Zeit und Raum in Rechnernetzen

Rüdiger Reischuk
Med. Universität zu Lübeck*

April 1996

Sie (die Zeit) hat nur Eine Dimension: verschiedene Zeiten sind nicht zugleich, sondern nach einander (so wie verschiedene Räume nicht nach einander, sondern zugleich sind). . . . Hier füge ich noch hinzu, daß der Begriff der Veränderung und, mit ihm, der Begriff der Bewegung (als Veränderung des Orts) nur durch und in der Zeitvorstellung möglich ist.

Immanuel Kant, Kritik der reinen Vernunft:
Der Transzendentalen Ästhetik zweiter Abschnitt Von der Zeit

Verteilte Rechnersysteme erweisen sich als eine attraktive Architektur mit einer Reihe von Vorteilen. Vor allem ist hier zu nennen die lokale Verfügbarkeit und die Möglichkeit des dezentralen Managements, so daß bei Ausfall einzelner Komponenten nicht zwangsläufig das gesamte System zum Stillstand kommen muß. Andererseits erzeugt gerade diese räumliche Trennung eine neuartige Synchronisationsproblematik. Wir diskutieren exemplarisch zwei Prototypprobleme: Konsistenz bei divergenter lokaler Information – auch Consensus oder das Problem der Byzantinischen Generäle genannt – und die zeitliche Synchronisation der lokalen Uhren in den einzelnen Knoten und gemeinsamer Aktionen. Grundlegend hierfür ist eine präzise Analyse wie aus lokalem Wissen der einzelnen Komponenten globales Wissen über den Gesamtzustand des Systems gewonnen werden kann, und zwar durch den Austausch von Nachrichten und das Eintreffen oder Nichteintreffen von Ereignissen.

Es zeigt sich, daß in Abhängigkeit von bestimmten Systemeigenschaften diese Ziele nur bis zu einem gewissen Grad erreichbar sind. Dabei können kryptografische und probabilistische Verfahren hilfreich sein. Auf die wesentlichen Systemparameter und ihre Bedeutung wird näher eingegangen. Fehlertoleranz ist in manchen Fällen überhaupt nicht erzielbar. Wir stellen in diesem Aufsatz einige der grundlegenden Ergebnisse und methodischen Ansätze vor.

1 Die Bedeutung Verteilter Systeme

Vernetzte Cluster von autonomen Rechnern sind heutzutage die beherrschende System-Architektur. Dies gilt nicht nur für Forschungseinrichtungen wie Universitäten und Forschungsgesellschaften, die bei dem Wandel weg vom zentralen Großrechner eine Vorreiterrolle übernommen haben, sondern inzwischen auch für die meisten Industrieunternehmen, die auf eine Vielzahl leistungsfähiger EDV-Arbeitsplätze angewiesen sind. Wenn wir etwas mehr globalisieren, so wäre das weltweite Internet mit seinen hunderttausenden von intelligenten Knoten als ein weiteres Beispiel anzuführen. Aber auch "im Kleinen" wie etwa in modernen Flug- und Raumfahrzeugen wird diese Architektur zunehmend verwandt.

*Institut für Theoretische Informatik, Wallstraße 40, 23560 Lübeck, Germany
email: reischuk @ informatik.mu-luebeck.de

Dezentrale Verfügbarkeit und die Zusammenarbeit einer Vielzahl von Maschinen kann eine enorme Leistungssteigerung bewirken, beispielsweise bei verteilten Datenbanksystemen. Räumliche Trennung und Duplizierung von Komponenten kann die Zuverlässigkeit des Gesamtsystems wesentlich steigern. Jedoch verlangt jede Art von Zusammenarbeit ein gewisses Maß an Synchronisation der einzelnen Agenten, sowohl im zeitlichen Ablauf als auch bei der Auswahl von Aktionen und Entscheidungen.

Diese Synchronisationsproblematik ist schon seit längerer Zeit ein zentrales Grundlagenthema der Informatik. Zunächst stellt sich die Aufgabe, reale Systeme adäquat zu modellieren. Noch mehr als bei der Untersuchung von sequentiellen Rechnern sind Abstraktionen und vereinfachende Annahmen notwendig, um stringent beweisbare Aussagen von grundsätzlicher Bedeutung überhaupt machen zu können. Solch ein abstraktes Modell wird durch Ausprägungen der wesentlichen Attribute beschrieben. Ein wichtiger Punkt ist die Kommunikation zwischen den einzelnen Komponenten: sei es durch globale Variablen oder durch Versenden von Nachrichten, im Fall von Nachrichten etwa durch paarweisen Informationsaustausch (eine Nachricht ist für genau einen Empfänger bestimmt) oder in Form von Broadcasts (das gleichzeitige Senden an mehrere Empfänger) und, wie schnell und wie zuverlässig eine einzelne Nachricht übermittelt wird.

