisG enthält genau $f - 1$ fehlerhafte Knoten. Folglich ist $f = 3$ und BasisG enthält genau $(f + 1) - (f - 1) = 2$ fehlerfreie Knoten.

Laut NC1 und SC2 sendet der letzte fehlerfreie Knoten N[i] in BasisG eine Data-Nachricht mit mindestens der folgenden Anzahl an Signaturen:

$$sigCount(M[i]) \geq 3$$

da die Nachricht M[i] von dem Quellknoten und mindestens den beiden fehlerfreien Knoten aus BasisG signiert worden ist.

Da ErwG höchstens einen fehlerhaften Knoten enthält, signieren und senden gemäß MD1 alle fehlerfreien Knoten in ErwG die Data-Nachricht. Unter der Annahme, dass die Formel 4.1 für beliebige $f \geq 0$ gilt, folgt, dass die Menge ErwG mindestens die folgende Anzahl an fehlerfreien Knoten enthält:

$$2(f-1) + \max\,(0, f-2) - 1$$

Weiterhin wissen wir, dass sich gemäß SC2 bei den fehlerfreien Knoten immer die Data-Nachricht durchsetzt, welche die meisten Signaturen aufweist. Folglich sendet der letzte fehlerfreie Knoten in ErwG eine Data-Nachricht x mit mindestens der folgenden Anzahl an Signaturen:

$$sigCount(x) \;\geq\; sigCount(M[i]) + 2(f-1) + max(0, f-2) - 1$$

$$\Rightarrow sigCount(x) \;\geq\; 2f + max(0, f-2) \;\geq\; f+1$$

Laut AG2 entscheiden sich demzufolge alle Knoten für denselben finalen Endwert, wodurch IC1 und IC2 erfüllt sind.

Fall 2.2: BasisG enthält genau f fehlerhafte Knoten.

Folglich ist $2 \leq f \leq 3$ und BasisG enthält genau $(f+1) - f = 1$ fehlerfreien Knoten. Laut NC1 und SC2 sendet der fehlerfreie Knoten N[i] in BasisG eine Data-Nachricht mit mindestens der folgenden Anzahl an Signaturen:

$$sigCount(M[i]) \geq 2$$

da die Nachricht M[i] von dem Quellknoten und mindestens dem fehlerfreien Knoten aus BasisG. signiert worden ist. Da ErwG keinen fehlerhaften Knoten enthält, signieren und senden gemäß MD1 alle fehlerfreien Knoten in ErwG die Data-Nachricht. Unter der Annahme, dass die Formel 4.1 für beliebige $f \geq 0$ gilt, folgt, dass die Menge ErwG mindestens die folgende Anzahl an fehlerfreien Knoten enthält:

$$2(f-1) + \max\,(0, f-2)$$

Gemäß SC2 sendet der letzte fehlerfreie Knoten in ErwG eine Data-Nachricht x mit mindestens der folgenden Anzahl an Signaturen:

$$sigCount(x) \;\geq\; sigCount(M[i]) + 2(f-1) + max(0, f-2)$$

$$\Rightarrow sigCount(x) \;\geq\; 2f + max(0, f-2) \;\geq\; f+1$$

Laut AG2 entscheiden sich demzufolge alle Knoten für denselben finalen Endwert, wodurch IC1 und IC2 erfüllt sind.

Behauptung 2. *Falls für $f \leq 3$ der Quellknoten fehlerhaft ist und mindestens 2 Knoten aus BasisG irgendeine Data-Nachricht senden, dann ist die Bedingung IC1 stets erfüllt.*

Behauptung 2 erfüllt (a) und das Komplement von (c).

Beweis: Der Beweis erfolgt durch Widerspruch. Wir nehmen an, dass die fehlerfreien Knoten die Bedingung IC1 nicht erfüllen.

Fall 1: Die restlichen $f-1$ fehlerhaften Knoten befinden sich in BasisG. Dies bedeutet, dass die Gruppe ErwG keine fehlerhaften Knoten enthält.

Fall 1.1: Kein Knoten aus ErwG signiert und sendet eine der Data-Nachrichten.

Wenn kein Knoten aus ErwG eine Data-Nachricht sendet, dann erfüllen alle fehlerfreien Knoten die Bedingung IC1.

Beweis: Der Beweis erfolgt durch Widerspruch. Wir nehmen an, es gibt mindestens zwei fehlerfreie Knoten N[j] und N[k], die sich für zwei verschiedene Endwerte entscheiden.

Falls es einen fehlerfreien Knoten N[i] in BasisG gibt, der eine Data-Nachricht sendet, dann müssten gemäß NC2 und SC2 alle fehlerfreien Nachfolger von N[i] ebenfalls eine Data-Nachricht signieren und senden, insbesondere auch die Nachfolger in ErwG. Dies stellt einen Widerspruch zu der Annahme in Fall 1.1 dar (**Widerspuch!**).

Fall 1.2: Mindestens ein Knoten aus ErwG signiert eine der Data-Nachrichten.

Wenn der letzte Knoten aus ErwG eine Data-Nachricht mit weniger als f Signaturen überträgt, dann ist AG1 niemals erfüllt und alle fehlerfreien Knoten entscheiden sich für die globale Konstante, wodurch IC1 erfüllt ist. In allen anderen Fällen sendet der letzte fehlerfreie Knoten eine Data-Nachricht, die von mindestens $f+1$ Knoten signiert worden ist. Weiterhin wissen wir, dass sich gemäß SC2 bei den fehlerfreien Knoten immer die Data-Nachricht durchsetzt, welche die meisten Signaturen aufweist. Folglich entscheiden sich laut AG2 alle fehlerfreien Knoten für den Wert des Quellknotens, der im Laufe der Protokoll-Ausführung die meisten Signaturen erhält, wodurch IC1 erfüllt ist.

Fall 2: BasisG enthalte höchstens $f - 2$ fehlerhafte Knoten.

Da $f \leq 3$ angenommen wird, bedeutet Fall 2, dass BasisG höchstens einen fehlerhaften Knoten enthält. Dann gibt es laut der Voraussetzung von Behauptung 2 mindestens einen fehlerfreien Knoten in BasisG, der eine Data-Nachricht an alle fehlerfreien Knoten sendet.

Fall 2.1: BasisG enthält keinen fehlerhaften Knoten.

> Hinweis: Laut Voraussetzung von Behauptung 2 empfangen und senden mindestens zwei Knoten aus BasisG eine Data-Nachricht vom Quellknoten. Da BasisG nur aus fehlerfreien Knoten besteht, sendet laut SC2 der letzte fehlerfreie Knoten in BasisG eine Data-Nachricht x mit mindestens 3 Signaturen (enthält die Signatur des Quellknotens sowie 2 Signaturen aus BasisG, weil gemäß Voraussetzung von Behauptung 2 mindestens zwei Knoten aus BasisG eine Data-Nachricht senden).

Laut NC2 und SC2 signieren und senden alle fehlerfreien Knoten in ErwG die Data-Nachricht x. Gemäß SC2 sendet der letzte fehlerfreie Knoten in ErwG die Data-Nachricht x mit mindestens der folgenden Signaturzahl:

$$sigCount(x) \geq 3 + 2(f - 1) + max(0, f - 2) - (f - 1)$$

Diese Ungleichung ist wie folgt zu begründen: Die Nachricht ist von dem Quellknoten und mindestens zwei Knoten aus BasisG signiert worden (in der Summe mindestens 3 Signaturen, siehe Hinweis zuvor). Hinzu kommen noch mindestens die Signaturen der fehlerfreien Knoten aus ErwG (höchstens $f - 1$ Knoten in ErwG sind fehlerhaft). Es gilt

$$sigCount(x) \geq 3 + 2f - 2 + max(0, f - 2) - (f - 1)$$

$$\Rightarrow sigCount(x) \geq 3 + f - 1 + max(0, f - 2)$$

$$\Rightarrow sigCount(x) \geq f + 2$$

Nach Ausführung der „Cleaning-Function“ CF enthält jeder fehlerfreie Knoten im PMB die Data-Nachricht x mit der folgenden Mindest-Signaturanzahl. Diese berechnet sich aus der Anzahl der Signaturen der Nachricht, abzüglich der Anzahl von Signaturen, die durch CF ignoriert werden (Diese Funktion ignoriert höchstens so viele

Signaturen, wie es fehlerhafte Knoten in ErwG gibt. Dies sind wegen $f \leq 3$ und der der Annahme eines fehlerhaften Quellknoten 2 fehlerhafte Knoten):

$$sigCount(x) \geq f + 2 - 2 = f$$

Gemäß AG2 entscheiden sich alle fehlerfreien Knoten für denselben Endwert x, womit IC1 erfüllt ist.

Hinweis: Die fehlerfreien Knoten entscheiden sich alle für die Data-Nachricht x, weil diese sowohl die notwendige (siehe AG1) als auch die hinreichende Bedingung (siehe AG2) erfüllt, die wie folgt formuliert worden sind:

1) AG1 besagt, dass die Data-Nachricht im PMB vor Ausführung der „Cleaning Function“ CF mindestens $f + 1$ Signaturen enthalten muss.
2) AG2 besagt, dass die Data-Nachricht im PMB nach Ausführung der „Cleaning Function“ CF mindestens f Signaturen enthalten muss oder im SMB eine Data-Nachricht mit mindestens $f + 1$ hinterlegt ist.

Fall 2.2: BasisG enthält genau einen fehlerhaften Knoten.

Wenn BasisG genau einen fehlerhaften Knoten enthält, sendet der letzte fehlerfreie Knoten N[i] in BasisG eine Data-Nachricht M[i] mit mindestens der folgenden Anzahl an Signaturen:

$$sigCount(M[i]) \geq 2$$

Hinweis: Da die Menge ErwG höchstens einen fehlerhaften Knoten enthält und Knoten N[i] eine Data-Nachricht mit mindestens 2 Signaturen sendet, signieren und senden laut SC2 alle fehlerfreien Knoten aus ErwG eine Data-Nachricht

Sei ErwG‘ die Menge der fehlerfreien Knoten in ErwG, dann enthält die Menge ErwG‘ mindestens die folgende Anzahl an fehlerfreien Knoten:

$$|ErwG'| \geq 2(f-1) + max(0, f-2) - (f-2)$$

$$\Rightarrow |ErwG'| \geq \mathrm{f} + max(0, f-2)$$

Gemäß S2 sendet der letzte fehlerfreie Knoten N[k] in ErwG‘ eine Data-Nachricht M[k] mit mindestens der folgenden Anzahl an Signaturen:

$$sigCount(M[k]) \geq sigCount(M[i]) + |ErwG'|$$

$$sigCount(M[k]) \geq 2 + f + max(0, f-2)$$

Diese Nachricht wird von allen fehlerfreien Knoten in PMB hinterlegt. Um IC1 zu verletzten, müsste gemäß SC2 der fehlerhafte Knoten in ErwG eine Data-Nachricht y oder eine Default-Nachricht senden, die mindestens so viele Signaturen enthält wie die von Knoten N[k] übertragene Data-Nachricht M[k]. Dies stellt aber ein Widerspruch zu der f-Fehlerannahme dar. Folglich, gemäß AG2, entscheiden sich alle fehlerfreien Knoten für denselben Endwert, womit IC1 erfüllt ist.

Behauptung 3. *Falls für $f \leq 3$ die Bedingungen (a) und (c) gelten und (b) nicht gilt, dann ist IC1 stets erfüllt.*

Diese Behauptung bezieht sich also auf die Annahme, dass der Quellknoten fehlerhaft ist und höchstens ein Knoten aus BasisG eine Data-Nachricht weiterleitet, wobei BasisG genau zwei fehlerhafte Knoten und ErwG keinen fehlerhaften Knoten enthält (hier gilt stets $f = 3$). Für $f = 3$ enthält BasisG stets genau $f + 1 = 4$ Knoten, also zwei fehlerhafte und zwei fehlerfreie Knoten.

Fall 1: Kein Knoten aus ErwG sendet eine Data-Nachricht.

Fall 1.1: Kein fehlerfreier Knoten in BasisG sendet eine Data-Nachricht.

Wegen der Fehlerannahme $f = 3$ kann kein fehlerhafter Knoten eine Data-Nachricht mit mehr als f Signaturen senden – und gemäß der Annahme in Fall 1 fügt kein fehlerfreier Knoten eine Signatur hinzu. Die durch AG1 gegebene Schwelle $f + 1$ der Signaturanzahl wird keinesfalls erreicht. Alle fehlerfreien Knoten entscheiden sich für die globale Konstante, wodurch IC1 erfüllt ist.

Fall 1.2: Es gibt mindestens einen fehlerfreien Knoten in BasisG, der eine Data-Nachricht sendet.

Dann müssten laut NC2 und SC2 alle fehlerfreien Knoten in ErwG ebenfalls eine Data-Nachricht sendet. Dies stellt einen Widerspruch zu der Annahme in Fall 1 dar (**Widerspuch!**).

Fall 2: Mindestens ein Knoten aus ErwG sendet eine Data-Nachricht.

Laut SC2 speichert jeder fehlerfreie Knoten im PMB die Data-Nachricht, die vom letzten fehlerfreien Knoten N[k] in ErwG gesendet wurde.

Fall 2.1: Der Knoten N[k] sendet eine Data-Nachricht mit höchstens f Signaturen.

Damit ist AG1 niemals erfüllt und alle fehlerfreien Knoten entscheiden sich für die globale Konstante, wodurch IC1 erfüllt ist.

Fall 2.2: Der Knoten N[k] sendet eine Data-Nachricht mit mindestens $f + 1$ Signaturen. Damit ist AG2 stets erfüllt und alle fehlerfreien Knoten entscheiden sich für die $(f + 1)$-fach signierte Data-Nachricht. Somit ist IC1 erfüllt.

Beweis: Aus den bewiesenen Behauptungen 1 bis 3, die von komplementären Annahmen ausgehen, und alle möglichen Fälle für $f \leq 3$ außer der Kombination (a), (b) und (c) abdecken, die durch das Uppaal-Modell verifiziert wurde, folgt jeweils, dass die in Abschnitt 5.2.1 getroffenen Annahmen hinreichend sind, um die Fehlertoleranzfähigkeit des Protokolls mittels UPPAAL nachzuweisen.

5.2.2 Methode

Nachdem der Algorithmus von ESSEN beschrieben wurde sowie die in der Simulation und Verifikation durchgeführten Abstraktionen erläutert und begründet worden sind (siehe hierzu auch Abschnitt 5.1), soll im nächsten Schritt die Vorgehensweise zur Verifikation des Protokolls anhand des Verifikationswerkzeugs UPPAAL (Bengtsson et al. 1996) erläutert werden. Hierzu wird zunächst eine kurze Einführung in UPPAAL gegeben, bevor anschließend das in UPPAAL modellierte und verifizierte Protokoll ESSEN präsentiert wird.

UPPAAL ist ein von den Universitäten Uppsala und Aalborg entwickeltes Validierungs- und Verifikationswerkzeug (kurz: „Model Checker“), mit dessen Hilfe Echtzeitsysteme sowohl modelliert und simuliert als auch verifiziert werden können (Behrmann et al. 2006). In UPPAAL wird das System als ein Netzwerk von zeitbehafteten endlichen Automaten („Timed Automata“) modelliert, bei denen die als Prozesse bezeichneten Automaten über Kanäle und/oder gemeinsame Variable miteinander kommunizieren bzw. sich synchronisieren. Nach (Behrmann et al. 2006) ist ein solcher zeitbehafteter endlicher Automat definiert als ein Tupel $\mathcal{A}$ = (L, l_0, C, A, E, I) mit:

- L als die Menge der Stellen
- $l_0 \in$ L der Startzustand,
- C die Menge der Uhren
- A die Menge der Aktionen
  $E \subseteq L \times A \times B(C) \times 2^C \times L$, die Menge der Transition zwischen je zwei Knoten mit einer Aktion, einer Transitionsbedingung („guard“) und einem Satz sich zurücksetzender Uhren sowie einer Funktion $I: L \rightarrow B(C)$, welche Stellen Zeitinvarianten zuordnet. Bei dem Ausdruck B(C) handelt es sich um eine Menge von Konjunktionen

über einfache Ausdrücke bzw. Bedingungen der Form x < c oder x ≤ c mit x, y ϵ C, c ϵ $\mathbb{N}$ und den folgenden Produktionsregeln P:

$B(C) \rightarrow B(C) \wedge B(C)$,

$B(C) \rightarrow B(C) \vee B(C)$,

$B(C) \rightarrow \neg B(C)$,

$B(C) \rightarrow \neg B(C)$,

$B(C) \rightarrow true$,

$B(C) \rightarrow false$,

$B(C) \rightarrow x < c$,

$B(C) \rightarrow x \leq c$

Zur Verifikation und Modellierung von Eigenschaften verwendet UPPAAL als Spezifikationssprache eine Untermenge der Time Computing Tree Logic (TCTL) (Behrmann et al. 2006). Ähnlich zu TCTL bestehen die Ausdrücke dieser Abfragesprache aus Temporaloperatoren und Pfadquantoren (Behrmann et al. 2006), wobei ersteres zur Beschreibung der individuellen Zustände und letzteres zur Beschreibung des Verhaltens eines Pfades im Modell verwendet wird. Die zu verifizierenden Eigenschaften des zu untersuchenden Modells werden mittels der von UPPAAL unterstützten TCTL-Abfragesprach spezifiziert und durch vollständige Exploration des Zustandsraumes bestätigt bzw. widerlegt. Die von UPPAAL unterstützen Ausdrücke mit ihren jeweiligen Bedeutungen sind in Tabelle 4 skizziert.

Die in UPPAAL spezifizierten Timed Automata dienen als Template, von dem eine oder mehrere Instanzen (auch Prozesse genannt) erzeugt werden können. Diese Prozesse agieren unabhängig voneinander (und sind zueinander asynchron), sofern diese nicht über Variablen, die sowohl global als auch lokal angelegt werden können (letzteres lässt sich nicht zur Synchronisation verwenden), oder über Kommunikationskanäle synchronisiert sind.

Außerdem besteht in UPPAAL die Möglichkeit, den Stellen des Automaten das Schlüsselwort „urgent“ oder „comitted“ zu zuweisen. Ersteres stellt sicher, dass die Transition zeitlos erfolgt, während letzteres zusätzlich ein sofortiges Verlassen der Stelle durchsetzt. Bezüglich der Synchronisation der Instanzen über Kanäle unterscheidet UPPAAL zwischen der Unicast- und Broadcast-Synchronisation. Bei der Unicast-Synchronisation handelt es sich um eine „Handshake“-Synchronisation, die ausschließlich zur Synchronisation zweier Prozesse eingesetzt wird. Möchte man jedoch mehrere Prozesse gleichzeitig über einen Kommunikationskanal

synchronisieren (beispielsweise zur abstrakten Darstellung der Uhrensynchronisation in verteilten Systems, sofern diese nicht Gegenstand der Untersuchung ist), so deklariert man den Kanal als Broadcast Channel. Soll die Synchronisation zusätzlich zeitlos erfolgen, so fügt man dem Kanal zusätzlich das Schlüsselwort „urgent“ zu.

Tabelle 4: Ein Überblick über die von UPPAAL unterstützte TCTL – Abfragesprache.

Temporaler Operator (Pfadquantor + Temporaloperatoren)	Bedeutung	UPPAAL Syntax
∀	Für alle Pfade	`A`
∃	Es gibt mindestens einen Pfad	`E`
□	Gilt immer	`[]`
◊	Irgendwann gilt	`<>`
¬	Es gilt nicht	`not`
$\Psi \rightarrow \varphi$	Wenn Ψ irgendwann gilt, dann wird auch φ irgendwann gelten.	`-->`
$\Psi \Rightarrow \varphi$	Wenn Ψ gilt, dann gilt auch φ.	`imply`
∀□φ	Für alle Pfade gilt immer φ.	`A[] p`
∀ ◊ φ	Für alle Pfade gilt irgendwann φ.	`A<> p`
∃□φ	Es gibt mindestens einen Pfad, auf dem immer φ gilt.	`E[] p`
∃ ◊ φ	Es gibt mindestens einen Pfad, auf dem irgendwann φ gilt.	`E<> p`

Für mehr Informationen zu den von UPPAAL angebotenen Möglichkeiten sowie Anschauungsbeispiele, welche den Umgang mit UPPAAL zeigen, sei der Leser auf die Quellen verwiesen (Bengtsson et al. 1996; Behrmann et al. 2006).

Mittels den in Abschnitt 5.1 getroffenen Vereinfachungen ist es nun möglich, im nächsten Schritt (siehe nächsten Abschnitt) das Übereinstimmungsprotokoll ESSEN mit Hilfe von

UPPAAL zu modeliieren und seine Fehlertoleranzeigenschaft für bis zu 3 Fehler formal zu verifizieren.

5.2.3 Modell

In den vorangegangen Abschnitten haben wir das Protokoll ESSEN vorgestellt und die zur Validierung und Verifizierung notwendigen Abstraktionen erläutert und begründet. Aufbauend auf diesen Erkenntnissen wird im Folgenden das UPPAAL-Modell im Detail besprochen. Der gesamte Quellcode sowie das UPPAAL-Modell selbst ist im elektronischen Anhang der Arbeit beigefügt.

Das UPPAAL-Modell besteht aus zwei Templates, die beide jeweils in Abbildung 14 und Abbildung 15 dargestellt sind. Abbildung 14 zeigt den „Control“-Prozess. Dieser ist sowohl für die Fehlerinjektion als auch zur Darstellung der Slot-Start- und Endzeitpunkte des verteilten Systems zuständig. Der „Node“-Prozess in Abbildung 15 bildet hingegen das Verhalten der fehlerfreien sowie fehlerhaften Knoten im verteilten System nach.

Der „Control“-Prozess führt zunächst, wie in Abbildung 14 zu sehen, in f zyklisch wiederkehrenden Folgen Fehlerinjektionen im verteilten System durch, welches aus |SendG| + |purSink| Knoten besteht mit |purSink| = 2. Hierzu wählt der Prozess durch eine indeterministische Auswahl innerhalb eines Zyklus aus einem Knotenfehlerinjektionsbereich KJB = {N[minNodeId], …, N[maxNodeId]}, mit minNodeID = 0 und maxNodeId = n – 1, einen Knoten aus und markiert diesen als fehlerhaft (siehe Funktion `injectNodeFault(...)` in Abbildung 14). Ein als fehlerhaft markierter Knoten ändert sein Empfangs- und Sendeverhalten im Vergleich zu einem nicht markierten Knoten „in beliebiger Weise“ (siehe Abbildung 15) und bilden das in Abschnitt 2.3.6 aufgeführte Empfangs- und Sendeverhalten nach.

Zwar kann es durchaus vorkommen, dass während der f Fehlerinjektionszyklen weniger als f Knoten ausgewählt werden (kurz: die Kardinalität der ausgewählten Knotenfehlermenge KFM aus KM kann echt kleiner als f sein, |KFM| < f), da die Auswahl der Knoten aus KJB mit Hilfe einer indeterministischen Auswahl immer wieder neu ausgewählt wird. Da die in UPPAAL durchgeführte Erreichbarkeitsanalyse (zur Prüfung der Fehlertoleranzeigenschaften) sämtliche mögliche Zustandsübergänge berücksichtigt, werden demzufolge alle $\binom{n}{f}$ Knotenfehlerkombinationen berücksichtigt.

Nach Beendigung der Knotenfehlerinjektion beginnt der „Control“-Prozess die Slot-Zeit zu initialisieren. Zeitgleich (siehe Abbildung 15) werden im „Node“-Prozess der Quellwert und die Verteilung der Zugriffe auf die Signaturschlüssel für jeden fehlerhaften Knoten einzeln festgelegt.

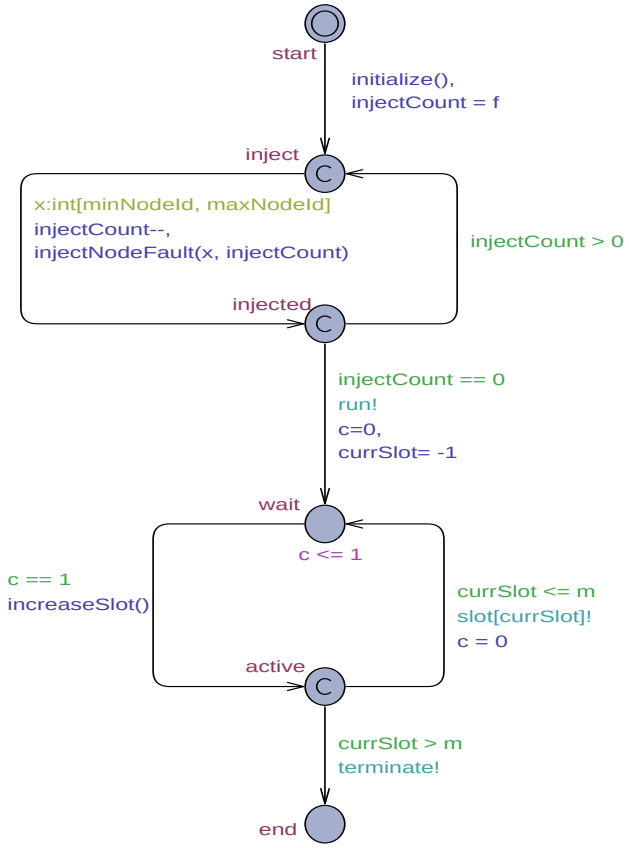


Abbildung 14: der Prozess Control.

Die Entscheidung, welcher Wert an welchen Knoten übertragen wird, wird über die Empfangsmethode `receiveMsg(...)` geregelt. Folglich sendet der fehlerhafte Quellknoten stets alle möglichen Quellwerte (hier: auf zwei Werte beschränkt). Alle als nicht fehlerhaft markierten Knoten wechseln sodann in den Zustand receive, von wo aus sie in Abhängigkeit vom jeweils aktuellen Slotwert entweder in ihren Sendezustand wechseln oder nach Aufruf der Methode `receiveMsg(...)` wieder in ihren vorherigen Zustand receive zurückkehren. Anders sieht es bei den fehlerhaften Knoten aus. Bevor diese erstmals den Zustand receive erreichen, wird zunächst der Zugriff auf die Signaturschlüssel mittels des Parameters `faultyCoop` und der Methode `keyUsageForwarder(...)` zwischen den fehlerhaften Knoten einzeln festgelegt, um „bösartige“ Kooperation auszudrücken.

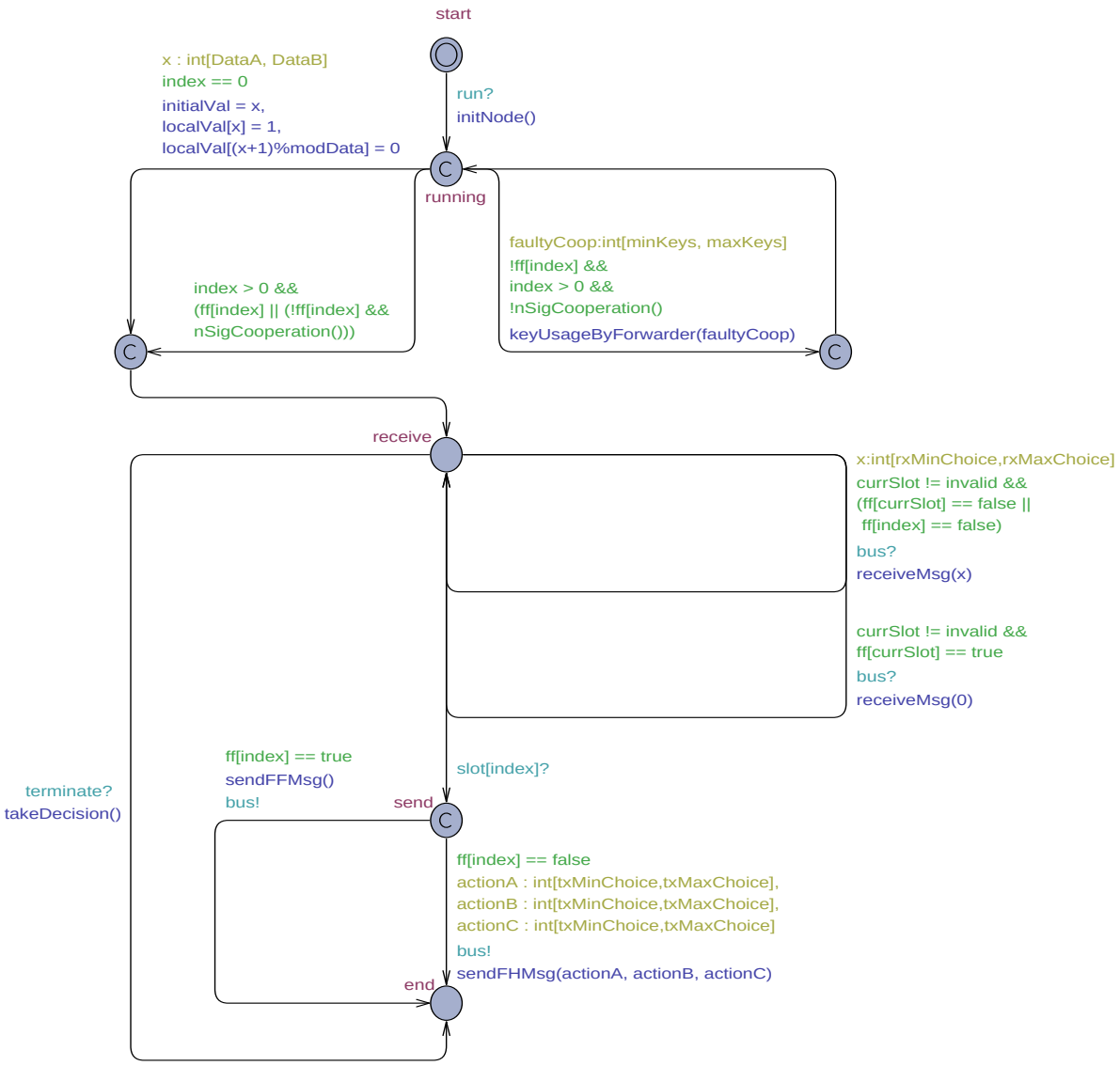


Abbildung 15: der Prozess Node

Hierzu wählt jeder fehlerhafte Knoten in $f-1$ sich wiederholenden Zyklen aus einem Intervall [minKeys = 0, maxKeys = $f-1$] die zur Signierung einer Nachricht verwendeten Signaturschlüssel aus einem globalen Array aus, das zuvor vom „Control“-Prozess beim Übergang vom Zustand **inject** zu Zustand **injected** initialisiert wurde. Die für die Signierung der Nachricht zur Verfügung stehenden Signaturschlüssel werden von den fehlerhaften Knoten einzeln verwaltet. Da außerdem die Auswahl zufällig erfolgt, können somit sämtliche möglichen Schlüsselkombinationen zwischen fehlerhaften Knoten („Node“-Prozessinstanzen) bei der Verifikation berücksichtigt bzw. untersucht werden.

Anschließend befinden sich alle fehlerfreien und fehlerhaften Knoten im Zustand **receive**. Von diesem Zustand aus signalisiert der „Control“-Prozess in m (hier: m = |SendG|) zyklisch

wiederkehrenden Folgen über den Kanal `bus` einen neuen Slot-Beginn. Abhängig vom aktuellen Slotwert, wechselt einer der Knoten („Node“-Prozessinstanzen) in den Zustand **send** (`N[index] == N[currSlot]`). Falls der Knoten fehlerfrei ist, so sendet dieser gemäß Protokollspezifikation entweder eine Data- oder Default-Nachricht. Dies wird durch den Aufruf der Methode `sendFFMsg(...)` erzielt. Die fehlerhaften Knoten rufen im Gegensatz zu den fehlerhaften Knoten Methode `sendFhMsg(...)` auf. Die Werte actionA, actionB und actionC (welche der Methode sendMsg(...) übergeben werden) steuern das Sendeverhalten eines fehlerhaften Knotens. Ein fehlerhafter Knoten im UPPAAL-Modell kann bis zu drei Nachrichten (zwei Data-Nachricht und eine Default-Nachricht) übertragen. Abhängig vom übergebenen Wert actionA, actionB und actionC wird das folgende Sendeverhalten beim fehlerhaften Knoten ausgelöst:

1. actionA = 0: Falls der fehlerhafte Knoten eine Data-Nachricht vom Wert 0 gespeichert hat, dann signiert er diese nur mit seinem privaten Signaturschlüssel.
2. actionA = 1: Falls der Sender eine Data-Nachricht vom Wert 0 gespeichert hat, dann signiert er diese mit den Signaturschlüsseln, auf die er Zugriff hat (d.h. insbesondere mit den Schlüsseln aller fehlerhaften Knoten, siehe ggf. Abbildung 14).
3. actionB = 0: Falls der fehlerhafte Knoten eine Data-Nachricht vom Wert 1 gespeichert hat, dann signiert er diese nur mit seinem privaten Signaturschlüssel.
4. actionB = 1: Falls der Sender eine Data-Nachricht vom Wert 1 gespeichert hat, dann signiert er diese mit den Signaturschlüsseln der fehlerhaften Knoten, auf die er Zugriff hat (siehe ggf. Abbildung 14).
5. actionC = 0: Der fehlerhafte Knoten signiert eine Default-Nachricht nur mit seinem privaten Signaturschlüssel.
6. actionC = 1: Der fehlerhafte Knoten signiert eine Default-Nachricht mit den Signaturschlüsseln der fehlerhaften Knoten, auf die er Zugriff hat.

Mit Hilfe der Methode `receiveMsg(...)` wird auf die Nachricht zugegriffen. Die Übertragung der Nachricht wird anschließend allen Knoten über den Kanal `bus` signalisiert. Anschließend wechselt der jeweilige Sendeknoten in den Zustand **end** und terminiert. Falls der Senderknoten fehlerfrei war, greifen die fehlerfreien Knoten über die Methode `receiveMsg(...)` auf die Nachricht des Senderknotens zu. Der Parameter x, der an die Methode `receiveMsg(...)` übergeben wird, wird im Falle eines fehlerfreien Sendeknotens nicht weiter berücksichtigt. Falls der Empfänger fehlerhaft ist, so steuert der Para-

meter x (dessen Wert im Intervall [0, 3] liegt) das Empfangsverhalten. Im Falle eines fehlerhaften Sendeknotens bestimmt der Parameter x das Empfangsverhalten eines jeden Empfangsknotens. Abhängig vom übergebenen Wert x wird das folgende Empfangsverhalten beim (fehlerhaften bzw. fehlerfreien) Empfänger ausgelöst:

7. x = 0: Der Empfänger erwartet den Empfang einer Data-Nachricht vom Wert 0, falls der Sendeknoten keine Data-Nachricht vom Wert 0 übertragen hat, ignoriert der jeweilige Empfänger den Nachrichtenempfang. Andernfalls liest der Empfänger die Nachricht und führt die in **Algorithmus 1** beschrieben Methoden aus.
8. x = 1: Analog zu Punkt 1, jedoch wird eine Data-Nachricht vom Wert 1 erwartet.
9. x = 2: Ähnlich zu Punkt 2, statt einer Data-Nachricht wird der Empfang einer Default-Nachricht erwartet.
10. x = 3: Symbolisiert den Nicht-Empfang einer Nachricht (ist mit dem Empfang von Rauschen geleichzusetzen).

Alle Knoten aus der Gruppe SendG betreten genau einmal den Zustand **send**, bevor sie in den Zustand **end** übergehen (siehe Abbildung 15 unten). Die Knoten aus der Gruppe PurSinkG hingegen wechseln vom Zustand **receive** in den Zustand **end**, ohne den Zustand **send** zu betreten. Jedoch rufen die Knoten aus der Gruppe PurSinkG im Gegensatz zu den Knoten aus der Gruppe SendG beim Übergang zum Zustand **end** die Methode `takedecision()` auf. Diese Methode entspricht den Zeilen 34 bis 41 in **Algorithmus 1**. Nach Aufruf der Methode `takedecision()` werden sodann die finalen Übereinstimmungswerte der PurSinkG-Knoten in `finalVal[2]`, welcher bis zu |PurSinkG| verschiedene Werte aufnehmen kann, hinterlegt (hier gilt: | PurSinkG | = 2).

5.2.4 Ergebnis

Sämtliche Verifikationsformeln, die mit „Validierung“ markiert sind, dienen ausschließlich zur Validierung des UPPAAL-Modells (siehe Tabelle 5). Sie prüfen die „safety“ und „liveness“ Eigenschaften des Modells. Die Letzten beiden Verifikationsformeln hingegen prüfen, ob die Bedingungen IC1 und IC2 auch im Fehlerfall stets erfüllt werden. Sämtliche geprüften Eigenschaften sind in Tabelle 5 zusammengefasst. Die Verifikation von IC1 und IC2 konnte für $0 \leq f \leq 3$ gezeigt werden. Die Verifikation für höhere Knotenfehleranzahl $f > 3$ konnten wegen der Zustandsraumexplosion mittels UPPAAL nicht untersucht werden.

Tabelle 5: Formale Verifikation zur Prüfung der Fehlertoleranzeigenschaften mittels UPPAAL.

Bezeichnung	Eigenschaft	Beschreibung
Validierung	`A[](!control.end)|| ((finalVal[0]==0)|| (finalVal[0]==1)|| (final[0] == 2))`	Andere finale Endwerte als {0, 1, 2} treten niemals auf.
Validierung	`E<> final[0] == 2`	Der finale Endwert 2 wurde geprüft.
Validierung	`E<> final[0] == 1`	Der finale Endwert 1 wurde geprüft.
Validierung	`E<> final[0] == 0`	Der finale Endwert 0 wurde geprüft.
Validierung	`E<> ( !ff[0] && !ff[f+1])`	Fehlerhafter Quell- und Basisknoten wurden geprüft
Validierung	`E<> (!ff[0] && ff[m-1])`	Fehler in QuellG, und ErwG wurden geprüft
Validierung	`A[] !control.end || (source.end && forw1.end && forw2.end && sink0.end && sink1.end)`	Alle Knoten erreichen ihren Endzustand
Verifikation IC2	`A[] ( !(slots.end && ffNode[0]))|| (finalVal[0] == initialVal && finalVal[1] == initialVal)`	Wenn der Quellknoten fehlerfrei ist, dann terminieren die SinkG-Knoten mit dem Initialwert des Quellknotens.
Verifikation IC1	`A[]!slots.end || (finalVal[0] == finalVal[1])`	Beide SinkG-Knoten terminieren mit demselben Endwert.

5.3 Simulation und Auswertung

Dieser Abschnitt befasst sich mit der simulativen Analyse des fehlertoleranten Verhaltens des Protokolls ESSEN für größere Anzahlen von Knotenfehlern ($0 \leq f_N \leq 14$).

5.3.1 Das Knoten- und Kanal-Fehlermodell für die Simulation

Der Simulator basiert auf dem UPPAAL-Modell. Folglich ist das Fehlermodell in der Simulation auf ähnliche Weise wie im UPPAAL-Modell realisiert worden. Der Unterschied zum UPPAAL-Modell besteht darin, dass in der Simulation die Anzahl der Kotenfehler im Modell auf $0 \leq f_N < 15$ beschränkt wurde (entspricht **FM0**). Die Untersuchung für höhere Knotenfehlerzahlen macht kaum Sinn, da die für aussagekräftige Ergebnisse benötigte Simulationszeit unverhältnismäßig groß wird.

Der in Java implementierte Simulator sowie die aus dem Simulator gewonnenen und ausgewerteten Ergebnisse werden in Abschnitt 5.3.2 und 5.3.3 näher erläutert.

5.3.2 Design des Simulators

Zur Durchführung der Experimente wurde eine Fehlerinjektionsumgebung gemäß (Mei-Chen Hsueh et al. 1997) in der Programmiersprache Java entwickelt. Es handelt sich hierbei um einen rein software-basierten Ansatz, auch bekannt als „software implemented fault injection“ (kurz: SWIFI). Folglich sind sämtliche Komponenten der Fehlerinjektionsumgebung nach (Mei-Chen Hsueh et al. 1997) in Software realisiert worden. Dazu zählt auch das Zielsystem (hier: das verteile System, auf dem das Übereinstimmungsprotokoll ESSSEN ausgeführt wird).

Im Folgenden werden die einzelnen Komponenten der in der Programmiersprache Java umgesetzten Fehlerinjektionsumgebung (siehe auch Abbildung 16) erläutert. Jede dieser Komponenten setzt sich aus einer oder mehreren Java Klassen zusammen (die aber nicht im Detail bzw. auf Quellcodebene besprochen werden):

- Controller (Kontroller): Diese Klasse ist für die Initialisierung und Steuerung der Experimente zuständig. Jedes Experiment enthält Informationen über die Fehlerart sowie den Fehlerort, an dem Fehler in das Zielsystem zu injizieren sind.
- Fault Injector (Fehlerinjektor): Mit Hilfe des Fehlerinjektors werden Fehler ins Zielsystem injiziert. Der hier verwendete Fehlerinjektor besteht aus zwei Teilen, und zwar

aus dem Fehlerinjektionsauslöser und dem Fehlerinjektionsmodul. Der Fehlerinjektionsauslöser legt den Zeitpunkt der Fehlerinjektion im Zielsystem fest und das Fehlerinjektionsmodul den zu injizierenden Fehlertyp (siehe Abschnitt 5.3.1).

- Workload Generator (Lastgenerierung): Diese Klasse ist gemäß dem Schema von (Mei-Chen Hsueh et al. 1997) für die Lastgenerierung im Zielsystem zuständig. Da aber die Last im Rahmen dieser Untersuchung nicht weiter betrachtet wird, wurde diese Klasse nicht implementiert. Begründung: Das Ziel der Analyse besteht nur darin herauszufinden, ob das Übereinstimmungsprotokoll über die geforderten Fehlertoleranzeigenschaften verfügt oder nicht.
- Monitor and Data Collector (Monitor und Datensammler): Die Aufgabe dieser Klasse besteht darin, die Ereignisse und Zustände des Zielsystems während der Fehlerinjektion aufzuzeichnen, sodass der Datenanalyser diese für die weitere Verarbeitung nutzen kann (siehe nächsten Unterpunkt).
- Data Analyser (Datenanalyser): Der Datenanalyser ist für die Auswertung der aufgezeichneten Daten zuständig. Im Rahmen dieser Arbeit führt der Datenanalyser insbesondere eine boolesche Auswertung durch und liefert dabei die folgenden beiden Ergebnisse (vgl. Quelle: Hsueh et al.):
    - In dem durchgeführten Fehlerinjektionsexperiment wurde Fehlertoleranz erzielt.
    - In dem durchgeführten Fehlerinjektionsexperiment wurde keine Fehlertoleranz erzielt.

  Im letztgenannten Fall wird das zu untersuchende Zielsystem terminiert und ein Trace ausgegeben. Dieser Trace enthält sämtliche Ereignisse (übertragene Nachrichten, empfangene Nachrichten, etc.) und Zustandsänderungen, die während des durchgeführten Fehlerinjektionsexperiments aufgetreten sind. Dieser Trace dient der Vereinfachung der anschließenden Ursachenanalyse.
- Das Zielsystem ist der Bereich, auf den die im Fehlerinjektor spezifizierten Fehler einwirken (hier: Zielsystem = das verteilte System, auf dem das Übereinstimmungsprotokoll ESSEN ausgeführt wird).

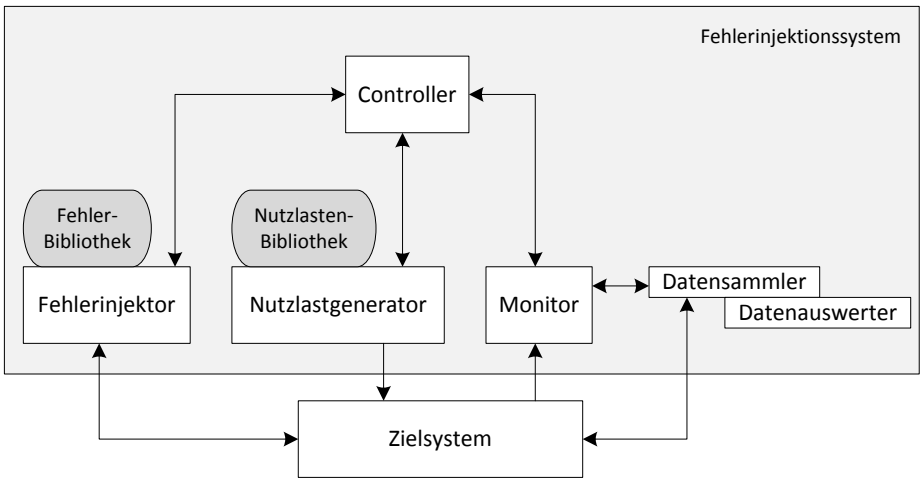


Abbildung 16: Basiskomponenten der Fehlerinjektionsumgebung gemäß (Mei-Chen Hsueh et al. 1997).

Bei der Durchführung der Experimente mussten, wie bereits in Abschnitt 5.3.1 ausgeführt, einige Einschränkungen vorgenommen werden. Andernfalls hätte die Anzahl der durchzuführenden Experimente den zeitlichen Rahmen dieser Arbeit gesprengt.

Der zur Überprüfung der Fehlertoleranzeigenschaft eingesetzte Simulator verfügt über diverse einzustellende Parameter (siehe auch Abschnitt 5.1), mit deren Hilfe die verschiedene Serien von Fehlerinjektionsexperimenten generiert und ausgeführt werden können. Im Folgenden werden nur die wichtigsten Parameter aufgeführt:

- Anzahl der Nachrichtentypen: *N_MESSAGE_TYPE(int)*
- Anzahl der fehlerhaften Knoten: *N_FAULTS(int)*
- Quellknoten als fehlerhaft/fehlerfrei konfigurieren: *SOURCE_PERMANENT_FAULT(int)*
- Anzahl der verwendeten Versenderknoten: *N_FORWARDER(int)*
- Anzahl der Knoten in Gruppe BasisG: *GROUP_ONE(int)*
- Anzahl der Knoten in Gruppe ErwG: *GROUP_TWO(int)*
- Anzahl der simulierten Runden: *RUNS*
- Inhalt der Nachricht im fehlerfreien und fehlerhaften Fall: Der Inhalt kann fest konfiguriert oder durch den Einsatz eines Pseudozufallsgenerators (hier: MRG32j3a (L'Ecuyer 1998)) per Zufallsauswahl ausgewählt werden.

5.3.3 Ergebnis der Simulation

Insgesamt wurden bis zu 10^9 Runden sowie maximal 54 Knoten simuliert ($0 \leq f_N \leq 14$, siehe Tabelle 6). In keinem einzigen der durchgeführten Simulationsexperimente konnte ein Gegenbeispiel gefunden werden, das die Fehlertoleranzeigenschaft des Protokolls verletzt hätte. Außerdem wurde im Rahmen der Untersuchung für verschiedene Anzahlen an Knotenfehlern ein Plausibilitätscheck durchgeführt. Mit Hilfe des Plausibilitätschecks konnte gezeigt werde, dass der Simulator für eine kleinere Knotenfehlerzahl (hier: $0 \leq f_N \leq 6$ Knotenfehler) ein Gegenbeispiel findet, wenn man statt der notwendigen Knotenzahl $n \geq 3f + max(0, f-2)$ eine um eins reduzierte Knotenzahl $n = 3f + max(0, f-2) - 1$ im Simulator konfiguriert (im Anhang sind die Tabelle mit den gefundenen Gegenbeispiele in der CD-ROM hinterlegt worden).