Für ein derartig spezifiziertes System kann man dann versuchen, geeignete Verfahren (d.h. Algorithmen, man spricht in diesem Fall auch von Protokollen) zu entwickeln, um die verschiedenartigen Synchronisationsaufgaben effizient zu lösen, beispielsweise zu erkennen, wann alle Komponenten des Systems die Bearbeitung einer Aufgabe beendet haben oder ob das System in einen unerwünschten Zustand wie einen Deadlock geraten ist. Dabei scheiden simple zentralistische Lösungsverfahren, bei denen das System global beobachtet werden muß und eine einzelne Komponente für die Durchführung verantwortlich ist, aus, sowohl aus technischen wie auch aus Effizienzgründen. Gesucht sind sogenannte verteilte Verfahren, bei denen jede Komponente aktiv mitarbeitet und aus ihrem lokalen Wissen über den Systemzustand eigenständig Entscheidungen fällt. Dadurch werden starke Lastunterschiede innerhalb des Systems und Totalausfall beim Fehler einer einzelnen Komponente, wie es bei einer zentralistischen Lösung zu befürchten ist, vermieden.

Der korrekte Entwurf solcher Protokolle scheint eine der anspruchsvollsten Aufgabenstellungen in der Informatik zu sein, wenn man als Meßlatte ansetzt den Anteil veröffentlichter Verfahren, die sich später als fehlerhaft erwiesen haben[1]. Darüberhinaus sind für manche Aufgabenstellungen bislang keine Lösungen bekannt. Dies muß nicht immer an der Unfähigkeit der Informatiker liegen, denn in einer Reihe von Fällen konnte gezeigt werden, daß ein Problem für gewisse Systemklassen überhaupt nicht lösbar ist. Derartige negative Resultate besitzen nicht nur beweistechnisches Interesse, sondern können auch praktisch relevant sein. Zunächst erübrigt sich damit eine weitere vergebliche Suche nach Protokollen mit den gewünschten Eigenschaften. Ein weiterer positiver Fortschritt kann dann darin bestehen, die Synchronisationsforderungen abzuschwächen oder technische Eigenschaften des Systems abzuändern. Hierfür werden wir Beispiele geben.

Ziel dieses Beitrages ist es, an einigen prototypischen Problemen die Besonderheiten dieser Fragestellungen zu illustrieren und Methoden zur ihrer Lösung zu skizzieren. Die Fachwelt hat sich dabei einer Reihe von Metaphern bedient, die Szenarien außerhalb der Rechnerwelt beschreiben, oftmals in Form einer militärtaktischen Aufgabenstellung. Da diese Beispiele die Problematik recht anschaulich verdeutlichen, werden wir auf einige etwas näher eingehen. Die Ethik verantwortungsvoller Informatiker möge daran keinen Anstoß nehmen.

2 Lokales und globales Wissen

Bereits vor 40 Jahren wurden Synchronisationsprobleme von Myhill, Waksman und anderen untersucht. Popularität hat das sogenannte *Firing-Squad-Problem* erlangt:

[1] Eines der wenigen Beispiele, wo ein Protokoll formal verifiziert worden ist und dabei eine Reihe von Ungenauigkeiten der ursprünglichen Spezifikation verbessert werden mußten, kann man in [Ru94] nachlesen

Ein General führt eine Armee von Soldaten. Erhält dieser einen Angriffsbefehl, so soll er seine Soldaten derart instruieren, daß alle gleichzeitig feuern.

Die Aufgabenstellung ist trivial lösbar in einem System mit Broadcast-Kommunikation: Der General läßt den Trompeter das Angriffssignal blasen, von dem die Soldaten wissen, daß dies eine unverzügliche Aktion verlangt.