Tabelle 6: Fehlerinjektionsexperimente.

n	f_N	Fehlerinjektionsexperimente
1	0	ca. $10^{2\,(3)}$
3	1	ca. $10^{3\,(3)}$
6	2	ca. $10^{4\,(3)}$
10	3	ca. $10^{5\,(3)}$
14	4	ca. 10^9
18	5	ca. 10^9
22	6	ca. 10^9
…	…	…
54	14	ca. 10^9

In allen durchgeführten Fehlerinjektionsexperimenten hat das Übereinstimmungsprotokoll ESSEN stets eine Übereinstimmung gemäß IC1 und IC2 erzielt. Folglich kann die Korrektheit des Protokolls auf statistischer Basis gefolgert werden – zusätzlich zu der formalen Verifikation mit UPPAAL für $f \leq 3$.

5.4 Korrektheitsbeweis

Obwohl das Protokoll ESSEN mit Hilfe des Werkzeugtools UPPAAL sowie mit Hilfe einer Simulation untersucht wurde, reichen die gewonnenen Ergebnisse nicht in allen Fällen aus,

[3] Diese Fälle wurden auch mittels UPPAAL verifiziert.

um daraus die Korrektheit des Protokolls formal zu schlussfolgern, jedoch können sie als sehr starkes Indiz für die Korrektheit des Protokolls (in dem Fall) für beliebige $f > 3$ betrachtet werden. Um die Korrektheit des Protokolls für beliebige Fehler (hier: $f > 3$) streng nachzuweisen, wird in Kapitel 6 ein formaler Beweis von ESSEN durchgeführt.

5.5 Zusammenfassung

Das Protokoll ESSEN wurde mit dem Ziel einer minimalen Kommunikationskomplexität bzgl. der Lösung des Byzantinischen Übereinstimmungsproblems entworfen, jedoch lässt es sich bzgl. der Kommunikationskomplexität weiter verbessern. Ein möglicher Ansatz, der sich während der Arbeit herauskristallisiert hat, ist der Einsatz einer sogenannten Signaturverschmelzungsfunktion. Die Idee dahinter ist simpel: Mit Hilfe eines speziellen Signaturverfahrens soll ein Knoten (empfangene bzw. gespeicherte Nachrichten) die Möglichkeit erhalten, Nachrichten miteinander zu kombinieren, wenn sich diese nur in den Signaturquellen unterscheiden.

Als Basis für diesen Ansatz wurde das in (Echtle und Kimmeskamp 2010; Echtle 2005) beschriebene Signaturverfahren verwendet. Das ursprüngliche Signaturverfahren wurde so modifiziert, dass es die Signaturverschmelzungseigenschaft besitzt. Die genaueren Details zum Signaturverfahren werden in Kapitel 7 beschrieben. Dadurch könnte im Falle von Übereinstimmungsprotokollen die Anzahl der Kommunikationsrunden (die Anzahl der Phasen im Falle von ESSEN) sowie die Anzahl der zu übertragenden Nachrichten weiter reduziert werden. Die Vor- und Nachteile dieses Ansatzes werden im Kapitel 7 ausführlich beleuchtet. Der Vollständigkeit halber sei vorab erwähnt, dass dieser Ansatz jedoch nicht auf Übereinstimmungsprotokolle angewendet werden kann, bei der die Entscheidung über den Endwert unabhängig von der Anzahl der Signaturen getroffen werden kann (wie etwa bei SM (Lamport et al. 1982) bzw. deren Derivaten (Jochim und Forest 2010)).

6 Korrektheitsbeweis von ESSEN

„Was beweisbar ist, soll in der Wissenschaft nicht ohne Beweis geglaubt werden. " – Richard Dedekind.

„Der Beweis ist der einzige Weg, um anderen Wahrheiten begreiflich zu machen, welche sie unmittelbar nicht einsehen können." – Arthur Schopenhauer

Zur Durchführung des Korrektheitsbeweises wird das synchrone verteilte System als Graph modelliert, bei dem lediglich die Reihenfolge der Sende-Slots betrachtet wird. Wie bereits erwähnt verfügt im Protokoll ESSEN jeder Knoten über genau einen Sendeslot, ausgenommen die Pure-Sink-Knoten, die überhaupt nicht senden. Die Knoten V des Graphen stellen die Knoten des verteilten Systems dar, während die Kanten E des Graphen die zeitliche Reihenfolge des Sendens (entsprechend dem Sendeslot) ausdrücken. Von jedem Knoten führt eine Kante zu dem Knoten, der im unmittelbar nachfolgenden Slot sendet. Folglich bilden die Kanten eine Kette, die von dem zuerst sendenden Knoten (dem Quellknoten) zu dem zuletzt sendenden Knoten führt. Weitere Kanten gibt es nicht. Ein solcher Graph wird als Kommunikationsgraph bezeichnet.

Definition 6.1

Ein Kommunikationsgraph G = (V, E) ist eine Kette mit einer endlichen Knotenmenge $V \neq \emptyset$ und einer endlichen Kantenmenge $E \subseteq \{(u, v) \mid u, v \in V, u \neq v\}$. Die Elemente von V heißen Knoten („vertex") von G und die Elemente von E heißen Kanten („edge") von G. Falls es eine Kante $e = (u, v)$ im Kommunikationsgraph G für zwei Knoten $u, v \in V$ gibt (kurz: e ∈ E mit $e = (u, v)$), dann heißt u Anfangsknoten und v Endknoten von e. Ein Kommunikationsgraph ist zyklenfrei. Für genau einen Knoten des Kommunika-tionsgraphen (den Anfangsknoten der Kette) gilt, dass keine Kante zu ihm hinführt und genau eine Kante von ihm wegführt. Für genau einen weiteren Knoten des Kommunikationsgraphen (den Endknoten der Kette) gilt, dass genau eine Kante zu ihm hinführt und keine Kante von ihm wegführt. Für jeden anderen Knoten eines Kommunikationsgraphen gilt, dass jeweils genau eine Kante zu ihm hinführt und genau eine Kante von ihm wegführt.

Definition 6.2

Den Anfangs- bzw. den Endknoten eines Subgraphen Q = (V', E') mit $V' \subseteq V$ und $E' \subseteq E$, wobei Q ein Kommunikationsgraph gemäß **Definition 6.1** ist, bezeichnen wir mit $v_Q^a = v_Q^0$ bzw. $v_Q^e = v_Q^{q-1}$ wobei q = $|V'|$ und im Subgraphen Q die Indizierung der Knoten $v_Q^0, v_Q^1, \ldots, v_Q^{q-1}$entsprechend ihrer Position in der Kette vorgenommen wird.

Definition 6.3

Die Menge der fehlerfreien bzw. fehlerhaften Knoten in einem Kommunikationsgraphen G = (V, E) bezeichnen wir mit

$V_{ff} \subseteq V$ bzw. $V_{fh} \subseteq V$, wobei $V_{ff} \cup V_{fh} = V$ und $V_{ff} \cap V_{fh} = \emptyset$. Dann ist $|V_{ff}| \geq n - f$ und $|V_{fh}| \leq f$, weil höchsten f Knoten als fehlerhaft angenommen werden.

Definition 6.4

Es sei G = (V, E) ein Kommunikationsgraph mit G = (V, E). Ein Weg in G ist eine endliche Folge

$$W_G = (v_V^0, e_V^0, v_V^1, \ldots, v_V^{k-2}, e_V^{k-2}, v_V^{k-1}) \quad \text{mit} \quad e_V^i = (v_V^i, v_V^{i+1}) \quad \text{für} \quad 0 \leq i < k$$

Man bezeichnet v_V^0 als den Anfangsknoten, v_V^{k-1} als den Endknoten und k als die Länge des Weges W_G (Anzahl der Knoten des Wegs, wobei Anfangs- und Endknoten mitgezählt werden).

Definition 6.5

Ein synchrones verteiltes System wird repräsentiert durch einen Kommunikationsgraphen G = (V, E), welcher aus einer endlichen Knotenmenge V, die durch ein Kommunikationsnetz miteinander verbunden sind, und einer endlichen Kantenmenge E besteht, welche die Zugriffsreihenfolge der Knoten auf den Kommunikationskanal abbildet. Für zwei beliebige Knoten v_V^i und v_V^j, die über einen Weg W_V = (v_V^i,…, v_V^j) der Länge j – i + 1 > 0 miteinander verbunden sind, gilt: Der Sende-Slot von Knoten v_V^i liegt zeitlich vor dem Sende-Slot von Knoten v_V^j. Daher wird Knoten v_V^i als Vorgänger von v_V^j und Knoten v_V^j als Nachfolger von v_V^i bezeichnet.

Definition 6.6

Es sei G = (V, E) ein Kommunikationsgraph und $v_V^i{}'$ ein beliebiger Knoten in $V' \subseteq V$. Dann liefert die Funktion Vorg($v_V^i{}'$) die Menge der Knoten in V', die Vorgänger von $v_V^i{}'$ sind (siehe ggf. **Tabelle 7**).

Definition 6.7

Es sei G = (V, E) ein Kommunikationsgraph und $v_V^i{}'$ ein beliebiger Knoten in $V' \subseteq V$. Dann liefert die Funktion Nachf($v_V^i{}'$) die Menge der Knoten in V', die Nachfolger von $v_V^i{}'$ sind (siehe ggf. **Tabelle 7**).

Definition 6.8

Es sei G = (V, E) ein Kommunikationsgraph, v_V ein beliebiger Knoten aus V und M eine Nachricht. Dann liefert die Funktion sign(v_V, M) den Wert true, wenn Knoten v_V die Nachricht M signiert hat, sonst liefert sign(v_V, M) den Wert false.

Definition 6.9

Es sei G = (V, E) ein Kommunikationsgraph und v_V ein beliebiger Knoten in V. Dann gibt die Funktion finalM(v_V) den Wert an, für den sich v_V nach Abschluss des Protokolls endgültig entschieden hat.

Definition 6.10

Es sei G = (V, E) ein Kommunikationsgraph und v_V ein beliebiger Knoten in G. Die Mengen V_{data}und $V_{default}$ bezeichnen die Menge der Knoten, die eine Data-Nachricht bzw. eine Default-Nachricht (ko)signiert und gesendet haben. Entsprechend gibt $|V_{data}|$ bzw. $|V_{default}|$ die Anzahl der Knoten an, die eine Data- bzw. eine Default-Nachricht (ko)signiert und gesendet haben. Des Weiteren wird definiert: Die Mengen $V_{ff,data}$ und $V_{ff,default}$ bzw. $V_{fh,data}$ und $V_{fh,default}$ bezeichnen die Menge der fehlerfreien bzw. fehlerhaften Knoten, die eine Data-Nachricht bzw. eine Default-Nachricht (ko)signiert und gesendet haben.

Definition 6.11

Es sei G = (V, E) ein Kommunikationsgraph und v_V ein beliebiger Knoten in V. Die von v_V gesandte Nachricht sein M. Dann bezeichnet signNum(v_V) die Anzahl der Knoten, die die Nachricht M signiert haben, die Knoten v_V gesendet hat (inklusive der Signatur von Knoten v_V). Ebenso bezeichnet signNum(M) die Anzahl der Knoten, die M signiert haben (Die Funktion signNum ist hinsichtlich ihrer Parametrisierung überladen).

Definition 6.12

Es sei G = (V, E) ein Kommunikationsgraph, v_V ein beliebiger Knoten in G und $Gruppe$ eine Teilmenge von V, ($Gruppe \subseteq V$). Dann liefert die Funktion $\text{signNum}_{|Gruppe}(v_V)$ die Anzahl der Knoten aus der $Gruppe$, die die Nachricht v_V signiert bzw. kosigniert haben. Analog bezeichnet $\text{signNum}_{|Gruppe}(M)$ die Anzahl der Knoten aus der $Gruppe$, die die Nachricht M signiert haben (Die Funktion $\text{signNum}_{|Gruppe}$ ist hinsichtlich ihrer Parametrisierung überladen).

Beispiel zu **Definition 6.12**:

Sei beispielweise die von Knoten v_V übertragene Nachricht vom

- Quellknoten v_V^0,
- höchstens $f - 2$ fehlerfreie Knoten aus der Gruppe V',
- sowie 2 Knoten aus der Gruppe V''

und von keinem weiteren Knoten signiert worden. Diese Information lässt sich mit Hilfe der Funktion $\text{signNum}_{|Gruppe}(v_V)$ wie folgt darstellen:

$$\text{signNum}(v_V) = \text{signNum}_{|\{v_V^0\}}(v_V) + \text{signNum}_{|V'_{ff}}(v_V) + \text{signNum}_{|V''}(v_V)$$
$$\leq 1 + (f-2) + 2 \;=\; f + 1$$

Definition 6.13
Es werden die folgenden Bezeichnungen verwendet: SendG = V, BasicG = V', ExtG = V''. Außerdem werden Knoten, die keine Nachrichten senden, nicht betrachtet, d.h. purSink = $\emptyset$.

Mit diesen Festlegungen gilt: SinkG = $V' \cup V''$ und QuellG = $V \setminus (V' \cup V'') \neq \emptyset$ mit $V' \cup V'' \subset V$ und $V' \cap V'' = \emptyset$. Die Knotenmenge V entspricht einen Kommunikationsgraphen, der die Definitionen **Definition 6.1** bis **Definition 6.13** erfüllt.

Definition 6.14 (Menge P, E, S)

- Mit $P = V'' \cap V_{Data} \cap V_{ff}$ bezeichnen wir die Menge der fehlerfreien Knoten in V'', die eine Data-Nachricht senden.
  Hinweis: Eine korrekte Data-Nachricht ist stets vom Quellknoten und mindestens einem Knoten aus V' signiert worden.
- Mit $R = V'' \cap V_{Default} \cap V_{ff} \cap Vorg(v_P^e)$ bezeichnen wir die Menge der fehlerfreien Knoten in V'', die eine Default-Nachricht senden und Vorgänger von v_P^e sind.
  Mit anderen Worten: R sind die fehlerfreien Default-Nachricht-Sender, denen noch mindestens ein fehlerfreier Data-Nachricht-Sender nachfolgt.
  Satz: *Für die Menge R gilt (siehe Abbildung 17):*
  $|V_P| \geq 1, |V_R| \geq 1 \Longrightarrow \forall v_R, \exists v_P : W_V(v_R, \dots, v_P)$.
  Beweis: Trivial. Folgt aus der Definition der Menge P und den **Definition 6.4** bis **Definition 6.7**.
- Mit $S = V'' \cap V_{Default} \cap V_{ff} \cap Nachf(v_P^e)$ bezeichnen wir die Menge der fehlerfreien Knoten in V'', die eine Default-Nachricht senden und Nachfolger von v_P^e sind. Mit anderen Worten: S sind die fehlerfreien Default-Nachricht-Sender, denen kein fehlerfreier Data-Nachricht-Sender nachfolgt.
  Satz: *Für die Menge S gilt (siehe Abbildung 18):*
  $|V_P| \geq 1, |V_S| \geq 1 \Longrightarrow \forall v_P : W_V(v_P, \dots, v_S^a)$.
  Beweis: Trivial. Folgt aus den Definitionen der Mengen P und R sowie den **Definition 6.4** bis **Definition 6.7**.

Definition 6.15 (Menge Q, T, Θ)

- Mit $Q = V'' \cap V_{Default} \cap V_{fh} \cap Vorg(v_P^e)$ bezeichnen wir die Menge der fehlerhaften Knoten in V'', die eine Default-Nachricht senden und Vorgänger von v_P^e sind.
  Mit anderen Worten: Q sind die fehlerhaften Default-Nachricht-Sender, die den fehlerfreien Data-Nachricht-Sender v_P^e vorangehen.
  Satz: *Für die Menge Q gilt (siehe Abbildung 18):*
  $|V_P| \geq 1, |V_Q| \geq 1 \Longrightarrow \forall v_Q : W_V(v_Q, \dots, v_P^e)$.
  Beweis: Trivial. Folgt aus der Definition der Menge P und den **Definition 6.4** bis **Definition 6.7**.
- Mit $T = V'' \cap V_{Default} \cap V_{fh} \cap Nachf(v_p^e)$ bezeichnen wir die Menge der fehlerhaften Knoten in V'', die eine Default-Nachricht senden und Nachfolger von v_P^e sind.
  Mit anderen Worten: T sind die fehlerhaften Default-Nachricht-Sender, denen kein fehlerfreier Data-Nachricht-Sender nachfolgt.
  Satz: *Für die Menge T gilt (siehe Abbildung 18):*
  $|V_P| \geq 1, |V_T| \geq 1 \Longrightarrow \forall v_P : W_V(v_P, \dots, v_T^a)$.
  Beweis: Trivial. Folgt aus den Definitionen der Mengen P und Q sowie den **Definition 6.4** bis **Definition 6.7**.
- Mit $\Theta = V'' \cap V_{Data} \cap V_{fh}$ bezeichnen wir die Menge der fehlerhaften Knoten in V'', die eine Data-Nachricht senden (siehe Abbildung 18).

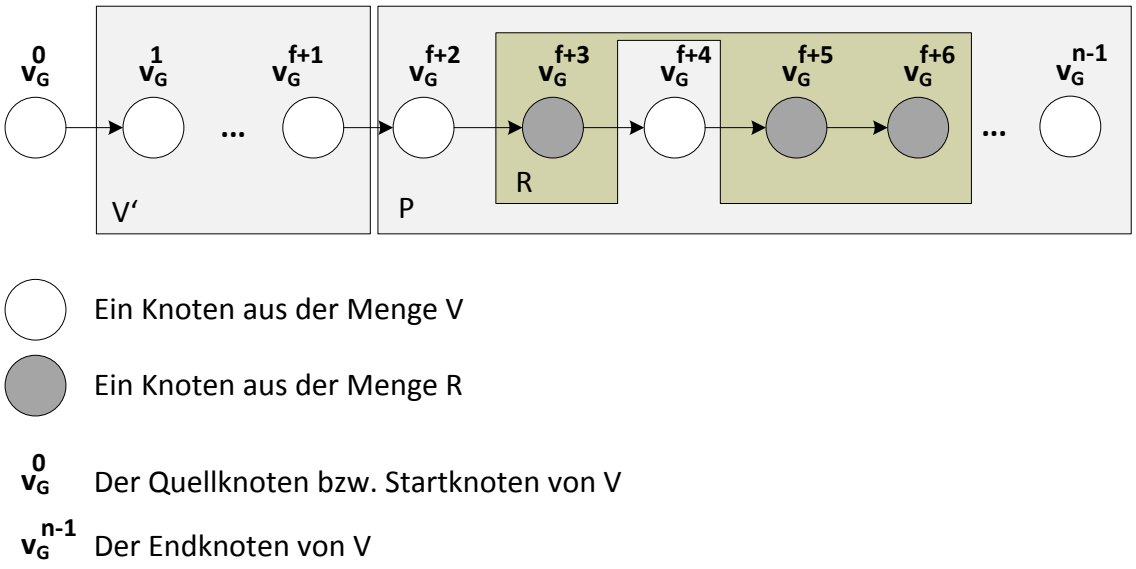


Abbildung 17: Die Mengen $R \subseteq V''$ und $P \subseteq V''$ im Graphen G = (V, E) mit $V' \cup V'' \subseteq V$.

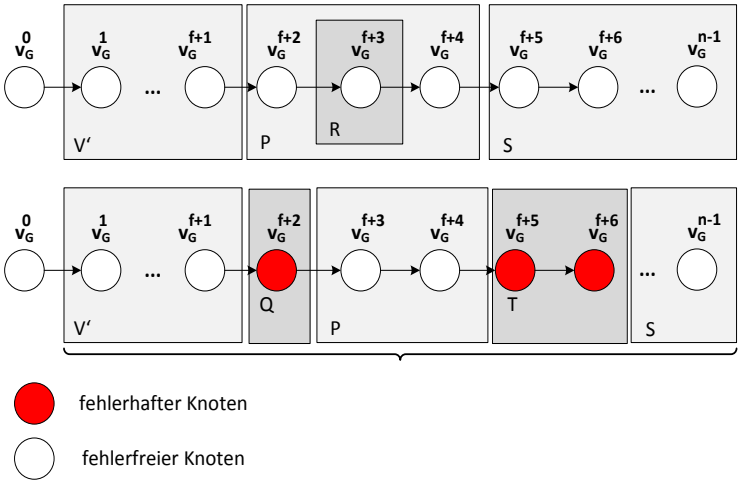


Abbildung 18: Kommunikationsgraph G = (V, E).

Tabelle 7: Vorgänger und Nachfolger des Knotens v_P^e (letzter fehlerfreier Knoten, der einen Data-Nachricht sendet)

Knotengruppe	Vorgänger von v_P^e	Nachfolger von v_P^e
$V''_{ff,Data}$	$Vorg(v_P^e) \cap V''_{ff,Data} = \mathrm{P}$	$Nachf(v_P^e) \cap V''_{ff,Data} = \emptyset$
$V''_{ff,Default}$	$Vorg(v_P^e) \cap V''_{ff,Default} = \mathrm{R}$	$Nachf(v_P^e) \cap V''_{ff,Default} = \mathrm{S}$
$V''_{fh,Data}$	$Vorg(v_P^e) \cap V''_{fh,Data} \subseteq \Theta$	$Nachf(v_P^e) \cap V''_{fh,Data} \subseteq \Theta$
$V''_{fh,Default}$	$Vorg(v_P^e) \cap V''_{fh,Default} = \mathrm{Q}$	$Nachf(v_P^e) \cap V''_{fh,Default} = \mathrm{T}$

LEMMA 1: *Gegeben ist ein Kommunikationsgraph G = (V, E). Sei v_V^0 der Quellknoten in V. Wenn $v_V^0 \in \mathrm{V}_{ff}$ gilt, dann gilt:*

$$\mathrm{IC2}: \Leftrightarrow \quad \left(\forall v_V^i \; mit \; i \neq 0: v_V^i \in \mathrm{V}_{ff} \quad \Rightarrow \quad \mathrm{finalM}\left(v_V^i\right) = \mathrm{finalM}(v_V^0)\right).$$

Der Beweis gliedert sich in Einzelbeweise für die folgenden Behauptungen, aus denen insgesamt der Beweis von Lemma 1 entwickelt wird.

Behauptung 1. Wenn $|V_{fh}| \leq 1$ gilt, dann folgt **LEMMA 1**.

Beweis: Für $f \leq 1$ gilt laut Protokollspezifikation $V'' = \emptyset$. Laut Annahme in **LEMMA 1** gilt $v_V^0 \in \mathrm{V}_{ff}$. Folglich empfangen alle fehlerfreien Knoten aus V' die gleiche korrekte Nachricht des Quellknotens. Kein Knoten aus V' empängt eine davon abweichende Nachricht, die der Quellknoten v_V^0 signiert hat. Gemäß SC2 sendet der

letzte fehlerfreie Knoten in V', der mit $v^e_{V_{ff}}$ bezeichnet wird, eine Data-Nachricht mit mindestens 2 Signaturen:

$$signNum\left(v^e_{V_{ff}}\right) = signNum_{|\{v^0_V\}}\left(v^e_{V_{ff}}\right) + signNum_{|V'}\left(v^e_{V_{ff}}\right) \qquad 6.1$$

$$\Rightarrow signNum\left(v^e_{V_{ff}}\right) \geq 1 + 1 \geq f + 1$$

Die Nachricht trägt also mindestens eine fehlerfreie Signatur. Da die Signatur eines fehlerfreien Knotens nicht unbemerkt gefälscht werden kann, folgt entsprechend AG1 die Behauptung 1.

Behauptung 2. Wenn $1 < |V_{fh}| \leq f$*, dann sendet kein fehlerfreier Knoten in* V'' *eine Default-Nachricht. Weiterhin gilt, dass der letzte fehlerfreie Knoten in* V' stets eine Data-Nachricht mit mindestens $signNum\left(v^e_{V'_{ff}}\right) \geq 2 + |V''_{fh}|$ Signaturen sendet.

Beweis: Zunächst wird der zweite Satz von Behauptung 2 bewiesen, indem die Anzahl der Signaturen bestimmt wird. Laut der f-Fehlerannahme enthält die Knotenmenge V' höchstens

$$|V'_{fh}| = |V_{fh}| - |V''_{fh}| \leq f - |V''_{fh}| \qquad 6.2$$

fehlerhafte Knoten und mindestens

$$|V'_{ff}| = |V'| - |V'_{fh}| \geq (f+1) - |V'_{fh}| \geq (f+1) - \left(f - |V''_{fh}|\right) = 1 + |V''_{fh}|$$

fehlerfreie Knoten. Folglich sendet der letzte fehlerfreie Knoten in V', der mit $v^e_{V'_{ff}}$ bezeichnet wird, eine Data-Nachricht mit mindestens

$$signNum\left(v^e_{V'_{ff}}\right) \geq \text{signNum}_{|\{v^0_V\}}\left(v^e_{V_{ff}}\right) + \text{signNum}_{|V'_{ff}}\left(v^e_{V_{ff}}\right) \qquad 6.3$$

$$\Longrightarrow signNum\left(v^e_{V'_{ff}}\right) \geq 1 + \left(1 + |V''_{fh}|\right) = 2 + |V''_{fh}|$$

Signaturen. Damit ist der zweite Satz von Behauptung 2 bewiesen. Der Beweis des ersten Satzes von Behauptung 2 erfolgt durch Widerspruch. Wir nehmen an, es gelte

$$\exists v \in V''_{ff} : send(v) = default. \tag{6.4}$$

Sei $v^i_{V''}$ der erste fehlerfreie Knoten in V'', der eine Default-Nachricht sendet. Dann müsste laut MD2 der Knoten $v^i_{V''}$ zuvor eine Default-Nachricht mit mindestens so vielen Signaturen wie die die erhaltene Data-Nachricht erhalten haben, also mindestens $signNum\left(v^e_{V'_{ff}}\right)$ Signaturen. Da bei der Erzeugung einer Default-Nachrichten keine Signaturen einer Data-Nachricht übernommen werden können, stammen die Signaturen der zuvor erhaltenen Default-Nachricht ausschließlich von fehlerhaften Knoten – und zwar von fehlerhaften Knoten in V'', da die Knoten in V' keine Default-Nachrichten erzeugen dürfen. Es muss also gelten:

$$\left|V''_{fh}\right| \geq signNum\left(v^e_{V'_{ff}}\right) \geq 2 + \left|V''_{fh}\right| \blacksquare \textbf{\textit{ Widerspruch !}} \tag{6.5}$$

Behauptung 3. Wenn $1 < \left|V_{fh}\right| \leq f$ *und* $\left|V'_{fh}\right| = 0$ *gilt, dann ist* **LEMMA 1** *stets erfüllt.*

Beweis: Laut Behauptung 3 sendet der letzte fehlerfreie Knoten in V' eine Data-Nachricht. Gemäß SC2 beträgt die Signaturanzahl dieser Data-Nachricht

$$signNum\left(v^e_{V'_{ff}}\right) \geq signNum_{|\{v^0_V\}}\left(v^e_{V_{ff}}\right) + signNum_{|V'_{ff}}\left(v^e_{V_{ff}}\right) \tag{6.6}$$

$$\Rightarrow signNum\left(v^e_{V'_{ff}}\right) \geq 1 + \left|V'_{ff}\right|$$

$$\Rightarrow signNum\left(v^e_{V'_{ff}}\right) \geq 1 + (f+1)$$

$$\Rightarrow signNum\left(v^e_{V'_{ff}}\right) \geq f + 2$$

Die Anzahl der fehlerfreien Knoten in V''_{ff} beträgt

$$\left|V''_{ff}\right| = |V''| - \left|V''_{fh}\right| \tag{6.7}$$

$$\left|V''_{ff}\right| = |V''| - \left|V''_{fh}\right|$$

$$\left|V''_{ff}\right| = [2(f-1) + max(0, f-2)] - \left|V''_{fh}\right|$$

$$\Rightarrow \left|V''_{ff}\right| \geq [2(f-1) + max(0, f-2)] - f$$

$$\Rightarrow \left|V''_{ff}\right| \geq f - 2 + max(0, f-2).$$

Der letzte fehlerfreie Knoten $v^e_{V''_{ff}}$ wählt gemäß SC2 die empfangene Data-Nachricht aus, die die meisten Signaturen enthält, kosigniert diese Data-Nachricht und sendet sie, so dass sie mindestens die folgenden Signaturanzahl aufweist:

$$signNum\left(v^e_{V''_{ff}}\right) \geq \tag{6.8}$$

$$signNum_{|\{v^0_V\}}\left(v^e_{V''_{ff}}\right) + signNum_{|V'_{ff}}\left(v^e_{V''_{ff}}\right) + signNum_{|V''_{ff}}\left(v^e_{V''_{ff}}\right)$$

$$\Rightarrow signNum\left(v^e_{V''_{ff}}\right) \geq 1 + (f+1) + \left|V''_{ff}\right|$$

$$\Rightarrow signNum\left(v^e_{V''_{ff}}\right) \geq 1 + (f+1) + [f - 2 + max(0, f-2)]$$

$$\Rightarrow signNum\left(v^e_{V''_{ff}}\right) \geq 2f + max(0, f-2)$$

Fall (**A1**): Falls Knoten $v^e_{V''_{ff}}$ keinen fehlerhaften Vorgänger aus der Menge Θ enthält (d. h. $Vorg\left(v^e_{V''_{ff}}\right) \cap V''_{fh,Data} = \emptyset$), dann hinterlegt jeder fehlerfreie Knoten die Data-Nachricht von $v^e_{V''_{ff}}$, die gemäß Annahme **A1** von mindestens

$$signNum\left(v^e_{V''_{ff}}\right) = signNum\left(v^e_{V'_{ff}}\right) + \left|V''_{ff}\right| \tag{6.9}$$

$$\Longrightarrow signNum\left(v^e_{V''_{ff}}\right) \geq (f+2) + (f - 2 + max(0, f-2))$$

$$\Longrightarrow signNum\left(v^e_{V''_{ff}}\right) \geq 2f + max(0, f-2)$$

fehlerfreien Knoten signiert worden ist, in seinem PMB oder SMB. Laut AG2, AG3 und der Annahme in **LEMMA 1**, dass der Quellknoten fehlerfrei ist, entscheiden sich alle fehlerfreien Knoten für den Wert des Quellknotens. Damit ist **LEMMA 1** erfüllt.

Fall (**A2**): Nehmen wir an, Knoten $v^e_{V''_{ff}}$ hat mindestens einen fehlerhaften Vorgänger v^i_Θ innerhalb von V'' (d. h. $v^i_\Theta \in \Theta$).

> Hinweis: Bei v muss es sich nicht unbedingt um den unmittelbaren Vorgänger handeln. Es kann auch ein indirekter Vorgänger sein.

Fall (**A2.1**): Falls die Nachricht von v^i_Θ von keinem fehlerfreien Nachfolger von v^i_Θ empfangen wird, dann entscheiden sich analog zu **A1** alle fehlerfreien Knoten für den Wert des Quellknotens. Damit ist **LEMMA 1** erfüllt.

Fall (**A2.2**): In allen anderen Fällen sendet der letzte fehlerfreie Knoten (laut obiger Gleichung) eine Data-Nachricht die von mindestens

$$signNum\left(v^e_{V''_{ff}}\right) \geq signNum\left(v^e_{V'_{ff}}\right) + \left|V''_{ff}\right| - \left|V''_{fh}\right| + 1 \qquad 6.10$$

$$\Longrightarrow signNum\left(v^e_{V''_{ff}}\right) \geq (f+2) + (f-2+max(0,f-2)) - f + 1$$

$$\Rightarrow signNum\left(v^e_{V''_{ff}}\right) \geq f + 1 + max(0,f-2)$$

fehlerfreien Knoten signiert worden ist. In diesem Formelansatz besagt „$-\left|V''_{fh}\right| + 1$“, dass schlechtestensfall kein fehlerhafter Knoten signiert außer Knoten v^i_Θ, der gemäß Annahme **A2.2** signiert. Diese derart signierte Nachricht wird von allen fehlerfreien Knoten mindestens in ihrem PMB oder SMB gespeichert werden. Laut AG2, AG3 und der Annahme in **LEMMA 1**, dass der Quellknoten fehlerfrei ist, entscheiden sich alle fehlerfreien Knoten für den Wert des Quellknotens. Damit ist **LEMMA 1** erfüllt.

Behauptung 4. Wenn $1 < \left|V_{fh}\right| \leq f$ *mit* $\left|V'_{fh}\right| \geq 1$ *gilt, dann sendet der letzte fehlerfreie Knoten* $v^e_{V_{ff}}$ *eine Data-Nachricht mit mindestens 2f Signaturen.*

Anmerkung: Der letzte fehlerfreie Knoten gehört stets zu V'', weil für $f \geq 2$ die Gruppe V'' mindestens f Knoten enthält (von denen sind laut Annahme in Behauptung 4 höchstens $f-1$ fehlerhaft).

Anmerkung: Für $f = 1$ ist **LEMMA 1** gemäß Behauptung 1 erfüllt. Daher wird dieser Fall hier nicht betrachtet.

Beweis: Gemäß Behauptung 2 sendet der letzte fehlerfreie Knoten $v^e_{V'_{ff}}$ in der Gruppe V' eine Data-Nachricht mit mindestens

$$signNum\left(v^{e}_{V'_{ff}}\right) \geq 2 + \left|V''_{fh}\right| \qquad 6.11$$

Signaturen. Da V'' mindestens

$$\left|V''_{ff}\right| = |V''| - \left|V''_{fh}\right| \geq [2(f-1) + max(0, f-2)] - \left|V''_{fh}\right| \qquad 6.12$$

fehlerfreie Knoten enthält, sendet der letzte fehlerfreie Knoten $v^{e}_{V''_{ff}}$ gemäß SC2 stets eine Data-Nachricht (und keine Default-Nachricht, wie aus der bewiesenen Behauptung 2 hervorgeht) mit mindestens

$$signNum\left(v^{e}_{V''_{ff}}\right) \geq signNum\left(v^{e}_{V'_{ff}}\right) + \left|V''_{ff}\right| \geq \left(2 + \left|V''_{fh}\right|\right) + \left|V''_{ff}\right| \qquad 6.13$$

$$signNum\left(v^{e}_{V''_{ff}}\right) \geq signNum\left(v^{e}_{V'_{ff}}\right) + \left|V''_{ff}\right| \geq \left(2 + \left|V''_{fh}\right|\right) + \left|V''_{ff}\right|$$

$$\Rightarrow signNum\left(v^{e}_{V''_{ff}}\right) \geq 2 + \left|V''_{fh}\right| + \left[2(f-1) + max(0, f-2) - \left|V''_{fh}\right|\right]$$

$$\Rightarrow signNum\left(v^{e}_{V''_{ff}}\right) \geq 2f + max(0, f-2)$$

$$\Rightarrow signNum\left(v^{e}_{V''_{ff}}\right) \geq 2f \blacksquare$$

Signaturen.

Behauptung 5. Wenn $1 < \left|V_{fh}\right| \leq f$ *mit* $\left|V'_{fh}\right| \geq 1$ *gilt, dann ist* **LEMMA 1** *stets erfüllt.*

Beweis: Laut bewiesener Behauptung 4 sendet der letzte fehlerfreie Knoten $v^{e}_{V''_{ff}}$ eine Data-Nachricht mit mindestens $2f$ Signaturen. Laut Annahme in Behauptung 5 enthält V' mindestens einen fehlerhaften Knoten. Daher enthält V'' höchstens $f-1$ fehlerhafte Knoten und somit enthält auch Θ höchstens $f-1$ fehlerhafte Knoten.

Fall (**A1**): Falls $|\Theta| = 0$ gilt, dann senden laut bewiesener Behauptung 2, alle fehlerfreien Knoten eine Data-Nachricht. Da V_{ff} mindestens $f+1$ fehlerfreie Knoten enthält, trägt die Data-Nachricht des letzten fehlerfreien Knotens mindestens $f+1$ Signaturen und wird daher von allen fehlerfreien Knoten akzeptiert, so dass **LEMMA 1** für $|\Theta| = 0$ erfüllt ist.

Fall (**A2**): Wir nehmen an, dass Knoten $v_{V_{ff}''}^{e}$ mindestens einen fehlerhaften Vorgänger $v_{\Theta}^{i} \in \Theta$ hat (wobei v_{Θ}^{i} ein indirekter Vorgänger sein kann). Es gilt also:

$$0 < |\Theta| \leq f - 1 \ \wedge \ Vorg\left(v_{V_{ff}''}^{e}\right) \cap V_{fh,Data}'' \geq 1. \tag{6.14}$$

Laut bewiesener Behauptung 4 sendet der letzte fehlerfreie Knoten $v_{V_{ff}''}^{e}$ eine Data-Nachricht mit der folgenden Mindest-Signaturanzahl:

$$signNum\left(v_{V_{ff}''}^{e}\right) \geq 2f \tag{6.15}$$

Fall (**A2.1**): Wenn die Nachricht von v_{Θ}^{i} von keinem fehlerfreien Nachfolger von v_{Θ}^{i} empfangen wird, dann wird die von Knoten $v_{V_{ff}''}^{e}$ gesendete Data-Nachricht von allen fehlerfreien Knoten mindestens in ihrem PMB oder SMB gespeichert. Laut AG2, AG3 und der Annahme in **LEMMA 1**, dass der Quellknoten ist fehlerfrei ist, entscheiden sich alle fehlerfreien Knoten für den Wert des Quellknotens. Damit ist **LEMMA 1** erfüllt.

Fall (**A2.2**): In allen anderen Fällen enthält nach Ausführung der „Clean-Function" CF jeder fehlerfreie Knoten im PMB bzw. im SMB die vom letzten fehlerfreien Knoten $v_{V_{ff}''}^{e}$ gesandte Data-Nachricht x mit der folgenden Mindest-Signaturanzahl. Diese berechnet sich aus der Anzahl der Signaturen der Nachricht, abzüglich der Anzahl von Signaturen, die durch CF ignoriert werden (Diese ignoriert höchstens so viele Signaturen, wie es fehlerhafte Knoten in V'' gibt):

$$signNum(x) \geq signNum\left(v_{V_{ff}''}^{e}\right) - signNum_{|\Theta}\left(v_{V_{ff}''}^{e}\right) \tag{6.16}$$

$$\Longrightarrow signNum(x) \geq (2f) - (f - 1)$$

$$\Longrightarrow signNum(x) \geq f + 1$$

Gemäß AG1 (d.h. wenn vor Ausführung der „Clean-Function" CF $\text{sigCount}(PMB) \geq f + 1$ gilt) und AG2 (d.h. wenn nach Ausführung der „Clean-Function" CF $\text{sigCount}(PMB) \geq f$ gilt), entscheiden sich somit alle fehlerfreien Knoten für die Nachricht x. Da der Quellknoten fehlerfrei ist und x signiert hat, ist x

die Nachricht finalM(v_V^0), für die sich der Quellknoten entschieden hat, womit die in **LEMMA 1** formulierte Bedingung IC2 erfüllt ist.

Behauptung 6. Lemma 1 ist stets korrekt.

Beweis: Behauptung 1 beweist Lemma 1 für den Fall, dass kein Fehler auftritt. Behauptung 3 beweist Lemma 1 für den Fall, dass Fehler nur in V'' auftreten. Behauptung 5 beweist Lemma 1 für den Fall, dass Fehler auch in V' auftreten. Damit ist Lemma 1 für jeden möglichen Fall bewiesen.

LEMMA 2: *Falls der Quellknoten fehlerhaft ist (kurz: $v_V^0 \in V_{fh}$) und alle Knoten aus V' fehlerfrei sind (kurz: $V' \cap V_{fh} = \emptyset$), dann erfüllen alle fehlerfreien Knoten die Bedingung IC1. Formal:*

$$v_V^0 \in V_{fh} \quad \wedge \quad V' \cap V_{fh} = \emptyset \quad \Rightarrow \quad \forall u, v \in V_{ff}: finalM(u) = finalM(v) \qquad 6.17$$

*Behauptung 1. Wenn kein Knoten in V' eine Data-Nachricht überträgt, dann ist **LEMMA 2** erfüllt.*

Beweis: Trivial. Da kein fehlerfreier Knoten aus V' eine Data-Nachricht sendet, wird die Bedingung NC2 niemals erfüllt. Folglich akzeptiert kein Knoten aus V'' eine Data-Nachricht. Damit ist auch AG1 niemals erfüllt und alle fehlerfreien Knoten entscheiden sich für die globale Konstante. Daraus folgt die Behauptung 1 und ***LEMMA 2*** ist erfüllt.

*Behauptung 2. Wenn alle fehlerfreien Knoten in V'' eine Data-Nachricht übertragen, dann ist **LEMMA 2** erfüllt.*

Beweis: Unter der Annahme, dass der Quellknoten fehlerhaft ist, gilt:

$$\left|V_{fh}''\right| \leq f - 1 \qquad 6.18$$

Sei $v_{V''}^i$ der erste fehlerfreie Knoten in V_{ff}'', der eine Data-Nachricht x sendet, die zuvor von mindestens zwei fehlerhaften Knoten signiert worden ist. (Zuvor können keine fehlerfreien Knoten in V_{ff}'' eine Default-Nachricht gesendet haben, weil Behauptung 2 davon ausgeht, dass alle fehlerfreien Knoten in V_{ff}'' eine Data-Nachricht senden. Es

können aber zuvor andere fehlerfreie Knoten in V''_{ff} eine Data-Nachricht gesendet haben, die von keinem fehlerhaften Knoten außer dem fehlerhaften Quellknoten signiert worden ist). Diese Nachricht x ist von höchstens f fehlerhaften Knoten signiert worden, darunter der fehlerhafte Quellknoten und maximal $f-1$ weitere fehlerhafte Knoten, die alle in V''_{fh} liegen, weil alle Knoten in V' hier als fehlerfrei angenommen wurden. Alle weiteren Signaturen in x stammen von fehlerfreien Vorgängern von $v^i_{V''}$, und zwar von mindestens einem fehlerfreien Knoten in V'_{ff} und von fehlerfreien Vorgängern in V''_{ff}.

Fall (**A1**): Für die Anzahl der fehlerfreien Vorgänger gelte (siehe ggf. Abbildung 19)

$$\left|Vorg\left(v^i_{V''}\right) \cap V''_{ff}\right| > f-2 \tag{6.19}$$

Dann wird gezeigt, dass alle fehlerfreien Knoten die Bedingungen IC1 erfüllen (und damit ***LEMMA 2*** erfüllt ist, d.h. $\Rightarrow \forall u,v \in V_{ff}\colon\ finalM(u) = finalM(v)$).

Der Beweis erfolgt durch Widerspruch. Wir nehmen an, es gelte

$$\exists u,v \in V_{ff}\colon\ finalM(u) \neq finalM(v) \tag{6.20}$$

Sei $v^{i-1}_{V''}$ der letzte fehlerfreie Vorgänger von $v^i_{V''}$. Ein solcher Knoten existiert, weil $\left|Vorg\left(v^i_{V''}\right) \cap V''_{ff}\right| > f-2$ und $f \geq 2$. Dann sendet $v^{i-1}_{V''}$ laut SC2 eine Data-Nachricht y mit mindestens

$$signNum\left(v^{i-1}_{V''}\right) \geq \tag{6.21}$$

$$signNum_{|\{v^0_V\}}\left(v^{i-1}_{V''}\right) + signNum_{|V'_{ff}}\left(v^{i-1}_{V''}\right) + \left|Vorg\left(v^i_{V''}\right) \cap V_{ff}\right|$$

$$signNum\left(v^{i-1}_{V''}\right) \geq 1 + signNum_{|V'_{ff}}\left(v^{i-1}_{V''}\right) + (f-1)$$

$$signNum\left(v^{i-1}_{V''}\right) \geq f+1$$

Signaturen. Diese Nachricht y würde von alle fehlerfreien Knoten mindestens in ihrem PMB oder SMB hinterlegt werden. Gemäß AG2 und AG3 würden sich alle fehlerfreien Knoten für denselben Endwert entscheiden (**Widerspruch!**).

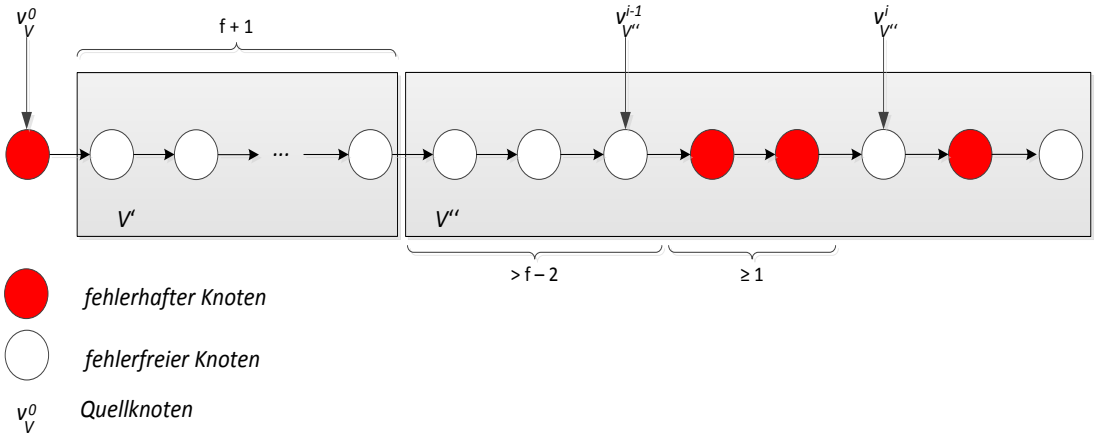


Abbildung 19: Der Kommunikationsgraph G=(V,E) für f = 4.

Fall (**A2**): Für die Anzahl der fehlerfreien Vorgänger gelte:

$$\left|Vorg\left(v_{V''}^{i}\right) \cap V_{ff}''\right| \leq f-2 \qquad 6.22$$

Dann wird gezeigt, dass alle fehlerfreien Knoten die Bedingungen IC1 erfüllen (und damit Lemma erfüllt ist, d.h. $\Rightarrow \forall u,v \in V_{ff}\colon\ finalM(u) = finalM(v)$). Der Beweis erfolgt durch Widerspruch. Wie nehmen an, es gelte

$$\exists u,v \in V_{ff}\colon\ finalM(u) \neq finalM(v)$$

Wenn Knoten $v_{V''}^{i}$ höchstens

$$\left|\mathrm{Vorg}\left(v_{V''}^{i}\right) \cap V_{ff}''\right| \leq f-2$$

fehlerfreie Vorgänger enthält, dann enthält $v_{V''}^{i}$ mindestens

$$\left|Nachf\left(v_{V''}^{i}\right) \cap V_{ff}\right| \geq |V''| - 1 - \left|Vorg\left(v_{V''}^{i}\right) \cap V_{ff}''\right| - \left|V_{fh}''\right| \qquad 6.23$$

fehlerfreie Nachfolger, berechnet durch die Gesamtanzahl der Knoten in V'', wovon „1“ für den Knoten selbst, die Anzahl der fehlerfreien Vorgänger und die Anzahl der fehlerhaften Knoten subtrahiert werden. Daraus folgt:

$$\left|Nachf\left(v_{V''}^{i}\right) \cap V_{ff}\right| \geq \qquad 6.24$$

$$2(f-1) + max(0, f-2) - 1 - \left|Vorg\left(v_{V''}^{i}\right) \cap V_{ff}''\right| - \left|V_{fh}''\right|$$

$$\Rightarrow \left|Nachf\left(v_{V''}^{i}\right)\right| \geq 2f - 2 + max(0, f-2) - 1 - (f-2) - (f-1)$$

$$\Rightarrow \left|Nachf\left(v_{V''}^{i}\right)\right| \geq max(0, f-2)$$

Folglich enthält die vom letzten fehlerfreien Knoten $v_{V_{ff}''}^{e}$ übertragene Data-Nachricht mindestens die folgende Anzahl von Signaturen:

$$signNum\left(v_{V_{ff}''}^{e}\right) \geq \tag{6.25}$$

$$signNum\left(v_{V_{ff}'}^{e}\right) + signNum_{|\{v_{V''}^{i}\}}\left(v_{V_{ff}''}^{e}\right) + signNum_{|V_{fh}''}\left(v_{V_{ff}''}^{e}\right) + \left|Nachf\left(v_{V''}^{i}\right)\right|$$

$$\Rightarrow signNum\left(v_{V_{ff}''}^{e}\right) \geq$$

$$signNum_{|\{v_{V}^{0}\}}\left(v_{V_{ff}'}^{e}\right) + signNum_{|V'}\left(v_{V_{ff}'}^{e}\right) + 1 + signNum_{|V_{fh}''}\left(v_{V_{ff}''}^{e}\right) + \left|Nachf\left(v_{V''}^{i}\right)\right|$$

$$\Rightarrow signNum\left(v_{V_{ff}''}^{e}\right) \geq$$

$$1 + 1 + 1 + signNum_{|V_{fh}''}\left(v_{V_{ff}''}^{e}\right) + max\ (0, f-2)$$

$$\Rightarrow signNum\left(v_{V_{ff}''}^{e}\right) \geq$$

$$3 + signNum_{|V_{fh}''}\left(v_{V_{ff}''}^{e}\right) + max\ (0, f-2)$$

Signaturen. Nach Ausführung der „Cleaning Function“ CF enthält jeder fehlerfreie Knoten in PMB eine Data-Nachricht mit mindestens

$$sigCount(PMB) \geq \tag{6.26}$$

$$signNum\left(v_{V_{ff}''}^{e}\right) - signNum_{|V_{fh}''}\left(v_{V_{ff}''}^{e}\right)$$

$$\Rightarrow sigCount(PMB) \geq$$

$$\left[3 + signNum_{|V''_{fh}}\left(v^e_{V''_{ff}}\right) + max\ (0, f-2)\right] - signNum_{|V''_{fh}}\left(v^e_{V''_{ff}}\right)$$

$$\Rightarrow sigCount(PMB) \geq 3 + max\ (0, f-2)$$

$$\Rightarrow sigCount(PMB) \geq f$$

Signaturen. Gemäß AG2 entscheiden sich somit alle fehlerfreien Knoten für denselben Endwert. Aus Fall **A1** und Fall **A2** folgt die Behauptung 2.

Behauptung 3. Wenn der letzte fehlerfreie Knoten $v^e_{V''_{ff}}$ eine Data-Nachricht sendet, dann gilt für diese Nachricht:

$$signNum_{|\{v^0_V\}\cup V_{ff}}\left(v^e_{V''_{ff}}\right) \geq f\ \wedge\ signNum\left(v^e_{V''_{ff}}\right) \geq f+1$$

Beweis:

Fall (**A1**): Der Quellknoten sendet keine Data-Nachricht.
Dann werden gemäß NC1 alle fehlerfreien Knoten in V'' eine Default-Nachricht senden. Folglich sendet der letzte fehlerfreie Knoten keine Data-Nachricht. Damit ist die Voraussetzung von Behauptung 3 nicht erfüllt und Behauptung 3 eine wahre Aussage.

Fall **(A2)**: Der Quellknoten sendet eine Data-Nachricht.
Definition: Die Anzahl der fehlerhaften Knoten, die eine Data-Nachricht senden, wird mit f^* bezeichnet, d.h. $f^* = |V_{fh,data}|$.
Man beachte, dass gemäß den Annahmen der Quellknoten fehlerhaft ist und bis zu $f-1$ Knoten in V'' fehlerhaft sein können, aber alle Knoten in V' fehlerfrei sind.
Wenn f^* fehlerhafte Knoten eine Data-Nachricht senden und der letzte fehlerfreie Knoten $v^e_{V''_{ff}}$ eine Data-Nachricht sendet, dann gilt:

$$signNum_{|\{v^0_V\}\cup V_{ff}}\left(v^e_{V''_{ff}}\right) \geq f \quad \wedge \quad signNum\left(v^e_{V''_{ff}}\right) \geq f+1 \qquad 6.27$$

Der Beweis dieser Aussage gliedert sich in die folgenden Fälle:
Fall (**A2.1**): Sei $f^* = |V_{fh,data}| = 0$. Daraus folgt, dass insbesondere der Quellknoten keine Data-Nachricht sendet. Analog zu Fall **(A1)** ist die Voraussetzung von Behauptung 3 nicht erfüllt und damit Behauptung 3 für $f^* = 0$ trivialerweise eine wahre Aussage.

Fall (**A2.2**): Sei $0 < f^* = \left|V_{fh,data}\right| \leq f$.

Der Quellknoten ist einer der fehlerhaften Data-Nachricht-Sender. Folglich gilt $\left|V''_{fh,data}\right| = f^* - 1 \leq f - 1$.

Wir führen einen Widerspruchsbeweis durch und nehmen dazu an, dass Folgendes gelte:

$$signNum_{|\{v_V^0\} \cup V_{ff}}\left(v^e_{V''_{ff}}\right) < f \quad \vee \quad signNum\left(v^e_{V''_{ff}}\right) < f + 1 \tag{6.28}$$

Laut der Annahme in Behauptung 3 ist der letzte fehlerfreie Knoten in V'' ein Element von P, d.h. ein Element der Menge der fehlerfreien Knoten in V'', die eine Data-Nachricht senden, kurz: $v^e_{V''_{ff}} \in P$. Da die Knoten der Menge S denen der Menge P stets nachfolgen, ist in diesem Fall die Menge S leer, kurz: $S = \emptyset$.

Fall (**A2.2.1**): Es gelte $|R| = 0$. Dies bedeutet, dass es in V'' keinen fehlerfreien Default-Nachricht-Sender gibt, dem noch mindestens ein fehlerfreier Data-Nachricht-Sender nachfolgt.

Dann senden wegen $S = \emptyset$ mindestens

$$|V_{data}| \geq 2 + \left|V''_{ff}\right| + \left|V''_{fh,data}\right| \quad \Leftrightarrow \quad |V_{data}| \geq 2 + \left|V''_{ff}\right| + f^* \tag{6.29}$$

Knoten eine Data-Nachricht (berechnet durch die Gesamtanzahl der fehlerfreien Knoten in V'', zuzüglich der f^* fehlerhaften Knoten in V'', die eine Data-Nachricht senden. Wegen NC1 und NC2 kommt noch der Quellknoten und mindestens einen Knoten aus V' hinzu).

Im Folgenden betrachten wir einen fehlerhaften Knoten v^i_Θ aus V'', der eine Data-Nachricht sendet. Es gibt mindestens einen solchen Knoten, wenn $f^* > 0$. Der Fall $f^* = 0$ wurde bereits durch Fall A2.1 behandelt. Es ist nicht notwendig festzulegen, welchen Knoten aus Θ der Knoten v^i_Θ repräsentiert, da fehlerhafte Knoten ohnehin miteinander kooperieren können.

Fall (**A2.2.1.1**): Die Nachricht von v^i_Θ wird von keinem fehlerfreien Knoten in V''_{ff} empfangen.

Dann sendet der letzte fehlerfreie Knoten eine Data-Nachricht mit der folgenden Anzahl an Signaturen:

$$signNum\left(v^e_{V''_{ff}}\right) \geq 2 + \left|V''_{ff}\right| \tag{6.30}$$

Diese Formel besagt, dass die fehlerfreien Knoten in V''_{ff} und der Quellknoten und noch mindestens ein Knoten aus V' die Data-Nachricht signiert haben. Daraus folgt:

$$signNum\left(v^{e}_{V''_{ff}}\right) \geq 2 + 2(f-1) + max(0, f-2) - (f-1) \qquad 6.31$$

$$\Rightarrow \text{signNum}\left(v^{e}_{V''_{ff}}\right) \geq f + 1$$

Dabei handelt es sich ausschließlich um die Signaturen von fehlerfreien Knoten mit Ausnahme der Signatur des Quellknotens, der fehlerhaft ist. Folglich gilt:

$$signNum_{|\{v^{0}_{V}\}\cup V_{ff}}\left(v^{e}_{V''_{ff}}\right) \geq f + 1 \qquad 6.32$$

Insgesamt gilt also:

$$signNum_{|\{v^{0}_{V}\}\cup V_{ff}}\left(v^{e}_{V''_{ff}}\right) \geq f + 1 \;\wedge\; signNum\left(v^{e}_{V''_{ff}}\right) \geq f + 1 \blacksquare \text{ (}\mathbf{Widerspruch!}\text{)}$$

Fall (**A2.2.1.2**): Die Nachricht von v^{i}_{Θ} wird von mindestens einem fehlerfreien Knoten in V''_{ff} empfangen.

Dann enthält die vom letzten fehlerfreien Knoten $v^{e}_{V''_{ff}}$ gesendete Data-Nachricht mindestens eine und höchstens f^* Signaturen von fehlerhaften Knoten aus $V''_{fh,data}$. Folglich enthält diese Data-Nachricht mindestens die folgende Anzahl von Signaturen:

$$signNum\left(v^{e}_{V''_{ff}}\right) \geq 2 + f^* + max\left(1, \left|V''_{ff}\right| - f^*\right) \qquad 6.33$$

Diese Formel ist wie folgt zu begründen: Die Nachricht ist von dem Quellknoten und mindestens einem Knoten aus V' signiert worden (in der Summe mindestens 2 Signaturen). Hinzu kommen Signaturen f^* fehlerhaften Knoten aus V''. Die übrigen $f - f^*$ fehlerhaften Knoten senden gemäß Definition von f^* keine Data-Nachricht und tragen folglich keine Signatur zur Data-Nachricht von $v^{e}_{V''_{ff}}$ bei. Von den $\left|V''_{ff}\right|$ fehlerfreien Knoten signieren nicht notwendigerweise alle die Data-Nachricht von $v^{e}_{V''_{ff}}$, weil die fehlerhaften Knoten bis zu f^* fehlerfreie Knoten (die in dem Kommunikationsgraphen unmittelbar aufeinander folgen) überbrücken können (d.h. ihnen keine Nachricht zusenden). In diesem Überbrückungsfall wird die Data-Nachricht von $v^{e}_{V''_{ff}}$ nur von $\left|V''_{ff}\right| - f^*$

fehlerfreien Knoten aus V'' signiert, insgesamt von mindestens $f^* + |V''_{ff}| - f^*$ Knoten aus V''. Wenn aber die f^* fehlerhaften Knoten mehr als f^* fehlerfreie Knoten überbrücken, dann weist ihre Nachricht am Zielknoten der Überbrückung weniger Signaturen auf als die (dort ebenfalls eintreffende) Nachricht der überbrückten fehlerfreien Knoten. Wegen SC2 verbleibt daher die Nachricht der f^* fehlerhaften Knoten nicht im PMB des fehlerfreien Knotens am Ende der Überbrückungsstrecke. Die Data-Nachricht von $v^e_{V''_{ff}}$ trägt dann mindestens $|V''_{ff}|$ Signaturen, was ebenfalls als $f^* + |V''_{ff}| - f^*$ geschrieben werden kann. Des Weiteren ist in jedem Fall $f^* + 1$ eine untere Schranke für die Anzahl der Signaturen aus V'', weil mindestens f^* fehlerhafte oder f^* überbrückte fehlerfreie Knoten und zusätzlich der letzte fehlerfreie Knoten $v^e_{V''_{ff}}$ die Data-Nachricht signiert haben. Insgesamt gilt also:

$$signNum\left(v^e_{V''_{ff}}\right) \geq 2 + f^* + max\left(1, |V''_{ff}| - f^*\right) \tag{6.34}$$

$$\Longrightarrow signNum\left(v^e_{V''_{ff}}\right) \geq 2 + max\left(1, |V''_{ff}| - f^*\right) + f^*$$

$$\Longrightarrow signNum\left(v^e_{V''_{ff}}\right) \geq$$

$$2 + max(1, [2(f-1) + max(0, f-2) - (f-1)] - f^*) + f^*$$

$$\Longrightarrow signNum\left(v^e_{V''_{ff}}\right) \geq 2 + max(1, f - 1 + max(0, f-2) - f^*) + f^*$$

$$\Longrightarrow signNum\left(v^e_{V''_{ff}}\right) \geq 2 + max(1 + f^*, f - 1 + max(0, f-2))$$

$$\Longrightarrow signNum\left(v^e_{V''_{ff}}\right) \geq 2 + max(2, f - 1 + max(0, f-2))$$

$$\Longrightarrow signNum\left(v^e_{V''_{ff}}\right) \geq 2 + max(2, f-1)$$

$$\Longrightarrow signNum\left(v^e_{V''_{ff}}\right) \geq f + 1$$

Aus der zuvor entwickelten Formel 6.33 erhält man:

$$signNum\left(v_{V''_{ff}}^{e}\right) \geq 2 + f^{*} + max(1, [2(f-1) + max(0, f-2) - (f-1)] - f^{*}) \quad 6.35$$

$$\Rightarrow signNum\left(v_{V''_{ff}}^{e}\right) \geq 2 + f^{*} + max(1, [(f-1) + max(0, f-2)] - f^{*})$$

$$\Rightarrow signNum_{|\{v_V^0\} \cup V_{ff}}\left(v_{V''_{ff}}^{e}\right) \geq$$

$$2 + f^{*} + max(1, [(f-1) + max(0, f-2)] - f^{*}) - f^{*}$$

$$\Longrightarrow signNum_{|\{v_V^0\} \cup V_{ff}}\left(v_{V''_{ff}}^{e}\right) \geq 2 + max\big(1, f - 1 + max(0, f-2) - (f-1)\big)$$

$$\Longrightarrow signNum_{|\{v_V^0\} \cup V_{ff}}\left(v_{V''_{ff}}^{e}\right) \geq 2 + max(1, max(0, f-2))$$

$$\Rightarrow signNum_{|\{v_V^0\} \cup V_{ff}}\left(v_{V''_{ff}}^{e}\right) \geq f$$

Die Zusammenfassung der beiden zuvor entwickelten Formeln ergibt:

$\Rightarrow signNum_{|\{v_V^0\} \cup V_{ff}}\left(v_{V''_{ff}}^{e}\right) \geq f \quad \wedge \quad signNum\left(v_{V''_{ff}}^{e}\right) \geq f + 1$ ■ (**Widerspruch!)**

Fall (**A2.2.2**): Es gelte $|R| \geq 1$. In diesem Fall gibt es mindestens einen fehlerfreien Default-Nachricht-Sender in V'', dem noch ein fehlerfreier Data-Nachricht-Sender nachfolgt.

Hinweis: Der Fall $|R| = 0$ ist bereits durch die bewiesene Behauptung 2 abgedeckt und muss hier nicht erneut aufgegriffen werden.

Laut MD2 sendet der letzte fehlerfreie Knoten v_R^e in der Knotenmenge R eine Default-Nachricht mit mindestens der folgenden Signaturanzahl:

$$signNum\ (v_R^e) \geq signNum\ _{|Q}(v_R^e) + |R| \quad 6.36$$

$$\Longrightarrow signNum\ (v_R^e) \geq |\{v_V^0\}| + \left|V'_{ff,data}\right| + |R|$$

$$\Longrightarrow signNum\ (v_R^e) \geq 2 + |R|$$

Diese Formel ist wie folgt zu begründen. Der erste fehlerfreie Knoten aus R sendet nur dann eine Default-Nachricht, wenn er eine Default-Nachricht von einem fehlerhaften Knoten aus Q erhalten hat, die mindestens so viele Signaturen enthält wie die zuvor

empfangene Data-Nachricht, also mindestens $\text{signNum}\left(v_{V'_{ff}}^{e}\right) \geq 2$ Signaturen. Bei der Weiterleitung der Default-Nachricht innerhalb von R kommen noch mindestens $|R|$ Signaturen hinzu.

Sei v_P^i der erste fehlerfreie Nachfolger von v_R^e, der eine Data-Nachricht von einem fehlerhaften Knoten v_Θ^i empfängt (d. h. Knoten v_Θ^i ist der letzte Vorgänger von v_P^i, d.h. der Vorgänger in Θ, der am nächsten bei v_P^i liegt).

> Hinweis: Wenn v_P^i keinen Vorgänger in Θ besitzt, dann würde der Knoten v_P^i sowie alle fehlerfreien Nachfolger von v_P^i laut SC3 eine Default-Nachricht übertragen. Damit wäre die Voraussetzung von Behauptung 3 nicht erfüllt (Behauptung, dass v_P^e eine Data-Nachricht sendet) und Behauptung 3 wäre trivialerweise eine wahre Aussage.

Laut Behauptung 3 und MD1 enthält diese Data-Nachricht, die Knoten v_P^i empfängt, mindestens die folgende Anzahl an Signaturen:

$$signNum\left(v_\Theta^i\right) \geq signNum\left(v_R^e\right) + 1 \geq 3 + |R| \qquad 6.37$$

Diese Formel ist wie folgt zu begründen. Der erste fehlerfreie Nachfolger von v_R^e sendet nur dann eine Data-Nachricht, wenn diese mehr Signaturen (echt größer) als die empfangene Default-Nachricht aufweist. Daher wird in obiger Formel 1 addiert.

Weiterhin gilt folgendes: Die vom Knoten v_Θ^i an Knoten v_P^i gesendete Data-Nachricht ist von mindestens 1 + $|R|$ fehlerhaften Knoten aus V'' signiert worden (Diese fehlerhaften Knoten liegen alle in Θ). Dies lässt sich wie folgt begründen (siehe Fälle A2.2.2.1 bis A2.2.2.3):

Fall (**A2.2.2.1**): Der Knoten v_R^e enthält keinen Vorgänger aus P.

In diesem Fall enthält die vom Knoten v_Θ^i gesendete Data-Nachricht keine Signatur aus der Menge P. Somit gilt:

$$signNum\left(v_\Theta^i\right) \geq signNum\left(v_R^e\right) + 1 \qquad 6.38$$

$$\Rightarrow signNum\left(v_\Theta^i\right) \geq |\{v_V^0\}| + \left|V'_{ff,data}\right| + |R| + 1$$

$$\Rightarrow signNum_{|\Theta}\left(v_\Theta^i\right) \geq |\{v_V^0\}| + \left|V'_{ff,data}\right| + |R| + 1 - \left(|\{v_V^0\}| + \left|V'_{ff,data}\right|\right)$$

$$\Rightarrow signNum_{|\Theta}\left(v_\Theta^i\right) \geq |R| + 1 \blacksquare$$

Fall (**A2.2.2.2**): Der Knoten v_R^e enthält genau einen Vorgänger aus P.
Fall (**A2.2.2.2.1**): Der Knoten v_P^a enthält keinen fehlerfreien Vorgänger aus R. Laut SC2 sendet Knoten v_P^a eine Data-Nachricht mit mindestens der folgenden Anzahl an Signaturen:

$$signNum\ (v_P^a) \geq \tag{6.39}$$

$$signNum_{\ |\{v_V^0\} \cup V'_{ff}}(v_P^a) + signNum_{\ |\{v_P^a\}}(v_P^a) + signNum_{\ |\Theta}(v_P^a)$$

$$\Rightarrow signNum\ (v_P^a) \geq |\{v_V^0\}| + \left|V'_{ff,data}\right| + 1$$

Laut MD2 empfängt der erste fehlerfreie Knoten in R eine Default-Nachricht mit mindestens so vielen Signaturen, wie diese Formel angibt. Der letzte fehlerfreie Knoten v_R^e aus R sendet laut SC3 eine Default-Nachricht mit mindestens der folgenden Anzahl an Signaturen:

$$signNum\ (v_R^e) \geq signNum\ (v_P^a) + |R| \tag{6.40}$$

$$\Rightarrow signNum\ (v_R^e) \geq |\{v_V^0\}| + \left|V'_{ff,data}\right| + 1 + |R|$$

Laut MD1 enthält die von Knoten v_Θ^i an Knoten v_P^i gesendete Data-Nachricht mindestens die folgende Anzahl an Signaturen:

$$signNum\ \left(v_\Theta^i\right) \geq signNum\ (v_R^e) + 1 \tag{6.41}$$

$$\Rightarrow signNum\ \left(v_\Theta^i\right) \geq signNum\ (v_P^a) + |R| + 1$$

$$\Rightarrow signNum\ \left(v_\Theta^i\right) \geq |\{v_V^0\}| + \left|V'_{ff,data}\right| + 1 + |R| + 1$$

$$\Rightarrow signNum\ \left(v_\Theta^i\right) \geq |\{v_V^0\}| + \left|V'_{ff,data}\right| + |R| + 2$$

Von diesen stammt insgesamt eine Signatur aus der Menge P, nämlich von Knoten v_P^a, eine Signatur vom Quellknoten sowie $\left|V'_{ff,data}\right|$ Signaturen aus V'_{ff}. Für die Anzahl der fehlerhaften Knoten aus Θ, die die Data-Nachricht an v_P^i signiert haben, gilt folglich:

$$signNum_{|\Theta}(v_\Theta^i) \geq \tag{6.42}$$
$$\left(|\{v_V^0\}| + \left|V'_{ff,data}\right| + |R| + 2\right) - \left(1 + |\{v_V^0\}| + \left|V'_{ff,data}\right|\right)$$

$$\Rightarrow signNum_{|\Theta}(v_\Theta^i) \geq |R| + 1 \blacksquare$$

Fall (**A2.2.2.2.2**): Der Knoten v_P^a besitzt mindestens einen fehlerfreien Vorgänger v_R^i aus R. Gemäß der Annahme für den Fall 2.2.2.2 besitzt er auch mindestens einen Nachfolger v_R^j aus R. Laut MD2 sendet Knoten v_R^i eine Default-Nachricht mit mindestens der folgenden Anzahl an Signaturen:

$$signNum\left(v_R^i\right) \geq signNum_{|Q}(v_R^i) + signNum_{|R}\left(v_R^i\right) \tag{6.43}$$

$$\Rightarrow signNum\left(v_R^i\right) \geq |\{v_V^0\}| + \left|V'_{ff,data}\right| + signNum_{|R}\left(v_R^i\right)$$

Gemäß MD1 sendet der Knoten v_P^a eine Data-Nachricht genau dann, wenn er zuvor eine Data-Nachricht mit mehr Signaturen (echt größer) als die empfangene Default-Nachricht von v_R^i empfangen hat (d. h. v_P^a empfängt eine Data-Nachricht mit mindestens signNum $\left(v_R^i\right) + 1$ Signaturen). Laut SC2 sendet v_P^a dann eine Data-Nachricht mit einer Signatur mehr, d.h. mit mindestens der folgenden Anzahl an Signaturen:

$$signNum\left(v_P^a\right) \geq signNum\left(v_R^i\right) + 2 \tag{6.44}$$

$$\Rightarrow signNum\left(v_P^a\right) \geq \left(|\{v_V^0\}| + \left|V'_{ff,data}\right| + signNum_{|R}\left(v_R^i\right)\right) + 2$$

Laut SC3 sendet der letzte Knoten v_R^e aus R eine Default-Nachricht mit mindestens der folgenden Anzahl an Signaturen:

$$signNum\left(v_R^e\right) \geq signNum\left(v_P^a\right) + \left(|R| - signNum_{|R}\left(v_R^i\right)\right) \tag{6.45}$$

Diese Formel lässt sich wie folgt begründen: Der Knoten v_P^a besitzt laut Fall A2.2.2.2.2 mindestens signNum $_{|R}\left(v_R^i\right) \geq 1$ fehlerfreie Vorgänger aus R. Folglich gilt für die Anzahl der fehlerfreien Nachfolger von v_P^a, die die Default-Nachricht signieren, Folgendes:

$$Nachfolg(v_P^a) \cap V''_{ff,default} = |R| - signNum_{|R}\left(v_R^i\right) \tag{6.46}$$

Damit die Nachfolger von v_P^a in R eine Default-Nachricht senden, muss der erste fehlerfreie Nachfolger von v_P^a gemäß MD2 eine Default-Nachricht mit mindestens signNum (v_P^a) Signaturen empfangen haben. Daraus folgt:

$$signNum\ (v_R^e) \geq \tag{6.47}$$

$$\left(|\{v_V^0\}| + \left|V'_{ff,data}\right| + signNum\ _{|R}\left(v_R^i\right)\right) + 2 + \left(|R| - signNum\ _{|R}\left(v_R^i\right)\right)$$

$$\Longrightarrow signNum\ (v_R^e) \geq |\{v_V^0\}| + \left|V'_{ff,data}\right| + 2 + |R|$$

Laut MD1 empfängt der erste fehlerfreie Nachfolger v_P^i von v_R^e eine Data-Nachricht mit mindestens einer Signatur mehr. Es gilt also:

$$signNum\left(v_\Theta^i\right) \geq signNum\ (v_R^e) + 1 \tag{6.48}$$

$$\Rightarrow signNum\left(v_\Theta^i\right) \geq |\{v_V^0\}| + \left|V'_{ff,data}\right| + 2 + |R| + 1$$

$$\Rightarrow signNum\left(v_\Theta^i\right) \geq |\{v_V^0\}| + \left|V'_{ff,data}\right| + 3 + |R|$$

Von diesen Signaturen stammt genau eine Signatur aus der Menge P (nämlich von einem Nachfolger von Knoten v_P^a), eine Signatur vom Quellknoten sowie $\left|V'_{ff,data}\right|$ Signaturen von V'_{ff}. Für die Anzahl der fehlerhaften Knoten in Θ, die die Data-Nachricht signNum $\left(v_\Theta^i\right)$ signiert haben, gilt folglich:

$$signNum\ _{|\Theta}\left(v_\Theta^i\right) \geq \tag{6.49}$$
$$\left(|\{v_V^0\}| + \left|V'_{ff,data}\right| + 3 + |R|\right) - \left(1 + |\{v_V^0\}| + \left|V'_{ff,data}\right|\right)$$

$$\Rightarrow signNum\ _{|\Theta}\left(v_\Theta^i\right) \geq |R| + 2 \blacksquare$$

Fall (**A2.2.2.3**): Der Knoten v_R^e besitzt mindestens zwei Vorgänger aus P.
In diesem Fall gilt für die Nachricht signNum $\left(v_\Theta^i\right)$ (die von Knoten v_P^i empfangen wird) stets, dass sie von mindestens 1 + $|R|$ fehlerhaften Knoten aus V'' signiert worden ist. Wenn n^* Knoten aus P Vorgänger von v_R^e sind, dann steigt die Anzahl der Knoten aus Θ, die die Nachricht signNum $\left(v_\Theta^i\right)$ signiert haben müssen, um n^*. Der Beweis lässt sich für jedes n^*auf analoge Weise wie in Fall A2.2.2.1 und A2.2.2.2 führen.

> Hinweis: Wenn für Knoten v_R^e die Anzahl der Vorgänger aus P wächst, dann wächst laut MD2 und SC3 in gleichem Maße die Anzahl der fehlerhaften Knoten in Q, die die Default-Nachricht v_R^e signiert. D.h. signNum (v_R^e) wächst in gleichem Maße. Wenn die Anzahl signNum (v_R^e) der Signaturen zunimmt, dann nimmt laut MD1 und SC2 die Anzahl der Knoten in Θ, die die Nachricht signNum $\left(v_\Theta^i\right)$ signieren, in gleichem Maße zu.

Ende dieser Fallunterscheidung, Fall (**A2.2.2**) wird jetzt fortgesetzt.

Es ist durch die Fälle A2.2.2.1 bis Fall A2.2.2.2 gezeigt worden, dass die von v_P^i empfangene Data-Nachricht mindestens $|R|+1$ Signaturen von fehlerhaften Knoten aus Θ enthält. Für die Anzahl der Signaturen, die Knoten $v_{V_{ff}''}^e$ sendet, gilt dann gemäß MD1 und SC2 folgendes:

$$signNum\left(v_{V_{ff}''}^e\right) \geq 2 + \left(\left|V_{ff}''\right| - |R|\right) + (|R|+1) \qquad 6.50$$

$$\Rightarrow signNum\left(v_{V_{ff}''}^e\right) \geq 3 + \left|V_{ff}''\right|$$

Diese Formel lässt sich wie folgt begründen: Die Nachricht ist von dem Quellknoten und mindestens einem Knoten aus V' signiert worden (in der Summe mindestens 2 Signaturen). Von den V_{ff}'' fehlerfreien Knoten werden $|R|$ fehlerfreie Knoten, die die Default-Nachricht signiert haben, subtrahiert. Da der erste fehlerfreie Nachfolger von R eine Data-Nachricht überträgt, muss dieser Knoten zuvor eine Data-Nachricht empfangen haben, die mehr Signaturen enthält als die zuvor empfangene Default-Nachricht. Von den empfangenen Signaturen stammen mindestens $|R|+1$ Signaturen von fehlerhaften Knoten aus V''. Insgesamt gilt also:

$$signNum\left(v_{V_{ff}''}^e\right) \geq 3 + \left|V_{ff}''\right| \qquad 6.51$$

$$\Longrightarrow signNum\left(v_{V_{ff}''}^e\right) \geq 3 + [2(f-1) + max(0, f-2) - (f-1)]$$

$$\Longrightarrow signNum\left(v_{V_{ff}''}^e\right) \geq 3 + (f-1) + max(0, f-2)$$

$$\Longrightarrow signNum\left(v_{V_{ff}''}^e\right) \geq f + 2 + max(0, f-2)$$

$$\Rightarrow signNum\left(v_{V''_{ff}}^{e}\right) \geq f + 1 \blacksquare$$

Von den mindestens $1 + \left|V''_{ff}\right|$ fehlerfreien Knoten (mindestens 1 fehlerfreier signierender Knoten in V') signieren nicht notwendigerweise alle die Data-Nachricht von $v_{V''_{ff}}^{e}$, weil die fehlerhaften Knoten bis zu f^* fehlerfreie Knoten überbrücken können (d.h. ihnen keine Nachricht zusenden). In diesem Überbrückungsfall wird die Data-Nachricht von $v_{V''_{ff}}^{e}$ nur von mindestens $1 + \left|V''_{ff}\right| - f^*$ fehlerfreien Knoten aus V'' und mindestens 1 fehlerfreien Knoten V' signiert. Nimmt man noch die Signatur des fehlerhaften Quellknotens hinzu, so erhält man

$$signNum_{|\{v_V^0\} \cup V_{ff}}\left(v_{V''_{ff}}^{e}\right) \geq 3 + \left|V''_{ff}\right| - f^* \qquad 6.52$$

$$\Rightarrow signNum_{|\{v_V^0\} \cup V_{ff}}\left(v_{V''_{ff}}^{e}\right) \geq 3 + \left|V''_{ff}\right| - (f-1)$$

$$\Rightarrow signNum_{|\{v_V^0\} \cup V_{ff}}\left(v_{V''_{ff}}^{e}\right) \geq 3 + [2(f-1) + max(0, f-2) - (f-1)] - (f-1)$$

$$\Rightarrow signNum_{|\{v_V^0\} \cup V_{ff}}\left(v_{V''_{ff}}^{e}\right) \geq 3 + [(f-1) + max(0, f-2)] - (f-1)$$

$$\Rightarrow signNum_{|\{v_V^0\} \cup V_{ff}}\left(v_{V''_{ff}}^{e}\right) \geq 3 + max(0, f-2)$$

$$\Rightarrow signNum_{|\{v_V^0\} \cup V_{ff}}\left(v_{V''_{ff}}^{e}\right) \geq f \blacksquare$$

Die Zusammenfassung der beiden zuvor entwickelten Formeln ergibt:

$$\Rightarrow signNum_{|\{v_V^0\} \cup V_{ff}}\left(v_{V''_{ff}}^{e}\right) \geq f \quad \wedge \quad signNum\left(v_{V''_{ff}}^{e}\right) \geq f + 1 \blacksquare$$

(**Widerspruch!**)

Behauptung 4. Wenn $\Theta = \emptyset$ gilt, dann ist die Bedingung IC1 erfüllt (d.h. die Endwerte der fehlerfreien Knoten stimmen überein).

Beweis:

Fall (**A1**): Der letzte fehlerfreie Knoten $v_{V''_{ff}}^{e}$ sendet eine Data-Nachricht.

Fall (**A1.1**): Es existiert ein fehlerfreier Knoten in V''_{ff}, der eine Default-Nachricht sendet.

Sei $v^i_{V''_{ff}}$ der erste fehlerfreie Knoten in V''_{ff}, der eine Default-Nachricht sendet. Dann senden gemäß Annahme in Behauptung 4 ($\Theta = \emptyset$) und gemäß MD2 alle fehlerfreien Nachfolger von $v^i_{V''_{ff}}$ eine Default-Nachricht. Folglich sendet auch der letzte fehlerfreie Knoten $v^e_{V''_{ff}}$ eine Default-Nachricht. Damit ist die Voraussetzung in Fall A1 nicht erfüllt, woraus gefolgert wird, dass der Fall (A1.1) nicht existiert (Behauptung 4 ist dann trivialerweise eine wahre Aussage).

Fall (**A1.2**): Alle fehlerfreien Knoten in V''_{ff}, senden eine Data-Nachricht.

Die Gültigkeit von IC1 wurde für diesen Fall bereits durch die bewiesene Behauptung 2 gezeigt. Folglich gilt für den Fall A2 die Behauptung 4.

Fall (**A2**): Der letzte fehlerfreie Knoten in V''_{ff} sendet eine Default-Nachricht.

Fall (**A2.1**): Die Anzahl der fehlerfreien Knoten, die die Data-Nachricht gesendet haben, ist echt kleiner als f. Anmerkung: Hier sind auch die fehlerfreien Knoten aus V' einbezogen.

In diesem Fall ist die Bedingung AG1 (die besagt, dass mindestens f + 1 Signaturen vorliegen müssen) in keinem fehlerfreien Knoten erfüllt, weil in diesem Fall nur die Signaturen von höchstens f fehlerfreien Knoten, die Signatur des Quellknotens, aber keine weiteren Signaturen von fehlerhaften Knoten vorliegen, da laut der Annahme in Behauptung 4 die Menge Θ leer ist. Folglich entscheiden sich alle fehlerfreien Knoten für die globale Konstante.

Fall (**A2.2**): Die Anzahl der fehlerfreien Knoten, die die Data-Nachricht gesendet haben, beträgt mindestens f.

Da alle Knoten in V' fehlerfrei sind, besitzt die von $v^e_{V'_{ff}}$ gesandte Data-Nachricht eine größere Signatur-Anzahl als jede andere von einem Knoten aus V' gesandte Nachricht. Da die vom fehlerfreien Knoten $v^e_{V'_{ff}}$ gesandte Nachricht an alle Knoten in V'' korrekt verteilt wird, akzeptieren diese keine andere Data-Nachricht (von den fehlerhaften Knoten kommen wegen $\Theta = \emptyset$ keine Signaturen hinzu). In diesem Fall sendet der letzte fehlerfreie Knoten $v^e_{V_{ff,data}}$ eine Data-Nachricht mit mindestens $f + 1$ Signaturen, da zur Annahme von f fehlerfreien signierenden Knoten aus Fall

A2.2 noch die Signatur des fehlerhaften Quellknotens hinzukommt. Gemäß SC2 wird diese Nachricht von allen fehlerfreien Knoten in PMB hinterlegt.

Gemäß AG2 entscheiden sich alle fehlerfreien Knoten für denselben Endwert, womit IC1 erfüllt ist.

Behauptung 5.Falls der letzte fehlerfreie Knoten eine Default-Nachricht überträgt, dann ist die Bedingung IC1 erfüllt (d.h. die Endwerte der fehlerfreien Knoten stimmen überein).

Beweis:

Fall (**A1**): Kein fehlerfreier Knoten aus V''_{ff} sendet eine Data-Nachricht, d.h. $|P| = 0$. In diesem Fall empfängt gemäß MD2 der erste fehlerfreie Knoten in V'' eine Default-Nachricht von einem fehlerhaften Knoten $v^i_{Q \cup T}$ mit mindestens der folgenden Anzahl an Signaturen:

$$signNum\left(v^i_{Q \cup T}\right) \geq signNum\left(v^e_{V'}\right) \tag{6.53}$$

Nach Ausführung der „Cleaning Function“ CF hat kein fehlerfreier Knoten in PMB oder SMB eine Data-Nachricht x mit mehr als

$$signNum(x) \leq \left(signNum\left(v^e_{V'}\right) + \left|V''_{fh}\right|\right) - signNum\left(v^i_{Q \cup T}\right) \tag{6.54}$$

$$\Rightarrow signNum(x) \leq \left(signNum\left(v^e_{V'}\right) + \left|V''_{fh}\right|\right) - signNum\left(v^e_{V'}\right)$$

$$\Rightarrow signNum(x) \leq \left|V''_{fh}\right|$$

$$\Rightarrow signNum(x) \leq f - 1$$

Signaturen gespeichert. Damit ist AG2 in keinem fehlerfreien Knoten erfüllt und alle fehlerfreien Knoten entscheiden sich für die globale Konstante, womit IC1 erfüllt ist.

Fall (**A2**): Mindestens ein fehlerfreier Knoten aus V''_{ff} sendet eine Data-Nachricht, d.h. $|P| > 0$.

In diesem Fall empfängt gemäß MD2 der erste fehlerfreie Nachfolger des letzten fehlerfreien Knotens v^e_P von P eine Default-Nachricht von einem fehlerhaften Knoten $v^i_{Q \cup T}$ mit mindestens der folgenden Anzahl an Signaturen:

$$signNum\left(v^i_{Q \cup T}\right) \geq signNum(v^e_P) \tag{6.55}$$

Nach Ausführung der „Cleaning Function“ CF enthält kein fehlerfreier Knoten in PMB oder SMB eine Data-Nachricht x mit mehr als

$$signNum(x) \leq \left(signNum(v_P^e) + \left|V_{fh}''\right|\right) - signNum\left(v_{Q \cup T}^i\right) \quad 6.56$$

$$\Rightarrow signNum(x) \leq \left(signNum(v_P^e) + \left|V_{fh}''\right|\right) - signNum(v_P^e)$$

$$\Rightarrow signNum(x) \leq \left|V_{fh}''\right|$$

$$\Rightarrow signNum(x) \leq f - 1$$

Signaturen gespeichert. Damit ist AG2 in keinem fehlerfreien Knoten erfüllt und alle fehlerfreien Knoten entscheiden sich für die globale Konstante, womit IC1 erfüllt ist.

Behauptung 6. Falls der letzte fehlerfreie Knoten eine Data-Nachricht sendet, dann ist die Bedingung IC1 erfüllt (d.h. die Endwerte der fehlerfreien Knoten stimmen überein).

Beweis:

Fall (**A1**): Es gelte $|R| = 0$. Dies bedeutet, dass kein fehlerfreier Knoten eine Default-Nachricht sendet.

Laut Fall A2.2.1 der bewiesenen Behauptung 3 sendet der letzte fehlerfreie Knoten $v_{V_{ff}''}^e$ eine Data-Nachricht, die von mindestens

$$signNum\left(v_{V_{ff}''}^e\right) \geq 2 + \left|V_{ff}''\right| \quad 6.57$$

Knoten signiert worden ist.

Damit enthält gemäß SC2 jeder fehlerfreie Knoten in PMB eine Data-Nachricht mit mindestens der folgenden Anzahl an Signaturen

$$sigCount(PMB) \geq 2 + \left|V_{ff}''\right| \quad 6.58$$

Dies lässt sich wie folgt begründen: Da $v_{V_{ff}''}^e$ fehlerfrei ist, wird die Data-Nachricht mit signNum $\left(v_{V_{ff}''}^e\right)$ Signaturen von allen fehlerfreien Knoten in V_{ff} empfangen. Diese Data-Nachricht wird von einem fehlerfreien Knoten genau dann im PMB hinterlegt, wenn diese mehr Signaturen (echt größer) als die in PMB hinterlegte Nachricht aufweist.

Wie bereits in Behauptung 3 gezeigt, gilt für die in PMB hinterlegte Nachricht:

$$signNum_{|\{v_V^0\}\cup V_{ff}}\left(v_{V_{ff}''}^e\right) \geq f \ \wedge \ signNum\left(v_{V_{ff}''}^e\right) \geq f+1 \qquad 6.59$$

Somit entscheiden sich gemäß AG2 alle fehlerfreien Knoten für dieselbe Data-Nachricht, wodurch IC1 erfüllt ist.

Fall (**A2**): Es gelte $|R| > 0$.

Laut Fall A2.2.2 der bewiesenen Behauptung 3 sendet der letzte fehlerfreie Knoten $v_{V_{ff}''}^e$ eine Data-Nachricht die von mindestens

$$signNum\ \left(v_{V_{ff}''}^e\right) \geq 3 + \left|V_{ff}''\right| \qquad 6.60$$

Knoten signiert worden ist. Damit enthält gemäß SC2 jeder fehlerfreie Knoten in PMB eine Data-Nachricht mit mindestens der folgenden Anzahl an Signaturen

$$sigCount(PMB) \geq 3 + \left|V_{ff}''\right| \qquad 6.61$$

Wir bereits in Behauptung 3 gezeigt, gilt für die in PMB hinterlegte Nachricht:

$$signNum_{|\{v_V^0\}\cup V_{ff}}\left(v_{V_{ff}''}^e\right) \geq f \ \wedge \ signNum\left(v_{V_{ff}''}^e\right) \geq f+1$$

Somit entscheiden sich gemäß AG2 alle fehlerfreien Knoten für denselben Data-Nachricht, wodurch IC1 erfüllt ist.

*Beweis von **LEMMA 2**:* Aus den bewiesenen Behauptungen Behauptung 1, Behauptung 5 und Behauptung 6, die von komplementären Annahmen ausgehen und alle möglichen Fälle abdecken, folgt jeweils, das IC1 erfüllt ist. Damit gilt **LEMMA 2**.

Theorem 1*: Wenn* $\left|V_{ff}'\right| = f+1$ *und 1)* $v_V^0 \in V_{ff}$ *oder 2)* $v_V^0 \in V_{fh}$ *gilt, dann sind die Bedingungen IC1 und IC2 erfüllt.*

Beweis: Theorem 1 folgt unmittelbar aus **LEMMA 1** und **LEMMA 2**

LEMMA 3: *Falls der Quellknoten* v_V^0 *fehlerhaft ist und zusätzlich Fehler in* V' *auftreten, d.h.* $1 \leq \left|V_{ff}'\right| < f+1$ *gilt, dann ist die Bedingung IC1 erfüllt (d.h. die Endwerte der fehlerfreien Knoten stimmen überein).*

*Behauptung 1. Falls kein fehlerfreier Knoten in V''_{ff} eine Data-Nachricht signiert, d.h. $P = \emptyset$, dann ist **LEMMA 3** erfüllt.*

Beweis:

Fall **A1**: Kein fehlerfreier Knoten in V'_{ff} sendet eine Data-Nachricht.

In diesem Fall gilt IC1, weil kein fehlerhafter Knoten in V_{fh} in der Lage ist, eine Data-Nachricht mit mehr als f Signaturen zu senden. Folglich hat kein fehlerfreien Knoten in V_{ff} im PMB oder SMB eine Data-Nachricht mit mehr als f Signaturen gespeichert. Damit ist AG1 niemals erfüllt und alle fehlerfreien Knoten entscheiden sich für die globale Konstante. Daraus folgt die Behauptung 1 für Fall A1 und **LEMMA 3** ist erfüllt.

Fall **A2**: Mindestens ein fehlerfreier Knoten aus V'_{ff} sendet eine Data-Nachricht.

Gemäß Annahme in Behauptung 1 empfängt der erste fehlerfreie Knoten in V''_{ff} eine Default-Nachricht von einem fehlerhaften Knoten $v^i_{Q\cup T}$ mit mindestens der folgenden Anzahl an Signaturen:

$$signNum\left(v^i_{Q\cup T}\right) \geq signNum\left(v^e_{V'_{ff}}\right) \tag{6.62}$$

Nach Ausführung der „Cleaning Function“ CF enthält kein fehlerfreier Knoten im PMB oder SMB eine Data-Nachricht x mit mehr als

$$\left(signNum\left(v^e_{V'_{ff}}\right) + \left|V''_{fh}\right|\right) - signNum\left(v^i_{Q\cup T}\right) \tag{6.63}$$

Signaturen. D.h. für alle fehlerfreien Knoten gilt

$$signNum(x) \leq \left(signNum\left(v^e_{V'_{ff}}\right) + \left|V''_{fh}\right|\right) - signNum\left(v^i_{Q\cup T}\right) \tag{6.64}$$

$$\Rightarrow signNum(x) \leq \left(signNum\left(v^e_{V'_{ff}}\right) + \left|V''_{fh}\right|\right) - signNum\left(v^e_{V'_{ff}}\right)$$

$$\Rightarrow signNum(x) \leq \left|V''_{fh}\right|$$

$$\Rightarrow signNum(x) \leq f - 1$$

Damit ist AG2 in keinem fehlerfreien Knoten erfüllt und alle fehlerfreien Knoten entscheiden sich für die globale Konstante, wodurch IC1 erfüllt ist.

Behauptung 2. *Die Menge V''_{ff} enthält mindestens $f + \max(0, f-2)$ fehlerfreie Knoten. Formal:*

$$\left|V''_{ff}\right| \geq f + max\ (0, f-2)$$

Beweis:

Gemäß der Voraussetzung von **LEMMA 3** gilt:

$$\left|V'_{fh} \cup \{v_V^0\}\right| \geq 2 \qquad 6.65$$

Laut der f-Fehlerannahme enthält die Knotenmenge V''_{fh} höchstens

$$\left|V''_{fh}\right| = \left|V_{fh}\right| - \left|V'_{fh} \cup \{v_V^0\}\right| \leq f-2 \qquad 6.66$$

fehlerhafte Knoten. Folglich enthält die Knotenmenge V''_{ff} mindestens

$$\left|V''_{ff}\right| \geq |V''| - \left|V'_{fh} \cup \{v_V^0\}\right| \qquad 6.67$$

$$\Rightarrow \left|V''_{ff}\right| \geq 2(f-1) + max(0, f-2) - (f-2)$$

$$\Rightarrow \left|V''_{ff}\right| \geq f + max(0, f-2)\ \blacksquare$$

fehlerfreie Knoten.

Behauptung 3. *Falls der letzte fehlerfreie Knoten eine Data-Nachricht sendet, dann gibt es keine zwei fehlerfreien Knoten u und v in V_{ff}, die in ihrem PMB unterschiedliche Data-Nachrichten gespeichert haben. Formal:*

$$\nexists u, v \in V''_{ff}: finalM(u) \neq finalM(v)\ mit\ finalM(u) \in Data, finalM(v) \in Data.$$

Beweis:

Der Beweis von Behauptung 3 erfolgt durch Widerspruch.

Wir nehmen an es gelte

$$\exists u, v \in V''_{ff}: \text{finalM}(u) \neq \text{finalM}(v) \text{ mit finalM}(u) \in Data, \text{finalM}(v) \in Data.$$

Fall **A1**: Sei $f = 2$. Dies bedeutet, dass der Quellknoten und ein Knoten aus der Menge V'_{fh} fehlerhaft sind.