Ist die Armee dagegen räumlich zerstreut und Informationsaustausch nur durch Botschafter oder Telephonverbindungen zwischen den einzelnen Truppenteilen möglich, wird das Problem um einiges schwerer. Information muß dann lokal weitergeleitet werden. Zunächst sei angenommen, daß diese Nachrichtenweitergabe zuverlässig ohne Verstümmlungen des Textes oder bewußte Sabotage funktioniert (eine in vielen Fällen sicherlich fragwürdige Annahme, wenn man sich die Verbreitung von Gerüchten oder das Kinderspiel "Stille Post" vor Augen hält) und daß jeder Soldat über eine eigene Uhr verfügt, die exakt die gleiche Zeit anzeigt wie die Uhr seines Generals. Kann der General nicht definitiv abschätzen, wann sein Befehl auch den letzten Soldaten erreicht hat, so genügt es selbst unter diesen Voraussetzungen nicht, etwa mittags um 12h einen Befehl abzusetzen, um 13h anzugreifen. Bei einem Teil der Truppe könnte dieser Befehl erst nach 13h eintreffen.

Zunächst mag der Eindruck eines rein zeitlichen Synchronisationsproblems gegeben sein. Ebenso wesentlich ist jedoch das Problem "gleiches Wissen aller Akteure". Jeder Soldat muß den Angriffsbefehl kennen, aber darüberhinaus auch wissen, daß jeder andere Soldat dies weiß, bevor er aktiv werden kann.

Das Firing-Squad-Problem wurde hauptsächlich für reguläre, synchron getaktete ein- und mehrdimensionale Prozessorfelder untersucht, deren genaue Größe vorab nicht bekannt ist. Es ist bereits lösbar, wenn die einzelnen Akteure nur simple Automaten sind, was auf den ersten Blick überraschen mag. Ziel einer Reihe von Untersuchungen war es, Lösungen mit möglichst einfachen Automaten, d.h. möglichst wenigen Zuständen zu finden. Wesentlich dabei ist es, daß das gesamte Systems zumindest soweit synchronisiert ist, daß alle Komponenten gleichförmig getaktet arbeiten. Eine Berechnung läßt sich dann in eine Folge von *Runden* zerlegen. Zu Beginn einer Runde generiert ein Prozessor Nachrichten und verschickt diese an seine Nachbarn. Die Länge einer Runde ist so bemessen, daß alle diese Nachrichten vor Ende der Runde die Empfänger erreichen.

Bei getakteten Systemen ist Informationsübertragung auch ohne direkten Datenaustausch möglich. Dies ist ein wesentlicher Vorteil synchroner Systeme. In diesem Fall können Mechanismen wie Timeouts eingesetzt werden. Einem möglichen Ereignis E wird eine Zeitspanne zugeordnet; wird das Eintreffen von E innerhalb dieser Zeit nicht beobachtet, so können die einzelnen Komponenten des Systems folgern, daß E nicht eingetreten ist. Ein wichtiges Anwendungsbeispiel sind Updates in verteilten Datenbanksystemen [La84]. Beim *Leader-Election-Problem* – der internen Auswahl eines einzelnen Prozessors aus einem gleichförmigen System, dessen Komponenten sich nur durch eindeutige Namen (IDs) unterscheiden – konnte für ringförmige Netzwerke bewiesen werden, daß der notwendige Kommunikationsaufwand im Vergleich zu asynchronen Systemen drastisch sinkt. Während ein System aus n Prozessoren im asynchronen Fall mindestens $n \log n$ Nachrichten zur Auswahl genau eines Prozessors benötigt, genügt unter der Annahme der Synchronität linearer Kommunikationsaufwand.

Das Thema "Knowledge and Action" in Verteilten Systemen ist sowohl hinsichtlich einer adäquaten Modellierung als auch bezüglich der korrekten logischen Schlußweisen äußerst komplex.

Das wohlbekannte *Gefangenen-Dilemma*[2] scheitert an seiner Spezifikation (in [Le84] wird dies Paradoxon unter dem Namen "unerwartete Prüfung" eingehender diskutiert).

[2]Ein zum Tode verurteiler Gefangener erhält bei Verkündung des Urteils den zusätzlichen Hinweis, daß er in der folgenden Woche gehängt werde, und zwar so, daß er den Tag seiner Hinrichtung nicht vorhersehen könne. Der Verurteilte überlegt sicht daraufhin, daß dies nicht der letzte Tag der kommende Woche sein könne, da er ansonsten nach Verstreichen des vorletzten Tages den Termin mit Sicherheit wissen würde. Aus einer ähnlichen Überlegung scheidet dann auch der vorletzte Tag aus, denn ansonsten wäre dieser Termin nach Verstreichen des drittletzten Tages eindeutig, usw. Für den Gefangenen vollkommen überraschend erscheint dann doch der Henker am folgenden Mittwoch.

Das Problem der *untreuen Ehemänner* ist dagegen realer Natur. Wir müssen uns hier auf eine Kurzfassung beschränken und verweisen den interessierten Leser bezüglich weiterer historischer Details auf die vorzügliche Darstellung in [MDH86].