> **Hinweis**: Der Fall $f = 1$ wurde bereits in **LEMMA 1** und **LEMMA 2** gezeigt.

Fall **A1.1**: Der (einzige) fehlerhafte Knoten aus V'_{fh} sendet keine Nachricht. Damit die Voraussetzung von Behauptung 3 erfüllt wird, muss mindestens ein fehlerfreier Knoten $v^i_{V'_{ff}}$ aus V'_{ff} die Data-Nachricht senden. Dann aber signieren wegen $\left|V''_{fh}\right| = 0$ alle fehlerfreien Nachfolger von $v^i_{V'_{ff}}$ die Data-Nachricht. Folglich wird die Data-Nachricht von $v^i_{V''_{ff}}$ von allen fehlerfreien Knoten in PMB hinterlegt und alle fehlerfreien Knoten entscheiden sich folglich für genau diese Nachricht. Dies stellt einen Widerspruch zu der Annahme in Fall A1 dar.

Fall **A1.2**: Der fehlerhafte Knoten aus V'_{fh} sendet eine Data-Nachricht. Laut SC2 setzt sich die Data-Nachricht durch, die von dem ersten fehlerfreien Knoten in V''_{ff} gesendet wird. Sei $v^i_{V''_{ff}}$ der erste fehlerfreie Knoten in V''_{ff}, der eine Data-Nachricht sendet. Laut SC2 sendet Knoten $v^i_{V''_{ff}}$ eine Data-Nachricht mit mindestens

$$signNum\left(v^i_{V''_{ff}}\right) \geq \tag{6.68}$$

$$signNum_{|\{v^0_V\}}\left(v^i_{V''_{ff}}\right) + signNum_{|V'}\left(v^i_{V''_{ff}}\right) + signNum_{|\left\{v^i_{V''_{ff}}\right\}}\left(v^i_{V''_{ff}}\right)$$

$$\Rightarrow signNum\left(v^i_{V''_{ff}}\right) \geq 1 + 1 + 1$$

$$\Rightarrow signNum\left(v^i_{V''_{ff}}\right) \geq f + 1$$

Signaturen (hier gilt $f = 2$). Wegen der f-Fehlerannahme signieren alle fehlerfreien Nachfolger von $v^i_{V''_{ff}}$ die Data-Nachricht signNum$\left(v^i_{V''_{ff}}\right)$. Folglich wird die Data-Nachricht von $v^i_{V''_{ff}}$ von allen fehlerfreien Knoten in PMB hinterlegt und alle fehlerfreien Knoten entscheiden sich folglich für genau diese Nachricht. Dies stellt einen Widerspruch zu der Annahme in Fall A1 dar.

Fall **A2**: Sei $f \geq 3$.

Laut SC2 sendet der letzte fehlerfreie Knoten $v_{V''_{ff}}^{e}$ eine Data-Nachricht mit mindestens der folgenden Signaturanzahl:

$$signNum\left(v_{V''_{ff}}^{e}\right) \geq \tag{6.69}$$

$$\begin{cases} signNum_{\,|\{v_V^0\}\cup V'}\left(v_{V''_{ff}}^{e}\right) + \left|V''_{ff}\right| - |R|, wenn\ signNum\left(v_R^e\right) \leq 1 \\ \left(1 + signNum\left(v_R^e\right)\right) + \left|V''_{ff}\right| - |R|, sonst \end{cases}$$

In dieser bedingten Ungleichung wird unterschieden, ob höchstens ein fehlerfreier Knoten oder mindestens zwei fehlerfreie Knoten (aus der Menge R) eine Default-Nachricht gesendet haben. Die obige Formel ist wie folgt zu begründen: Die Data-Nachricht von Knoten $v_{V''_{ff}}^{e}$ ist von dem Quellknoten und mindestens einem Knoten aus V' signiert worden (in der Summe mindestens 2 Signaturen). Für signNum $(v_R^e) > 1$ folgt daraus, dass der erste fehlerfreie Nachfolger von v_R^e nur dann eine Data-Nachricht sendet, wenn die zuvor empfangene Data-Nachricht mehr Signaturen (echt größer) als eine möglicherweise empfangene Default-Nachricht aufweist. Daher wird im Fall signNum $(v_R^e) > 1$ in der der obigen bedingten Ungleichung der Wert $\left(1 + \text{signNum}\ (v_R^e)\right)$ addiert. Weiterhin signieren laut SC2 und der Annahme in Behauptung 3 mindestens $\left|V''_{ff}\right| - |R|$ Knoten aus V'' die Data-Nachricht (einschließlich des Knotens $v_{V''_{ff}}^{e}$).

> Anmerkung: Der Wert $\left|V''_{ff}\right| - |R|$ in der obigen Formel besagt nur, dass die Anzahl der Signaturen um weitere $\left|V''_{ff}\right| - |R|$ Signaturen erhöht wird. Dies impliziert aber nicht, dass die Data-Nachricht von $v_{V''_{ff}}^{e}$ stets $\left|V''_{ff}\right| - |R|$ Signaturen von fehlerfreien Knoten enthält (einige der Signaturen können auch von fehlerhaften Knoten stammen). Wir wissen aber, dass mindestens $\left|V''_{ff}\right| - |R|$ fehlerfreie Knoten eine Data-Nachricht signieren, unabhängig davon, welche von ihnen tatsächlich die Data-Nachricht von Knoten $v_{V''_{ff}}^{e}$ signieren. Weiterhin wissen wir, dass sich gemäß SC2 bei den fehlerfreien Knoten immer die Data-Nachricht durchsetzt, welche die meisten Signaturen aufweist.

Daher werden in der obigen Ungleichung für beide Fälle (d.h. signNum $(v_R^e) > 1$ und signNum $(v_R^e) \leq 1$) zusätzlich $\left|V''_{ff}\right| - |R|$ Signaturen addiert.
Daraus folgt für den Fall signNum $(v_R^e) \leq 1$:

$$signNum\left(v_{V''_{ff}}^{e}\right) \geq signNum_{|\{v_V^0\}\cup V'}\left(v_{V''_{ff}}^{e}\right) + \left|V''_{ff}\right| - |R| \qquad 6.70$$

$$\Rightarrow signNum\left(v_{V''_{ff}}^{e}\right) \geq$$

$$signNum_{|\{v_V^0\}\cup V'}\left(v_{V''_{ff}}^{e}\right) + [2(f-1) + max(0, f-2) - (f-2) - |R|]$$

$$\Rightarrow signNum\left(v_{V''_{ff}}^{e}\right) \geq$$

$$signNum_{|\{v_V^0\}\cup V'}\left(v_{V''_{ff}}^{e}\right) + [2(f-1) + max(0, f-2) - (f-2) - 1]$$

$$\Rightarrow signNum\left(v_{V''_{ff}}^{e}\right) \geq 2 + [f - 1 + max(0, f-2)]$$

$$\Rightarrow signNum\left(v_{V''_{ff}}^{e}\right) \geq f + 1 + max(0, f-2)$$

und für den Fall signNum $(v_R^e) > 1$:

$$signNum\left(v_{V''_{ff}}^{e}\right) \geq \qquad 6.71$$

$$\left(1 + signNum\,(v_R^e)\right) + [2(f-1) + max(0, f-2) - (f-2) - |R|]$$

$$\Rightarrow signNum\left(v_{V''_{ff}}^{e}\right) \geq \left(1 + signNum\,(v_R^e)\right) + f + max(0, f-2) - |R|$$

$$\Rightarrow signNum\left(v_{V''_{ff}}^{e}\right) \geq (signNum\,(v_R^e) - |R|) + (f+1) + max(0, f-2)$$

$$\Leftrightarrow signNum\left(v_{V''_{ff}}^{e}\right) \geq \left(signNum_{|V''_{fh}}(v_R^e)\right) + (f+1) + max(0, f-2)$$

Man beachte, dass der Ausdruck „signNum $(v_R^e) - |R|$“ durch den gleichgroßen Ausdruck „signNum $_{|V''_{fh}}(v_R^e)$“ ersetzt wurde. Dabei gibt signNum $_{|V''_{fh}}(v_R^e)$ die Anzahl der fehlerhaften Knoten an, die die Default-Nachricht von Knoten v_R^e signiert haben (es gilt also signNum $_{|V''_{fh}}(v_R^e) \geq 0$).

Im nächsten Schritt zeigen wir, dass alle fehlerfreien Knoten sich für eine Data-Nachricht als Endwert entscheiden.

Laut Voraussetzung in Behauptung 3 sendet der letzte fehlerfreie Knoten $v^e_{V''_{ff}}$ eine Data-Nachricht. Gemäß der obigen Herleitung (siehe Formeln 6.70 und 6.71) enthält die von Knoten $v^e_{V''_{ff}}$ gesendete Data-Nachricht mindestens die folgende Signaturanzahl

$$signNum\left(v^e_{V''_{ff}}\right) \geq \begin{cases} f+1+max(0,f-2)\,,\ wenn\ signNum\,(v^e_R) \leq 1 \\ signNum_{\,|V''_{fh}}(v^e_R)+(f+1)+max(0,f-2)\,,\ sonst \end{cases} \quad 6.72$$

Gemäß SC2 enthält jeder fehlerfreie Knoten in PMB eine Data-Nachricht, die mindestens so viele Signaturen enthält wie die von Knoten $v^e_{V''_{ff}}$ übertragene Data-Nachricht. Nach Ausführung der „Cleaning Function“ CF enthält jeder fehlerfreier Knoten in PMB eine Data-Nachricht x mit mindestens der folgenden Anzahl an Signaturen:
Für den Fall signNum $(v^e_R) \leq 1$ folgt:

$$signNum(x) \geq (f+1)+max(0,f-2)-signNum_{\,|V''_{fh}}(x) \quad 6.73$$

$$\Rightarrow signNum(x) \geq (f+1)+max(0,f-2)-(f-2)$$

$$\Rightarrow signNum(x) \geq 3+max(0,f-2)$$

$$\Rightarrow signNum(x) \geq f+1$$

und für den Fall signNum $(v^e_R) > 1$ folgt:

$$signNum\left(v^e_{V''_{ff}}\right) \geq \quad 6.74$$

$$signNum_{\,|V''_{fh}}(v^e_R)+(f+1)+max(0,f-2)-signNum_{\,|V''_{fh}}(x)$$

$$signNum\left(v^e_{V''_{ff}}\right) \geq$$

$$signNum_{\,|V''_{fh}}(v^e_R)+(f+1)+max(0,f-2)-signNum_{\,|V''_{fh}}(x)$$

$$\Rightarrow signNum\left(v^e_{V''_{ff}}\right) \geq signNum_{\,|V''_{fh}}(v^e_R)+(f+1)+max(0,f-2)-(f-2)$$

$$\Rightarrow signNum\left(v_{V_{ff}''}^{e}\right) \geq signNum_{|V_{fh}''}(v_R^e) + 3 + max(0, f-2)$$

$$\Rightarrow signNum\left(v_{V_{ff}''}^{e}\right) \geq 3 + max(0, f-2)$$

$$\Rightarrow signNum(x) \geq f + 1$$

Gemäß AG2 entscheiden sich somit alle fehlerfreien Knoten für eine Data-Nachricht als Endwert.
Es gibt mindestens zwei fehlerfreie Knoten, die sich für zwei unterschiedliche Data-Nachrichten x und y entscheiden.

Seien Knoten u und v zwei fehlerfreie Knoten aus V_{ff}, die sich für zwei verschiedene Data-Nachrichten x und y mit x $\neq y$ entscheiden. Es gilt also:

$$\exists u, v \in V_{ff}'': finalM(u) = x, finalM(v) = y \; mit \; finalM(u) \neq finalM(v) \qquad 6.75$$

Gemäß Formel 6.72 sendet der letzte fehlerfreie Knoten eine Data Nachricht mit mindestens

$$signNum\left(v_{V_{ff}''}^{e}\right) \geq \begin{cases} f + 1 + max(0, f-2)\,, \; wenn \; signNum\,(v_R^e) \leq 1 \\ signNum_{|V_{fh}''}(v_R^e) + (f+1) + max(0, f-2)\,, \; sonst \end{cases}$$

Signaturen. O.B.d.A. entscheidet sich Knoten u für die von Knoten $v_{V_{ff}''}^{e}$ übertragene Data-Nachricht x und Knoten v für eine andere Data-Nachricht y, die von einem fehlerhaften Knoten $v_{V_{fh}}^{i}$ von $v_{V_{ff}''}^{e}$ an Knoten v übertragen wurde. Damit die Nachricht von Knoten $v_{V_{fh}}^{i}$ nicht durch die Nachricht von $v_{V_{ff}''}^{e}$ ersetzt wird, muss die von Knoten $v_{V_{fh}}^{i}$ übertragene Nachricht y mindestens die folgende Signaturanzahl aufweisen:

$$signNum\,(y) \geq signNum\left(v_{V_{ff}''}^{e}\right) \qquad 6.76$$

Dies ist nur dann erfüllt, wenn die Nachricht y von mindestens der folgenden Anzahl an fehlerfreien Knoten aus V_{ff} signiert worden ist:
Für den Fall signNum $(v_R^e) \leq 1$ gilt

$$signNum_{|V_{ff}}(y) \geq f + 1 + max(0, f-2) - |V_{fh}| \tag{6.77}$$

$$\Rightarrow signNum_{|V_{ff}}(y) \geq (f+1) + max(0, f-2) - f$$

$$\Rightarrow signNum_{|V_{ff}}(y) \geq 1 + max(0, f-2)$$

Und für den Fall signNum $(v_R^e) > 1$ gilt

$$signNum_{|V_{ff}}(y) \geq signNum_{|V''_{fh}}(v_R^e) + (f+1) + max(0, f-2) - |V_{fh}| \tag{6.78}$$

$$\Rightarrow signNum_{|V_{ff}}(y) \geq signNum_{|V''_{fh}}(v_R^e) + (f+1) + max(0, f-2) - f$$

$$\Rightarrow signNum_{|V_{ff}}(y) \geq signNum_{|V''_{fh}}(v_R^e) + 1 + max(0, f-2)$$

Insgesamt gilt also:

$$signNum_{|V_{ff}}(y) \geq \begin{cases} 1 + max(0, f-2)\,, & wenn\ signNum\ (v_R^e) \leq 1 \\ signNum_{|V''_{fh}}(v_R^e) + 1 + max(0, f-2)\,, & sonst \end{cases} \tag{6.79}$$

Fall **A2.1**: Sei signNum $(v_R^e) \leq 1$. Dies bedeutet, dass höchstens ein fehlerfreier Knoten in R eine Default-Nachricht sendet. Gemäß MD2 und der Annahme in Fall A2.1 enthält Knoten v_R^e keine Vorgänger aus der Menge P.

> Anmerkung: Im Fall A2.1 sendet der Knoten v_R^e eine Default-Nachricht mit höchstens signNum $(v_R^e) \leq 1$ Signaturen. Da der erste fehlerfreie Knoten aus P laut NC2 und SC2 stets eine Data-Nachricht mit mindestens
>
> $$\text{signNum}(v_P^a) \geq \text{signNum}_{|\{v_V^0\} \cup V'} \left(v_{V''_{ff}}^e\right) + 1 \geq 3$$
>
> Signaturen sendet, kann demnach Knoten v_R^e keine Vorgänger aus der Menge P besitzen.

Sei $v_{V_{ff}}^j$ der letzte fehlerfreie Knoten in V_{ff}, der die Data-Nachricht y signiert und sendet. Laut SC2 muss der erste fehlerfreie Nachfolger von $v_{V_{ff}}^j$ zuvor eine Data-Nachricht x mit mindestens der folgenden Anzahl an Signaturen empfangen haben:

$$signNum(x) \geq signNum_{|\{v_V^0\} \cup V'_{fh}}(x) + signNum_{|V_{ff}}(y) \tag{6.80}$$

Diese Formel lässt sich wie folgt begründen: Laut NC1 ist die Data-Nachricht x von dem Quellknoten signiert. Da $\text{signNum}\,(v_R^e) \leq 1$gilt, folgt unmittelbar, dass kein Knoten aus V'_{ff} die Data-Nachricht x signiert haben kann. Andernfalls hätte laut MD2 der Knoten v_R^e zuvor eine Default-Nachricht mit mindestens $\text{signNum}_{|\{v_V^0\}\cup V'_{fh}}(v)$ Signaturen empfangen haben müssen. Zur Begründung dieser Aussage nehmen wir an, dass ein fehlerfreier Knoten aus V'_{ff} die Data-Nachricht signiert habe. Dann würde Knoten v_R^e im PMB eine Data-Nachricht mit mindestens $\text{signNum}_{|\{v_V^0\}\cup V'_{ff}}\left(v^e_{V'_{ff}}\right)$ hinterlegt haben. Laut der Voraussetzung in Fall A2.1 wissen wir, dass Knoten v_R^e eine Default-Nachricht sendet. Wegen SC3 müsste der Knoten, um eine Default-Nachricht zu senden, eine Default-Nachricht mit mindestens so vielen Signaturen wie in PMB gespeichert empfangen haben. Dies würde zu einem Widerspruch führen, da Fall A2.1 annimmt, dass $\text{signNum}\,(v_R^e) \leq 1$ gilt.

Dann würde aber Knoten v_R^e laut SC3 eine Default-Nachricht mit mindestens 3 Signaturen senden. Dies steht im Widerspruch zur Annahme in Fall A2.1. Daher wird in der obigen Ungleichung (siehe Formel 6.80) $\text{signNum}_{|\{v_V^0\}\cup V'_{fh}}(x)$ addiert. Da laut Fall A2 (siehe Formel 6.77 und 6.78) $\text{signNum}_{|V_{ff}}(y) \geq 1 + \max(0, f-2)$ fehlerfreie Knoten die Data-Nachricht y signiert haben, muss laut SC2 der Nachfolger von $v^j_{V_{ff}}$ eine Data-Nachricht x empfangen haben, die von mindestens so vielen Knoten signiert wurde wie die zuvor empfangene Data-Nachricht y. Daher wird in der obigen Ungleichung (siehe Formel 6.80) $\text{signNum}_{|V_{ff}}(y)$ addiert. Insgesamt gilt also:

$$\Rightarrow \text{signNum}(x) \geq \text{signNum}_{|\{v_V^0\}}(x) + \text{signNum}_{|V'_{fh}}(x) + 1 + \max(0, f-2)$$

In dieser Formel ersetzt der Summand „$1 + \max(0, f-2)$“ den Ausdruck „$\text{signNum}_{|V_{ff}}(y)$“ aus der Formel zuvor. Die Ersetzung ist zulässig, weil gemäß der Formel 6.79 stets $\text{signNum}_{|V_{ff}}(y) \geq 1 + \max(0, f-2)$ gilt.

Sei $V_{data\,y}$ die Menge der fehlerfreien Knoten in V, die die Data-Nachricht y signiert haben. Da die Nachricht x von keinem der fehlerfreien Knoten in $V_{data\,y}$ empfangen bzw. signiert wurde, muss laut SC2 die Nachricht x, die der erste fehlerfreie Nachfolger von $v^j_{V_{ff}}$signiert und sendet, in der Summe mindestens $1 + \max(0, f-2)$ fehlerfreie Knoten überbrückt haben. Folglich müssen mindestens die folgende Anzahl an fehlerhaften Knoten aus $V'_{fh} \cup V''_{fh}$, die Data-Nachricht x signiert haben:

$$signNum_{|V'_{fh} \cup V''_{fh}}(x) \geq signNum_{|V'_{fh}}(x) + 1 + max(0, f-2) \quad 6.81$$

$$\Rightarrow signNum_{|V'_{fh} \cup V''_{fh}}(x) \geq 1 + 1 + max(0, f-2)$$

$$\Rightarrow signNum_{|V'_{fh} \cup V''_{fh}}(x) \geq f$$

Dies ist ein Widerspruch zu der f-Fehlerannahme, denn die Menge $V'_{fh} \cup V''_{fh}$ enthält höchstens $f-1$ Knoten, da der Quellknoten als fehlerhaft angenommen wurde. Folglich entscheiden sich alle fehlerfreien Knoten für dieselbe Data-Nachricht.

Fall **A2.2**: Sei signNum $(v_R^e) \geq 2$. Dies bedeutet, dass der letzte fehlerfrei Knoten in R eine Default-Nachricht mit mindestens 2 Signaturen sendet.

Sei $v_{V_{ff}}^j$ der letzte fehlerfreie Knoten in V_{ff}, der die Data-Nachricht y signiert.

Fall **A2.2.1**: Sei Knoten v_R^e Vorgänger von $v_{V_{ff}}^j$.

Sei $v_{V'_{fh} \cup V''_{fh}}^i$ ein fehlerhafter Knoten aus der Menge $V'_{fh} \cup V''_{fh}$, der eine Data-Nachricht an den ersten fehlerfreien Nachfolger von v_R^e sendet.

> Hinweis: Falls es einen solchen Knoten $v_{V'_{fh} \cup V''_{fh}}^i$ nicht gibt, dann würde laut SC3 der erste fehlerfreie Nachfolger von v_R^e die Default-Nachricht signieren, was ein Widerspruch ist, weil dieser Knoten dann ebenfalls zur Menge R gehören würde. Aus dem gleichen Grund würden auch alle nachfolgenden fehlerfreien Knoten bis hin zum fehlerfreien Knoten $v_{V_{ff}}^j$ die Default-Nachricht signieren. Damit wäre die Voraussetzung von Behauptung 3 nicht erfüllt (Behauptung, dass es zwei fehlerfreie Knoten u und v gibt, die sich für zwei unterschiedliche Data-Nachrichten x und y entscheiden) und Behauptung 3 im Fall A2.2.1 wäre trivialerweise eine wahre Aussage.

Laut MD1 empfängt der erste fehlerfreie Nachfolger von v_R^e eine Data-Nachricht mit mindestens

$$signNum\left(v_\Theta^i\right) \geq 1 + signNum(v_R^e) \quad 6.82$$

Signaturen. Sei $V_{ff,data\ y}$ die Menge der fehlerfreien Knoten in V_{ff}, die die Data-Nachricht y signiert haben. Dann sendet der Knoten $v_{V_{ff}}^j$ gemäß SC2 eine Data-Nachricht y mit mindestens

$$signNum\left(v_{V_{ff}}^{j}\right) \geq 1 + signNum(v_R^e) + signNum_{|\left\{v_{V_{ff}}^{j}\right\}}\left(v_{V_{ff}}^{j}\right) \qquad 6.83$$

$$\Rightarrow \text{signNum}\left(v_{V_{ff}}^{j}\right) \geq \text{signNum}_{|\{v_V^0\}}(y) + \text{signNum}_{|V' \cup \left\{v_{V'_{fh} \cup V''_{fh}}^{i}\right\}}(y) + 1 + \max(0, f-2) - \left|V'_{ff,data\ y}\right|$$

Signaturen.

Diese Formel lässt sich wie folgt begründen: Der fehlerfreie Knoten $v_{V_{ff}}^{j}$ sendet eine Data-Nachricht y welche von mindestens $1 + \max(0, f-2)$ fehlerfreien Knoten signiert worden ist. Dies gilt unabhängig von der Anzahl der Knoten in R (siehe ggf. Formel 6.78). Folglich enthält $\text{signNum}(y)$ mindestens 1 + $\max(0, f-2)$ Signaturen von fehlerfreien Knoten. Hinzu kommen noch mindestens die Signatur des Quellknotens und die des fehlerhaften Knotens $v_{V'_{fh} \cup V''_{fh}}^{i}$. Weiterhin kann die Data-Nachricht y von weiteren fehlerhaften wie auch fehlerfreien Knoten aus V' signiert worden sein. Daher wird in der obigen Ungleichung $\text{signNum}_{|V'}(y)$ addiert. Um zu verhindern, dass Signaturen aus $V'_{ff,data\ y}$ doppelt gezählt werden, muss von den $1 + max(0, f-2)$ Signaturen, die Anzahl der fehlerfreien Knoten aus $V'_{ff,data\ y}$ subtrahiert werden.

Laut Annahme in Fall A2.2 muss der erste fehlerfreie Nachfolger von $v_{V_{ff}}^{j}$ zuvor eine Data-Nachricht x mit der folgenden Anzahl an Signaturen empfangen haben

$$signNum(x) \geq signNum\left(v_{V_{ff}}^{j}\right) \iff signNum(x) \geq signNum(y) \qquad 6.84$$

$$signNum(x) \geq signNum_{|\{v_V^0\}}(y) + signNum_{|V' \cup \left\{v_{V'_{fh} \cup V''_{fh}}^{i}\right\}}(y) + 1 + max(0, f-2) - \left|V'_{ff,data\ y}\right|$$

$$\Longrightarrow signNum(x)$$
$$\geq signNum_{|\{v_V^0\}}(y) + signNum_{|V' \cup \left\{v^i_{V'_{fh} \cup V''_{fh}}\right\}}(y) + 1$$
$$+ max(0, f-2) - \left|V'_{ff,data\ y}\right|$$

In der folgenden Formel wird V' in V'_{ff} und V'_{fh} unterteilt:

$$\Longrightarrow signNum(x)$$
$$\geq signNum_{|\{v_V^0\}}(y) + signNum_{|V'_{ff}}(y)$$
$$+ signNum_{|V'_{fh} \cup \left\{v^i_{V'_{fh} \cup V''_{fh}}\right\}}(y)$$
$$+ 1 + max(0, f-2) - \left|V'_{ff,data\ y}\right|$$

Da die Nachricht x von keinem der fehlerfreien Knoten in $V_{ff,data\ y}$ akzeptiert, signiert und weitergesendet wurde, muss laut SC2 die Nachricht x, die der erste fehlerfreie Nachfolger von $v^j_{V_{ff}}$ signiert und sendet, in der Summe mindestens

$$signNum_{|V'_{ff}}(y) + 1 + max(0, f-2) - \left|V'_{ff,data,y}\right| \qquad 6.85$$

fehlerfreie Knoten überbrückt haben, da keiner dieser fehlerfreien Knoten die Data-Nachricht x signiert und weitergeleitet hat. Folglich müssen mindestens die folgende Anzahl an fehlerhaften Knoten aus $V'_{fh} \cup V''_{fh}$, die Data-Nachricht x signiert haben:

$$signNum_{|V'_{fh} \cup V''_{fh}}(x) \geq \qquad 6.86$$

$$signNum_{|V'_{ff}}(y) + signNum_{|V'_{fh} \cup \left\{v^i_{V'_{fh} \cup V''_{fh}}\right\}}(y) + 1 + max(0, f-2) - \left|V'_{ff,data\ y}\right|$$

Die Ersetzung von $\text{signNum}_{|V'_{ff}}(y)$ durch $\left|V'_{ff,data,y}\right|$ liefert:

$$\text{signNum}_{|V'_{fh} \cup V''_{fh}}(x) \geq$$
$$signNum_{|V'_{fh} \cup \left\{v^i_{V'_{fh} \cup V''_{fh}}\right\}}(y) + \left|V'_{ff,data,y}\right| + (1 + max(0, f-2)) - \left|V'_{ff,data\ y}\right|$$

$$\Rightarrow \text{signNum}_{|V'_{fh} \cup V''_{fh}}(x) \geq \text{signNum}_{|V'_{fh} \cup \left\{v^i_{V'_{fh} \cup V''_{fh}}\right\}}(y) + (1 + \max(0, f-2))$$

$$\Rightarrow \text{signNum}_{|V'_{fh} \cup V''_{fh}}(x) \geq 1 + (1 + \max(0, f-2))$$

$$\Rightarrow \text{signNum}_{|V'_{fh} \cup V''_{fh}}(x) \geq 2 + \max(0, f-2)$$

$$\Rightarrow \text{signNum}_{|V'_{fh} \cup V''_{fh}}(x) \geq f$$

Dies ist ein Widerspruch zu der f-Fehlerannahme, denn die Menge $V'_{fh} \cup V''_{fh}$ enthält höchstens $f-1$ Knoten, da der Quellknoten als fehlerhaft angenommen wurde. Folglich entscheiden sich alle fehlerfreien Knoten für dieselbe Data-Nachricht.

Fall **A2.2.2**: Sei Knoten v_R^e ein Nachfolger von $v_{V_{ff}}^j$, wobei $v_{V_{ff}}^j$ der letzte fehlerfreie Knoten in V_{ff} ist, der die Data-Nachricht y signiert. Dies bedeutet, dass zwischen $v_{V_{ff}}^j$ und v_R^e weitere fehlerfreie Knoten liegen können. Mit $P_{data\,x}$ bezeichnen wir die Menge der fehlerfreien Knoten, die zwischen Knoten $v_{V_{ff}}^j$ und v_R^e liegen und die die Data-Nachricht x signieren, wobei $|P_{data\,x}| \geq 0$. Gemäß MD2 und SC3 sendet v_R^e eine Default-Nachricht mit mindestens so vielen Signaturen wie die vom fehlerfreien Vorgänger gesandte Data-Nachricht x, also mit mindestens $\text{signNum}\left(v_{P_{data\,x}}^e\right)$ Signaturen. Es muss daher gelten:

$$signNum(v_R^e) \geq signNum\left(v_{V_{ff}}^j\right) + signNum\left(v_{P_{data\,x}}^e\right) \qquad 6.87$$

$$\Rightarrow signNum(v_R^e) \geq \qquad 6.88$$

$$\begin{cases} signNum_{|\{v_V^0\}}(y) + (1 + max(0, f-2)) + |P_{data\,x}|, & wenn\ \left|V'_{ff,data\,y}\right| > 0 \\ signNum_{|\{v_V^0\}}(y) + signNum_{|V'_{fh}}(y) + (1 + max(0, f-2)) + |P_{data\,x}|, & sonst \end{cases}$$

Diese Formel lässt sich wie folgt begründen: Die von Knoten v_R^e zuletzt empfangene Data-Nachricht ist von dem Quellknoten und mindestens einem Knoten aus V' signiert worden. Die von Knoten $v_{V_{ff}}^j$ gesendete Data-Nachricht y ist von mindestens $1 + \max(0, f-2)$ fehlerfreien Knoten signiert worden. Falls $\left|V'_{ff,data\,y}\right| = 0$ gilt (siehe untere Zeile der obigen Formel 6.88), dann muss laut NC2 mindestens ein fehlerhafter Knoten aus V'_{fh} die Data-Nachricht y signiert haben. Daher wird in der obigen Formel 6.88 für den Fall $\left|V'_{ff,data\,y}\right| = 0$ der Wert $\text{signNum}_{|V'_{fh}}(y)$ addiert. Zwischen den beiden

Knoten $v_{V_{ff}}^{j}$ und v_R^e können weitere fehlerfreie Knoten liegen, die die Data-Nachricht x signieren. Ein Knoten signiert die Data-Nachricht x genau dann, wenn sie mindestens so viele Signaturen enthält wie die zuletzt gesendete Data-Nachricht y. Daher wird in der obigen Formel 6.88 für beide Fälle, d.h für $\left|V'_{ff,data\,y}\right| > 0$ und für $\left|V'_{ff,data\,y}\right| = 0$, der Wert $|P_{data\,x}|$ addiert.

Sei v_Θ^i ein fehlerhafter Knoten, der eine Data-Nachricht x an den ersten fehlerfreien Nachfolger von $v_{V_{ff}}^{j}$ sendet.

> Hinweis: Falls es einen solchen Knoten v_Θ^i nicht gibt, dann würden laut SC3 alle fehlerfreien Nachfolger von v_R^e (es gilt $|P_{data\,x}| \geq 0$) die Default-Nachricht signieren. Damit wäre die Voraussetzung von Behauptung 3 nicht erfüllt (Behauptung, dass der letzte fehlerfreie Knoten $v_{V''_{ff}}^{e}$ eine Data-Nachricht sendet) und Behauptung 3 im Fall A2.2.2 wäre trivialerweise eine wahre Aussage.

Laut MD1 und SC2 enthält die von Knoten v_Θ^i gesendete Data-Nachricht x mindestens die folgende Anzahl an Signaturen:

$$signNum\left(v_\Theta^i\right) \geq 1 + signNum\left(v_R^e\right) \tag{6.89}$$

$$\Rightarrow signNum\left(v_\Theta^i\right) \geq$$

$$\begin{cases} 1 + \left(signNum_{\,|\{v_V^0\}}(y) + (1 + max(0, f-2)) + |P_{data\,x}|\right), & wenn\ \left|V'_{ff,data\,y}\right| > 0 \\ 1 + \left(signNum_{\,|\{v_V^0\}}(y) + signNum_{\,|V'_{fh}}(y) + (1 + max(0, f-2)) + |P_{data\,x}|\right), & sonst \end{cases}$$

Daraus folgt für den Fall $\left|V'_{ff,data\,y}\right| > 0$:

$$signNum\left(v_\Theta^i\right) \geq \tag{6.90}$$
$$1 + signNum_{\,|\{v_V^0\}}(y) + (1 + max(0, f-2)) + |P_{data\,x}|$$

Da die Nachricht x, die von Knoten v_Θ^i gesendet wird, von keinem der fehlerfreien Knoten in $V_{data\,y}$ akzeptiert, signiert und weitergesendet wurde, muss laut SC2 die Nachricht x, die der erste fehlerfreie Nachfolger von $v_{V_{ff}}^{j}$ signiert und sendet, in der Summe mindestens so viele Knoten überbrückt haben, wie fehlerfreie Knoten die Data-Nachricht y signiert

haben. Folglich kann die Anzahl der fehlerhaften Knoten aus $V'_{fh} \cup V''_{fh}$, die die Data-Nachricht x signiert haben, wie folgt bestimmt werden: Aus der Formel 6.90 werden die Signatur des Quellknotens und die Signaturen der Knoten aus der Menge $P_{data\,x}$ entfernt, da keiner dieser Knoten in der Menge $V'_{fh} \cup V''_{fh}$ enthalten ist. Die Anzahl der Signaturen beträgt also:

$$signNum_{|V'_{fh} \cup V''_{fh}}(v_\Theta^i) \geq 1 + (1 + max(0, f-2)) \tag{6.91}$$

$$\Rightarrow signNum_{|V'_{fh} \cup V''_{fh}}(v_\Theta^i) \geq 2 + max(0, f-2)$$

$$\Rightarrow \text{signNum}_{|V'_{fh} \cup V''_{fh}}(v_\Theta^i) \geq f$$

und für den Fall $\left|V'_{ff,data\,y}\right| = 0$

$$signNum\,(v_\Theta^i) \geq \tag{6.92}$$

$$1 + \left(signNum_{|\{v_V^0\}}(y) + signNum_{|V'_{fh}}(y) + (1 + max(0, f-2))\right) + |P_{data\,x}|$$

$$\Rightarrow signNum\,(v_\Theta^i) \geq 1 + \left(signNum_{|\{v_V^0\}}(y) + 1 + (1 + max(0, f-2))\right) + |P_{data\,x}|$$

$$\Rightarrow signNum\,(v_\Theta^i) \geq signNum_{|\{v_V^0\}}(y) + 3 + max(0, f-2) + |P_{data\,x}|$$

Folglich (Begründung analog zu Formel 6.90 und 6.91) müssen mindestens die folgende Anzahl an fehlerhaften Knoten aus $V'_{fh} \cup V''_{fh}$, die Data-Nachricht x signiert haben:

$$signNum_{|V'_{fh} \cup V''_{fh}}(v_\Theta^i) \geq 0 + 3 + max(0, f-2) + 0 \tag{6.93}$$

$$\Rightarrow \text{signNum}_{|V'_{fh} \cup V''_{fh}}(v_\Theta^i) \geq f + 1$$

Beide Fälle, d.h. $\left|V'_{ff,data\,y}\right| > 0$ und $\left|V'_{ff,data\,y}\right| = 0$, stehen im Widerspruch zu der f-Fehlerannahme, denn die Menge $V'_{fh} \cup V''_{fh}$ enthält höchstens $f-1$ Knoten, da der Quellknoten als fehlerhaft angenommen wurde. Folglich entscheiden sich alle fehlerfreien Knoten für dieselbe Data-Nachricht.

Behauptung 4. Falls der letzte fehlerfreie Knoten eine Data-Nachricht sendet, dann ist die Bedingung IC1 erfüllt (d.h. die Endwerte der fehlerfreien Knoten stimmen überein).

Beweis:

Der Beweis erfolgt analog zu **LEMMA 2** mit dem Unterschied, dass in **LEMMA 3** die Anzahl der fehlerhaften Knoten in V''_{fh} höchstens $f-2$ beträgt. Dies bedeutet, dass die Menge V''_{fh}, anders als in **LEMMA 2**, einen fehlerhaften Knoten weniger enthält, weil hier angenommen wird, dass mindestens einer der Knoten in V' fehlerhaft ist.

Folglich enthält die von Knoten $v^e_{V_{ff}}$ übertragene Data-Nachricht, anders als in **LEMMA 2**, höchstens $f-2$ fehlerhafte Signaturen aus V''_{fh}.

Dies lässt sich wie folgt begründen: Es gilt $\left|V'_{fh}\right| \geq 1 \ \wedge \ v_V^0 \in V_{fh}$. Folglich enthält die Menge V''_{fh} laut f-Fehlerannahme höchstens

$$\left|V''_{fh}\right| \leq f - \left(\left|V'_{fh} \cup \{v_V^0\}\right|\right) \qquad 6.94$$

$$\Rightarrow \left|V''_{fh}\right| \leq f-2$$

fehlerhafte Knoten. Gemäß bewiesener Behauptung 3 haben alle fehlerfreien Knoten in PMB dieselbe Data-Nachricht gespeichert. Diese Data-Nachricht enthält im Vergleich zu Behauptung 3 von **LEMMA 2** mindestens eine fehlerfreie Signatur mehr. In Behauptung 3 von **LEMMA 2** wurde die Mindestanzahl der betreffenden Signaturen mit

$$signNum_{|\{v_V^0\} \cup V_{ff}}\left(v^e_{V''_{ff}}\right) \geq f \ \wedge \ signNum\left(v^e_{V''_{ff}}\right) \geq f+1$$

quantifiziert. Diese Anzahl wird „erst recht" überschritten, wenn die Data-Nachricht eine Signatur mehr als $signNum_{|\{v_V^0\} \cup V_{ff}}\left(v^e_{V''_{ff}}\right) \geq f$ enthält. Folglich entscheiden sich die fehlerfreien Knoten in dem von **LEMMA 3** vorausgesetzten Fall (analog zu **LEMMA 2**) für dieselbe Data-Nachricht. Damit gilt **LEMMA 3** unter der Voraussetzung von Behauptung 3.

Behauptung 5. *Falls der letzte fehlerfreie Knoten eine Default-Nachricht sendet, dann ist die Bedingung IC1 erfüllt.*

Beweis:

Der Beweis erfolgt analog zu **LEMMA 2** mit dem Unterschied, dass in **LEMMA 3** die Anzahl der fehlerhaften Knoten in V''_{fh} höchstens $f-2$ beträgt (d. h. die Menge V''_{fh}

enthält, anders als in **LEMMA 2**, einen fehlerhaften Knoten weniger, weil hier angenommen wird, dass mindestens einer der Knoten in V'' fehlerhaft ist).

Fall (**A1**): Kein fehlerfreier Knoten aus V''_{ff} sendet eine Data-Nachricht, d.h. $|P| = 0$.

Fall (**A1.1**): Sei $\left|V'_{ff,data}\right| = 0$. Dies bedeutet, dass kein fehlerfreier Knoten aus V'_{ff} eine Data-Nachricht sendet.

Wegen der f-Fehlerannahme kann kein fehlerhafter Knoten eine Data-Nachricht mit mehr als f Signaturen senden. Damit ist AG1 niemals erfüllt und alle fehlerfreien Knoten entscheiden sich für die globale Konstante, wodurch IC1 erfüllt ist.

Fall (**A1.2**): Sei $\left|V'_{ff,data}\right| > 0$. Dies bedeutet, dass mindestens ein fehlerfreier Knoten aus V'_{ff} eine Data-Nachricht sendet.

Sei $v^i_{Q\cup T}$ ein fehlerhafter Knoten in V''_{ff}, der eine Default-Nachricht an den ersten fehlerfreien Knoten in V''_{ff} sendet.

> Hinweis: Es kann ausgeschlossen werden, dass die erste fehlerfreie Default-Nachricht von einem fehlerfreien Knoten aus V''_{ff} gesendet wird, weil dieser gemäß der Annahme in Fall A1.2 mindestens eine Data-Nachricht von einem Knoten aus V'_{ff} empfangen hat und diese signieren und weiterleiten würde.

Gemäß MD2 und SC3 empfängt der erste fehlerfreie Knoten in V'' eine Default-Nachricht mit mindestens der folgenden Anzahl an Signaturen:

$$signNum\left(v^i_{Q\cup T}\right) \geq signNum\left(v^e_{V'}\right) \tag{6.95}$$

Nach Ausführung der „Cleaning Function“ CF hat kein fehlerfreier Knoten in PMB oder SMB eine Data-Nachricht x mit mehr als

$$signNum(x) \leq \left(signNum\left(v^e_{V'}\right) + \left|V'_{fh} \cup V''_{fh}\right|\right) - signNum\left(v^i_{Q\cup T}\right) \tag{6.96}$$

Diese Formel ist wie folgt zu begründen: Der letzte fehlerfreie Knoten in V' sendet eine Data-Nachricht, die gemäß NC1 vom fehlerhaften Quellknoten signiert worden ist. Da die fehlerhaften Knoten in $V'_{fh} \cup V''_{fh}$ miteinander kooperieren können, ist jeder fehlerhafte Nachfolger von $v^e_{V'_{ff}}$ in der Lage, die Data-Nachricht von $v^e_{V'_{ff}}$ um weitere $\left|V'_{fh} \cup V''_{fh}\right|$ Signaturen zu ergänzen. Daher wird in obiger Formel $\left|V'_{fh} \cup V''_{fh}\right|$ addiert.

Weiterhin muss laut SC3 und der Annahme in Fall A.1 der fehlerhafte Knoten $v^i_{Q\cup T}$ eine Default-Nachricht mit mindestens $signNum_{|\{v^0_V\}\cup V'_{ff}}(v^e_{V'})$ Signaturen senden. Es gilt also $signNum\left(v^i_{Q\cup T}\right) \geq signNum_{|\{v^0_V\}\cup V'_{ff}}(v^e_{V'})$. Mit dieser Ungleichung lässt aus der obigen Formel folgern:

$$signNum(x) \leq \left(signNum_{|\{v^0_V\}\cup V'_{ff}}(v^e_{V'}) + \left|V'_{fh} \cup V''_{fh}\right|\right) - signNum_{|\{v^0_V\}\cup V'_{ff}}(v^e_{V'}) \quad 6.97$$

$$\Rightarrow signNum(x) \leq \left|V'_{fh} \cup V''_{fh}\right|$$

$$\Rightarrow signNum(x) \leq f - 1$$

Signaturen gespeichert. Damit ist AG2 in keinem fehlerfreien Knoten erfüllt und alle fehlerfreien Knoten entscheiden sich für die globale Konstante, womit IC1 erfüllt ist.

Fall (A2): Mindestens ein fehlerfreier Knoten aus V''_{ff} sendet eine Data-Nachricht, d.h. $|P| > 0$.

Sei $v^i_{Q\cup T}$ ein fehlerhafter Knoten in V''_{fh}, der eine Default-Nachricht an den ersten fehlerfreien Nachfolger von v^e_P sendet. Gemäß MD2 signiert der erste fehlerfreie Nachfolger des letzten fehlerfreien Knotens v^e_P von P die Default-Nachricht von $v^i_{Q\cup T}$, wenn diese mindestens die folgende Signaturanzahl aufweist:

$$signNum\left(v^i_{Q\cup T}\right) \geq signNum(v^e_P) \quad 6.98$$

Nach Ausführung der „Cleaning Function“ CF enthält kein fehlerfreier Knoten in PMB oder SMB eine Data-Nachricht x mit mehr als

$$signNum(x) \leq \quad 6.99$$

$$\left(signNum(v^e_P) + \left|V'_{fh} \cup V''_{fh}\right| - signNum_{|V'_{fh}\cup V''_{fh}}(v^e_P)\right) - signNum\left(v^i_{Q\cup T}\right)$$

Diese Formel ist wie folgt zu begründen: Der letzte Knoten aus P sendet eine Data-Nachricht mit $signNum(v^e_P)$ Signaturen. Diese Nachricht ist von dem fehlerhaften Quellknoten und mindestens einem Knoten aus V' signiert worden. Da die fehlerhaften Knoten in $V'_{fh} \cup V''_{fh}$ miteinander kooperieren können, ist jeder fehlerhafte Nachfolger von

v_P^e in der Lage, die Data-Nachricht von v_P^e um weitere $\left|V'_{fh} \cup V''_{fh}\right| - signNum_{|V'_{fh} \cup V''_{fh}}(v_P^e)$ Signaturen zu ergänzen. Daher wird in obiger Formel $\left|V'_{fh} \cup V''_{fh}\right| - signNum_{|V'_{fh} \cup V''_{fh}}(v_P^e)$ addiert. Aus der obigen Formel folgt:

$$\begin{aligned}\text{signNum}(x) &\leq signNum_{|\{v_V^0\} \cup V' \cup P}(v_P^e) + \left(\left|V'_{fh} \cup V''_{fh}\right| - signNum_{|V'_{fh} \cup V''_{fh}}(v_P^e)\right) \\ &\quad - signNum_{|\{v_V^0\} \cup V'_{ff} \cup P}(v_P^e)\end{aligned}$$

Diese Formel ist wie folgt zu begründen: Die Data-Nachricht x ist von dem Quellknoten und höchstens $f + 1$ Knoten aus V' signiert worden. Hinzu kommen die Signaturen der fehlerfreien Knoten aus P. Weiterhin enthält laut SC3 die Default-Nachricht $signNum\left(v_{Q \cup T}^i\right)$ mindestens so viele Signaturen (siehe ggf. Formel 6.99) wie fehlerfreie Knoten überbrückt worden sind. Daher wird in obiger Formel $signNum_{|\{v_V^0\} \cup V'_{ff} \cup P}(v_P^e)$ subtrahiert und es gilt sodann:

$$signNum(x) \leq$$

$$signNum_{|\{v_V^0\} \cup V' \cup P}(v_P^e) + \left(\left|V'_{fh} \cup V''_{fh}\right| - signNum_{|V'_{fh} \cup V''_{fh}}(v_P^e)\right) - signNum_{|\{v_V^0\} \cup V'_{ff} \cup P}(v_P^e)$$

$$\Rightarrow signNum(x) \leq$$

$$\left(signNum_{|V'_{fh} \cup V''_{fh}}(v_P^e) + signNum_{|\{v_V^0\} \cup V'_{ff} \cup P}(v_P^e)\right) + \left(\left|V'_{fh} \cup V''_{fh}\right| - signNum_{|V'_{fh} \cup V''_{fh}}(v_P^e)\right) - signNum_{|\{v_V^0\} \cup V'_{ff} \cup P}(v_P^e)$$

$$\Rightarrow signNum(x) \leq \left|V'_{fh} \cup V''_{fh}\right|$$

$$\Rightarrow signNum(x) \leq f - 1$$

Signaturen gespeichert. Damit ist AG2 in keinem fehlerfreien Knoten erfüllt und alle fehlerfreien Knoten entscheiden sich für die globale Konstante, womit IC1 erfüllt ist.