> *Der Inselstaat Mamajorca besitzt eine matriacharische Gesellschaftsform. Königin Henrietta I regiert mit großer Weisheit. Alle Untertaninnen gehorchen ihr aufs Wort und beherrschen das logische Schließen perfekt. Darüberhinaus ist noch wichtig zu wissen, daß die Insel so klein ist, daß ein lautes Ereignis, das sich dort ereignet, überall wahrgenommen werden kann.*
>
> *Eines Tages hält die Königin eine Versammlung aller Frauen auf dem Marktplatz ab und gibt Folgendes bekannt:*
>
> - *Mindestens einer Eurer Ehemänner ist untreu gewesen.*
> - *Keine von Euch weiß, ob Ihr eigener Gatte treu ist oder nicht, aber jede weiß dies von allen anderen Männern.*
> - *Ich gestatte keine öffentliche Diskussion über dies Thema.*
> - *Sobald eine Frau weiß, daß ihr Gatte untreu ist, so hat sie diesen um Mitternacht desselben Tages zu erschießen.*
>
> *Daraufhin vergehen neun Tage ohne Besonderheiten. In der zehnten Nacht konnte man eine Reihe von Schüssen hören.*

Frage, was ist passiert? Die Antwort lautet, daß zehn Männer untreu gewesen sind und daß alle in dieser Nacht ihr Ende finden. Allgemein gilt folgender Sachverhalt:

Theorem 1 *Es sei t die exakte Anzahl untreuer Ehemänner. Dann werden genau diese in der t-ten Nacht nach der Versammlung erschossen.*

Für $t = 1$ ergeben sich nämlich folgende Schlüsse. Alle Frauen bis auf eine wissen von genau einen Mann, daß er untreu ist. Dessen Frau dagegen kennt nur treue Ehemänner. Da nach Aussage der Königin mindestens ein untreuer existiert, schließt sie, daß ihr Mann der einzige derartige ist, und befolgt den Befehl in der ersten Nacht.

Aus diesem Vorfall können nun die anderen Frauen schließen, daß ihr eigener Ehemann treu gewesen sein muß. Denn andernfalls hätte die Vollstreckerin auch mindestens einen untreuen Mann gekannt und hätte daher am ersten Tag nicht die Gewißheit erlangen können, daß ihr Mann untreu war.

Nehmen wir nun induktiv an, daß es für $t > 1$ bis zur $(t-1)$-ten Nacht ruhig geblieben ist, so ergibt sich für die Untertaninnen am Tag t das folgende Bild. Die Frauen mit untreuen Männern kennen $t-1$ andere untreue Männer. Da keiner in der vorangegangen Nacht erschossen wurde, konnte bislang keine Frau Gewißheit erlangt haben. Dies impliziert jedoch, daß jede Frau mindestens $t-1$ untreue Männer kennen muß. Der Gatte einer Frau, die nur $t-1$ untreue Männer kennt, muß daher zu den untreuen gehören und wird folglich in derselben Nacht bestraft.

Global betrachtet stellen wir Folgendes fest: Die genaue Zahl t der Ungetreuen ist zunächst den Frauen nicht bekannt. Globales Wissen ist auf Grund der Verkündung der Königin nur, daß $t \geq 1$. Was weiß eine Frau, die t untreue Männer kennt? Die korrekte Zahl muß entweder t oder $t+1$ betragen – abhängig von ihrem eigenen Mann. Darüberhinaus kann sie sicher sein, daß jede andere Frau einerseits mindestens $t-1$ untreue Männer kennt und andererseits weiß, daß die anderen Frauen mindestens $t-2$ untreue Männer kennen. Nach Ablauf der t-ten Nacht kennen schließlich alle Frauen die korrekte Zahl – dies wird damit globales Wissen –, aber sie wissen darüberhinaus auch, daß alle anderen Frauen diese Zahl kennen und zusätzlich wissen, daß alle anderen dies Wissen besitzen. Informationsübertragung hat in diesem Fall ohne physikalische Datenübermittlung stattgefunden und zu einem einheitlichen globalen Wissensstand aller Beteiligten geführt.

Im allgemeinen wächst bei der Beschreibung des lokalen und globalen Wissens in einem Verteilten System die Rekursionstiefe mit der Größe des Systems. Derartiges wird in der menschlichen Umgangssprache sehr schnell nicht mehr überschaubar. Eine gute Formalisierung und strenge logische Schlußweisen sind dann unverzichtbar. Dem an näheren Details interessierten Leser empfehlen wir die Arbeiten [HM90, CM86, MT88, HF89, FHV92, MB94], wo geeignete Modellierungen vorgestellt werden.