> Anmerkung In dem Beweis wurde der Fall betrachtet, dass ein fehlerfreier Nachfolger von v_P^e eine Default-Nachricht sendet, welche die Nachricht von v_P^e durch eine höhere Signaturanzahl „überstimmt“. Tatsächlich könnte auch der Fall auftreten, dass das „Überstimmen“ durch einen fehlerhaften Nachfolger erfolgt. Dieser Fall muss hier nicht weiter betrachtet werden, denn laut der Voraussetzung in Behauptung 5 und der Annahme in Fall A.2 können wir davon ausgehen, dass v_P^e mindestens einen fehlerfreien Nachfolger hat, der eine Default-Nachricht sendet. Dieser Knoten muss laut SC3 zuvor eine Default Nachricht empfangen haben, die mindestens so viele Signaturen aufweist wie v_P^e. Ob zwischen v_P^e und dem fehlerfreien Nachfolger von v_P^e weitere fehlerhafte Knoten existieren, ist hier nicht relevant.

*Beweis von **LEMMA 3**:* Behauptung 1 beweist **LEMMA 3** für den Fall, dass kein fehlerfreier Knoten in V_{ff}'' die Data-Nachricht signiert. Behauptung 4 beweist **LEMMA 3** für den Fall, dass der letzte fehlerfreie Knoten eine Data-Nachricht überträgt. Behauptung 5 beweist **LEMMA 3** für den Fall, dass der letzte fehlerfreie Knoten eine Default-Nachricht überträgt. Damit ist dieses Lemma für jeden möglichen Fall bewiesen.

Theorem 2*: Wenn die Knotenmenge V höchstens f fehlerhafte Knoten enthält, dann gibt es keine zwei fehlerfreien Knoten die sich für unterschiedliche Endwerte entscheiden. Falls der Quellknoten fehlerfrei ist, setzt sich sein Wert als Endwert durch. Formal:*

1. *IC1*: $\Leftrightarrow v_V^0 \in V_{fh}\ und\ |V_{fh}| \leq f \Rightarrow \forall u, v\ \in V_{ff}: finalM(u) = finalM(v)$.
2. *IC2*: $\Leftrightarrow v_V^0 \in V_{ff}\ und\ |V_{fh}| \leq f \Rightarrow \forall u \in V_{ff}: finalM(u) = send(v_V^0)$.

Theorem 2 folgt unmittelbar aus **Theorem 1** und **LEMMA 3**

7 SigSeam – Ein neues Signaturverfahren

„An algorithm must be seen to be believed. “ – Donald Ervin Knuth.

7.1 Idee und Motivation

Übereinstimmungsprotokolle kommen in verteilten Rechensystemen immer dann zum Einsatz, wenn eine globale Konsistenz auf fehlertolerante Weise im System erreicht werden soll. Allerdings stellt das Lösen des Übereinstimmungsproblems, wenn es auf effiziente Weise bzgl. der Kommunikationskomplexität (siehe Punkte 1 und 2 unten) erfolgen soll, immer noch eine anspruchsvolle Aufgabe dar. Der praktische Einsatz von Übereinstimmungsprotokollen in (synchronen) verteilten Systemen wird wesentlich durch zwei Faktoren bestimmt, nämlich:

1. die Festlegung der Fehlertypen, die von dem Protokoll toleriert werden sollen (omission, byzantine, etc.), und
2. der vom Protokoll verursachte Overhead in Form von Nachrichtenübertragungen, Speicheraufwand (zu speichernde Nachrichten) und redundanten Komponenten (zusätzliche Kommunikationskanäle und Knoten).

Ohne Zweifel leisten digitale Signaturen in Übereinstimmungsprotokollen einen erheblichen Beitrag zur Reduktion der Kommunikationskomplexität (siehe Punkt 2 oben), da sie die Möglichkeit eröffnen, die Herkunft der Nachricht gegen nicht detektierbare Verfälschungen zu schützen, wenn eine Nachrichtweiterleitung von Knoten zu Knoten erfolgt (Lamport et al. 1982). Ohne den Einsatz von Signaturen kann dieser Schutz nur durch einen entsprechenden hohen Nachrichtenaufwand (ggf. höheren Einsatz von redundanten Komponenten) kompensiert werden (siehe z.B.“ oral message“ Protokoll in (Lamport et al. 1982)).

Beim Einsatz „herkömmlicher“[4] Signaturverfahren in Übereinstimmungsprotokollen tritt häufig das Problem auf, dass Nachrichten mit identischem Inhalt in mehreren Phasen und/oder Runden wiederholt übertragen werden müssen. Der Grund hierfür liegt unter ande-

[4] Das sind solche Verfahren, die ausschließlich zum Zweck der Fehlererkennung konzipiert worden sind. D. h. beim Einsatz solcher Signaturverfahren besteht nicht die Möglichkeit, Nachrichten, die sich nur in der Signatur unterscheiden, zu einer einzigen Nachricht zu verschmelzen, ohne dabei die Nachricht selbst bzw. den Aufwand zur Speicherung der Nachricht zu vergrößern.

rem in der fehlenden Möglichkeit, mehrere Nachrichten, welche sich nur in der Signatur unterscheiden, zu einer einzigen Nachricht zusammenzufassen, ohne die Nachricht in ihrer Länge zu verändern. Wenn beispielweise ein Knoten zwei Nachrichten $\sigma_i(v)$ und $\sigma_j(v)$ besitzt, dann kann er zwar die Nachricht zu zusammenfassen, in dem er die Signaturen $\sigma_j(v)$ an die Nachricht $\sigma_i(v)$ „anhängt“. Jedoch würde die Nachricht mit jeder „angehängten“ Signaturen länger werden. Das Ziel besteht also darin, eine Funktion V („Verschmelzungsfunktion“ genannt) zu entwerfen, die mehrere Signaturen auf eine neue Signatur k abbildet
(V: $D \subseteq \mathbb{N}^n \rightarrow \mathbb{N}$ mit $n \geq 3f + max(0, f-2)$) sowie alle gewünschten Signatureigenschaften (wie etwa Datenintegrität) aufweist.

In diesem Kapitel präsentieren wir ein neuartiges Signaturgenerierungsverfahren SigSeam (Bousbiba 2015b), das die Signaturverschmelzung ermöglicht. Dadurch wird einem Knoten die Möglichkeit eröffnet, mehrere empfangene Nachrichten, die sich nur in ihren Signaturen unterscheiden, zu einer einzigen Nachricht zu verschmelzen, ohne dabei die Nachrichtengröße zu verändern. Wie wir später sehen werden (siehe Abschnitt 7.10), sind solche Signaturverschmelzungsverfahren in der Lage, einen signifikanten Beitrag zur Reduzierung des Nachrichtenoverheads zu leisten.

An späterer Stelle werden die Vorteile des neuen Signaturverfahrens anhand zweier existierender Übereinstimmungsprotokolle (Turquois (Moniz et al. 2013) und ESSEN (Bousbiba 2015a)) demonstriert sowie der Tradeoff zwischen SigSeam und dem Verfahren aus (Echtle 2005) sowie (Echtle und Kimmeskamp 2010) veranschaulicht.

Die typischen Sequenzen von Aktionen, die während der Ausführung eines Übereinstimmungsprotokolls ausgeführt werden, lassen sich in der Regel in die folgenden vier Schritten gliedern:

1. Sende eine signierte Nachricht an einen oder mehreren Nachbarknoten.
2. Leite die Nachricht von Knoten zu Knoten weiter, wobei jeder Versenderknoten die Nachricht kosigniert.
3. Sammele die eintreffenden Nachrichten (diese können zahlreich sein) und führe jeweils Signaturchecks aus.
4. Triff eine lokale Entscheidung bzgl. der Nachricht, die in der nächsten Runde versendet werden soll.
5. Terminiere mit einem Konsistenzvektor.

Sei M die Menge aller übertragenen Nachrichten, die während einer Protokollausführung ausgetauscht werden. Eine Nachricht, die von einem Knoten N[i] übertragen wird, bezeichnen wir im Folgenden mit M[i]. Die Nachricht M[i], die von einem Knoten N[i] übertragen und von einem Knoten N[j] über einen Kommunikationskanal empfangen wird, bezeichnen wir mit M[i][j].

Die Schritte 1 bis 4 werden abhängig vom verwendeten Übereinstimmungsprotokoll mehrfach wiederholt (z.B. wie in (Moniz et al. 2013; Jochim und Forest 2010)). Typischerweise tritt die folgende Situation häufig auf: In einer bestimmten Phase empfängt ein Knoten N[i] verschiedene Nachrichten M[1][i], … , M[k][i] für i, k $\in \mathbb{N}$ mit M[i] ≠ M[k] sowie i ≠ k, wobei all diese Nachrichten an einen Nachbarknoten N[j] weitergeleitet werden müssen. Wenn die Nachrichten M[1][i], … , M[k][i]) von verschiedenen Knoten N[1], … , N[k] signiert wurden, wobei die Nutzdaten der Nachrichten (der Signaturwert zählt nicht dazu) den gleichen Inhalt A aufweisen (siehe Abbildung 20), wäre der Knoten N[i] nicht in der Lage, die Nachrichten zu einer einzigen Nachricht zu verschmelzen und diese an Knoten N[j] zu übertragen. Stattdessen muss der Knoten N[i] entweder die k Nachrichten einzeln oder eine entsprechend lange Nachricht (in dem vorliegenden Fall eine Nachricht mit k – 1 „angehängten“ Signaturen) übertragen.

Folglich haben Signaturverfahren, die in der Lage sind, mehrere Nachrichten zu einer einzigen Nachricht zusammenzufassen, das Potential, den Kommunikationsoverhead von Übereinstimmungsprotokollen stark zu reduzieren. Abbildung 20 veranschaulicht anhand eines fiktiven Beispiels die Idee, die hinter der Signaturverschmelzung steckt.

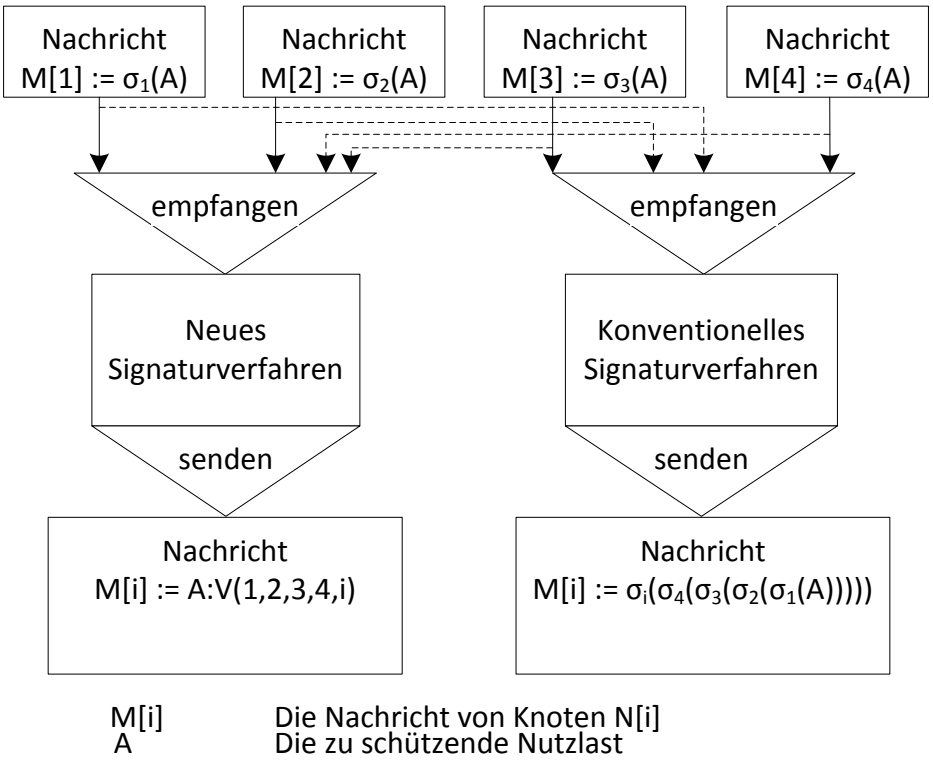


Abbildung 20: Ein Knoten N[i] empfängt vier Nachrichten, signiert von vier verschiedenen Knoten. links das neue, rechts das konventionelle Signaturverfahren.

7.2 Das Konzept der Signaturverschmelzung

Wenn in verteilten Rechensystemen Nachrichten über Knoten weitergeleitet werden müssen, verhindert der Einsatz von Signaturen nicht, dass die im System übertragenen Daten

- durch fehlerhafte Knoten verfälscht werden,
- nicht an Nachbarknoten weitergeleitet werden,
- zu unzulässigen Zeitpunkten gesendet werden,
- Nachrichtenüberflutung in Form von Babbling (Temple 1998) stattfindet.

Jedoch kann der Einsatz von Signaturen einen Beitrag zur Fehlererkennung leisten. Zum Zweck der Fehlertoleranz ist der Einsatz kryptographisch starker Signaturverfahren jedoch nicht zwingend erforderlich (Leu 1994), um die Integrität einer Nachricht zu gewährleisten. Typischerweise werden in Fehlertoleranz-Verfahren (wie etwa in Übereinstimmungsprotokollen) Signaturverfahren zum Schutz gegen „dumme Fehler“, nicht aber gegen „intelligente Angriffe“ eingesetzt. Daher lassen sich rechnerisch aufwandsarme Signaturverfahren einsetzen (bzgl. der Generierung und dem Check der Signatur).

Durch den Einsatz existierender Signaturverfahren (Beispielsweise solche wie in (Leu 1994; Echtle 1999) beschrieben) werden einem Knoten nur zwei Möglichkeiten angeboten, um auf eine bzw. mehrere empfangenen Nachrichten zu reagieren:

1. Entweder speichert der Knoten jede der empfangenen Nachrichten einzeln ab, wie beispielsweise (in ähnlicher Weise) in Turquois (Moniz et al. 2013) verfahren wird, oder
2. eine Art von Filtermechanismus muss angewendet werden. Beispielsweise wird von den empfangenen Nachrichten nur die mit den meisten Signaturen oder die zuerst empfangene und als gültige bzw. fehlerfrei eingestufte Nachricht tatsächlich im Speicher hinterlegt.

Nichtsdestotrotz kann es in beiden Fällen zu einem hohen Kommunikationsaufwand kommen. Bei dem hier vorgestellten Signaturverschmelzungsverfahren erhält der Knoten die Möglichkeit, die Nachrichten in einer einzigen Nachricht zu verschmelzen, ohne die Informationen bzgl. der Signaturquellen zu beeinflussen bzw. zu ändern. Das bedeutet, dass die neue Signatur die Informationen bzgl. aller verschmolzenen Signaturen enthält (wie in Abbildung 20 dargestellt).

Bei der Entwicklung des neuen Signaturverfahrens gehen wir von denselben Annahmen aus wie in (Echtle 1999; Leu 1994). D. h. zum Zwecke der Fehlertoleranz reicht es, wenn das Verfahren so konzipiert es, dass es einen ausreichenden Schutz gegen willkürliche technische Fehler bietet. Das neue Signaturverfahren SigSeam basiert auf einem ähnlichen Multiplikationsschema wie in (Echtle 2005; Echtle und Kimmeskamp 2010) beschrieben.

Im Folgenden wird die Signaturgenerierung, Kosignierung und Signaturprüfung in SigSeam näher erläutert.

7.3 Signaturgenerierung und Signaturprüfung

Alle Berechnungen finden in Modulo m statt, wobei m eine Zweierpotenz ist ($m = 2^x$ mit $x \in \mathbb{N}$). Typischer Wert für m sind $m = 2^{16}$, $m = 2^{32}$ oder $m = 2^{64}$. Zur Generierung der privaten und öffentlichen Schlüssel werden zunächst zwei ungerade natürliche Zahlen $a, b \in \mathbb{N}_u$ mit $\mathbb{N}_u = \{x \in \mathbb{N} \mid x \equiv 1 \bmod 2\}$ aus dem Bereich $m_u = [m/16, m/2] \cap \mathbb{N}_u$ bestimmt. Anschließend wird das Produkt $c = a \cdot b$ bestimmt (wobei hier wie auch bei allen folgenden Rechenschritten modulo m gerechnet wird). Der Parameter a dient als privater Schlüssel, während das Paar

(b, c) als öffentlicher Schlüssel an alle teilnehmenden Knoten zur Signaturprüfung statisch zugeordnet. Anmerkung: Die statische Schlüsselzuordnung während der Systemkonfigurierung ist für ein Automatisierungssystem angemessen, da die Signaturen nicht vor unlauteren Benutzern, sondern vor technischen Fehlern schützen sollen. Ein Protokoll zur Schlüsselverteilung ist nicht erforderlich.

Die Güte des Signaturverfahrens wird im Wesentlichen durch den Parameter m bestimmt.

Im Folgenden steht der Parameter D repräsentativ für die im Übereinstimmungsprozess übertragenen Nutzdaten und SeqNr für die fortlaufende Sequenznummer. Es sei angemerkt, dass sich im Falle von Übereinstimmungsprotokollen die Sequenznummer nur zu bestimmten Kommunikationsrunden ändern darf (z.B. nach f + 1 Phasen, siehe Definition 2.2), siehe z.B. (Jochim und Forest 2010)), andernfalls würde die Kosignierung nicht ohne weiteres gelingen[5].

Die folgende Signaturfunktion σ_0 und eine gewöhnliche CRC-Funktion werden vom Erstsignierer verwendet (z. B. vom Quellknoten, der mit der Zahl 0 indiziert ist).

Der Erstsignierer verwendet die folgende Signierfunktion, um den zu schützenden Nachrichteninhalt vor Verfälschung zu sichern:

$$\sigma_0 \coloneqq CRC(SeqNr, D) \cdot a_0 \tag{7.1}$$

Um die Signatur der Quellnachricht (das ist die Nachricht, die von keinen Knoten zuvor signiert wurde) zu bestimmen, wird zunächst die CRC-Funktion über die Nutzlast D und die Sequenznummer SeqNr gebildet. Das Ergebnis wird im Anschluss mit dem privaten Schlüssel a_0 multipliziert und mit Modulo m abgeschlossen.

Knoten N[i], der den Schlüssel a_i besitzt und eine Nachricht empfangen hat, die die Sequenznummer SeqNr, den Nutzinhalt D und die Signatur σ_j enthält, kosigniert diese Nachricht, indem er die folgende Kosignaturfunktion σ_i bildet:

$$\sigma_i \coloneqq \begin{cases} \sigma_j + CRC(SeqNr, D) \cdot (a_i + 1), wenn\ j \equiv 1\ mod\ 2 \\ \sigma_j + CRC(SeqNr, D) \cdot (a_i - 1), sonst \end{cases} \quad mit\ j < i. \tag{7.2}$$

[5] Außer wenn die Signaturen an die Nachricht angehängt werden, d. h. wenn die Nachricht neben den Nutzdaten zusätzlich mehrere Signaturen enthält. Dieser Ansatz würde jedoch die Nachricht unnötigerweise vergrößern.

Abhängig von der Anzahl der in σ_j gespeicherten Signaturen inkrementiert ($a_i + 1$) bzw. dekrementiert ($a_i - 1$) der kosignierende Knoten den Wert seines privaten Schlüssels, bevor der neue Signaturwert gebildet wird. Bei ungerader Signaturanzahl wird inkrementiert, andernfalls dekrementiert. Damit es möglich ist, diese beiden Fälle zu unterscheiden, muss neben der Signatur σ_j selbst auch die Anzahl der darin verschmolzenen Signaturen aus der Nachricht hervorgehen. Ein Verfahren dafür wird im Folgenden vorgestellt. Nachdem σ_i erzeugt worden ist, sendet der Knoten N[i] die kosignierte Nachricht, die SeqNr, D und σ_i enthält.

In dem hier vorgeschlagen Signaturverschmelzungsschema ersetzt zwar die neue Kosignatur σ_i die vorhandene Kosignatur σ_j, jedoch müssen die Indizes aller signierenden Knoten in die Nachricht eingetragen werden. Andernfalls ist ein Empfänger der Nachricht nicht in der Lage, eine Signaturprüfung durchzuführen. Folglich gibt es in jeder Nachricht eine Liste, die aussagt: „Wer hat diese Nachricht signiert“. Diese Liste wird im Allgemeinen von allen Signaturverfahren benötigt, da andernfalls die Signaturprüfung unnötig erschwert werden würde. Bevor die Signier- und Prüfunktionen näher erläutert werden, soll zunächst das von SigSeam verwendete Nachrichtenformat (siehe ggf. Abschnitt 7.6) beschrieben werden.

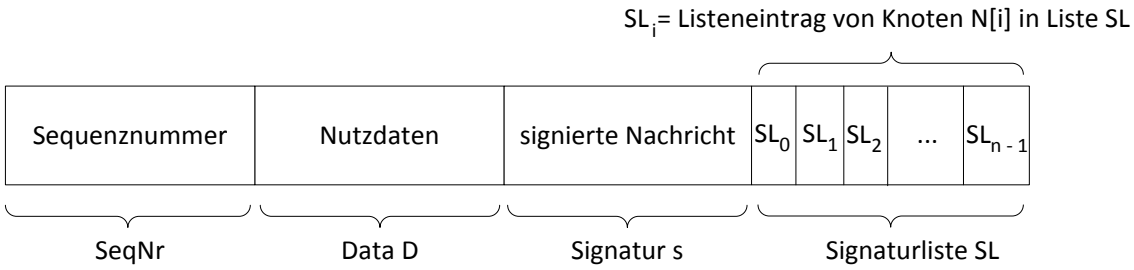


Abbildung 21: Zeigt den generellen Aufbau einer Nachricht

Wie in Abbildung 21 zu sehen besteht die Nachricht aus einer Sequenznummer SeqNr, den Nutzdaten D, der Signatur s und der Signaturliste SL (Diese Liste wird in Abschnitt 7.6 erläutert). Bis auf die Signaturlist SL fließen alle Bereiche (SeqNr, D und s) in die Berechnung der Signatur s ein. Die Signaturliste SL beeinflusst lediglich die Art und Weise, wie Signatur und Ko-Signatur miteinander verknüpft werden. Anders als bei der Signaturgenerierung fließen die einzelnen Signierer-Einträge SL_i in der Signaturliste SL bei der

Signaturprüfung (siehe Formel 7.5) mit ein. Dabei gibt der Wert SL_i an, ob die Signaturquelle N[i] in der Signaturberechnung überhaupt nicht ($SL_i = 0$), einfach ($SL_i = 1$) oder doppelt ($SL_i = 2$) eingeflossen ist.

Die Verschmelzungsfunktion für mehrere Nachrichten wird durch die folgende Verschmelzungsfunktion V gebildet, wobei $\#: \mathbb{N} \to \mathbb{N}$ eine Funktion ist, die die Anzahl der Signaturen von σ_i als Ergebnis zurückliefert.

$$V(i,j) \coloneqq \begin{cases} \sigma_i + \sigma_j, & \#(\sigma_i)\ ist\ ungerade\ und\ \#(\sigma_j)\ ist\ gerade \\ \sigma_i + \sigma_j, & \#(\sigma_i)\ ist\ gerade\ und\ \#(\sigma_j)\ ist\ ungerade \\ \sigma_i + \sigma_j - CRC(SeqNr, D), & \#(\sigma_i)\ und\ \#(\sigma_j)\ sind\ gerade \\ \sigma_i + \sigma_j + CRC(SeqNr, D), & \#(\sigma_i)\ und\ \#(\sigma_j)\ sind\ ungerade \end{cases} \quad 7.3$$

Die Signatur s einer Nachricht, die von j Knoten mit $j \geq 0$ signiert wurde, lässt sich durch die folgende Funktion ausdrücken:

$$s \coloneqq CRC(SeqNr, D) \cdot \left((j+1)\ mod\ 2 + \sum_{i=0}^{j} a_i\right). \quad 7.4$$

Ein Empfängerknoten prüft die Korrektheit der Signatur s einer empfangenen Nachricht mit Nutzinhalt D und Sequenznummer SeqNr mittels der folgenden Signaturcheckfunktion $\tau(SeqNr, D, s)$:

$$\tau(SeqNr, D, s) \coloneqq \quad 7.5$$

$$s \cdot \prod_{i=0}^{j} b_i = CRC(SeqNr, D) \cdot \left[e \cdot \prod_{i=0}^{j} b_i + \sum_{i=0}^{j} \left((c_i \cdot SL_i) \cdot \prod_{k=0, k \neq i}^{j} b_k \right) \right]$$

$$mit\ \mathrm{e} = \begin{cases} 0, & \#(s) \equiv 1\ mod\ 2 \\ 1, & \#(s) \not\equiv 1\ mod\ 2 \end{cases}$$

Wenn s eine ungerade Anzahl an Signaturen enthält, dann wird der Parameter e auf den Wert 0 gesetzt. Andernfalls wird e auf den Wert 1 gesetzt. Der Korrektheitsbeweis der Signaturprüfung lässt sich einfach führen. Für den Fall, dass die Nachricht von einer ungeraden Knotenzahl (ko-) signiert wurde, erhalten wir die folgende Gleichung:

$$s \cdot \prod_{i=0}^{j} b_i = CRC(SeqNr, D) \cdot \left[e \cdot \prod_{i=0}^{j} b_i + \sum_{i=0}^{j} \left((c_i \cdot SL_i) \cdot \prod_{k=0, k \neq i}^{j} b_k \right) \right]$$

$$s \cdot \prod_{i=0}^{j} b_i = CRC(SeqNr, D) \cdot \left(0 \cdot \prod_{i=0}^{j} b_i + \sum_{i=0}^{j} a_i \cdot SL_i \right) \cdot \prod_{i=0}^{j} b_i \Leftrightarrow$$

$$CRC(SeqNr, D) \cdot \sum_{i=0}^{j} a_i \cdot SL_i \cdot \prod_{i=0}^{j} b_i \Leftrightarrow$$

$$CRC(SeqNr, D) \cdot \sum_{i=0}^{j} (a_i \cdot b_i \cdot SL_i) \cdot \prod_{k=0, k \neq i}^{j} b_k \Leftrightarrow$$

$$CRC(SeqNr, D) \cdot \sum_{i=0}^{j} \left((c_i \cdot SL_i) \prod_{k=0, k \neq i}^{j} b_k \right) \quad Q.E.D.$$

Bei gerader Anzahl an (Ko-) Signierern lässt sich die Korrektheit in analoger Weise zeigen.

7.4 Fehlerinjektionsumgebung

Im Rahmen dieser Arbeit wurde die Fehlererfassungsfähigkeit („fault coverage“) des neuen Signaturverfahrens SigSeam, welches auf den in (Echtle 2005; Echtle und Kimmeskamp 2010) beschriebenen Verfahren zum Teil beruht, untersucht und den von (Echtle 2005; Echtle und Kimmeskamp 2010) beschriebenen Verfahren bzgl. der erkannten Fehler gegenübergestellt.

Zur Durchführung und Auswertung der Fehlerinjektionsexperimente wurde eine auf (Mei-Chen Hsueh et al. 1997) basierende Fehlerinjektionsumgebung (hier kurz: Simulator) in der Programmiersprache Java realisiert. Im Folgenden soll das Gerüst des Simulators, welches zur Auswertung und Vergleich der beiden untersuchten Signaturverfahren verwendet wurde, erläutert werden, wobei die dem Simulator zu Grunde liegende Modellwelt mit ihren Annahmen und Vereinfachungen zunächst kurz dargelegt werden soll.

7.5 Annahmen und Vereinfachungen

Das Zielsystem im Simulator bildet ein verteiltes Rechensystem nach, welches zur Absicherung der Nachrichten digitale Signaturen einsetzt. Bzgl. des verteilten Rechensystems und der im Fokus der Untersuchung (vom verteilten Rechensystem) stehenden Signaturverfahren wurden die folgenden Einschränkungen bzw. Vereinfachungen vorgenommen:

- Im Simulator senden die Knoten des verteilten Rechensystems jeweils einmal. Dabei wird die Sendereihenfolge zentral gesteuert. Die Realisierung der Uhrensynchronisation entfällt im Simulator. Da der Fokus dieser Untersuchung in der Bestimmung der Fehlererfassungsfähigkeit der implementierten Signaturverfahren liegt, reicht es vollkommen aus, das verteilte Rechensystem nur insoweit zu implementieren, als dies für die Untersuchung der Fehlererfassungsfähigkeit der implementierten Signaturverfahren notwendig ist. Demzufolge reicht es aus, nur die im verteilten Rechensystem auftretenden und auf Nachrichten einwirkenden im Simulator adäquat abzubilden (Echtle und Silva 1998, 1998, Mei-Chen Hsueh et al. 1997).
- Nachrichtenausfälle werden im Simulator nicht modelliert, da sie vom Empfänger durch das Ausbleiben einer Nachricht (bzw. durch den fehlenden Eintrag in der Signaturliste) leicht erkannt werden können.
- Die Fehlerinjektionszeitpunkte sind konfigurierbar. Dies dient ausschließlich der Vereinfachung der späteren Auswertung, wie etwa der Untersuchung der Auswirkungen der Kosignaturen auf die Fehlererfassung.
- Die Signaturlängen wurden im Rahmen dieser Untersuchungen auf 8-, 12- und 16-Bit eingeschränkt. Die Untersuchung der Fehlererfassungsfähigkeit für längere Signaturen wurde aus Gründen des Simulationsaufwands nicht durchgeführt.
- Die maximale Länge der Nutzdaten inklusive der Sequenznummer beträgt 512 Bit. Die Fehlererfassungsfähigkeit der implementierten Signaturverfahren wurde für verschiedene Nutzdatenlängen untersucht. Auch hier ist als Grund für die Einschränkung die höhere Simulationsaufwand zu nennen.
- Als Pseudozufallsgenerator zur Generierung gleichverteilter Zahlen wurde wegen seiner guten Eigenschaften der MRG32j3a (L'Ecuyer 1998) verwendet.

Die zur Fehlererkennung untersuchenden Verfahren (Bousbiba 2015b; Echtle 2005; Echtle und Kimmeskamp 2010) wurden bis auf die Einschränkung der Signaturlänge auf maximal 16 Bit ohne Vereinfachungen vollständig implementiert.

7.6 Nachrichtenformat

Im Folgenden wird das in der Simulation verwendete Nachrichtenformat dargestellt, das für die Fehlerinjektionsexperimente gewählt wurde. In beiden untersuchten Signaturverfahren wurde dasselbe Nachrichtenformat verwendet (siehe Abbildung 22).

Das in der Simulation verwendete Nachrichtenformat setzt sich aus den folgenden Teilen zusammen:

- Nutzdaten D: 8 bis 512 Bit und enthält den zu schützenden Nutzdatenbereich (inklusive der Sequenznummer).
- Signatur S: 8 bis 16 Bit und enthält den Signaturwert und wird zur Sicherung des Nachrichteninhalts D verwendet.
- Signaturliste SL: n bzw. 2n Bit, enthält einen Bitvektor, in dem für jeden der n Knoten, die die Nachricht signiert hat, je nach Verfahren ein oder zwei Bits gesetzt werden. Die Signaturliste ist indirekt durch die Signatur S geschützt, da die Werte in der Signaturliste für die Signaturprüfung verwendet werden. Verfälschungen in der Signaturliste sollten daher zu einer inkorrekten Signatur S‘ mit S‘ ≠ S führen. Bei (Bousbiba 2015b) sind pro Signierer zwei Bits und bei (Echtle 2005; Echtle und Kimmeskamp 2010) pro Signierer ein Bit gesetzt. Beim erstgenannten Verfahren wird das zweite Bit verwendet, um Signaturquellen, die bei der Verschmelzung doppelt eingerechnet worden sind, in der Signaturliste korrekt darzustellen.

Alle drei Abschnitte (D, SW, SL) bilden einen kritischen Bereich, dessen Fehler vom Signaturverfahren mit hoher Fehlerabdeckung[6] („fault coverage“) erkannt werden müssen.

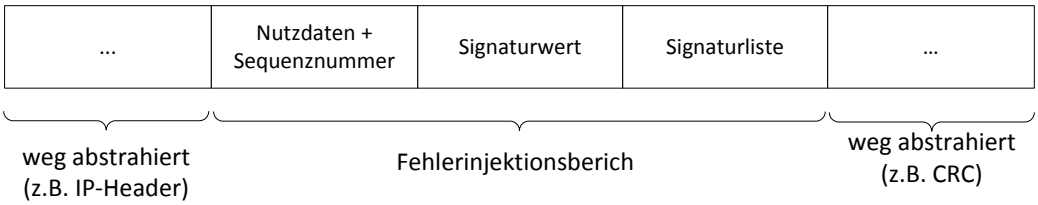


Abbildung 22: Nachrichtenaufbau und Fehlerinjektionsbereiche.

[6]Die Fehlerabdeckung liefert das Verhältnis von nicht erkannten Fehlern zu allen erkennbaren Fehlern. Sie wird hier als Maß für die Güte des Signaturverfahrens verwendet.

7.7 Komponenten des Fehlerinjektionssimulators

Zur Durchführung und Auswertung der Fehlerinjektionsexperimente wurde ein in Java geschriebener Simulator verwendet, der sich aus den folgenden fünf Teilen zusammensetzt (wobei jeder Teil durch eine oder mehrere Java-Klassen abgebildet wird):

- User Interface (Benutzerschnittstelle): Hier werden sämtliche Parameter, die zur Durchführung der Fehlerinjektionsexperimente benötigt werden, eingestellt (z.B. die maximale Nutzdatenlänge oder die Anzahl der fehlerfreien (Ko-) Signierungsläufe[7]).
- Controller (Kontroller): Ist für die Überwachung und Steuerung der Fehlerinjektions-experimente zuständig.
- Fault Injector (Fehlerinjektor): Implementiert sämtliche zu untersuchenden Fehlerarten (siehe Abschnitt 7.8) und führt die vom Controller ausgewählten Fehlerinjektion ins Zielsystem aus.
- Evaluation (Auswertung): Ist für die Auswertung und Speicheprung der Fehlerinjek-tionsexperimente zuständig. Die vom Simulator gewonnenen Rohdaten dienen als Grundlage zur Ermittlung der Fehlererfassungsfähigkeit der zu untersuchenden Signaturverfahren. Die Rohdaten enthalten neben der Anzahl der durchgeführten Simulationsläufe zusätzlich (für jedes durchgeführte Fehlerinjektionsexperiment) die Anzahl der nicht erkannten Fehler.
- Zielsystem (siehe Abschnitt 7.5).

7.8 Parameterraum des Simulators

Mit Hilfe der aus der Simulation gewonnen Daten soll evaluiert werden, in wie weit sich die beiden untersuchten Signaturverfahren bzgl. der Fehlererfassung unterscheiden. Um eine Aussage bzgl. Fehlererfassung treffen zu können, wurden im Rahmen der Untersuchung unterschiedliche Fehlerinjektionstypen sowie kombinierte Fehlertypen (Doppelfehler) injiziert und ausgewertet.

Der Parameterraum zur Konfiguration der Fehlerinjektionsexperimente setzt sich aus den folgenden Parametern zusammen:

[7] Die Fehlerinjektionen werden ausgeführt, wenn die Nachricht eine bestimmte Signaturschranke erreicht hat. Die Signaturschranke legt die Anzahl der Signaturen einer Nachricht fest.

1) Stichprobenumfang: Legt die Anzahl der Fehlerinjektionen für ein Fehlerinjektionsexperiment fest. Wenn z.B. im Simulator als Stichprobenumfang der Wert 100.000 konfiguriert und als Fehlerinjektionsexperiment Bit-Flip ausgewählt wird, dann injiziert der Simulator insgesamt 100.000 Mal den Fehlertyp Bit-Flip ins Zielsystem.
2) Fehlerfreie Kosignierungsläufe: Legt die Anzahl der Signaturen (pro Signatur signiert genau ein fehlerfreier Knoten die Nachricht) einer Nachricht vor einer Fehlerinjektion fest. Die Fehlerinjektion erfolgt erst dann, wenn die Anzahl der Signaturen in der Nachricht dem konfigurierten Wert des Parameters „fehlerfreie Kosignierungsläufe“ entspricht. Vorher werden keine Fehlerinjektionen gestartet.
3) Fehlerinjektionsparameter: Sämtliche zu untersuchenden Fehler werden hier definiert. Dieses Modul bildet die Fehlerbibliothek des Fehlerinjektors ab (siehe ggf. (Mei-Chen Hsueh et al. 1997)).
   a) Bit-Flip-Injektion: Bei einer Bit-Flip-Injektion wird die Information bitweise invertiert. Die Anzahl der zu injizierenden Bit-Flips entspricht der Anzahl der zu ändernden Bitinformationen. Beispielsweise muss der Fehlerinjektor bei einer 2-Bit-Flip-Injektion sicherstellen, dass die zu injizierenden Bit-Flips an zufälligen, aber unterschiedlichen Bitstellen der Nachricht wirken.
   b) 0/1-Burstfehler: Beim Burstfehler treten Bitverfälschungen (im Gegensatz zu Bit-Flips) stets gebündelt auf. D. h. ein Bereich der übertragenen Daten (kurz: Fehlerbreich) enthält die Bitverfälschung, während der übrige Bereich fehlerfrei bleibt. Die Länge eines Bursts legt die Anzahl der zu verfälschenden Bits fest. Vom 0/1-Burstfehler spricht man genau dann, wenn sämtliche Bits im Fehlerbereich auf den Wert 0 bzw. 1 gesetzt werden.
   c) Fehlerhafte Signierer (kann Quell- oder Kosignierer sein): Dieser Fehlertyp tritt auf, wenn die Signaturliste und der Signaturwert nicht übereinstimmen (z.B. wenn der Signatureintrag in der Liste durch ein Bit-Flip „entfernt“ worden ist). Der Fehler wird wie folgt umgesetzt:
      i) Die Signaturliste wird verfälscht (an einer oder mehreren Stellen), z.B. wird ein beliebiger Eintrag in der Signaturliste fehlerbedingt „entfernt“ (auf den Wert 0 gesetzt) oder hinzugefügt (ein Eintrag mit dem Wert 0 wird auf dem Wert 1 gesetzt).
      ii) Die Nachricht wird signiert, aber der Eintrag in der Signaturliste entfällt.
      iii) Die Nachricht wird signiert, jedoch nicht in der Signaturliste vermerkt (siehe 3d.i).
   d) Nutzdatenverfälschung: Die vom Quellknoten ursprünglich versendeten Nutzdaten D entsprechen nicht den empfangenen Nutzdaten D‘ (kurz: $D \neq D'$). Der Inhalt kann

teilweise oder vollständig verfälscht sein. Der Fehlerinjektor ersetzt den Wert von D durch einen beliebigen (mit Hilfe des Pseudozufallsgenerators MRG32j3a erzeugten) Wert gleicher Bitbreite. Alle anderen Bereiche (SW und SL) werden nicht verändert.

e) Doppel-Fehler: Dieser Fehlertyp besteht aus zwei Fehlern (siehe 3.a – 3.i.). Die Fehlertypen die zu einem Doppel-Fehler zusammengesetzt werden, sind stets unterschiedlich.

4) Signaturparameter:
   a) Signaturverfahren: Dieser Parameter wählt das zu untersuchende Signaturverfahren aus. Im vorliegenden Fall stehen nur zwei Verfahren zu Auswahl, nämlich: (Echtle 2005; Echtle und Kimmeskamp 2010) und SigSeam (Bousbiba 2015b).
   b) Bit-Breite der Signatur: Legt die Bit-Breite des Signaturwerts fest. Die Signatur Bit-Breite ist konfigurierbar. Im Rahmen dieser Untersuchung wird die Signatur Bit-Breite (aus Gründen des Simulationsaufwandes) auf 8-Bit, 12-Bit und 16-Bit eingeschränkt.
   c) Bit-Breite des CRC-Polynoms: Die Auswahl der CRC-Polynome ist konfigurierbar. Im Rahmen dieser Untersuchung werden ausschließlich 8 Bit, 12 Bit und 16 Bit breite Polynome (auch hier aus Gründen des Simulationsaufwandes) eingesetzt (wobei jeweils wie bei CRC-Polynomen üblich noch ein führendes 1-Bit hinzukommt).
5) Bit-Breite der Nutzdaten: Legt die Bit-Breite der zu schützenden Nutzdaten fest.

Die Ergebnisse der Fehlerinjektionsexperimente werden vom Simulator in eine CSV- Datei geschrieben, die mit Hilfe eines Tabellenkalkulationsprogramms (hier: Microsoft Excel 2010) weiterverarbeitet wird. Zur Auswertung und zum Vergleich der untersuchten Signaturverfahren wird die vom Simulator aufgezeichnete Fehlererfassung als Kenngröße verwendet. Sie enthält den Anteil der Fälle (als Prozentangabe), in denen das betreffende Signaturverfahren einen injizierten Fehler nicht erkannt hat. Nehmen wir als Beispiel den Fehlertyp Bit-Flip. Wenn von 100.000 injizierten Bit-Flips (Für dieses Beispiel ist die Anzahl der Bit-Flips irrelevant, dies gilt aber nicht für die spätere Auswertung) 200 nicht erkannt wurden, so weist die Fehlererfassung einen Wert von

$$100\% \cdot \left(1 - \frac{200}{100.000}\right) = 100\% \cdot (1 - 0{,}0002) = 99{,}98\%$$ auf.

7.9 Fehlerinjektionsexperimente und Ergebnis

In diesem Kapitel werden die verschiedenen durchgeführten Fehlerinjektionsexperimente zur Bewertung und zum Vergleich der beiden Verfahren (Bousbiba 2015b; Echtle 2005; Echtle und Kimmeskamp 2010) präsentiert. Zu jedem Fehlerinjektionsexperiment werden sowohl das Fehlerbild als auch die ausgewerteten Simulationsergebnisse dargelegt. In den nachfolgenden Tabellen sind die Anzahl der unerkannten Fehler (kurz: 1 – Fehlererfassung) für die jeweils untersuchten Fehlerinjektionsexperimente dargestellt. Genauer gesagt enthält jeder Zelleneintrag das 95%-Konfidenzintervall der nicht erkannten Fehler für die betreffenden Simulationsläufe (siehe Zeile „Simulations-Läufe“ in Tabelle 3). Die Anzahl der Simulationsläufe wurde bis auf den fehlerfreien Fall auf maximal 10.000.000 nach oben und 10.000 nach unten beschränkt. Die in diesem Kapitel präsentierten Ergebnisse enthalten nur einen Teil der tatsächlich durchgeführten Fehlerinjektionsexperimente und die Ergebnisse werden in den Abschnitten 7.9.3 bis 7.9.8 näher diskutiert. Sämtliche ausgewerteten Fehlerinjektionsexperimente sind jedoch der Vollständigkeit halber im Anhang dieser Arbeit aufgeführt.

7.9.1 Glossar zu den Ergebnistabellen

Im Folgenden werden zunächst die Informationen, die in den jeweiligen Zeilen und Spalten der Tabellen (siehe Tabelle 3 – Tabelle 33) eingetragen sind, näher erläutert. Im Anschluss daran werden die Ergebnisse der einzelnen Auswertungen dargelegt.

<table>
<tr><td></td><td colspan="2">Kopfbereich der Tabelle:
Stellt Informationen bereit, die für die gesamte Tabelle gelten
Quelleverweise, CRC & Signatur –
1) Enthält Informationen über das verwendete Signaturverfahren (Quellenangabe des Signaturverfahrens).
2) Liefert Auskunft über die verwendete Signaturschlüssellänge und die Länge des CRC-Prüfpolynoms.
3) Gibt die Bedingung an, wann eine Fehlerinjektion gestartet wird. Wir verwenden hierfür den Begriff Signaturfehlerinjektionsschranke SFT. Der Wert von SFT gibt an, wie viele Knoten zuvor die Nachricht signiert haben müssen, bevor eine Fehlerinjektion gestartet wird. Nehmen wir als Beispiel den Wert SFT = 5 an. Für den Wert SFT = 5 würde der Fehlerinjektor eine Fehlerinjektion erst dann ausführen, wenn die Data-Nachricht 5 Signaturen enthält bzw. von genau 5 Knoten signiert worden ist.
4) Ggf. werden hier sämtliche Fehlertypen aufgelistet, deren Parameter auf einen festen Wert vorkonfiguriert wurden.</td></tr>
<tr><td rowspan="2">DATA</td><td colspan="2">Parameter, der in der Tabelle variiert wurde:
Jeder Parameterwert wird in einer eigenen Spalte dargestellt
Dieser Bereich liefert detaillierte Informationen über den Parameter, der meistens ein Fehlertyp ist. Nehmen wir als Beispiel den Fehlertyp Bit-Flips. Für diesen Fehlertyp werden die einzelnen ausgewerteten Bit-Flip-Längen spaltenweise aufgelistet (siehe ggf. Tabelle 10 und Tabelle 11)</td></tr>
<tr><td>Parameterwert 1</td><td>Parameterwert 2, usw.</td></tr>
<tr><td>Nutzdatenlänge in Bits:
Listet zeilenweise die für die Fehlerinjektion untersuchten Nutzdatenlängen auf.</td><td>Jeder Zelleneintrag enthält das 95%-Konfidenz-intervall der nicht erkannten Fehler. Das 95%-Konfidenzintervall wird in eckigen Klammern angegeben. Das Zeichen „%“ wird in dem je-weiligen Zelleneintrag nicht explizit angege-ben. Der Eintrag „82“ bedeutet beispielsweise 0,82 = 82%.</td><td>…</td></tr>
<tr><td>Simulationsläufe</td><td colspan="2">Anzahl der ausgeführten Simulationsläufe:
Listet die Anzahl der durchgeführten Simulationsläufe pro Nutzdatenlänge und Wert des variierten Parameters auf.</td></tr>
</table>

7.9.2 Ergebnis im fehlerfreien Fall

Im fehlerfreien Fall werden keine der in Abschnitt 7.8 beschriebenen Fehlerinjektionen ausgeführt. Wie in Tabelle 3 ersichtlich, liefert der fehlerfreie Betrieb keine unvorhergesehenen Überraschungen. Alle vom Simulator generierten fehlerfreien Nachrichten wurden von den jeweiligen Signaturverfahren auch als solchen erkannt.

Tabelle 8: Ergebnisse ohne durchgeführte Fehlerinjektionen (fehlerfreie Simulationsläufe) (Bousbiba 2015b)

	(Bousbiba 2015b), CRC = 8 – 16 Bit Signatur = 8 – 16 Bit fehlerfreier Betrieb
DATA	
8 Bit	[0, 0]
16 Bit	[0, 0]
32 Bit	[0, 0]
64 Bit	[0, 0]
128 Bit	[0, 0]
256 Bit	[0, 0]
512 Bit	[0, 0]
Simulations-Läufe	100.000

Tabelle 9: Ergebnisse ohne durchgeführte Fehlerinjektionen (fehlerfreie Simulationsläufe) (Echtle 2005; Echtle und Kimmeskamp 2010).

	(Echtle 2005; Echtle und Kimmeskamp 2010), CRC = 8 – 16 Bit Signatur = 8 – 16 Bit fehlerfreier Betrieb
DATA	
8 Bit	[0, 0]
16 Bit	[0, 0]
32 Bit	[0, 0]
64 Bit	[0, 0]
128 Bit	[0, 0]
256 Bit	[0, 0]
512 Bit	[0, 0]
Simulations-Läufe	100.000

Die in den jeweiligen Zellen in Tabelle 3 und Tabelle 9 eingetragenen Werte, sind wie folgt zu deuten: In keinem der durchgeführten Simulationsläufe konnte eines der beiden Signaturverfahren einen Fehler feststellen, wie es im fehlerfreien Fall zu erwarten war.

7.9.3 Ergebnis der Untersuchung von Bit-Flips

Es wird die Fehlererfassung der implementierten Signaturverfahren bei Bit-Flip-Injektion in den Bereichen D, S und SL einer Nachricht untersucht. Hierzu wurden abhängig von der Anzahl der zu injizierenden Bit-Flips bis zu 10.000.000 Fehlerinjektionen ausgeführt (siehe unteren Zeileneintrag „Simulationsläufe“ in Tabelle 10). In den nachfolgenden Tabellen werden die Ergebnisse präsentiert, bei denen der CRC und der Signaturwert dieselbe Bitbreite aufweisen.

Tabelle 10: Anzahl der nicht erkannten Bit Flip Injektionen bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Bousbiba 2015b).

	(Bousbiba 2015b), CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 10				
	Bit-Flips in Bits				
DATA	1	2	3	4	5
8 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0}{10^4}, \frac{0.5}{10^4}\right]$	$\left[\frac{0.45}{10^4}, \frac{0.57}{10^4}\right]$	$\left[\frac{0.41}{10^4}, \frac{0.45}{10^4}\right]$	$\left[\frac{0.37}{10^4}, \frac{0.41}{10^4}\right]$
16 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0.40}{10^4}, \frac{1.20}{10^4}\right]$	$\left[\frac{0.54}{10^4}, \frac{0.66}{10^4}\right]$	$\left[\frac{0.41}{10^4}, \frac{0.45}{10^4}\right]$	$\left[\frac{0.36}{10^4}, \frac{0.40}{10^4}\right]$
32 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0.51}{10^4}, \frac{1.21}{10^4}\right]$	$\left[\frac{0.36}{10^4}, \frac{0.44}{10^4}\right]$	$\left[\frac{0.30}{10^4}, \frac{0.34}{10^4}\right]$	$\left[\frac{0.27}{10^4}, \frac{0.30}{10^4}\right]$
64 Bit	$\left[\frac{0}{10^4}, \frac{2.00}{10^4}\right]$	$\left[\frac{0.17}{10^4}, \frac{0.5}{10^4}\right]$	$\left[\frac{0.24}{10^4}, \frac{0.28}{10^4}\right]$	$\left[\frac{0.21}{10^4}, \frac{0.24}{10^4}\right]$	$\left[\frac{0.18}{10^4}, \frac{0.20}{10^4}\right]$
128 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0.14}{10^4}, \frac{0.32}{10^4}\right]$	$\left[\frac{0.16}{10^4}, \frac{0.19}{10^4}\right]$	$\left[\frac{0.15}{10^4}, \frac{0.18}{10^4}\right]$	$\left[\frac{0.14}{10^4}, \frac{0.17}{10^4}\right]$
256 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0.12}{10^4}, \frac{0.21}{10^4}\right]$	$\left[\frac{0.15}{10^4}, \frac{0.17}{10^4}\right]$	$\left[\frac{0.15}{10^4}, \frac{0.18}{10^4}\right]$	$\left[\frac{0.12}{10^4}, \frac{0.14}{10^4}\right]$
512 Bit	$\left[\frac{0}{10^4}, \frac{2.00}{10^4}\right]$	$\left[\frac{0.15}{10^4}, \frac{0.20}{10^4}\right]$	$\left[\frac{0.16}{10^4}, \frac{0.18}{10^4}\right]$	$\left[\frac{0.14}{10^4}, \frac{0.17}{10^4}\right]$	$\left[\frac{0.15}{10^4}, \frac{0.17}{10^4}\right]$
Simulations-Läufe	10^4	10^5	10^7	10^7	10^7

In Tabelle 10 und Tabelle 11 sind die Anzahl der unerkannten Fehler für verschiedene Datenlängen und Bit Flip Injektionen aufgeführt. Beide Signaturverfahren weisen bis zu einer Datenlänge von 64-bit eine ähnliche Fehlererfassung auf. Mit wachsender Datenlänge (> 64-Bit) weist das Signaturverfahren (Echtle 2005; Echtle und Kimmeskamp 2010) im Vergleich zu (Bousbiba 2015b) jedoch eine um Faktor 5 bessere Fehlererfassung auf (d.h. der Anteil der

unerkannten Fehler ist nur 1/5 so groß). Insgesamt ist die Fehlererfassung bei beiden Verfahren hoch.

Ein möglicher Grund für die schlechtere Fehlererfassung von (Bousbiba 2015b) liegt in der längeren Signaturliste (siehe Abschnitt 7.6). Anders als in (Echtle 2005; Echtle und Kimmeskamp 2010) verändern die Bit-Flip-Injektionen im Bereich der Signaturliste (SL) beim Signaturverfahren (Bousbiba 2015b) nicht zwangsläufig die Signaturquellen[8] (siehe Abschnitt 7.6). D. h. die Information über die Signierquellen kann trotz Bit-Flip-Injektion in der Signaturliste SL beim Signaturverfahren (Bousbiba 2015b) erhalten bleiben (genaueres siehe Abschnitt 7.6).

Tabelle 11: Anteil der nicht erkannten Bit Flip Injektion bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Echtle 2005; Echtle und Kimmeskamp 2010).

	(Echtle 2005; Echtle und Kimmeskamp 2010), CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 10				
	Bit-Flips in Bits				
DATA	1	2	3	4	5
8 Bit	$\left[\frac{1.27}{10^4}, \frac{4.73}{10^4}\right]$	$\left[\frac{0.69}{10^4}, \frac{1.81}{10^4}\right]$	$\left[\frac{0.54}{10^4}, \frac{0.67}{10^4}\right]$	$\left[\frac{0.50}{10^4}, \frac{0.55}{10^4}\right]$	$\left[\frac{0.50}{10^4}, \frac{0.43}{10^4}\right]$
16 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0.12}{10^4}, \frac{0.68}{10^4}\right]$	$\left[\frac{0.61}{10^4}, \frac{0.73}{10^4}\right]$	$\left[\frac{0.45}{10^4}, \frac{0.49}{10^4}\right]$	$\left[\frac{0.27}{10^4}, \frac{0.30}{10^4}\right]$
32 Bit	$\left[\frac{0}{10^4}, \frac{2.00}{10^4}\right]$	$\left[\frac{0.39}{10^4}, \frac{1.03}{10^4}\right]$	$\left[\frac{0.35}{10^4}, \frac{0.42}{10^4}\right]$	$\left[\frac{0.38}{10^4}, \frac{0.42}{10^4}\right]$	$\left[\frac{0.19}{10^4}, \frac{0.22}{10^4}\right]$
64 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0.17}{10^4}, \frac{0.5}{10^4}\right]$	$\left[\frac{0.21}{10^4}, \frac{0.25}{10^4}\right]$	$\left[\frac{0.33}{10^4}, \frac{0.37}{10^4}\right]$	$\left[\frac{0.13}{10^4}, \frac{0.16}{10^4}\right]$
128 Bit	$\left[\frac{0}{10^4}, \frac{2.00}{10^4}\right]$	$\left[\frac{0.05}{10^4}, \frac{0.18}{10^4}\right]$	$\left[\frac{0.11}{10^4}, \frac{0.14}{10^4}\right]$	$\left[\frac{0.26}{10^4}, \frac{0.29}{10^4}\right]$	$\left[\frac{0.10}{10^4}, \frac{0.12}{10^4}\right]$
256 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0.04}{10^4}, \frac{0.09}{10^4}\right]$	$\left[\frac{0.06}{10^4}, \frac{0.07}{10^4}\right]$	$\left[\frac{0.21}{10^4}, \frac{0.24}{10^4}\right]$	$\left[\frac{0.07}{10^4}, \frac{0.08}{10^4}\right]$
512 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0.01}{10^4}, \frac{0.03}{10^4}\right]$	$\left[\frac{0.02}{10^4}, \frac{0.04}{10^4}\right]$	$\left[\frac{0.22}{10^4}, \frac{0.25}{10^4}\right]$	$\left[\frac{0.03}{10^4}, \frac{0.05}{10^4}\right]$
Simulations-Läufe	10^4	10^5	10^7	10^7	10^7

[8] Die Knoten, die die Nachricht signiert bzw. kosigniert haben.

7.9.4 Ergebnis der Untersuchung von Bursts

Es wird die Fehlererfassung der implementierten Signaturverfahren bei Injektion eines Bursts im Bereich D, S und SL untersucht. Bei Burst-Fehlerinjektionen führt der Simulator unabhängig von der Burstlänge jeweils 10.000.000 Fehlerinjektionsexperimente aus. Die Ergebnisse der Auswertung sind in Tabelle 12 und Tabelle 13 angegeben.

Tabelle 12: Anteil der nicht erkannten Burst Injektion bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Bousbiba 2015b).

	(Bousbiba 2015b), CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 5				
	Burstlänge in Bits				
DATA	8	12	16	20	24
8 Bit	$\left[\frac{7.60}{10^5}, \frac{8.17}{10^5}\right]$	$\left[\frac{6.78}{10^5}, \frac{7.32}{10^5}\right]$	$\left[\frac{6.42}{10^5}, \frac{6.94}{10^5}\right]$	$\left[\frac{6.08}{10^5}, \frac{6.58}{10^5}\right]$	$\left[\frac{6.22}{10^5}, \frac{6.73}{10^5}\right]$
16 Bit	$\left[\frac{10.51}{10^5}, \frac{11.17}{10^5}\right]$	$\left[\frac{9.41}{10^5}, \frac{10.32}{10^5}\right]$	$\left[\frac{8.46}{10^5}, \frac{9.06}{10^5}\right]$	$\left[\frac{9.09}{10^5}, \frac{8.67}{10^5}\right]$	$\left[\frac{7.79}{10^5}, \frac{8.36}{10^5}\right]$
32 Bit	$\left[\frac{8.79}{10^5}, \frac{9.40}{10^5}\right]$	$\left[\frac{7.79}{10^5}, \frac{8.36}{10^5}\right]$	$\left[\frac{6.43}{10^5}, \frac{6.95}{10^5}\right]$	$\left[\frac{6.76}{10^5}, \frac{7.29}{10^5}\right]$	$\left[\frac{6.56}{10^5}, \frac{7.08}{10^5}\right]$
64 Bit	$\left[\frac{5.88}{10^5}, \frac{6.38}{10^5}\right]$	$\left[\frac{5.24}{10^5}, \frac{5.70}{10^5}\right]$	$\left[\frac{4.42}{10^5}, \frac{4.86}{10^5}\right]$	$\left[\frac{5.62}{10^5}, \frac{6.10}{10^5}\right]$	$\left[\frac{4.97}{10^5}, \frac{5.43}{10^5}\right]$
128 Bit	$\left[\frac{3.66}{10^5}, \frac{4.06}{10^5}\right]$	$\left[\frac{2.78}{10^5}, \frac{3.12}{10^5}\right]$	$\left[\frac{2.82}{10^5}, \frac{3.16}{10^5}\right]$	$\left[\frac{3.68}{10^5}, \frac{4.08}{10^5}\right]$	$\left[\frac{3.81}{10^5}, \frac{4.21}{10^5}\right]$
256 Bit	$\left[\frac{2.26}{10^5}, \frac{2.58}{10^5}\right]$	$\left[\frac{1.85}{10^5}, \frac{2.13}{10^5}\right]$	$\left[\frac{1.67}{10^5}, \frac{1.93}{10^5}\right]$	$\left[\frac{2.53}{10^5}, \frac{2.86}{10^5}\right]$	$\left[\frac{2.47}{10^5}, \frac{2.79}{10^5}\right]$
512 Bit	$\left[\frac{1.19}{10^5}, \frac{1.42}{10^5}\right]$	$\left[\frac{0.81}{10^5}, \frac{1.00}{10^5}\right]$	$\left[\frac{0.90}{10^5}, \frac{1.10}{10^5}\right]$	$\left[\frac{1.95}{10^5}, \frac{2.23}{10^5}\right]$	$\left[\frac{1.97}{10^5}, \frac{2.27}{10^5}\right]$
Simulations-Läufe	10^7	10^7	10^7	10^7	10^7

In Tabelle 12 und Tabelle 13 sind die Anzahl der unerkannten Fehler für verschiedene Daten- und Burstlängen aufgeführt. Insgesamt weist das Signaturverfahren (Echtle 2005; Echtle und Kimmeskamp 2010) eine um Faktor 2 bis Faktor 3 bessere Fehlererfassung auf. Insgesamt ist die Fehlererfassung bei beiden Verfahren sehr hoch.

Die schlechtere Fehlererfassung von (Bousbiba 2015b) lässt sich auf ähnliche Weise wie zuvor genannt begründen. Die Vermutung liegt nahe, dass Verfälschungen im Bereich der Signaturliste generell bei (Bousbiba 2015b) im Vergleich zu (Echtle 2005; Echtle und Kimmeskamp 2010) schlechter erkannt werden und folglich zu einer insgesamt schlechteren Fehlererfassung führen.

Tabelle 13: Anteil der nicht erkannten Burst Injektion bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Echtle 2005; Echtle und Kimmeskamp 2010).

	(Echtle 2005; Echtle und Kimmeskamp 2010), CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 5				
	Burstlänge in Bits				
DATA	8	12	16	20	24
8 Bit	$\left[\frac{4.20}{10^5}, \frac{4.62}{10^5}\right]$	$\left[\frac{3.98}{10^5}, \frac{4.39}{10^5}\right]$	$\left[\frac{3.24}{10^5}, \frac{3.62}{10^5}\right]$	$\left[\frac{2.37}{10^5}, \frac{2.69}{10^5}\right]$	$\left[\frac{1.71}{10^5}, \frac{1.99}{10^5}\right]$
16 Bit	$\left[\frac{6.22}{10^5}, \frac{6.72}{10^5}\right]$	$\left[\frac{4.97}{10^5}, \frac{5.43}{10^5}\right]$	$\left[\frac{3.92}{10^5}, \frac{4.33}{10^5}\right]$	$\left[\frac{3.68}{10^5}, \frac{4.08}{10^5}\right]$	$\left[\frac{2.40}{10^5}, \frac{2.72}{10^5}\right]$
32 Bit	$\left[\frac{3.84}{10^5}, \frac{4.24}{10^5}\right]$	$\left[\frac{3.85}{10^5}, \frac{4.25}{10^5}\right]$	$\left[\frac{2.34}{10^5}, \frac{2.66}{10^5}\right]$	$\left[\frac{2.94}{10^5}, \frac{3.29}{10^5}\right]$	$\left[\frac{1.96}{10^5}, \frac{2.26}{10^5}\right]$
64 Bit	$\left[\frac{2.77}{10^5}, \frac{3.11}{10^5}\right]$	$\left[\frac{2.45}{10^5}, \frac{2.77}{10^5}\right]$	$\left[\frac{1.83}{10^5}, \frac{2.11}{10^5}\right]$	$\left[\frac{2.26}{10^5}, \frac{2.58}{10^5}\right]$	$\left[\frac{1.18}{10^5}, \frac{2.06}{10^5}\right]$
128 Bit	$\left[\frac{1.73}{10^5}, \frac{2.01}{10^5}\right]$	$\left[\frac{1.35}{10^5}, \frac{1.59}{10^5}\right]$	$\left[\frac{1.25}{10^5}, \frac{1.49}{10^5}\right]$	$\left[\frac{1.63}{10^5}, \frac{1.89}{10^5}\right]$	$\left[\frac{1.59}{10^5}, \frac{1.85}{10^5}\right]$
256 Bit	$\left[\frac{0.83}{10^5}, \frac{1.03}{10^5}\right]$	$\left[\frac{0.69}{10^5}, \frac{0.87}{10^5}\right]$	$\left[\frac{0.54}{10^5}, \frac{0.69}{10^5}\right]$	$\left[\frac{1.52}{10^5}, \frac{1.78}{10^5}\right]$	$\left[\frac{1.48}{10^5}, \frac{1.74}{10^5}\right]$
512 Bit	$\left[\frac{0.44}{10^5}, \frac{0.58}{10^5}\right]$	$\left[\frac{0.43}{10^5}, \frac{0.57}{10^5}\right]$	$\left[\frac{0.25}{10^5}, \frac{0.37}{10^5}\right]$	$\left[\frac{1.41}{10^5}, \frac{1.65}{10^5}\right]$	$\left[\frac{1.42}{10^5}, \frac{1.66}{10^5}\right]$
Simulations-Läufe	10^7	10^7	10^7	10^7	10^7

7.9.5 Ergebnis der Untersuchung von Fehlen bezüglich der Signaturquelle

Es wird die Fehlererfassung der implementierten Signaturverfahren im Falle einer fehlerhaften Signaturquelle untersucht. Die Ergebnisse der Auswertung sind in den nachfolgenden Tabelle 14 und Tabelle 15 aufgeführt.

In der Tabelle 14 und Tabelle 15 sind die Anzahl der unerkannten Fehler für den Fehlertyp „fehlerhafter Quellknoten“ für verschiedene Datenlängen aufgeführt. Insgesamt weisen die von beiden Verfahren generierten Signaturen eine ähnlich gute Fehlererfassung auf. Der Fehlertyp (vergleich auch hierzu Abschnitt 7.8) falsche Quelle wird bei beiden Signaturverfahren gleich gut erkannt.

Tabelle 14: Anteil der nicht erkannten „fehlerhafte Quelle“-Fehlerinjektion bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Bousbiba 2015b).

	(Bousbiba 2015b), CRC = 16 Bit Signatur = 16 Bit fh Quelle		
	Signaturfehlerinjektionsschranke SFT		
DATA	5	10	15
16 Bit	$\left[\frac{2.22}{10^4}, \frac{2.52}{10^4}\right]$	$\left[\frac{2.35}{10^4}, \frac{2.67}{10^4}\right]$	$\left[\frac{2.40}{10^4}, \frac{2.72}{10^4}\right]$
32 Bit	$\left[\frac{2.22}{10^4}, \frac{2.52}{10^4}\right]$	$\left[\frac{2.23}{10^4}, \frac{2.53}{10^4}\right]$	$\left[\frac{2.23}{10^4}, \frac{2.53}{10^4}\right]$
64 Bit	$\left[\frac{2.34}{10^4}, \frac{2.66}{10^4}\right]$	$\left[\frac{2.30}{10^4}, \frac{2.62}{10^4}\right]$	$\left[\frac{2.31}{10^4}, \frac{2.62}{10^4}\right]$
128 Bit	$\left[\frac{2.37}{10^4}, \frac{2.69}{10^4}\right]$	$\left[\frac{2.58}{10^4}, \frac{2.91}{10^4}\right]$	$\left[\frac{2.39}{10^4}, \frac{2.71}{10^4}\right]$
256 Bit	$\left[\frac{2.31}{10^4}, \frac{2.63}{10^4}\right]$	$\left[\frac{2.56}{10^4}, \frac{2.89}{10^4}\right]$	$\left[\frac{2.14}{10^4}, \frac{2.44}{10^4}\right]$
512 Bit	$\left[\frac{2.35}{10^4}, \frac{2.67}{10^4}\right]$	$\left[\frac{2.06}{10^4}, \frac{2.36}{10^4}\right]$	$\left[\frac{2.59}{10^4}, \frac{2.93}{10^4}\right]$
Simulations-Läufe	10^6	10^6	10^6

Tabelle 15: Anteil der nicht erkannten „fehlerhafte Quelle“-Fehlerinjektion bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Echtle 2005; Echtle und Kimmeskamp 2010).

	(Echtle 2005; Echtle und Kimmeskamp 2010), CRC = 16 Bit Signatur = 16 Bit fh Quelle		
	Signaturfehlerinjektionsschranke SFT		
DATA	5	10	15
16 Bit	$\left[\frac{2.27}{10^4}, \frac{2.59}{10^4}\right]$	$\left[\frac{2.33}{10^4}, \frac{2.65}{10^4}\right]$	$\left[\frac{2.72}{10^4}, \frac{3.06}{10^4}\right]$
32 Bit	$\left[\frac{2.48}{10^4}, \frac{2.80}{10^4}\right]$	$\left[\frac{2.06}{10^4}, \frac{2.36}{10^4}\right]$	$\left[\frac{1.83}{10^4}, \frac{2.11}{10^4}\right]$
64 Bit	$\left[\frac{2.28}{10^4}, \frac{2.59}{10^4}\right]$	$\left[\frac{2.06}{10^4}, \frac{2.36}{10^4}\right]$	$\left[\frac{2.28}{10^4}, \frac{2.59}{10^4}\right]$
128 Bit	$\left[\frac{2.35}{10^4}, \frac{2.67}{10^4}\right]$	$\left[\frac{2.14}{10^4}, \frac{2.44}{10^4}\right]$	$\left[\frac{2.24}{10^4}, \frac{2.54}{10^4}\right]$
256 Bit	$\left[\frac{2.36}{10^4}, \frac{2.68}{10^4}\right]$	$\left[\frac{2.39}{10^4}, \frac{2.71}{10^4}\right]$	$\left[\frac{2.39}{10^4}, \frac{2.71}{10^4}\right]$
512 Bit	$\left[\frac{2.10}{10^4}, \frac{2.40}{10^4}\right]$	$\left[\frac{2.34}{10^4}, \frac{2.66}{10^4}\right]$	$\left[\frac{2.29}{10^4}, \frac{2.61}{10^4}\right]$
Simulations-Läufe	10^6	10^6	10^6

7.9.6 Ergebnis der Untersuchung von Fehlern bezüglich der Kosignierer

Es wird die Fehlererfassung der implementierten Signaturverfahren für den Fehlerinjektionstyp „fehlerhafter Kosignierer“ in diesem Abschnitt untersucht. Die Ergebnisse der Auswertung sind in den Tabelle 16 und Tabelle 17 aufgeführt.

Tabelle 16: Anteil der nicht erkannten „fehlerhafter Kosignierer“-Fehlerinjektion bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Bousbiba 2015b).

	(Bousbiba 2015b) CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 5				
	Anzahl der fehlerhaften Kosignierer				
DATA	1	2	3	4	5
8 Bit	$\left[\frac{0.81}{10^4}, \frac{1.00}{10^4}\right]$	$\left[\frac{0.56}{10^4}, \frac{0.72}{10^4}\right]$	$\left[\frac{0.61}{10^4}, \frac{0.77}{10^4}\right]$	$\left[\frac{0.71}{10^4}, \frac{0.89}{10^4}\right]$	$\left[\frac{0.49}{10^4}, \frac{0.65}{10^4}\right]$
16 Bit	$\left[\frac{0.93}{10^4}, \frac{1.13}{10^4}\right]$	$\left[\frac{1.18}{10^4}, \frac{1.40}{10^4}\right]$	$\left[\frac{1.11}{10^4}, \frac{1.33}{10^4}\right]$	$\left[\frac{1.19}{10^4}, \frac{1.41}{10^4}\right]$	$\left[\frac{1.02}{10^4}, \frac{1.24}{10^4}\right]$
32 Bit	$\left[\frac{1.22}{10^4}, \frac{1.46}{10^4}\right]$	$\left[\frac{1.07}{10^4}, \frac{1.29}{10^4}\right]$	$\left[\frac{1.36}{10^4}, \frac{1.60}{10^4}\right]$	$\left[\frac{0.95}{10^4}, \frac{1.15}{10^4}\right]$	$\left[\frac{1.31}{10^4}, \frac{1.55}{10^4}\right]$
64 Bit	$\left[\frac{0.87}{10^4}, \frac{1.07}{10^4}\right]$	$\left[\frac{0.90}{10^4}, \frac{1.10}{10^4}\right]$	$\left[\frac{1.04}{10^4}, \frac{1.26}{10^4}\right]$	$\left[\frac{0.93}{10^4}, \frac{1.13}{10^4}\right]$	$\left[\frac{1.07}{10^4}, \frac{1.29}{10^4}\right]$
128 Bit	$\left[\frac{0.87}{10^4}, \frac{1.07}{10^4}\right]$	$\left[\frac{0.96}{10^4}, \frac{1.16}{10^4}\right]$	$\left[\frac{1.05}{10^4}, \frac{1.27}{10^4}\right]$	$\left[\frac{1.12}{10^4}, \frac{1.35}{10^4}\right]$	$\left[\frac{1.18}{10^4}, \frac{1.40}{10^4}\right]$
256 Bit	$\left[\frac{1.05}{10^4}, \frac{1.27}{10^4}\right]$	$\left[\frac{1.00}{10^4}, \frac{1.22}{10^4}\right]$	$\left[\frac{1.17}{10^4}, \frac{1.39}{10^4}\right]$	$\left[\frac{1.12}{10^4}, \frac{1.34}{10^4}\right]$	$\left[\frac{1.02}{10^4}, \frac{1.24}{10^4}\right]$
512 Bit	$\left[\frac{0.61}{10^4}, \frac{0.77}{10^4}\right]$	$\left[\frac{1.04}{10^4}, \frac{1.26}{10^4}\right]$	$\left[\frac{1.22}{10^4}, \frac{1.46}{10^4}\right]$	$\left[\frac{1.22}{10^4}, \frac{1.46}{10^4}\right]$	$\left[\frac{1.19}{10^4}, \frac{1.41}{10^4}\right]$
Simulations-Läufe	10^7	10^7	10^7	10^7	10^7

In der Tabelle 16 und Tabelle 17 sind die Anzahl der unerkannten Fehler für den Fehlertyp „fehlerhafter Kosignierer“ für verschiedene Datenlängen aufgeführt. Gut zu erkennen ist (siehe Tabelle 16 und Tabelle 17), dass mit steigender Anzahl an fehlerhaften Kosignierungen das Signaturverfahren (Echtle 2005; Echtle und Kimmeskamp 2010) eine um Faktor 3 bis 4 bessere Fehlererfassung aufweist. Bei nur wenigen fehlerhaften Kosignierern schneidet jedoch das neue Verfahren (Bousbiba 2015b) besser ab.

Tabelle 17: Anteil der nicht erkannten „fehlerhafter Kosignierer“-Fehlerinjektion bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Echtle 2005; Echtle und Kimmeskamp 2010).

	(Echtle 2005; Echtle und Kimmeskamp 2010) CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 5				
	Anzahl der fehlerhaften Kosignierer				
DATA	1	2	3	4	5
8 Bit	$\left[\frac{1.58}{10^4}, \frac{1.66}{10^4}\right]$	$\left[\frac{0.70}{10^4}, \frac{0.76}{10^4}\right]$	$\left[\frac{0.44}{10^4}, \frac{0.48}{10^4}\right]$	$\left[\frac{0.28}{10^4}, \frac{0.31}{10^4}\right]$	$\left[\frac{0.17}{10^4}, \frac{0.20}{10^4}\right]$
16 Bit	$\left[\frac{2.57}{10^4}, \frac{2.68}{10^4}\right]$	$\left[\frac{1.18}{10^4}, \frac{1.25}{10^4}\right]$	$\left[\frac{0.73}{10^4}, \frac{0.79}{10^4}\right]$	$\left[\frac{0.44}{10^4}, \frac{0.48}{10^4}\right]$	$\left[\frac{0.27}{10^4}, \frac{0.31}{10^4}\right]$
32 Bit	$\left[\frac{2.50}{10^4}, \frac{2.61}{10^4}\right]$	$\left[\frac{1.21}{10^4}, \frac{1.28}{10^4}\right]$	$\left[\frac{0.76}{10^4}, \frac{0.81}{10^4}\right]$	$\left[\frac{0.44}{10^4}, \frac{0.49}{10^4}\right]$	$\left[\frac{0.29}{10^4}, \frac{0.33}{10^4}\right]$
64 Bit	$\left[\frac{2.49}{10^4}, \frac{2.59}{10^4}\right]$	$\left[\frac{1.17}{10^4}, \frac{1.24}{10^4}\right]$	$\left[\frac{0.72}{10^4}, \frac{0.77}{10^4}\right]$	$\left[\frac{0.44}{10^4}, \frac{0.49}{10^4}\right]$	$\left[\frac{0.27}{10^4}, \frac{0.31}{10^4}\right]$
128 Bit	$\left[\frac{2.59}{10^4}, \frac{2.69}{10^4}\right]$	$\left[\frac{1.18}{10^4}, \frac{1.25}{10^4}\right]$	$\left[\frac{0.69}{10^4}, \frac{0.74}{10^4}\right]$	$\left[\frac{0.43}{10^4}, \frac{0.47}{10^4}\right]$	$\left[\frac{0.28}{10^4}, \frac{0.31}{10^4}\right]$
256 Bit	$\left[\frac{2.52}{10^4}, \frac{2.63}{10^4}\right]$	$\left[\frac{1.17}{10^4}, \frac{1.24}{10^4}\right]$	$\left[\frac{0.75}{10^4}, \frac{0.80}{10^4}\right]$	$\left[\frac{0.41}{10^4}, \frac{0.45}{10^4}\right]$	$\left[\frac{0.26}{10^4}, \frac{0.29}{10^4}\right]$
512 Bit	$\left[\frac{2.46}{10^4}, \frac{2.56}{10^4}\right]$	$\left[\frac{1.18}{10^4}, \frac{1.25}{10^4}\right]$	$\left[\frac{0.72}{10^4}, \frac{0.77}{10^4}\right]$	$\left[\frac{0.42}{10^4}, \frac{0.47}{10^4}\right]$	$\left[\frac{0.29}{10^4}, \frac{0.33}{10^4}\right]$
Simulations-Läufe	10^7	10^7	10^7	10^7	10^7

7.9.7 Ergebnis der Untersuchung von Nutzdatenverfälschungen

Es wird die Fehlererfassung der implementierten Signaturverfahren für den Fehlerinjektionstyp Nutzdatenverfälschung untersucht. Beim diesem Fehlertyp ersetzt der Fehlerinjektor die Nutzdaten durch ein zufälliges Bitmuster gleicher Bitbreite (siehe ggf. Abschnitt 7.8). Die vom Simulator ausgeführten Fehlerinjektionen verändern nur den Nachrichtenbereich D, alle anderen Bereiche (SW und SL) wurden während der Fehlerinjektion nicht verändert. Die Ergebnisse der Auswertung sind in den Tabelle 18 und Tabelle 19 aufgeführt.

Tabelle 18: Anteil der nicht erkannten Nutzdatenverfälschungen bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Bousbiba 2015b).

	(Bousbiba 2015b) CRC = 16 Bit Signatur = 16 Bit Nutzdatenverfälschung (zufällige Werte)		
	Signaturfehlerinjektionsschranke SFT		
DATA	5	10	15
16 Bit	$\left[\frac{0.0}{10^4}, \frac{0.0}{10^4}\right]$	$\left[\frac{0.07}{10^4}, \frac{0.13}{10^4}\right]$	$\left[\frac{0.0}{10^4}, \frac{0.0}{10^4}\right]$
32 Bit	$\left[\frac{0.09}{10^4}, \frac{0.15}{10^4}\right]$	$\left[\frac{0.08}{10^4}, \frac{0.14}{10^4}\right]$	$\left[\frac{0.13}{10^4}, \frac{0.21}{10^4}\right]$
64 Bit	$\left[\frac{0.17}{10^4}, \frac{0.27}{10^4}\right]$	$\left[\frac{0.16}{10^4}, \frac{0.24}{10^4}\right]$	$\left[\frac{0.13}{10^4}, \frac{0.21}{10^4}\right]$
128 Bit	$\left[\frac{0.14}{10^4}, \frac{0.22}{10^4}\right]$	$\left[\frac{0.09}{10^4}, \frac{0.17}{10^4}\right]$	$\left[\frac{0.17}{10^4}, \frac{0.27}{10^4}\right]$
256 Bit	$\left[\frac{0.10}{10^4}, \frac{0.18}{10^4}\right]$	$\left[\frac{0.09}{10^4}, \frac{0.17}{10^4}\right]$	$\left[\frac{0.10}{10^4}, \frac{0.18}{10^4}\right]$
512 Bit	$\left[\frac{0.13}{10^4}, \frac{0.21}{10^4}\right]$	$\left[\frac{0.09}{10^4}, \frac{0.15}{10^4}\right]$	$\left[\frac{0.12}{10^4}, \frac{0.20}{10^4}\right]$
Simulations-Läufe	10^7	10^7	10^7

Tabelle 19: Anteil der nicht erkannten Nutzdatenverfälschungen bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Echtle 2005; Echtle und Kimmeskamp 2010).

	(Echtle 2005; Echtle und Kimmeskamp 2010) CRC = 16 Bit Signatur = 16 Bit Nutzdatenverfälschung (zufällige Werte)		
	Nutzdatenverfälschung		
DATA	5	10	15
16 Bit	$\left[\frac{0.0}{10^4}, \frac{0.0}{10^4}\right]$	$\left[\frac{0.0}{10^4}, \frac{0.0}{10^4}\right]$	$\left[\frac{0.0}{10^4}, \frac{0.0}{10^4}\right]$
32 Bit	$\left[\frac{1.41}{10^4}, \frac{1.65}{10^4}\right]$	$\left[\frac{1.47}{10^4}, \frac{1.72}{10^4}\right]$	$\left[\frac{1.31}{10^4}, \frac{1.55}{10^4}\right]$
64 Bit	$\left[\frac{1.15}{10^4}, \frac{1.37}{10^4}\right]$	$\left[\frac{1.42}{10^4}, \frac{1.66}{10^4}\right]$	$\left[\frac{1.46}{10^4}, \frac{1.72}{10^4}\right]$
128 Bit	$\left[\frac{1.36}{10^4}, \frac{1.60}{10^4}\right]$	$\left[\frac{1.19}{10^4}, \frac{1.42}{10^4}\right]$	$\left[\frac{1.35}{10^4}, \frac{1.59}{10^4}\right]$
256 Bit	$\left[\frac{1.31}{10^4}, \frac{1.55}{10^4}\right]$	$\left[\frac{1.26}{10^4}, \frac{1.49}{10^4}\right]$	$\left[\frac{1.29}{10^4}, \frac{1.53}{10^4}\right]$
512 Bit	$\left[\frac{1.28}{10^4}, \frac{1.52}{10^4}\right]$	$\left[\frac{1.44}{10^4}, \frac{1.68}{10^4}\right]$	$\left[\frac{1.16}{10^4}, \frac{1.38}{10^4}\right]$
Simulations-Läufe	10^7	10^7	10^7

In der Tabelle 18 und Tabelle 19 sind die Anzahl der unerkannten Fehler für den Fehlertyp „Nutzdatenverfälschung“ für verschiedene Datenlängen aufgeführt. Insgesamt zeigt das Signaturverfahren (Bousbiba 2015b) im Vergleich zu (Echtle 2005; Echtle und Kimmeskamp 2010) eine um Faktor 10 bessere Fehlererfassung. Eine mögliche Erklärung hierfür könnte in der Art und Weise, wie Signatur und Ko-Signatur in (Bousbiba 2015b) und (Echtle 2005; Echtle und Kimmeskamp 2010) miteinander verknüpft werden, liegen. Denn beim Fehlertyp „Nutzdatenverfälschung“ werden nur die Nutzdaten der Nachricht verändert, nicht aber der Signaturwert oder die Signaturliste der Nachricht. Diese bleiben vor und nach der Fehlerinjektion unverändert. Daher liegt die Vermutung nahe, dass das in (Bousbiba 2015b) beschriebene Signaturverfahren (siehe ggf. Abschnitt 7.3) im Vergleich zu (Echtle 2005; Echtle und Kimmeskamp 2010) die Menge der gültigen Signaturen über den Daten-Bereich D besser verteilt, was im Falle der Nutzdatenverfäschung zu einer möglicherweise niedrigeren Kollisionswahrscheinlichkeit der Nutzdaten-Signatur-Paare führt. Auch hier weisen beide Signaturverfahren insgesamt eine hohe Fehlererfassung auf.

7.9.8 Ergebnis der Untersuchung von kombinierten Fehlertypen

Nachfolgend werden die Ergebnisse der kombinierten Fehlertypen in den Tabelle 20 bis Tabelle 33 aufgeführt. Eine Zusammenfassung der Ergebnisse wird ebenfalls am Ende dieses Abschnittes gegeben. Es wird die Tendenz erwartet, dass die kombinierten Fehlertypen insgesamt schlechter erfasst werden als die untersuchten Einfach-Fehlertypen.

Tabelle 20: Anteil der nicht erkannten Doppelfehler (Quelle, Bit Flips) bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Bousbiba 2015b).

	(Bousbiba 2015b) CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 10 fehlerhafte Quelle				
	Bit-Flips in Bits				
DATA	1	2	3	4	5
32 Bit	$\left[\frac{2.00}{10^4}, \frac{5.99}{10^4}\right]$	$\left[\frac{0.98}{10^4}, \frac{1.88}{10^4}\right]$	$\left[\frac{1.41}{10^4}, \frac{1.55}{10^4}\right]$	$\left[\frac{1.47}{10^4}, \frac{1.55}{10^4}\right]$	$\left[\frac{1.48}{10^4}, \frac{1.56}{10^4}\right]$
128 Bit	$\left[\frac{0}{10^4}, \frac{2.00}{10^4}\right]$	$\left[\frac{1.29}{10^4}, \frac{1.78}{10^4}\right]$	$\left[\frac{1.36}{10^4}, \frac{1.44}{10^4}\right]$	$\left[\frac{1.39}{10^4}, \frac{1.46}{10^4}\right]$	$\left[\frac{1.31}{10^4}, \frac{1.38}{10^4}\right]$
512 Bit	$\left[\frac{0}{10^4}, \frac{2.00}{10^4}\right]$	$\left[\frac{1.29}{10^4}, \frac{1.43}{10^4}\right]$	$\left[\frac{1.39}{10^4}, \frac{1.46}{10^4}\right]$	$\left[\frac{1.48}{10^4}, \frac{1.56}{10^4}\right]$	$\left[\frac{1.41}{10^4}, \frac{1.49}{10^4}\right]$
Simulations-Läufe	10^4	10^6	10^7	10^7	10^7

Tabelle 21: Anteil der nicht erkannten Doppelfehler (Quelle, Bit Flips) bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Echtle 2005; Echtle und Kimmeskamp 2010).

	(Echtle 2005; Echtle und Kimmeskamp 2010) CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 10 fehlerhafte Quelle				
	Bit-Flips in Bits				
DATA	1	2	3	4	5
32 Bit	$\left[\frac{0.59}{10^4}, \frac{3.41}{10^4}\right]$	$\left[\frac{1.22}{10^4}, \frac{2.21}{10^4}\right]$	$\left[\frac{1.52}{10^4}, \frac{1.66}{10^4}\right]$	$\left[\frac{1.41}{10^4}, \frac{1.49}{10^4}\right]$	$\left[\frac{1.39}{10^4}, \frac{1.46}{10^4}\right]$
128 Bit	$\left[\frac{0}{10^4}, \frac{2.00}{10^4}\right]$	$\left[\frac{1.29}{10^4}, \frac{1.78}{10^4}\right]$	$\left[\frac{1.36}{10^4}, \frac{1.44}{10^4}\right]$	$\left[\frac{1.36}{10^4}, \frac{1.43}{10^4}\right]$	$\left[\frac{1.34}{10^4}, \frac{1.42}{10^4}\right]$
512 Bit	$\left[\frac{0}{10^4}, \frac{2.00}{10^4}\right]$	$\left[\frac{1.24}{10^4}, \frac{1.38}{10^4}\right]$	$\left[\frac{1.39}{10^4}, \frac{1.47}{10^4}\right]$	$\left[\frac{1.35}{10^4}, \frac{1.42}{10^4}\right]$	$\left[\frac{1.36}{10^4}, \frac{1.44}{10^4}\right]$
Simulations-Läufe	10^4	10^6	10^7	10^7	10^7

In der Tabelle 20 und Tabelle 21 sind die Anzahl der unerkannten Fehler für die kombinierten Fehlertypen „fehlerhafte Quelle“ und „Bit Flips“ aufgeführt. Insgesamt weisen beide Signaturverfahren eine ähnlich gute Fehlererfassung auf. Wie zu erwarten, wird der kombinierte Fehler schlechter erkannt als der Fehlertyp „Bit Flip“, jedoch besser erkannt als der Fehlertyp „fehlerhafte Quelle“. Die Fehlererfassung bei kombinierten Fehlertypen liegt anhand der durchgeführten Fehlerinjektionsexperiment zwischen den beiden untersuchten Einfach-Fehlertypen (hier: Fehlertyp Bit Flip und Fehlertyp falsche Quelle).

Tabelle 22: Anteil der nicht erkannten Doppelfehler (Kosignierer, Bit Flips) bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Bousbiba 2015b).

	(Bousbiba 2015b) CRC = 16 Bit Signatur = 16 Bit Anzahl der fehlerhaften Kosignierer = 3 Signaturfehlerinjektionsschranke SFT = 10				
	Bit-Flips in Bits				
DATA	1	2	3	4	5
32 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0.18}{10^4}, \frac{0.67}{10^4}\right]$	$\left[\frac{0.42}{10^4}, \frac{0.49}{10^4}\right]$	$\left[\frac{0.43}{10^4}, \frac{0.47}{10^4}\right]$	$\left[\frac{0.32}{10^4}, \frac{0.36}{10^4}\right]$
128 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0.29}{10^4}, \frac{0.55}{10^4}\right]$	$\left[\frac{0.26}{10^4}, \frac{0.29}{10^4}\right]$	$\left[\frac{0.26}{10^4}, \frac{0.29}{10^4}\right]$	$\left[\frac{0.24}{10^4}, \frac{0.27}{10^4}\right]$
512 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0.28}{10^4}, \frac{0.35}{10^4}\right]$	$\left[\frac{0.25}{10^4}, \frac{0.28}{10^4}\right]$	$\left[\frac{0.26}{10^4}, \frac{0.29}{10^4}\right]$	$\left[\frac{0.27}{10^4}, \frac{0.31}{10^4}\right]$
Simulations-Läufe	10^4	10^6	10^7	10^7	10^7

Tabelle 23: Anteil der nicht erkannten Doppelfehler (Kosignierer, Bit Flips) bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Echtle 2005; Echtle und Kimmeskamp 2010).

	(Echtle 2005; Echtle und Kimmeskamp 2010) CRC = 16 Bit Signatur = 16 Bit Anzahl der fehlerhaften Kosignierer = 3 Signaturfehlerinjektionsschranke SFT = 10				
	Bit-Flips in Bits				
DATA	1	2	3	4	5
32 Bit	$\left[\frac{0}{10^4}, \frac{2.00}{10^4}\right]$	$\left[\frac{0.08}{10^4}, \frac{0.49}{10^4}\right]$	$\left[\frac{0.17}{10^4}, \frac{0.23}{10^4}\right]$	$\left[\frac{0.14}{10^4}, \frac{0.16}{10^4}\right]$	$\left[\frac{0.14}{10^4}, \frac{0.17}{10^4}\right]$
128 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0.14}{10^4}, \frac{0.32}{10^4}\right]$	$\left[\frac{0.15}{10^4}, \frac{0.17}{10^4}\right]$	$\left[\frac{0.14}{10^4}, \frac{0.17}{10^4}\right]$	$\left[\frac{0.12}{10^4}, \frac{0.17}{10^4}\right]$
512 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0.12}{10^4}, \frac{0.17}{10^4}\right]$	$\left[\frac{0.14}{10^4}, \frac{0.17}{10^4}\right]$	$\left[\frac{0.14}{10^4}, \frac{0.16}{10^4}\right]$	$\left[\frac{0.14}{10^4}, \frac{0.17}{10^4}\right]$
Simulations-Läufe	10^4	10^6	10^7	10^7	10^7

In der Tabelle 22 und Tabelle 23 sind die Anzahl der unerkannten Fehler für den kombinierten Fehlertyp „fehlerhafter Kosignierer“ und „Bit Flips“ für verschiedene Datenlängen und Bit Flip Längen aufgeführt. Insgesamt zeigt das Signaturverfahren von (Echtle 2005; Echtle und Kimmeskamp 2010) im Vergleich zu (Bousbiba 2015b) eine um den Faktor 2 bessere Fehlererfassung. Der Grund hierfür liegt in der insgesamt besseren Fehlererfassung der Einfach-Fehlertypen „Bit Flip“ und „fehlerhafter Kosignierer“. Auch hier weisen beide Verfahren insgesamt eine hohe Fehlererfassung auf. Man kann anhand der Tabellen (siehe Tabelle 20 bis

Tabelle 23) davon ausgehen, dass bei den verbleibenden Doppel-Fehlerinjektions-Untersuchungen ein ähnliches Verhalten zu beobachten ist. Daher werden im nachfolgenden nur die Ergebnisse der verbleibenden Fehlerinjektionsuntersuchungen aufgelistet sowie ein Gesamtresümee am Ende des Abschnittes gezogen.

Tabelle 24: Anteil der nicht erkannten Doppelfehler (Quelle, Burst) bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Bousbiba 2015b)

	(Bousbiba 2015b) CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 5 fehlerhafte Quelle		
	Burstlänge in Bits		
DATA	16	20	24
32 Bit	$\left[\frac{1.63}{10^4}, \frac{1.72}{10^4}\right]$	$\left[\frac{1.61}{10^4}, \frac{1.69}{10^4}\right]$	$\left[\frac{1.63}{10^4}, \frac{1.71}{10^4}\right]$
128 Bit	$\left[\frac{1.46}{10^4}, \frac{1.54}{10^4}\right]$	$\left[\frac{1.50}{10^4}, \frac{1.58}{10^4}\right]$	$\left[\frac{1.42}{10^4}, \frac{1.50}{10^4}\right]$
512 Bit	$\left[\frac{1.35}{10^4}, \frac{1.43}{10^4}\right]$	$\left[\frac{1.38}{10^4}, \frac{1.46}{10^4}\right]$	$\left[\frac{1.39}{10^4}, \frac{1.47}{10^4}\right]$
Simulations-Läufe	10^7	10^7	10^7

Tabelle 25: Anteil der nicht erkannten Doppelfehler (Quelle, Burst) bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Echtle 2005; Echtle und Kimmeskamp 2010).

	(Echtle 2005; Echtle und Kimmeskamp 2010) CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 5 fehlerhafte Quelle		
	Burstlänge in Bits		
DATA	16	20	24
32 Bit	$\left[\frac{1.42}{10^4}, \frac{1.49}{10^4}\right]$	$\left[\frac{1.39}{10^4}, \frac{1.47}{10^4}\right]$	$\left[\frac{1.41}{10^4}, \frac{1.48}{10^4}\right]$
128 Bit	$\left[\frac{1.42}{10^4}, \frac{1.50}{10^4}\right]$	$\left[\frac{1.39}{10^4}, \frac{1.46}{10^4}\right]$	$\left[\frac{1.35}{10^4}, \frac{1.43}{10^4}\right]$
512 Bit	$\left[\frac{1.46}{10^4}, \frac{1.54}{10^4}\right]$	$\left[\frac{1.36}{10^4}, \frac{1.43}{10^4}\right]$	$\left[\frac{1.34}{10^4}, \frac{1.41}{10^4}\right]$
Simulations-Läufe	10^7	10^7	10^7

Tabelle 26: Anteil der nicht erkannten Doppelfehler (Kosignierer, Burst) bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Bousbiba 2015b).

	(Bousbiba 2015b) CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 3 Anzahl der fehlerhaften Kosignierer = 3		
	Burstlänge in Bits		
DATA	16	20	24
32 Bit	$\left[\frac{0.67}{10^4}, \frac{0.72}{10^4}\right]$	$\left[\frac{0.61}{10^4}, \frac{0.66}{10^4}\right]$	$\left[\frac{0.55}{10^4}, \frac{0.60}{10^4}\right]$
128 Bit	$\left[\frac{0.40}{10^4}, \frac{0.44}{10^4}\right]$	$\left[\frac{0.40}{10^4}, \frac{0.45}{10^4}\right]$	$\left[\frac{0.37}{10^4}, \frac{0.41}{10^4}\right]$
512 Bit	$\left[\frac{0.24}{10^4}, \frac{0.27}{10^4}\right]$	$\left[\frac{0.24}{10^4}, \frac{0.28}{10^4}\right]$	$\left[\frac{0.25}{10^4}, \frac{0.28}{10^4}\right]$
Simulations-Läufe	10^7	10^7	10^7

Tabelle 27: Anteil der nicht erkannten Doppelfehler (Kosignierer, Burst) bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Echtle 2005; Echtle und Kimmeskamp 2010).

	(Echtle 2005; Echtle und Kimmeskamp 2010), CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 3 Anzahl der fehlerhaften Kosignierer = 3		
	Burstlänge in Bits		
DATA	16	20	24
32 Bit	$\left[\frac{0.09}{10^4}, \frac{0.11}{10^4}\right]$	$\left[\frac{0.05}{10^4}, \frac{0.06}{10^4}\right]$	$\left[\frac{0.05}{10^4}, \frac{0.06}{10^4}\right]$
128 Bit	$\left[\frac{0.10}{10^4}, \frac{0.12}{10^4}\right]$	$\left[\frac{0.01}{10^4}, \frac{0.13}{10^4}\right]$	$\left[\frac{0.10}{10^4}, \frac{0.12}{10^4}\right]$
512 Bit	$\left[\frac{0.14}{10^4}, \frac{0.16}{10^4}\right]$	$\left[\frac{0.13}{10^4}, \frac{0.16}{10^4}\right]$	$\left[\frac{0.09}{10^4}, \frac{0.11}{10^4}\right]$
Simulations-Läufe	10^7	10^7	10^7

Tabelle 28: Anteil der nicht erkannten Doppelfehler (Quelle, Kosignierer) bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Bousbiba 2015b).

	(Bousbiba 2015b) CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 5 fehlerhafte Quelle				
	Anzahl der fehlerhaften Kosignierer				
DATA	1	2	3	4	5
32 Bit	$\left[\frac{19.67}{10^4}, \frac{20.56}{10^4}\right]$	$\left[\frac{17.48}{10^4}, \frac{18.33}{10^4}\right]$	$\left[\frac{17.84}{10^4}, \frac{18.70}{10^4}\right]$	$\left[\frac{17.78}{10^4}, \frac{18.64}{10^4}\right]$	$\left[\frac{18.83}{10^4}, \frac{19.71}{10^4}\right]$
128 Bit	$\left[\frac{20.29}{10^4}, \frac{21.20}{10^4}\right]$	$\left[\frac{17.88}{10^4}, \frac{18.74}{10^4}\right]$	$\left[\frac{18.94}{10^4}, \frac{19.82}{10^4}\right]$	$\left[\frac{18.04}{10^4}, \frac{18.90}{10^4}\right]$	$\left[\frac{18.42}{10^4}, \frac{19.28}{10^4}\right]$
512 Bit	$\left[\frac{19.35}{10^4}, \frac{20.23}{10^4}\right]$	$\left[\frac{17.72}{10^4}, \frac{18.58}{10^4}\right]$	$\left[\frac{17.63}{10^4}, \frac{18.47}{10^4}\right]$	$\left[\frac{17.02}{10^4}, \frac{17.86}{10^4}\right]$	$\left[\frac{18.28}{10^4}, \frac{19.14}{10^4}\right]$
Simulations-Läufe	10^7	10^7	10^7	10^7	10^7

Tabelle 29: Anteil der nicht erkannten Doppelfehler (Quelle, Kosignierer) bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Echtle 2005; Echtle und Kimmeskamp 2010).

	(Echtle 2005; Echtle und Kimmeskamp 2010) CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 5 fehlerhafte Quelle				
	Anzahl der fehlerhaften Kosignierer				
DATA	1	2	3	4	5
32 Bit	$\left[\frac{20.04,20.94}{10^4}\right]$	$\left[\frac{16.64}{10^4}, \frac{17.46}{10^4}\right]$	$\left[\frac{15.63}{10^4}, \frac{16.43}{10^4}\right]$	$\left[\frac{14.52}{10^4}, \frac{15.30}{10^4}\right]$	$\left[\frac{13.17}{10^4}, \frac{13.91}{10^4}\right]$
128 Bit	$\left[\frac{20.20}{10^4}, \frac{21.10}{10^4}\right]$	$\left[\frac{16.97}{10^4}, \frac{17.81}{10^4}\right]$	$\left[\frac{15.30}{10^4}, \frac{16.10}{10^4}\right]$	$\left[\frac{14.57}{10^4}, \frac{15.35}{10^4}\right]$	$\left[\frac{14.22}{10^4}, \frac{14.98}{10^4}\right]$
512 Bit	$\left[\frac{20.10}{10^4}, \frac{21.00}{10^4}\right]$	$\left[\frac{17.46}{10^4}, \frac{18.30}{10^4}\right]$	$\left[\frac{15.02}{10^4}, \frac{15.81}{10^4}\right]$	$\left[\frac{14.29}{10^4}, \frac{15.05}{10^4}\right]$	$\left[\frac{13.08}{10^4}, \frac{13.82}{10^4}\right]$
Simulations-Läufe	10^7	10^7	10^7	10^7	10^7

Tabelle 30: Anteil der nicht erkannten Doppelfehler (Datenverfälschung, Quelle) bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Bousbiba 2015b).

	(Bousbiba 2015b) CRC = 16 Bit Signatur = 16 Bit Nutzdatenverfälschung (zufällige Werte) Fehlerhafte Quelle (anderer Quellknoten)	
	Signaturfehlerinjektionsschranke SFT	
DATA	5	10
32 Bit	$\left[\frac{1.44}{10^4}, \frac{1.52}{10^4}\right]$	$\left[\frac{1.46}{10^4}, \frac{1.54}{10^4}\right]$
128 Bit	$\left[\frac{1.36}{10^4}, \frac{1.43}{10^4}\right]$	$\left[\frac{1.35}{10^4}, \frac{1.42}{10^4}\right]$
512 Bit	$\left[\frac{1.41}{10^4}, \frac{1.48}{10^4}\right]$	$\left[\frac{1.40}{10^4}, \frac{1.48}{10^4}\right]$
Simulations-Läufe	10^7	10^7

Tabelle 31: Anteil der nicht erkannten Doppelfehler (Datenverfälschung, Quelle) bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Echtle 2005; Echtle und Kimmeskamp 2010).

	(Echtle 2005; Echtle und Kimmeskamp 2010) CRC = 16 Bit Signatur = 16 Bit Nutzdatenverfälschung (zufällige Werte) Fehlerhafte Quelle (anderer Quellknoten)	
	Signaturfehlerinjektionsschranke SFT	
DATA	5	10
32 Bit	$\left[\frac{14.41}{10^4}, \frac{15.17}{10^4}\right]$	$\left[\frac{13.46}{10^4}, \frac{14.20}{10^4}\right]$
128 Bit	$\left[\frac{13.68}{10^4}, \frac{14.42}{10^4}\right]$	$\left[\frac{13.64}{10^4}, \frac{14.38}{10^4}\right]$
512 Bit	$\left[\frac{13.46}{10^4}, \frac{14.20}{10^4}\right]$	$\left[\frac{14.00}{10^4}, \frac{14.76}{10^4}\right]$
Simulations-Läufe	10^7	10^7

Tabelle 32: Anteil der nicht erkannten Doppelfehler (Datenverfälschung, Kosignierer) bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Bousbiba 2015b).

	(Bousbiba 2015b) CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 5 Nutzdatenverfälschung (zufällige Werte)				
	Anzahl der fehlerhaften Kosignierer				
DATA	1	2	3	4	5
32 Bit	$\left[\frac{0}{10^4}, \frac{2.00}{10^4}\right]$	$\left[\frac{0.08}{10^4}, \frac{0.49}{10^4}\right]$	$\left[\frac{0.46}{10^4}, \frac{0.53}{10^4}\right]$	$\left[\frac{0.33}{10^4}, \frac{0.37}{10^4}\right]$	$\left[\frac{1.44}{10^4}, \frac{1.52}{10^4}\right]$
128 Bit	$\left[\frac{0.59}{10^4}, \frac{3.41}{10^4}\right]$	$\left[\frac{0.19}{10^4}, \frac{0.42}{10^4}\right]$	$\left[\frac{0.24}{10^4}, \frac{0.28}{10^4}\right]$	$\left[\frac{0.18}{10^4}, \frac{0.21}{10^4}\right]$	$\left[\frac{0.19}{10^4}, \frac{0.22}{10^4}\right]$
512 Bit	$\left[\frac{0}{10^4}, \frac{0}{10^4}\right]$	$\left[\frac{0.16}{10^4}, \frac{0.22}{10^4}\right]$	$\left[\frac{0.19}{10^4}, \frac{0.21}{10^4}\right]$	$\left[\frac{0.17}{10^4}, \frac{0.19}{10^4}\right]$	$\left[\frac{0.19}{10^4}, \frac{0.22}{10^4}\right]$
Simulations-Läufe	10^7	10^7	10^7	10^7	10^7

Tabelle 33: Anteil der nicht erkannten Doppelfehler (Datenverfälschung, Kosignierer) bei Einsatz eines 16 Bit Signaturverfahrens gemäß (Echtle 2005; Echtle und Kimmeskamp 2010).

	(Echtle 2005; Echtle und Kimmeskamp 2010) CRC = 16 Bit Signatur = 16 Bit Signaturfehlerinjektionsschranke SFT = 5 Nutzdatenverfälschung (zufällige Werte)				
	Anzahl der fehlerhaften Kosignierer				
DATA	1	2	3	4	5
32 Bit	$\left[\frac{1.44}{10^4}, \frac{1.70}{10^4}\right]$	$\left[\frac{1.26}{10^4}, \frac{1.50}{10^4}\right]$	$\left[\frac{1.33}{10^4}, \frac{1.57}{10^4}\right]$	$\left[\frac{1.27}{10^4}, \frac{1.51}{10^4}\right]$	$\left[\frac{1.19}{10^4}, \frac{1.41}{10^4}\right]$
128 Bit	$\left[\frac{1.38}{10^4}, \frac{1.62}{10^4}\right]$	$\left[\frac{1.57}{10^4}, \frac{1.83}{10^4}\right]$	$\left[\frac{1.19}{10^4}, \frac{1.41}{10^4}\right]$	$\left[\frac{1.16}{10^4}, \frac{1.38}{10^4}\right]$	$\left[\frac{1.42}{10^4}, \frac{1.66}{10^4}\right]$
512 Bit	$\left[\frac{1.36}{10^4}, \frac{1.60}{10^4}\right]$	$\left[\frac{1.57}{10^4}, \frac{1.83}{10^4}\right]$	$\left[\frac{1.47}{10^4}, \frac{1.73}{10^4}\right]$	$\left[\frac{1.41}{10^4}, \frac{1.65}{10^4}\right]$	$\left[\frac{1.33}{10^4}, \frac{1.57}{10^4}\right]$
Simulations-Läufe	10^7	10^7	10^7	10^7	10^7

Insgesamt verfügen beide Signaturverfahren über eine hohe Fehlererfassung. Das Signaturverfahren (Bousbiba 2015b) weist aber im Vergleich zu (Echtle 2005; Echtle und Kimmeskamp 2010) eine leicht verschlechterte Fehlererfassung auf, die unter anderem in der längeren Signaturliste begründet liegt.

Um eine höhere bzw. gleichgute Fehlererfassung zu erzielen, muss das Signaturverfahren von (Bousbiba 2015b) im Vergleich zu (Echtle 2005; Echtle und Kimmeskamp 2010) eine um ca.

25-Prozent längere Signatur (verbunden mit entsprechend längeren Schlüsseln) verwenden (siehe Ergebnisliste auf der beigefügten CD-ROM).

Insgesamt lässt sich folgendes festhalten: Beide Verfahren sind für den Einsatz in fehlertoleranten verteilten Systemen gleichermaßen gut geeignet. Die Defizite[9] von (Bousbiba 2015b) können durch den Einsatz eines längeren Signaturschlüssels (siehe beigefügte CD-ROM) gut kompensiert werden. Folglich ist die Wahl, welches Signaturverfahren verwendet werden soll, auf den jeweiligen Einsatzzweck abzustimmen (Anzahl der zu reduzierenden Nachrichten sowie die um 25-Prozent höhere Informationsredundanz pro Nachricht).

Das Verfahren von (Bousbiba 2015b) benötigt eine längere Signaturliste und eine um ca. 25% längere Signatur, ermöglicht aber die Verschmelzung von verschieden signierten Nachrichten mit gleichem Inhalt. Insgesamt kann daraus ein bedeutender Aufwandsvorteil entstehen, wie nachfolgend gezeigt wird.

Im folgenden Abschnitt soll die positive Eigenschaft des Signaturverfahrens (Bousbiba 2015b) auf die Kommunikationskomplexität (Anzahl der Nachrichten, Anzahl der redundanten Knoten sowie die Anzahl der zu speichernden Nachrichten) von Übereinstimmungsprotokollen näher untersucht werden.

7.10 Modifizierte Übereinstimmungsprotokolle – Motivation

Seit das Übereinstimmungsproblem erstmals von Lamport et. al. (Lamport et al. 1982) angesprochen und gelöst wurde, sind dazu viele Beiträge für verschiedene Systemmodelle vorgestellt worden – unter anderem auch solche Arbeiten, die die Grenzen von Übereinstimmung für verschiedene Systemmodelle aufzeigen (siehe (Fischer et al. 1985c; Santoro und Widmayer 2007, 1989)). Unter den Beiträgen (seit der ersten Veröffentlich 1982) befinden sich auch viele Arbeiten, die als Ziel die Reduzierung der Kommunikationskomplexität anvisiert haben. Dazu zählen die folgenden Arbeiten: (Jochim und Forest 2010; Tulone 2004; Ben-Or und Hassidim; Nazreen Banu et al. 2012; Khosravi und Kavian 2012).

Es hat sich herausgestellt, dass die signaturbasierten Übereinstimmungsprotokolle eine signifikant geringere Anzahl an Nachrichten und redundanten Knoten benötigen als solche, die ohne Signatur auskommen. Allerdings gibt es ohne die Möglichkeit der Signaturver-

[9] Die Fehlererfassung von (Bousbiba 2015b) schwankt im Vergleich zu (Echtle 2005; Echtle und Kimmeskamp 2010) je nach untersuchtem Fehlertyp zwischen Faktor 1 bis 10.

schmelzung eine Einschränkung bzgl. der Reduktion der Kommunikationskomplexität: Wenn ein Knoten feststellt, dass eine bestimmte Nachricht von einem Knoten x signiert wurde und eine inhaltsgleiche Nachricht von einem Knoten y signiert wurde, dann kann er diese Information nur durch den Transfer von beiden Signaturen weitergeben. Er kann nicht durch eine einzige Signatur ausdrücken „x und y haben signiert“. Dies ist nur durch Signaturverschmelzung möglich.

In dieser Arbeit wurde das Signaturverschmelzungsverfahren SigSeam, welches bereits in Kapitel 7 vorgestellt wurde, auf zwei Übereinstimmungsprotokollen angewendet, nämlich Turquois und ESSEN. Anhand dieser beiden Protokolle sollen die Vorteile aufgezeigt werden, die das Signaturverschmelzungsverfahren SigSeam bietet. Für eine detaillierte Beschreibung des Protokolls Turquois sei auf (Moniz et al. 2013) verwiesen. Im Folgenden werden nur die Teile des Algorithmus beschrieben, die durch den Einsatz des neuen Signaturverfahrens zum Zweck der Effizienzverbesserung angepasst werden konnten.

7.11 ESSEN* und Turquois*

Mit Hilfe des neuen Signaturverfahrens SigSeam lässt sich die Kommunikationskomplexität von ESSEN (siehe Kapitel 4) weiter reduzieren. Hierzu wurden einige Änderungen am Protokoll vorgenommen, auf die im Folgenden näher eingegangen wird. Wir bezeichnen mit ESSEN* die durch den Einsatz von SigSeam bzgl. der Kommunikationskomplexität verbesserte Version von ESSEN:

- Nachrichtenspeicherung: Eine Data-Nachricht wird in den sekundären Puffer hinterlegt, wenn neben den vier Bedingungen (siehe Kapitel 4) zusätzlich die folgenden zwei Bedingungen mit erfüllt werden:
  1. Der Empfängerknoten gehört zu Gruppe ErwG.
  2. Der Empfängerknoten hat zuvor noch keine Nachricht übertragen.

  In allen anderen Fällen, wird die Data Nachricht verworfen (für mehr Details siehe Kapitel 4). Durch den Einsatz des neuen Signaturverfahrens wird die Bedingungen AG3 nicht mehr benötigt, da die Signaturen in SMB und PMB miteinander zu einer Signatur verschmolzen und übertragen werden. Daher löscht jeder fehlerfreie Knoten, der die Bedingung für die Nachrichtenverschmelzung nicht erfüllt, den Inhalt im SMB.

- Bedingung für die Nachrichtenverschmelzung: Ein Knoten verschmelzt die beiden Nachrichten in PMB und SMB genau dann, wenn beide Nachrichten zusammen mindestens $2f-2$ verschiedene Signaturquellen enthalten. Sobald ein fehlerfreier Knoten eine solche Nachricht überträgt, enthält sie mindesten f Signaturen von fehlerfreien Knoten aus ForwG. Folglich, gemäß AG2, ist durch das Senden einer solchen Nachricht die Bedingung IC1 und IC2 stets erfüllt. Die Verschmelzungsbedingung lautet formal:

$$sigCount\big(sig(MsgPMB) \cup \big(sig(MsgSMB) \setminus sig(MsgPMB)\big)\big) \geq 2f - 2. \qquad 7.6$$

  Unabhängig davon, ob zum Sendezeitpunkt eines Knotens N[i] die Bedingung für die Nachrichtenverschmelzung erfüllt ist oder nicht, löscht Knoten N[i] (bzw. verwendet nicht) den Inhalt seines Sekundärpuffers. Das Löschen des Sekundärpuffers erfolgt genau dann, wenn das Ende des Sendeslots von Knoten N[i] erreicht wurde. Folglich wird zur Bestimmung des endgültigen Übereinstimmungswerts nur der SMB und der DMB verwendet. Alle anderen Teile des Algorithmus von ESSEN bleiben unverändert.

Turquois ist ein Protokoll, welches das k-Konsens-Problem für asynchrone Systeme löst, die sich aus n Ad-hoc-Knoten zusammensetzen, von denen ein Teil f_N (mit $f_N < \frac{n}{3}$) in willkürlicher Weise ausfallen können. Außerdem ist es die erste Arbeit, welche das Problem des Konsensproblems in Anwesenheit von Omission-Fehler aufgreift und löst.

Allerdings löst Turquois das Konsensproblem unter Einsatz eines hohen Kommunikations- und Speicheroverheads. Der hohe Nachrichtenaufwand wird durch den verwendeten Nachrichtenvalidierungsprozess verursacht. Im schlimmsten Fall muss ein Knoten bis zu $\frac{n+f}{2}$ Nachrichten in einer Runde übertragen. Im Folgenden bezeichnen wir mit Turquois* das Derivat von Turquois, das die Signaturverschmelzung benutzt. Ansonsten besteht zwischen den beiden Protokollvarianten kein Unterschied.

Durch den Einsatz des neuen Signaturverfahrens SigSeam ist Turquois* in der Lage, die in jeder Runder empfangenen Nachrichten (mit identischem Inhalt) zu einer einzigen Nachricht zusammenzufassen (anstatt diese wie imcursprünglichen Ansatz einzeln abzuspeichern), was zu einer hohen Speicherreduktion resultiert. Nun muss ein Knoten unabhängig von der Anzahl

der zu tolerierenden Fehlern nicht mehr als zwei verschiedene Nachrichten pro Runde speichern und im ungünstigsten Fall nicht mehr als drei Nachrichten in einer Runde senden. Alle anderen in Turquois beschriebenen Prozesse (wie etwa der Validierungsprozess) bleiben unverändert, mehr hierzu siehe (Moniz et al. 2013).

7.12 Vergleich der Kommunikationskomplexität der untersuchten Übereinstimmungsprotokolle

Die Kommunikationskomplexität bzgl. der Knoten und Nachrichtenoverheads wurde mit Hilfe eines Simulators (siehe Kapitel 4) quantifiziert. Die modifizieren Protokollvarianten wurden mit ihren ursprünglichen Versionen vergleichen. Die Ergebnisse sind in Abbildung 23 und Abbildung 24 zu sehen. Insgesamt zeigen die Ergebnisse, dass durch den Einsatz des neuen Signaturverfahrens eine Verbesserung der Kommunikationskomplexität in beiden untersuchten Protokollvarianten erzielt werden konnten.

Im Falle von ESSEN konnte durch den Einsatz von SigSeam die Anzahl der redundanten Knoten sowie die Anzahl der benötigten Nachrichtenübertragungen von $3f + max(0, f - 2)$ auf $3f + max(0, f - 2) - 1$ reduziert werden (siehe Abbildung 23 und Abbildung 24).

Im Protokoll von Turquois konnte durch die Nachrichtenverschmelzung die hohe Anzahl von $1 + (3f + 1) \cdot \frac{(n+f+2)}{2}$ Nachrichten-übertragungen im ungünstigsten Fall auf 3 Nachrichten (unabhängig von der Anzahl der zu tolerierenden Fehlern) pro Knoten in einer Runde reduziert werden (insgesamt senden $3f + 1$ Knoten $9f + 3$ Nachrichten im worst case).

7.14 Die Herleitung der Knotenzahl in ESSEN*

Bei ESSEN* handelt es sich um ein Derivat von ESSEN, welches durch den Einsatz des Signaturverschmelzungsverfahrens SigSeam bzgl. Kommunikationskomplexität weiter optimiert wurde (siehe ggf. Abschnitt 7.11). Auf die Herleitung der Knotenzahl für ESSEN* wird im Folgenden näher eingegangen (siehe Tabelle 2).

Tabelle 34: Parameterwerte b, e (die mit Hilfe simulativer Analyse bestimmt worden sind) in Abhängigkeit von der Anzahl f der zu tolerierenden Fehler (Bousbiba 2015b).

f	Anzahl der Knoten in BasisG (b_f)	Anzahl der Knoten in ErwG (e_f)	Anzahl der Knoten in ForwG ($b_f + e_f$)
1	$b_1 = 2$	$e_1 = 0$	2
2	$b_2 = 3$	$e_2 = 2$	5
3	$b_3 = 4$	$e_3 = 4$	8
4	$b_4 = 5$	$e_4 = 7$	12
5	$b_5 = 6$	$e_5 = 10$	16
Die Werte b_i und e_i heißen Folgenglieder mit $1 \leq i \leq f$. Sie bestimmten die notwendige Anzahl der Knoten für die Gruppen BasisG und ErwG. Der Index i in e_i bzw. b_i gibt die Anzahl f der zu tolerierenden Fehler an.			

Im nächsten Schritt muss für die Folgeglieder e_f (wie in Abschnitt 4.7 bereits beschrieben) ein Bildungsgesetzt $<e_f>$ bestimmt werden. Das folgende Bildungsgesetzt $<e_f>$ für ESSEN* wurde mit Hilfe einer simulativen Analyse ermittelt (es wurde jeweils geprüft, bis zu welchen Höchstwerten von b und e sich das Protokoll ESSEN* fehlertolerant verhält):

$$< e_1 > = 0$$

$$< e_2 > = 2$$

...

$$< e_f > = \begin{cases} 2 \cdot (f-1) + max(0, f-2) - 1, & \text{für } f \geq 3 \\ 2 \cdot (f-1) - 1, & sonst \end{cases}$$

$$b \leq \begin{cases} 2 \cdot (f-1) + max(0, f-2) - 1, & \text{für } f \geq 3 \\ 2 \cdot (f-1) - 1, & sonst \end{cases} \blacksquare$$

Durch den Einsatz des neuen Signaturverfahrens benötigt ESSEN* genau einen Knoten weniger als das ursprüngliche Protokoll ESSEN. Eine mögliche Erklärung hierfür sind neben den Änderungen, die am Protokoll ESSEN vorgenommen wurden (siehe Abschnitt 7.11 und Formel 7.6), der Einsatz des neuen Signaturverfahrens SigSeam, welches dem Sender ermöglicht, frühzeitig eine Nachricht mit hoher Signaluranzahl zu senden (siehe Formel 7.6).

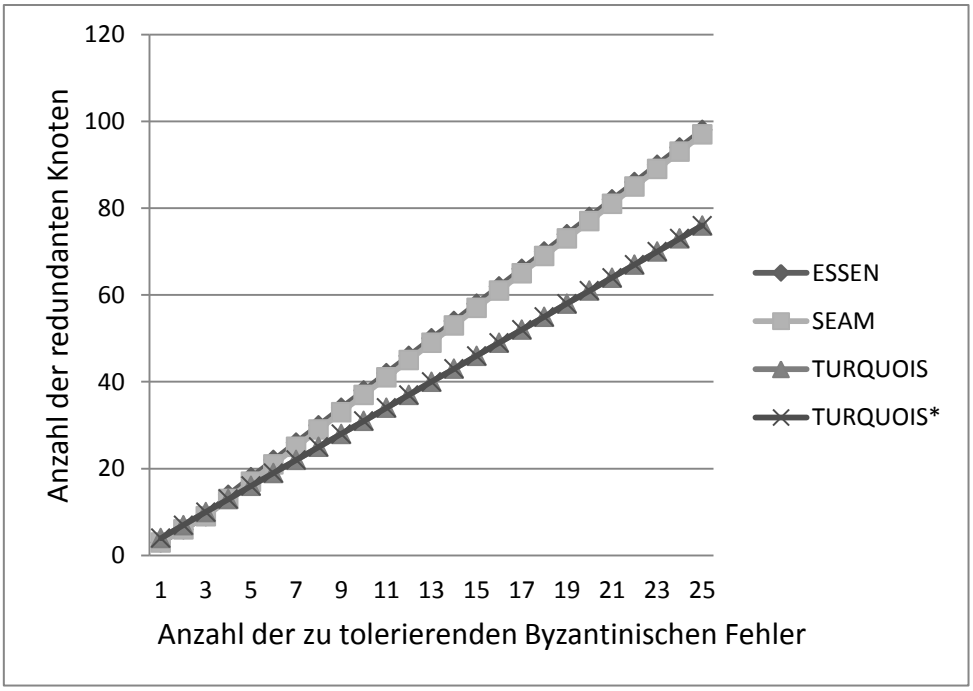


Abbildung 23: Der Einsatz von Signaturverschmelzung zur Reduzierung der Knotenzahl in Übereinstimmungsprotokollen (Bousbiba 2015b).

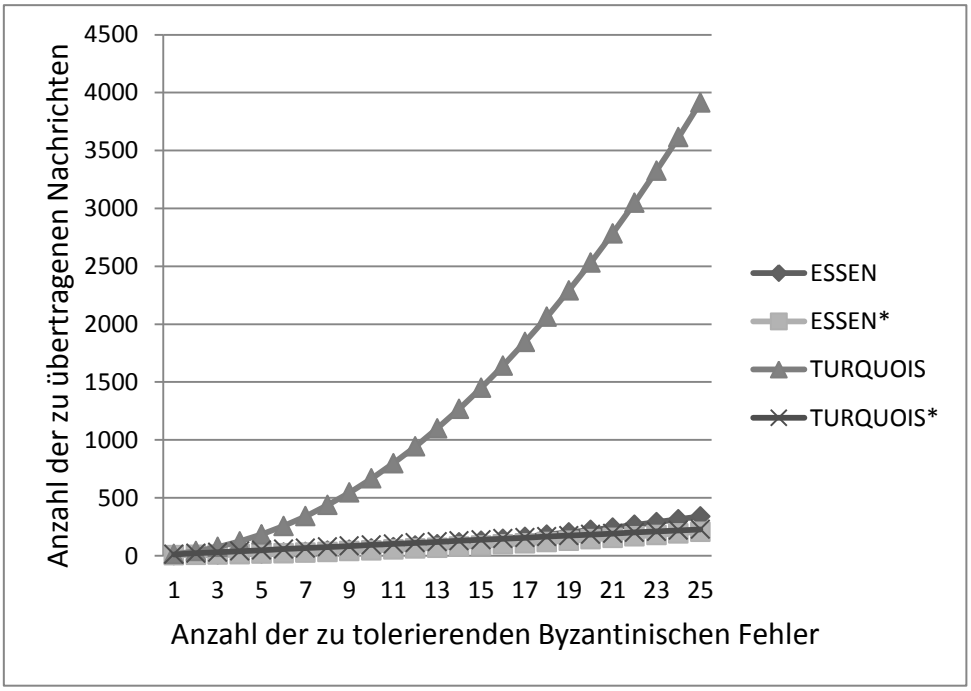


Abbildung 24: Einsatz von Signaturverschmelzung zur Reduzierung des Nachrichtenoverheads in Übereinstimmungsprotokollen (1) (Bousbiba 2015b).

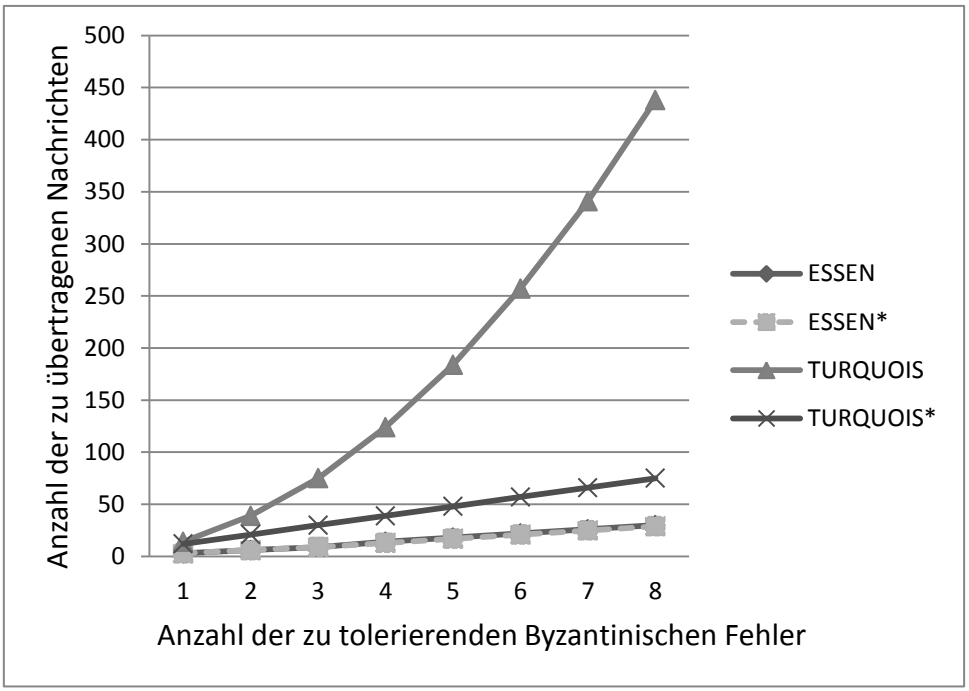


Abbildung 25: Einsatz von Signaturverschmelzung zur Reduzierung des Nachrichtenoverheads in Übereinstimmungsprotokollen (2) (Bousbiba 2015b).

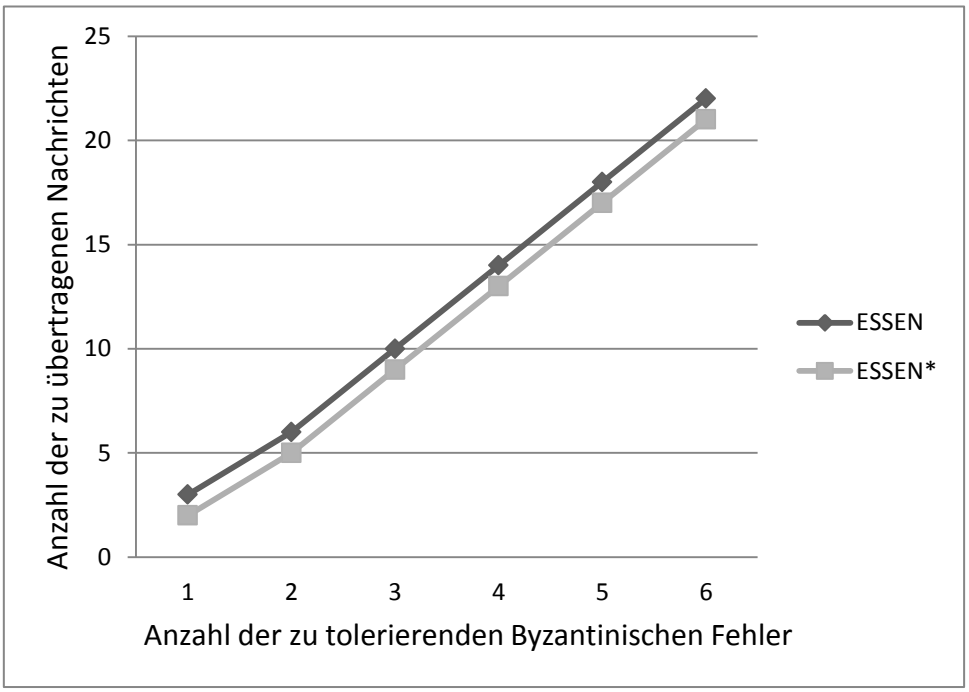


Abbildung 26: Einsatz von Signaturverschmelzung zur Reduzierung des Nachrichtenoverheads in Übereinstimmungsprotokollen (3) (Bousbiba 2015b).

7.15 Zusammenfassung

Die Ergebnisse aus Abschnitt 7.9 und Abschnitt 7.10 haben sowohl die Vorteile als auch die Nachteile des neuen Signaturverfahrens aufgezeigt. Insgesamt lässt sich zusammenfassen, dass durch den Einsatz des neuen Signaturverfahrens SigSeam eine signifikante Verbesserung der Kommunikationskomplexität existierender Protokolle erzielt wurde. Das Fehlermodell des Übereinstimmungsprotokolls wird durch den Einsatz des neuen Signaturverfahrens in keiner Weise beeinflusst. Jedoch muss man beim Einsatz von SigSeam im Vergleich zu herkömmlichen Signaturverfahren eine um 25-Prozent längere Signatur verwenden, wenn eine gleich gute Fehlererfassung erzielt werden soll.

8 Fazit und Ausblick

„When we recognize the battle against chaos, mess, and unmastered complexity as one of computing science's major callings, we must admit that 'Beauty Is Our Business'"– Edsger W. Dijkstra.

„Forschung ist Fortsetzung der Neugier mit anderen Mitteln" – Hans-Jürgen Quadbeck-Seeger

8.1 Zusammenfassung der Ergebnisse

Man kann davon ausgehen, dass der Einsatz von Übereinstimmungsprotokollen langfristig gesehen immer mehr in den Fokus des industriellen Einsatzes rücken wird, wie bereits im Einführungskapitel (siehe Kapitel 1) und zum Teil in Kapitel 3 motiviert. Außerdem gibt es bereits heute Bestrebungen, die in der Industrie verwendeten drahtgebundenen Systeme teilweise bzw. vollständig durch drahtlose Systeme zu ersetzen. Dieser Trend ist durch die folgenden Faktoren motiviert:

- **Kosten**: Viele drahtgebundene Systeme verursachen hohe Installationskosten. Außerdem können je nach Verkabelungskomplexität (wie es etwa in Flugzeugen der Fall ist) und die damit verbundene Gewichtszunahme zusätzliche Kosten entstehen (z. B. in Form von höherem Kraftstoffverbrauch bei Flugzeugen, Autos, etc.),
- **Flexibilität und Effizienz**: Der Austausch von Komponenten ist in drahtlosen Systemen im Vergleich zu drahtgebundenen Systemen einfacher zu realisieren,
- **Neuartige Anwendungen**: Beispielsweise ließe sich die Produktionsleistung in Industrieanlagen weiter steigen, wenn intelligente kooperierende Gabelstapler zum Einsatz kommen, die auf optimierten Routen Waren transportieren.

Ferner unterliegen viele (sicherheitskritische) Anwendungen in der Industrie (harten) Echtzeitanforderungen. Demzufolge müssen effiziente Lösungen des Übereinstimmungsproblems entwickelt und eingesetzt werden. Hierzu wurden im Rahmen dieser Arbeit neue Techniken und Strategien entwickelt.

Im ersten Teil der Arbeit wurden zunächst (neben der Motivation der Arbeit) sämtliche notwendigen Grundlagen und Begriffserklärungen erläutert. Nachdem dieses Fundament für das weitere Verständnis der Arbeit gelegt war, wurde darauf aufbauend der Stand der Forschung (siehe Kapitel 3) präsentiert sowie die gemäß dem Stand der Forschung bisher noch nicht beantwortete Forschungsfrage formuliert. Mögliche Ansätze zur Lösung des Problems wurden ebenfalls in diesem Kapitel kurz resümiert.

Der zweite Teil der Arbeit (Kapitel 4 bis 7) beschäftigte sich mit neuen Techniken und Strategien zur Entwicklung effizienter Übereinstimmungsprotokolle sowie zur Reduzierung des Aufwands neuer und bestehender Lösungen durch die Technik der Signaturverschmelzung. Die erarbeiteten Lösungen liefern zu den folgenden beiden formulierten Fragestellungen (siehe Punkt 1 und 2 unten) eine Antwort:

- **Entwicklung eines effizienten Übereinstimmungsprotokolls**: In der Literatur gibt es bereits eine Vielzahl an Lösungen zum Byzantinischen Übereinstimmungsproblem für verschiedene Systemmodellannahmen. Diese Lösungen erzielen in mehreren Runden eine Übereinstimmung in fehlertoleranter Weise.
  Lässt sich ein Übereinstimmungsprotokoll mit hohem Fehlertoleranzgrad entwickeln, welches das Übereinstimmungsproblem unabhängig von der Anzahl der zu tolerierenden Fehler stets in einer Runde löst und in der Summe eine niedrigere Kommunikationskomplexität als andere Lösungen gemäß dem Stand der Technik aufweist?
- **Entwicklung eines neuartigen Signaturverfahrens zur Verbesserung der Kommunikationskomplexität von Übereinstimmungsprotokollen**: In vielen Übereinstimmungsprotokollen leiten die Knoten in wiederkehrenden Runden ein Teil der aus den vorangegangenen Runden empfangen Nachrichten weiter (siehe SM (Jochim und Forest 2010), Turquois (Moniz et al. 2013), etc.).
  Lässt sich mit Hilfe eines geeigneten Signaturverfahrens die Kommunikationskomplexität von Übereinstimmungsprotokollen signifikant verbessern?

In Kapitel 4 bis 6 wurde das neue Byzantinische Übereinstimmungsprotokoll ESSEN („An **E**fficient **S**ingle Round **S**ignature Protected Message **E**xchange Agreement Protocol for Wireless Distributed **N**etworks“) inklusive Korrektheitsnachweis vorgestellt, welches das Byzantinische Übereinstimmungsproblem effizient löst. Folgenden Kernideen haben zur neuen Lösung des Übereinstimmungsproblems in O(1) Runden beigetragen:

1. Statische Runden: Zur Lösung des Übereinstimmungsproblems werden statische statt dynamische Runden (siehe Abschnitt 4.2) verwendet. In verteilten synchronen (Echtzeit-) Systemen ist der zeitliche Vorteil, den man bei dynamischen Runde erhält, gegenüber der statischen Runde nicht groß (siehe Argumentation in Abschnitt 4.2). Außerdem besteht bei Verwendung dynamischer Runden ein großer Nachteil darin, dass man zur Lösung des Byzantinischen Übereinstimmungsproblems stets $f + 1$ Phasen (siehe Definition 2.2) bzw. $f + 1$ dynamische Runden benötigt.
2. Default Nachrichten: Der Einsatz von Default-Nachrichten (mit der die Knoten eine Art „Veto“ ausdrücken können) soll sicherstellen, dass die Bedingung IC1 im Falle eines fehlerhaften Quellknotens stets eingehalten wird. Die Anzahl der Signaturen in einer Default-Nachricht bildet einen Schwellwert, der verhindern soll, dass fehlerhafte Knoten zu viele fehlerfreie Knoten unerkannt überbrücken können. Die fehlerfreien Knoten „melden sich mit Default-Nachrichten zu Wort“ und bilden dadurch ein Gegengewicht zu einer Data-Nachricht, die fehlerfreie Knoten überbrückt hat und dadurch zu wenige Signaturen fehlerfreien Knoten trägt. Allein aus der Signaturanzahl der Data.Nachricht könnte man nicht erkennen, wie viele diese Signaturen von fehlerfreien Knoten stammen. Deshalb wurde das neuartige Mittel der Default-Nachrichten geschaffen, dessen Signaturen als „aktive Gegenstimmen“ zählen. Obwohl auch bei den Default-Nachrichten nicht erkennbar ist, welche der Signaturen von fehlerfreien Knoten stammen, kann der Algorithmus, der die Cleaning-Funktion CF enthält eine Verletzung der Bedingung IC1 garantiert verhindern, wie formal bewiesen wurde.
3. Lokale Fehlerdiagnose: Durch den Einsatz der Reinigungsfunktion können die fehlerfreien Knoten, ihre Entscheidung auf Basis der fehlerfreien Knoten treffen.

Zur Tolerierung von f willkürlichen Fehlern benötigt ESSEN mindestens

$$n \geq 3f + max(0, f - 2)$$

Knoten. Außerdem terminiert das Übereinstimmungsprotokoll ESSEN stets in einer Runde – unabhängig von der Anzahl f der zu tolerierenden Fehler, wobei jeder Knoten nicht mehr als drei Nachrichten speichern muss und höchstens eine Nachricht in einer Runde überträgt. Die Korrektheit des Protokolls wurde im Rahmen der Arbeit sowohl validiert als auch formal verifiziert.

Insgesamt lässt sich zusammenfassen, dass die ursprüngliche Aufgabenstellung, nämlich die Entwicklung eines effizienten Übereinstimmungsprotokolls, in der vorliegenden Arbeit mit dem Protokoll ESSEN vollständig gelöst wurde. In der Praxis dürfte der lineare Anstieg der Knotenanzahl um den Faktor 4 oftmals keine allzu große Rolle spielen, weil man typischerweise von einer kleinen Anzahl von gleichzeitig auftretenden Fehlern in einem System, das ohnehin aus vielen Knoten besteht, ausgeht.

Obwohl das Protokoll ESSEN das Byzantinische Übereinstimmungsproblem effizient löst, lag die Vermutung nahe, dass durch den Einsatz eines geeigneten Signaturverfahrens eine weitere Verbesserung bzgl. der Kommunikationskomplexität erzielt werden kann (siehe Punkt 2 oben, „Default Nachrichten“). Folglich wurde im Rahmen dieser Arbeit ein neues Signaturverfahren entwickelt, welches erstmalig die Signaturverschmelzung umsetzt. Die Idee dahinter ist recht einfach: Einem Nachrichtenempfänger soll das Signaturverfahren die Möglichkeit eröffnen, die neu eintreffende Nachricht und bereits gespeicherte Nachrichten zu verschmelzen, wenn die Nachrichteninhalte bis auf die in ihnen enthaltenen Signaturen gleich sind. Die resultierende Nachricht wird dabei nicht länger als die Originalnachrichten.

Der letzte Schritt zur Entwicklung eines effizienten Übereinstimmungsprotokolls wurde mit dem neuen entwickelten Signaturverfahren SigSeam (“Reducing the Communication Complexity of Agreement Protocols By Applying A New Signature called SigSeam”) vollzogen (siehe mehr hierzu in Kapitel 7). Der positive Einfluss von SigSeam auf die Kommunikationskomplexität wurden in Kapitel 7 anhand zweier Übereinstimmungsprotokolle ESSEN und Turquois demonstriert. Im Falle von ESSEN konnte die ursprüngliche Kommunikationskomplexität von $n \geq 3f + max(0, f-2)$ auf $n \geq 3f + max(0, f-2) - 1$ reduziert werden. Im Fall von Turquois ließ sich die hohe Anzahl von $1 + (3f+1) \cdot \frac{(n+f+2)}{2}$ Nachrichtenübertragung, die (innerhalb einer Runde) im ungünstigsten stattfinden, auf 3 Nachrichten pro Knoten reduziert werden.

Insgesamt lässt sich festhalten, dass das Prinzip der Signaturverschmelzung zur Reduzierung der Kommunikationskomplexität prinzipiell auf einen Großteil der existierenden Übereinstimmungsprotokolle angewendet werden kann (Näheres hierzu siehe Kapitel 7). Das neue Signaturverfahren SigSeam konnte die Kommunikationskomplexität der untersuchten Übereinstimmungsprotokolle signifikant verbessern, weist aber eine im Vergleich zu „herkömmlichen“ Signaturverfahren leicht verschlechterte Fehlererfassung auf. Jedoch lässt sich dieses

Problem durch eine im Vergleich zu „herkömmlichen“ Signaturverfahren um ca. 25 Prozent erhöhte Signaturlänge (verbunden mit entsprechend längeren Schlüsseln) gut lösen.

Die vorliegende Arbeit hat gezeigt, dass die Effizienzsteigerung nicht mit einer Verschlechterung der Fehlertoleranz einhergehen muss. Weiterhin konnte gezeigt werden, dass der Einsatz von Übereinstimmungsprotokollen auch in verteilten Echtzeitsystemen mit harten Echtzeitanforderungen genutzt werden kann.

8.2 Offene Fragen

Zum Schluss soll noch auf offene Fragen eingegangen werden, die sich während der Bearbeitung der Arbeit ergeben haben und denen aufgrund der begrenzten Zeit nicht in der notwendigen Tiefe nachgegangen werden konnte.

Entwurf von effizienten fehlertoleranten Protokollen zur Tolerierung dynamischer Fehler (synonym: „mobiler Virus“; bösartiger „Agent“)
Es handelt sich hierbei um ein zu (Warns 2010) ähnliches Fehlermodell. Das Besondere an diesem Fehlermodell (näheres hierzu siehe (Buhrman et al. 1995)) besteht darin, dass das Byzantinische Übereinstimmungsproblem in Anwesenheit von bis zu f bösartigen (malicious) „Agenten“ (d.h. Knoten) gelöst werden muss, die in der Lage sind:

- Knoten zu infizieren (in einem fehlerhaften Zustand versetzen),
- wobei sie sich mit einer vom Parameters c (eine Konstante des Fehlermodells) abhängige Wandergeschwindigkeit im verteilten System bewegen, um weitere Knoten zu infizieren. Der Parameter c gibt an, wie viele Runden ein bösartiger „Agent“ benötigt, um von einem Knoten zum anderen Knoten zu wandern. Gemäß (Buhrman et al. 1995) wird der Wert c wie folgt bestimmt:

$$c = \frac{Anzahl\ Runden}{Anzahl\ Knoten} \text{ mit } c \in \mathbb{N}\backslash\{0\} \qquad 8.1$$

  Unabhängig vom Wert c kann ein bösartiger „Agent“ höchstens einen Knoten infizieren (siehe ggf. (Buhrman et al. 1995)).

Zwar kann das Problem als grundsätzlich gelöst betrachtet werden (siehe (Buhrman et al. 1995; Nazreen Banu et al. 2012)), jedoch ist der Knoten- und Kommunikationsoverhead sehr groß. Es stellt sich die Frage, inwieweit die hier vorgestellten Lösungsstrategien (SigSeam

und ESSEN) einen Beitrag zur effizienten Lösung des Problems leisten können. Dies bedarf jedoch einer (an diese Arbeit anknüpfenden) weiteren Untersuchung.

Entwurf von effizienten fehlertoleranten Protokollen für Multi-Hop-Netze
Die Entwicklung effizienter fehlertoleranter Protokolle für Multi-Hop Netze kann als noch nicht abgeschlossen betrachtet werden. Hier besteht weiterer Handlungsbedarf und zwar in Bezug auf die Entwicklung effizienter Fehlertoleranz-Verfahren wie etwa:

- Fehlertoleranzstrategien bei Broadcast für drahtgebundenen/drahtlosen verteilten *Multihop-Systemen*,
- Fehlertolerante effiziente Datenaggregationen für Sensornetzwerke,
- Gossip-basiertes fehlertolerantes Membership Management Protokoll für (Echtzeit-) Cloud Computing Umgebungen.

Es stellt sich die Frage, inwieweit die hier entwickelten Lösungsstrategien hierzu einen Beitrag liefern bzw. für die Entwicklung effizienter Fehlertoleranz-Verfahren genutzt werden können. Hierzu müssten noch weitere (an die Arbeit anknüpfende) Untersuchungen erfolgen.

Abschließend sei festgestellt: Mit ESSEN und SigSeam wurden zwei Verfahren entworfen, die ein effizienzsteigerndes Potential beinhalten, das durch Anpassung an gegebene Systemeigenschaften weiter erhöht werden kann.

Literaturverzeichnis

Abdelhakim, Mai; Lightfoot, Leonard E.; Li, Tongtong: Reliable data fusion in wireless sensor networks under Byzantine attacks. In: MILCOM 2011 - 2011 IEEE Military Communications Conference. Baltimore, MD, USA, S. 810–815.

Agmon, Noa; Peleg, David (2006): Fault-Tolerant Gathering Algorithms for Autonomous Mobile Robots. In: *SIAM J. Comput.* 36 (1), S. 56–82. DOI: 10.1137/050645221.

Anwar, Sohel (2012): Fault Tolerant Drive By Wire Systems: Impact on Vehicle Safety and Reliability: BENTHAM SCIENCE PUBLISHERS.

Awerbuch, Baruch; Holmer, David; Nita-Rotaru, Cristina; Rubens, Herbert: An on-demand secure routing protocol resilient to byzantine failures. In: Douglas Maughan und Nitin H. Vaidya (Hg.): the ACM workshop. Atlanta, GA, USA, S. 21–30.

Baron, Joshua; El Defrawy, Karim; Lampkins, Joshua; Ostrovsky, Rafail: How to withstand mobile virus attacks, revisited. In: Magnús Halldórsson und Shlomi Dolev (Hg.): the 2014 ACM symposium. Paris, France, S. 293–302.

Basir, O. A.; Shen, H. C.: Sensory data fusion: a team consensus approach. In: [Proceedings] 1992 IEEE International Conference on Systems, Man, and Cybernetics. Chicago, IL, USA, 18-21 Oct. 1992, S. 290–296.

Behrmann, Gerd; David, Alexandre; Larsen, Kim G. (2004): A Tutorial on Uppaal. In: David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell et al. (Hg.): Formal Methods for the Design of Real-Time Systems, Bd. 3185. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture notes in computer science), S. 200–236.

Behrmann, Gerd; David, Re; Larsen, Kim G. (2006): A Tutorial on Uppaal 4.0. Online verfügbar unter http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.222.9120.

Bengtsson, Johan; Larsen, Kim; Larsson, Fredrik; Pettersson, Paul; Yi, Wang (1996): UPPAAL — a tool suite for automatic verification of real-time systems. In: G. Goos, J. Hartmanis, J. van Leeuwen, Rajeev Alur, Thomas A. Henzinger und Eduardo D. Sontag (Hg.):

Hybrid Systems III, Bd. 1066. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture notes in computer science), S. 232–243.

Ben-Or, Michael; Hassidim, Avinatan: Fast quantum byzantine agreement. In: Hal Gabow und Ronald Fagin (Hg.): the thirty-seventh annual ACM symposium. Baltimore, MD, USA, S. 481.

Bousbiba (2015a): ESSEN - An Efficient Single Round Signature Protected Message Exchange Agreement Protocol for Wireless Distributed Networks. Architecture of Computing Systems. Proceedings, ARCS 2015 - The 28th International Conference on. Berlin: VDE Verl.

Bousbiba (2015b): Reducing the Communication Complexity of Agreement Protocols By Applying A New Signature Scheme called SIGSEAM. DEPEND 15 - The Eighth International Conference on Dependability: ThinkMind.

Buhrman, H.; Garay, J. A.; Hoepman, J.-H.: Optimal resiliency against mobile faults. In: Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers. Pasadena, CA, USA, 27-30 June 1995, S. 83–88.

Buhrman, H.; Garay, J. A.; Hoepman, J.-H. (1995): Optimal resiliency against mobile faults. In: *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium*, S. 83–88. DOI: 10.1109/FTCS.1995.466995.

Cachin, Christian; Guerraoui, Rachid; Rodrigues, Luís (2011): Introduction to Reliable and Secure Distributed Programming. Berlin, Heidelberg: Springer Berlin Heidelberg.

Capella, J. V.; Bonastre, A.; Ors, R.: Industrial applications of wireless networks: a bridge crane distributed control system based on bluetooth. In: 2004 IEEE International Conference on Industrial Technology, 2004. IEEE ICIT '04. Hammamet, Tunsia, Dec. 8-10, 2004, S. 824–829.

Chae, M. J.; Yoo, H. S.; Kim, J. Y.; Cho, M. Y. (2012): Development of a wireless sensor network system for suspension bridge health monitoring. In: *Automation in Construction* 21, S. 237–252. DOI: 10.1016/j.autcon.2011.06.008.

Chandra, Tushar Deepak; Toueg, Sam (1996): Unreliable failure detectors for reliable distributed systems. In: *J. ACM* 43 (2), S. 225–267. DOI: 10.1145/226643.226647.

Coelho, Tito E.; Macedo, Ricardo; Carvalhal, Paulo; Afonso, Jose A.; Silva, Luis F.; Almeida, Heitor et al.: A Fly-By-Wireless UAV Platform Based on a Flexible and Distributed System Architecture. In: 2006 IEEE International Conference on Industrial Technology. Mumbai, India, S. 2359–2364.

Dinh-Khanh Dang; Mifdaoui, Ahlem; Gayraud, Thierry: Fly-By-Wireless for next generation aircraft: Challenges and potential solutions. In: 2012 IFIP Wireless Days (WD). Dublin, Ireland, S. 1–8.

Dolev, D.; Strong, H. R. (1983): Authenticated Algorithms for Byzantine Agreement. In: *SIAM J. Comput.* 12 (4), S. 656–666. DOI: 10.1137/0212045.

Dolev, Danny; Dwork, Cynthia; Stockmeyer, Larry (1987a): On the minimal synchronism needed for distributed consensus. In: *J. ACM* 34 (1), S. 77–97. DOI: 10.1145/7531.7533.

Dolev, Danny; Dwork, Cynthia; Stockmeyer, Larry (1987b): On the minimal synchronism needed for distributed consensus. In: *J. ACM* 34 (1), S. 77–97. DOI: 10.1145/7531.7533.

Dolev, Danny; Strong, H. Raymond (1982): Polynomial algorithms for multiple processor agreement. In: *Lewis, Simons et al. (Hg.) – the fourteenth annual ACM symposium*, S. 401–407. DOI: 10.1145/800070.802215.

Echtle: Distance agreement protocols. In: [1989] The Nineteenth International Symposium on Fault-Tolerant Computing. Digest of Papers. Chicago, IL, USA, 21-23 June 1989, S. 191–198.

Echtle (1990a): Fehlertoleranzverfahren. Berlin, Heidelberg, New York, London, Paris, Tokyo, Hong Kong, Barcelona: Springer (Studienreihe Informatik).

Echtle (1990b): Fehlertoleranzverfahren. Berlin, Heidelberg, New York, London, Paris, Tokyo, Hong Kong, Barcelona: Springer (Studienreihe Informatik).

Echtle (1999): Avoiding Malicious Byzantine Faults by a New Signature Generation Technique. In: Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Jan Hlavička, Erik Maehle und András Pataricza (Hg.): Dependable Computing — EDCC-3, Bd. 1667. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture notes in computer science), S. 106–123.

Echtle (2005): Fehlertolerante Protokolle. Standardprotokolle. Online verfügbar unter http://dc.informatik.uni-essen.de/group/stud/teaching/ft_prot/FP_3q.pdf, zuletzt geprüft am 16.04.2016.

Echtle; Kimmeskamp (2010): Fault-Tolerant and Fail-Safe Control Systems Using Remote Redundancy. In: *FERS-Mitteilungen* 28 (1), S. 38–43. DOI: 10.1007/BF03345442.

Echtle; Silva (1998): Fehlerinjektion - ein Mittel zur Bewertung der Maßnahmen gegen Fehler in komplexen Rechensystemen. In: *Informatik-Spektrum* 21 (6), S. 328–336. DOI: 10.1007/s002870050113.

Elgezabal Gomez, Oroitz: Fly-by-wireless: Benefits, risks and technical challenges. In: 2010 Caneus Fly by Wireless Workshop (FBW). Orono, ME, USA, S. 14–15.

Elrahim, Adel Gaafar A.; Elsayed, Hussein A.; Ramly, Salwa El; Ibrahim, Magdy M.: Improving TCP congestion control for wireless sensor networks. In: 2011 4th Annual Caneus Fly by Wireless Workshop (FBW). Montreal, QC, Canada, S. 1–6.

Fan, Rui; Chakraborty, Indraneel; Lynch, Nancy (2004): Clock Synchronization for Wireless Networks. In: *IN PROC. 8TH INTERNATIONAL CONFERENCE ON PRINCIPLES OF DISTRIBUTED SYSTEMS (OPODIS*, S. 400–414. Online verfügbar unter http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.116.203.

Fischer, Michael J.; Lynch, Nancy A.; Paterson, Michael S. (1985a): Impossibility of distributed consensus with one faulty process. In: *J. ACM* 32 (2), S. 374–382. DOI: 10.1145/3149.214121.

Fischer, Michael J.; Lynch, Nancy A.; Paterson, Michael S. (1985b): Impossibility of distributed consensus with one faulty process. In: *J. ACM* 32 (2), S. 374–382. DOI: 10.1145/3149.214121.

Fischer, Michael J.; Lynch, Nancy A.; Paterson, Michael S. (1985c): Impossibility of distributed consensus with one faulty process. In: *J. ACM* 32 (2), S. 374–382. DOI: 10.1145/3149.214121.

Garay, Juan A.; Moses, Yoram (1998): Fully Polynomial Byzantine Agreement for Processors in Rounds. In: *SIAM J. Comput.* 27 (1), S. 247–290. DOI: 10.1137/S0097539794265232.

Ghajari, Houman; Yu, Haobin; Nummey, David; Mahbobi, Kamran: Airborne-ViFi™: Airborne V-band wireless network structure. In: 2011 4th Annual Caneus Fly by Wireless Workshop (FBW). Montreal, QC, Canada, S. 1–4.

Guerrero, Jose Alfredo; Garcia, Pedro Castillo; Challal, Yacine (2013): Quadrotors Formation Control. In: *J Intell Robot Syst* 70 (1-4), S. 221–231. DOI: 10.1007/s10846-012-9707-4.

Halldórsson, Magnús; Dolev, Shlomi (Hg.): the 2014 ACM symposium. Paris, France.

Hsieh, Hui-Ching; Chiang, Mao-Lun (2011): A new solution for the Byzantine agreement problem. In: *Journal of Parallel and Distributed Computing* 71 (10), S. 1261–1277. DOI: 10.1016/j.jpdc.2011.06.002.

Jalali, Mohsen; Moupfouma, Fidele; Dauterstedt, Sven; Wuthrich, Rolf: More wireless systems onboard of aircraft with metallic nanoparticles. In: 2011 4th Annual Caneus Fly by Wireless Workshop (FBW). Montreal, QC, Canada, S. 1.

Jeffery, Casey M.; Figueiredo, Renato J. O.: Towards Byzantine Fault Tolerance in Many-Core Computing Platforms. In: 13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007). Melbourne, Australia, S. 256–259.

Jochim, Markus; Forest, Thomas M. (2010): An Efficient Implementation of the SM Agreement Protocol for a Time Triggered Communication System. In: *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 3 (2), S. 106–116. DOI: 10.4271/2010-01-2320.

Kai Wang; Aidong Xu; Hong Wang: Avoiding the babbling idiot failure in a communication system based on flexible time division multiple access: A bus guardian solution. In: 2009 IEEE International Symposium on Industrial Electronics (ISIE 2009). Seoul, South Korea, S. 1292–1297.

Kai Wang; Aidong Xu; Hong Wang: Design and quantitative evaluation of a novel FlexRay Bus Guardian. In: 2010 IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS 2010). Xiamen, China, S. 782–786.

Khosravi, A.; Kavian, Y. S. (2012): Reaching an agreement in a distributed environment in absence of omission faults. In: *2012 8th International Symposium*, S. 1–4. DOI: 10.1109/CSNDSP.2012.6292709.

Koptez, H. (1997): Real-Time Systems.

Kotla, Ramakrishna; Alvisi, Lorenzo; Dahlin, Mike; Clement, Allen; Wong, Edmund: Zyzzyva. In: Thomas C. Bressoud und M. Frans Kaashoek (Hg.): twenty-first ACM SIGOPS symposium. Stevenson, Washington, USA, S. 45.

Kurose, James F.; Ross, Keith W. (2014): Computernetzwerke. Der Top-Down-Ansatz. 6., aktualisierte Aufl. Hallbergmoos: Pearson Studium (Pearson Studium - Informatik).

Lamport, L. (1983): The Weak Byzantine Generals Problem. In: *J. ACM* 30 (3), S. 668–676. DOI: 10.1145/2402.322398.

Lamport, Leslie; Shostak, Robert; Pease, Marshall (1982): The Byzantine Generals Problem. In: *ACM Trans. Program. Lang. Syst.* 4 (3), S. 382–401. DOI: 10.1145/357172.357176.

Lasassmeh, Sami M.; Conrad, James M.: Time synchronization in wireless sensor networks: A survey. In: SOUTHEASTCON 2010. Concord, NC, USA, S. 242–245.

L'Ecuyer, Pierre (1998): Random Number Generation. In: Jerry Banks (Hg.): Handbook of Simulation. Hoboken, NJ, USA: John Wiley & Sons, Inc, S. 93–137.

Leitenberger, B. (2015): Das ATV und die Versorgung der ISS: Die Versorgungssysteme der Raumstation: Books on Demand. Online verfügbar unter https://books.google.de/books?id=aUZzBwAAQBAJ.

Leu, Martin (1994): Relative signatures for fault tolerance and their implementation. In: Gerhard Goos, Juris Hartmanis, Jan Leeuwen, Klaus Echtle, Dieter Hammer und David Powell (Hg.): Dependable Computing — EDCC-1, Bd. 852. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture notes in computer science), S. 561–580.

Liu, Hugh H.T.: Aircraft wireless sensor network for in-flight fault diagnostics and control. In: 2010 Caneus Fly by Wireless Workshop (FBW). Orono, ME, USA, S. 22–23.

Lynch, Nancy A. (2010): Distributed algorithms. transf. to digital print. San Francisco, Calif.: Elsevier, Kaufmann (The Morgan Kaufmann series in data management systems).

Malcolm, Michael; Strong, Ray (Hg.) (1985): Simple constant-time consensus protocols in realistic failure models. Proceedings of the fourth annual ACM symposium on Principles of distributed computing. Minaki, Ontario, Canada. New York: ACM.

Mei-Chen Hsueh; Tsai, T. K.; Iyer, R. K. (1997): Fault injection techniques and tools. In: *Computer* 30 (4), S. 75–82. DOI: 10.1109/2.585157.

Michael Ben-Or (2005): Fast quantum byzantine agreement. In: *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, S. 481–485. DOI: 10.1145/1060590.1060662.

Moniz, Henrique; Neves, Nuno F.; Correia, Miguel (2013): Byzantine Fault-Tolerant Consensus in Wireless Ad Hoc Networks. In: *IEEE Trans. on Mobile Comput.* 12 (12), S. 2441–2454. DOI: 10.1109/TMC.2012.225.

Nazreen Banu; Samia Souissi; Taisuke Izumi (2012): An Improved Byzantine Agreement Algorithm for Synchronous Systems with Mobile Faults. In: *An Improved Byzantine Agreement Algorithm for Synchronous Systems with Mobile Faults* Volume 43 - Number 22, 2012, S. 1–7. Online verfügbar unter http://www.ijcaonline.org/archives/volume43/number22/6400-8878, zuletzt geprüft am 01.02.2016.

Qu, Yaohong; Zhu, Xu; Zhang, Youmin M. (2012): Cooperative Control for UAV Formation Flight Based on Decentralized Consensus Algorithm. In: David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell et al. (Hg.): Intelligent Robotics and Applications, Bd. 7506. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture notes in computer science), S. 357–366.

Rabin, Michael O.: Randomized byzantine generals. In: 24th Annual Symposium on Foundations of Computer Science (sfcs 1983). Tucson, AZ, USA, S. 403–409.

Rausch, Mathias (2008): FlexRay. Grundlagen, Funktionsweise, Anwendung : 59 Tabellen. München [u.a.]: Hanser.

Razi, Abolfazl; Abedi, Ali: Interference reduction in Wireless Passive Sensor Networks using directional antennas. In: 2011 4th Annual Caneus Fly by Wireless Workshop (FBW). Montreal, QC, Canada, S. 1–4.

Saini, Poonam; Singh, Awadhesh Kumar: Hash Based Byzantine Fault Tolerant Agreement with Enhanced View Consistency. In: 2011 12th International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT). Gwangju, South Korea, S. 368–374.

Santoro, Nicola (2006): Distributed Algorithms for Autonomous Mobile Robots. In: Gonzalo Navarro, Leopoldo Bertossi und Yoshiharu Kohayakawa (Hg.): Fourth IFIP International Conference on Theoretical Computer Science- TCS 2006, Bd. 209. Boston, MA: Springer US (IFIP International Federation for Information Processing), S. 11.

Santoro, Nicola; Widmayer, Peter (1989): Time is not a healer. In: B. Monien und R. Cori (Hg.): STACS 89, Bd. 349. Berlin/Heidelberg: Springer-Verlag (Lecture notes in computer science), S. 304–313.

Santoro, Nicola; Widmayer, Peter (2007): Agreement in synchronous networks with ubiquitous faults. In: *Structural Information and Communication Complexity (SIROCCO 2005)* 384 (2–3), S. 232–249. DOI: 10.1016/j.tcs.2007.04.036.

Tanenbaum, Andrew S.; van Steen, Maarten (2008): Verteilte Systeme. Prinzipien und Paradigmen. 2., aktualisierte Aufl. München [u.a.]: Pearson Studium (Informatik).

Temple, C. (1998): Avoiding the babbling-idiot failure in a time-triggered communication system, S. 218–227. DOI: 10.1109/FTCS.1998.689473.

Tulone, Daniela (2004): Enhancing Efficiency of Byzantine-Tolerant Coordination Protocols via Hash Functions. In: David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell et al. (Hg.): Euro-Par 2004 Parallel Processing, Bd. 3149. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture notes in computer science), S. 587–595.

Walter Lange, Martin Bogda: Entwurf und Synthese von Eingebetteten Systemen: Ein Lehrbuch.

Wang, Na; He, Haihui: Time Synchronization for Failure Tolerance in Wireless Sensor Network. In: 2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD). Kyoto, Japan, S. 181–184.

Warns, Timo (2010): Structural Failure Models for Fault-Tolerant Distributed Computing. Wiesbaden: Vieweg+Teubner.

Welch, Jennifer Lundelius; Lynch, Nancy (1988): A new fault-tolerant algorithm for clock synchronization. In: *Information and Computation* 77 (1), S. 1–36. DOI: 10.1016/0890-5401(88)90043-0.

Wörn, Heinz: Echtzeitsysteme. Grundlagen, Funktionsweisen, Anwendungen.

Young, Maxwell; Boutaba, Raouf (2011): Overcoming Adversaries in Sensor Networks: A Survey of Theoretical Models and Algorithmic Approaches for Tolerating Malicious Interference. In: *IEEE Commun. Surv. Tutorials* 13 (4), S. 617–641. DOI: 10.1109/SURV.2011.041311.00156.

Anhang A: Kurzreferenz für ESSEN

Empfang einer **Data**-Na x:

Wenn x enthält nicht die Sig des Quellknotens *NC1*
oder x enthält keine Sig aus BasisG und Empfänger ∈ ExtG *NC2*
oder SigAnz(x) < f + 1 und SigAnz(x) ≤ SigAnz(PMB) und SigAnz(x) ≤ SigAnz(SMB)
dann ➔ x verwerfen *SC1, SC2*
sonst wenn SigAnz(x) > SigAnz(PMB) *SC2*
dann ➔ x in **PMB** speichern
sonst wenn SigMenge(x) \ SigMenge(PMB) ≠ ∅ und SigMenge(x) > SigMenge(SMB) *SC2*
dann ➔ x in **SMB** speichern
sonst ➔ x verwerfen

Empfang einer **Default**-Na x:

Wenn x enthält die Sig des Quellknotens oder eines Basis-Knotens *NC3*
oder SigAnz(x) < SigAnz(DMB) *SC3*
dann ➔ x verwerfen
sonst ➔ x in **DMB** speichern

Senden:

Wenn	Sender ist Quellknoten	dann ➔	erzeuge und sende Data-Na	
sonst wenn	SigAnz(PMB) > SigAnz(DMB)	dann ➔	sende Data-Na aus **PMB**	*MD1*
sonst wenn	DMB enthält Default-Na	dann ➔	sende Default-Na aus **DMB**	*MD2*
sonst wenn	DMB ist leer	dann ➔	erzeuge und sende Default-Na	

Entscheidung:

Wenn SigAnz(PMB) < f + 1 dann ➔ entscheide für **Ersatzwert** *AG1*
sonst ➔ gemeinsame Sig in PMB und DMB aus der Na in PMB ignorieren, *CF*
gemeinsame Sig in SMB und DMB aus der Na in SMB ignorieren

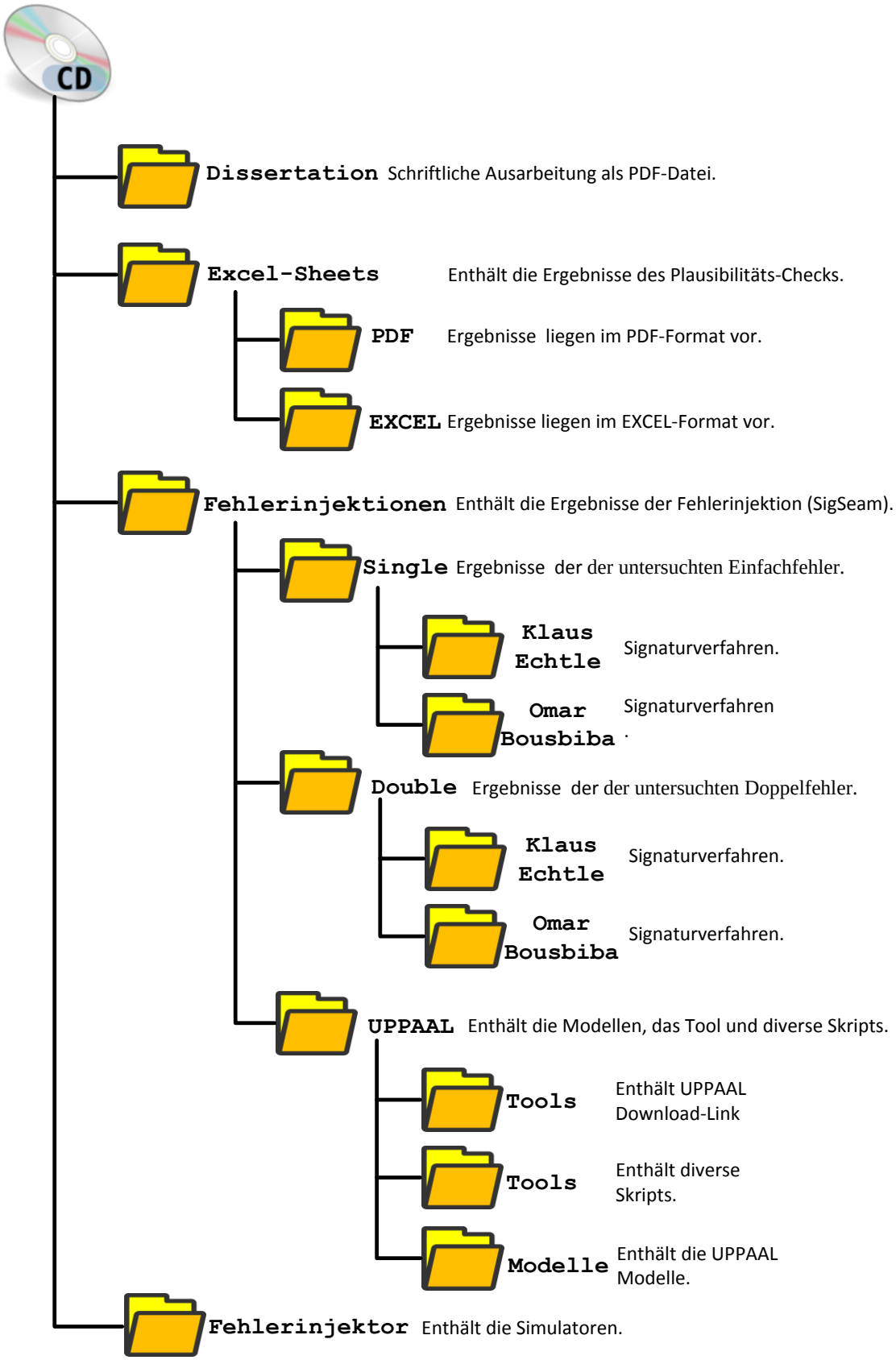
Wenn	SigAnz(PMB) < f	dann ➔	entscheide für **Ersatzwert**	*AG2*
sonst wenn	SigAnz(SMB) ≤ f	dann ➔	entscheide für **Ersatzwert**	*AG3*
sonst ➔			entscheide für **PMB**	

(Für den Wert in SMB wird niemals entschieden)

Anzahl der Knoten: 1 Quellknoten, f + 1 Knoten in BasisG, 2(f – 1) + max(0, f – 2) Knoten in ExtG:

Knotengruppe	Vorgänger von v_P^e	Nachfolger von v_P^e
$V''_{ff,Data}$	$Vorg(v_P^e) \cap V''_{ff,Data} = \mathrm{P}$	$Nachf(v_P^e) \cap V''_{ff,Data} = \emptyset$
$V''_{ff,Default}$	$Vorg(v_P^e) \cap V''_{ff,Default} = \mathrm{R}$	$Nachf(v_P^e) \cap V''_{ff,Default} = \mathrm{S}$
$V''_{fh,Data}$	$Vorg(v_P^e) \cap V''_{fh,Data} \subseteq \Theta$	$Nachf(v_P^e) \cap V''_{fh,Data} \subseteq \Theta$
$V''_{fh,Default}$	$Vorg(v_P^e) \cap V''_{fh,Default} = \mathrm{Q}$	$Nachf(v_P^e) \cap V''_{fh,Default} = \mathrm{T}$

Anhang B: Inhalt der beiliegenden CD-ROM



Glossar

Symbole für Übereinstimmungsprobleme und Akteure

actSinkG	Die Menge der aktiven Knoten (dürfen Nachrichten versenden), die Übereinstimmung gemäß IC1 und IC2 erzielen müssen. Die Menge actSinkG ist die Vereinigung der Mengen BasisG und ErwG (kurz: $actSinkG = BasisG \cup ErwG$).
BAP	Die Klasse von Übereinstimmungsproblemen, die die Lösung des Problems der Byzantinischen Übereinstimmung (Byzantine Agreement Problem) zum Gegenstand haben ($BAP \subset ICP$). BAP löst das Single – Commander Problem, wobei gilt: $\|QuellG\| = 1$.
BasisG	Die Menge der Versender-Knoten (eine Teilmenge von „lieutnant generals“ in (Lamport et al. 1982)). Ein Knoten aus dieser Menge empfängt und leitet den empfangenen Wert konsistent und zuverlässig an alle anderen Knoten weiter. Die Mengen QuellG und BasisG müssen nicht disjunkt sein.
CP	Die Klasse von Übereinstimmungsproblemen, die die Lösung des Konsens-Problems (Consensus Problem) zum Gegenstand haben. CP ist auch als schwaches interaktives Konsistenzproblem bekannt (Lamport 1983).
ErwG	Die Menge der Versender-Knoten (eine Teilmenge von „lieutnant general“ in (Lamport et al. 1982)). Knoten in ErwG besitzen im Gegensatz zu denen aus BasisG zusätzliche Privilegien, nämlich die Möglichkeit „Default“-Nachrichten zu versenden. Weiterhin gilt: $BasisG \cap ErwG = \emptyset$. Die Mengen QuellG und ErwG müssen nicht disjunkt sein.

ForwG	Die Menge der aktiven Knoten (vgl. „lieutnant general“ in (Lamport et al. 1982)), die zur Menge BasisG und ErwG gehören (kurz: ForwG = BasisG ⋃ ErwG).
G	Die Menge aller passiven und aktiven Knoten.
IC1	Notwendige Bedingung zur Lösung des Übereinstimmungsproblems im Falle eines fehlerfreien Quellknotens. Hinreichende Bedingung im Falle eines fehlerhaften Quellknotens.
IC2	Notwendige Bedingung zur Lösung des Übereinstimmungsproblems im Falle eines fehlerfreien Quellknotens.
ICP	Die Klasse von Übereinstimmungsproblemen, die das Problem der interaktiven Konsistenz (Interactive Consistency Problem) zum Gegenstand hat. ICP löst das Multiple-Commander.-Problem, wobei gilt: |QuellG| > 1.
k-CP	Abgeschwächte Form des Konsens-Problems. Statt n – f müssen mindestens k > $\frac{n}{2}$ fehlerfreie Knoten einen Konsens erzielen.
purSinkG	Die Menge der passiven Knoten (dürfen keine Nachrichten versenden), die Übereinstimmung gemäß IC1 und IC2 erzielen müssen. Die Menge PurSinkG ist keine Teilmenge von QuellG, ErwG und BasisG.
QuellG	Die Menge der Quellknoten (vgl. „commander general“ in (Lamport et al. 1982). Jeder fehlerfreie Knoten aus dieser Menge verteilt seinen Wert konsistent und zuverlässig an alle anderen Knoten.
SendG	Die Gesamtmenge der aktiven Knoten („commander general“ und „lieutnant general“). Folglich ist die Menge

	SendG die Vereinigungsmenge von QuellG, BasisG und ErwG (kurz: SendG = QuellG ∪ BasisG ∪ ErwG).
SinkG	Die Gesamtmenge der Knoten, die Übereinstimmung gemäß IC1 und IC2 erzielen müssen. Folglich ist die Menge SinkG die Vereinigungsmenge von actSinkG und purSinkG (kurz: SinkG = actSinkG ⋃ purSink).

<u>Symbole für Komponenten</u>

b	Die Anzahl der Knoten in Gruppe B. Entspricht die Kardinalität von B.
B = {B[0],…,B[b – 1]}	Die Menge der Knoten, die zu Gruppe BasisG gehören.
CH = {CH[0],…, CH[k – 1]}	Die Menge der zu Verfügung stehenden Kommunikationskanäle.
CH[i]	i-ter Kommunikationkanal, über den die Knoten Nachrichten versenden können.
e	Die Anzahl der Knoten in Gruppe E. Entspricht die Kardinalität von E.
E = {E[0],…,E[e – 1]}	Die Menge der Knoten, die zu Gruppe ErwG gehören.
g	Gesamtzahl aller passiven und aktiven Knoten, die am Übereinstimmungsprotokoll teilnehmen (g entspricht der Kardinalität von G).
k	Die Anzahl der zu Verfügung stehenden Kanäle (k entspricht der Kardinalität von CH).
q	Die Anzahl der Knoten in Gruppe Q q (entspricht der Kardinalität von Q).
Q = {Q[0],…,Q[q – 1]}	Die Menge der im Protokoll vorkommenden Quellknoten mit q ≥ 1.

Q[i]	i-ter Quellknoten, identifiziert anhand seines Index i.
n	Anzahl der Knoten, die aktiv am Übereinstimmungsprotokoll teilnehmen, d.h. Nachrichten senden (n entspricht der Kardinalität von N).
N = {N[0],…, N[n – 1]}	Die Menge der Knoten die aktiv am Übereinstimmungsprotokoll teilnehmen, d.h. Nachrichten versenden.
N[i]	Protokollausführender Knoten, identifiziert anhand seines Index i.

Symbole für Nachrichten

A	Entspricht den Anfangswert des Quellknotens.
A[i]	Anfangswert des Quellknotens i.
K	Konsistenzvektor. Dieser speichert den Anfangswert aller am Protokoll teilnehmenden Quellknoten ab. Terminiert das Übereinstimmungsprotokoll, dann enthält K für einen Knoten i die finalen Übereinstimmungswerte aller vorkommenden Quellknoten.
K[i] = {K[i][0],…, K[i][n – 1]}	Der von Knoten i gespeicherte Konsistenzvektor. Jeder Eintrag K[i][j] im Vektor K[i] entspricht dem Anfangswert A[j] eines Quellknotens j.
MN = {M[0],…, M[n – 1]}	die Menge der übertragenen Broadcast-Nachrichten. Pro aktivem Knoten wird im Protokoll ESSEN genau eine Nachricht übertragen.
M[i]	Die von Knoten N[i] gesendete Nachricht.
M[i][j]	Die von Knoten N[i] gesendeet und von Knoten N[j] empfangene Nachricht. Falls der Knoten N[i] fehlerhaft ist, dann ist die Bedingung M[i][j] = M[i][k] mit j ≠ k für

	zwei beliebige fehlerfreie Empfangsknoten N[j] und N[k] nicht immer erfüllt.
M	Eine beliebige Nachricht. Die Bezeichnung M ist keinem Knoten zugeordnet. Der Begriff M wird verwendet, wenn der Absender bzw. der Empfänger der Nachricht nicht im Fokus steht.
M_{data}	Steht für eine Data-Nachricht. Das Symbol M wird verwendet, wenn der Absender bzw. der Empfänger der Nachricht nicht im Fokus steht.
$M_{default}$	Steht für eine Default-Nachricht. Das Symbol M wird verwendet, wenn der Absender bzw. der Empfänger der Nachricht nicht im Fokus steht.
Max(A[i],…, A[j])	Bestimmt aus den Werten A[i] bis A[j] mit i ≤ j den größten Anfangswert.
Min(A[i],…, A[j])	Bestimmt aus den Werten A[i] bis A[j] mit i ≤ j den kleinsten Anfangswert.

Symbole für fehlerhafte Komponenten

f	Gesamtzahl der fehlerhaften Komponenten (Knoten und kanalbedingte Nachrichtenausfälle pro Slot).
f_B	Anzahl der zu tolerierenden Blockierungsfehler. Bei einem Blockierungsfehler verhält sich der betroffene fehlerfreie Sendeknoten ähnlich wie ein gutartiger Byzantinisch fehlerhafter Knoten, weil alle Kanäle fehlerhaft sind, über die er sendet.
f_N	Gesamtzahl der fehlerhaften Knoten.
FN = {FN[0],…, FB[$f_N - 1$]}	Die Menge der fehlerhaften Knoten. Beispiel für f_N = 2: FN[1] = N[3], FN[2] = N[5].

FN[i]	Fehlerhafte Knoten mit Index i.

Symbole für Fehlertypen

FM = {FM0,…,FMi}	Die Menge der zu betrachtenden Fehlermodi.
FMi	Fehlermodus i.
FMAi	Addition (Einfügefehler) in Fehlermode FMi.
FMCi	Corruption (Verfälschung) in Fehlermode FMi.
FMOi	Omission (Nachrichtenausfall) in Fehlermode FMi.

Symbole für Kommunikationszeiten

bcr	Die Anzahl der reservierten Slots pro Kommunikationsrunde. Es gilt: bcr = | Broadcastrunde |.
Broadcastrunde = {Slot[i],…,Slot[j]}	Die Menge der reservierten Slots pro Kommunikationsrunde.
Slot[i]	Sendeslot für einen Knoten N[i]. Nur Knoten N[i] hat Zugriff auf den Slot, um eine Sendeoperation auszuführen. Alle anderen haben nur lesenden Zugriff, d.h. sie empfangen Nachrichten.

spezifische Symbole für das Protokoll ESSEN

Data	Teil der Anfangsnachricht, enthält mindestens den Anfangswert und den dazugehörigen Signaturwert des Quellknotens der Nachricht.
Data[i]	Die von Knoten N[i] weitergeleitete Anfangsnachricht.
Default	Default-Nachricht (um ein Veto auszudrücken), enthält nur Signaturen aus der Menge E.
Default[i]	Die von Knoten N[i] gesendete Default-Nachricht.

DMB[i]	Default Message Buffer (Default Nachrichtenpuffer) von Knoten N[i].
MsgPMB[i]	Der Inhalt von PMB [i] (die in PMB[i] gespeicherte Nachricht).
MsgSMB[i]	Der Inhalt von SMB[i] (die in SMB[i] gespeicherte Nachricht).
MsgDMB[i]	Der Inhalt von DMB[i] (die in DMB[i] gespeicherte Nachricht).
PMB[i]	Primary Message Buffer (Primärer Nachrichtenpuffer) von Knoten N[i].
SMB[i]	Secondary Message Buffer (Sekundärer Nachrichtenpuffer) von Knoten N[i].

Funktionen und Werte

$\sigma_i(\ldots)$	Die von Knoten N[i] verwendete Signaturfunktion.
$\sigma_i(v) \circ \sigma_j(v)$	Die von Knoten N[i] und N[j] signierte Nachricht v.
$\sigma_i(\sigma_j(v))$	Äquivalent zu $\sigma_i(v) \circ \sigma_j(v)$.
$\tau_i(\ldots)$	Die von Knoten N[i] verwendete Prüffunktion. Die Prüffunktion liefert genau dann den Wert *False*, wenn eine Verfälschung in der Nachricht erkannt wird, sonst liefert sie den Wert *True*. Aus *True* kann geschlossen werden, dass die Nachricht (mit hoher Wahrscheinlichkeit) ohne jegliche Verfälschungen korrekt übertragen wurde.
CRC(…)	Funktion, die zur Berechnung des CRC aufgerufen wird. Alle innerhalb der Klammer angegebenen Werte werden als ein einziger Bitvektor interpretiert. Bei Aufruf von CRC(x, y) für zwei Bitvektoren x und y, wird zunächst x

	und y zu einem Bitvektor konkateniert, bevor der CRC von x und y bestimmt wird, d. h. CRC(x,y) ist identisch zu CRC(x∘y).
equalContent(M[i], M[j])	Die Funktion equalContent(M[i], M[j]) liefert *True*, falls die Nutzdaten der beiden Nachrichten M[i] und M[j] inhaltlich gleich sind, sonst *False*.
Input()	Funktion, die den zu verteilenden Anfangswert A[i] liefert, wenn Knoten N[i] ein Quellknoten ist und diese Funktion aufruft. Nicht-Quellknoten sind nicht autorisiert, diese Funktion aufzurufen.
max(x, y)	Liefert den größeren von zwei Werten x, y $\in \mathbb{N}_0$.
M[i].isSigned(u_i)	Liefert den Wert *True* genau dann, wenn die Nachricht M[i] von Knoten N[i] signiert wurde, sonst liefert sie den Wert *False*.
receive()	Liefert die vom Broadcast-Kanal empfangene Nachricht.
s_i	Privater Signierschlüssel von Knoten N[i].
sig(M[i] $\in$ M)	Liefert die Menge der Knoten, die die Nachricht M[i] signiert haben, kurz: sig(MsgPMB)$\subseteq$ SendG.
sigCount(M[i] $\in$ M)	Liefert die Anzahl der in M[i] enthaltenen Signaturen.
syntacticallyWrong(M[i])	Liefert den Wert *True* genau dann, wenn die Nachricht M[i] keine syntaktischen Fehler enthält, sonst liefert sie den Wert *False*.
t_E	Schwellwert eines Knotens. t_E gibt die minimale Anzahl an Signaturen an, die die Quellnachricht (Nachricht des Quellknotens) mindestens enthalten muss, um vom entsprechenden Knoten weitergeleitet zu werden.

u_i	Öffentlicher Schlüssel von Knoten N[i].
unequal(M[i], M[j])	Die Funktion unequal (...) liefert *True*, falls die Nutzdaten der beiden Nachrichten voneinander verschieden sind, sonst *False*.
v: V(i, j)	Die von Knoten N[i] und N[j] signierte Nachricht v. Die Funktion V wird als Verschmelzungsfunktion bezeichnet und bildet die beiden Signaturen $\sigma_i(v)$ und $\sigma_j(v)$ auf eine neue signierte Nachricht v: V(i, j) ab.
valueOf(M[i])	Extrahiert die Nutzdaten aus der Nachricht M[i].
Value	Nutzdaten
validSig(M[i])	Prüft die Korrektheit der empfangenen Nachricht. Die Prüffunktion liefert genau dann den Wert *False*, wenn eine Verfälschung in der Nachricht erkannt wird, sonst liefert sie den Wert *True*. Aus *True* kann geschlossen werden, dass die Nachricht ohne jegliche Verfälschung korrekt übertragen wurde.
#(M)	Liefert die Anzahl der Signaturquellen von M als Ergebnis. Dies ist äquivalent zu dem funktionswert von sigCount(M).
$<e_f>$	Liefert das Bildungsgesetz für die Knotenzahl in der Gruppe ErwG. Der Index f gibt die Anzahl der zu tolerierenden Fehler an.
$<b_f>$	Liefert das Bildungsgesetz für die Knotenzahl in Gruppe BasisG. Der Index f gibt die Anzahl der zu tolerierenden Fehler an.

Signaturbezogene Symbole

a_i	Privater Signierschlüssel von Knoten N[i].

b_i	Teil des öffentlichen Testschlüssels zur Prüfung der Signatur von Knoten N[i].
c_i	Zweiter Teil des öffentlichen Testschlüssels zur Prüfung der Signatur.
D	Äquivalent zu Value.
SW	Signaturwert.
SeqNr	Sequenznummer einer Nachricht.
SL	Signaturliste.
SL_i	Entspricht den i-ten Eintrag in der Liste SL. Der Wert von SL_i fließt in die Signaturprüfung mit ein.

Abkürzungen

CC	Kommunikationskomplexität („communication complexity“). Diese beschreibt den Aufwand, den man benötigt, um das Übereinstimmungsproblem zu lösen. Sie lässt sich durch die folgenden drei Parameter Nachrichtenoverhead CS („communication step“), Speicheroverhead SH und Anzahl der benötigten Kommunikationsrunden CR spezifizieren.
CCM	Nachrichtenoverhead. Dieser beschreibt die Anzahl der Nachrichten, die in einer Phase ausgetauscht werden.
CCR	Kommunikationsrunde. Diese beschreibt die Anzahl der benötigten Kommunikationsschritte, um das Übereinstimmungsproblem zu lösen. In der Regel gilt CR = $f + 1$, wobei f die Anzahl der zu tolerierenden Fehler im System ist.

CCS	Speicheroverhead. Dieser gibt die Anzahl der zu speichernden Nachrichten pro Knoten an.

Protokolleigenschaften von ESSEN

CF(M_{data}, $M_{default}$)	Cleaning Function. Wenn es mindestens eine gemeinsame Signaturquelle sq in einer der beiden Nachrichten M_{data} und $M_{default}$ (Data und Default) gibt, dann wird die Signaturquelle sq aus der Data-Nachricht M_{data} entfernt.
AG1	Bedingung: Bevor CF ausgeführt wird, enthält die Data-Nachricht mindestens $f + 1$ Signaturen. Andernfalls (d.h. bei ¬AG1) entscheidet sich der jeweilige fehlerfreie Knoten für eine vordefinierte globale Konstante.
AG2	Bedingung: Nach Ausführung von CF enthält die Data-Nachricht in MsgPMB mindestens f Signaturen.
AG3	Bedingung: Nach Ausführung von CF enthält die Data-Nachricht in MsgPMB höchstens $f - 1$ Signaturen und die Nachricht in MsgSMB mindestens $f + 1$ Signaturen. Sonst (d.h. bei ¬AG3) entscheidet sich der jeweilige fehlerfreie Knoten für eine vordefinierte globale Konstante.
MD1	Bedingung: Ein fehlerfreier Knoten aus ErwG überträgt die MsgPMB Nachricht genau dann, wenn die Anzahl der Signaturen in MsgPMB größer ist als die in MsgDMB.
MD2	Bedingung: Ein fehlerfreier Knoten aus ErwG sendet die Default-Nachricht MsgDMB genau dann, wenn sie mindestens soviele Signaturen enthält wie die in MsgPMB gespeicherte Nachricht.
SC1(RxMsg)	Bedingung: Die Data-Nachricht RxMsg enthält mehr Signaturen als vom jeweiligen Empfangsknoten zuvor in MsgPMB oder MsgSMB gespeichert worden sind. Andernfalls (d.h. bei ¬SC1) wird die Nachricht verworfen.
SC2(RxMsg)	Bedingung: Die Default-Nachricht RxMsg enthält mehr Signaturen als vom jeweiligen Empfangsknoten zuvor

	gespeichert worden sind. Andernfalls (d.h. bei ¬SC2) wird die Nachricht verworfen.
NC1	Bedingung: Eine Data-Nachricht enthält mindestens die Signatur des Quellknotens. Andernfalls (d.h. bei ¬NC1) wird die Nachricht von jedem fehlerfreien Knoten aus BasicG verworfen.
NC2	Bedingung: Eine Data-Nachricht enthält mindestens die Signatur des Quellknotens und mindestens eine Signatur von einen Knoten aus BasisG. Andernfalls (d.h. bei ¬NC2) wird die Nachricht von keinem fehlerfreien Knoten aus ErwG akzeptiert.
NC3	Bedingung: Eine Default-Nachricht wird als valide betrachtet, wenn sie ausschließlich von Knoten aus ErwG signiert bzw. cosigniert wurde. Andernfalls (d.h. bei ¬NC3) wird die Nachricht von keinen fehlerfreien Knoten aus SendG akzeptiert.

Abbildungsfunktionen

$\hat{b} : \{0,\ldots, b-1\} \rightarrow \{0,\ldots, n-1\}$	ist eine injektive Abbildung, die jedem Basisknoten-Index den entsprechenden Knotenindex zuordnet.
$\hat{e} : \{0,\ldots, e-1\} \rightarrow \{0,\ldots, n-1\}$	ist eine injektive Abbildung, die jedem Knoten-Index aus Erw den entsprechenden Knotenindex zuordnet.
$\hat{f} : \{0,\ldots, f_N-1\} \rightarrow \{0,\ldots, n-1\}$	ist eine injektive Abbildung, die jedem Fehlerknoten-Index den entsprechenden Knotenindex zuordnet.
$\hat{q} : \{0,\ldots, q-1\} \rightarrow \{0,\ldots, n-1\}$	ist eine injektive Abbildung, die jedem Quellknoten-Index den entsprechenden Knotenindex zuordnet.

Sonstige Symbole

$x \gg y$	x ist viel größer als y.