3 Kommunikationsverzögerungen

Im nächsten Jahrhundert setzt Henriette II die Treue-Kampagne ihrer Mutter fort, allerdings mit neuerer Technologie.

- Sie installiert ein Post-System auf der Insel, das jeden Brief in endlicher Zeit dem Empfänger zustellt.

Anstelle einer Versammlung auf dem Marktplatz geschieht die Unterrichtung aller Untertaninnen zum Thema "untreue Ehemänner" nun dergestallt, daß die Königin zunächst Briefe an alle versendet mit Informationen über das neue Mail-System. Anschließend erhält jede Frau einen Brief mit dem gleichem Wortlaut wie das, was Henriette I verkündet hatte

Das Ergebnis ist niederschmetternd, es passiert nämlich nichts. Denn bei Verzicht auf die Versammlung – oder wie ein Informatiker sagen würde, Ersatz des synchronen Broadcasts durch paarweise Kommunikation mit unbeschränkter Verzögerung – ergibt sich, daß bei mehr als einem untreuen Ehemann keiner bestraft wird. Eine Frau kann sich nämlich nicht sicher sein, daß alle anderen den Befehl der Königin bereits erhalten haben und dementsprechend gehandelt haben. Aus der Asynchronität des Systems können daher keine Schlußfolgerungen über das Nichteintreten von Ereignissen gezogen werden.

Henriette III hat aus den Fehlern ihrer Mutter gelernt und verbessert den Post-Service.

- Jeder Brief wird nun garantiert noch am gleichen oder spätestens am nächsten Tag zugestellt.

Anders ausgedrückt gibt es eine Zeitspanne von $u = 2$ Tagen, innerhalb derer eine Nachricht eintreffen kann. Diese Unsicherheit u ist wesentlicher als die Mindestzeit zur Datenübermittlung. Qualitativ gleiche Resultate würden sich ergeben, wenn jeder Brief beispielsweise am dritten oder vierten Tag nach Aufgabe beim Empfänger einträfe.

Die Unterrichtung aller Frauen zum Thema "untreue Ehemänner" unter ihrer Herrschaft beginnt mit der Information über das verbesserte Mail-System gefolgt von einem Brief an alle Untertaninnen mit dem gleichem Wortlaut wie der ihrer Großmutter.

Das Ergebnis ist noch immer nicht zufriedenstellend, denn es geschieht eine große Ungerechtigkeit: Es werden einige, aber nicht alle untreuen Männer bestraft. Wieso?

Die Analyse dieser Situation ist wesentlich komplexer und soll hier nur angedeutet werden. Die Unsicherheit, wann die Briefe der Königin bei den Untertaninnen eintreffen, impliziert zum einen, daß sich der Fortschritt beim globalen Wissen nur mit Geschwindigkeit u entwickeln kann. Bei jeder Überlegung muß eine Frau darauf Rücksicht nehmen, daß andere diese Erkenntnisse erst einen Tag später erlangt haben könnten, und daher einen Tag abwarten. So wäre es beispielsweise fatal, wenn eine Frau, die nur einen untreuen Mann kennt und in der ersten Nacht keine Schüsse hört, daraus schließen würde, daß ihr eigener Mann ebenfalls untreu sein müsse und diesen daraufhin in der zweiten Nacht bestrafen würde. Die Frau des untreuen Mannes könnte den Brief der Königin erst am zweiten Tag erhalten habe und, angenommen alle andere Männer seien treu, erst in der zweiten Nacht handeln. Es vergehen daher mindestens $2t - 1$ Tage, bis betrogene Ehefrauen ihren Mann entlarven können.

Darüberhinaus kann eine betrogene Ehefrau, die den Brief der Königin später als ihre Leidensgenossinen erhält, keine Gewißheit über die Untreue ihres Gatten erlangen. Denn die Schüsse, die in der Nacht $2t-1$ fallen – für diese Frau wäre es erst die $(2t-2)$-te Nacht nach Erhalt des Briefes der Königin – könnten ebenso aus einer Situation resultieren, wo es nur $t-1$ untreue Männer gibt und alle deren Ehefrauen den Befehl der Königin mit einem Tag Verspätung erhalten.

Man kann dies Problem erstaunlicherweise dadurch lösen, daß man den Frauen eine zusätzliche Wartezeit $d \geq u-1$ auferlegt, bevor sie zur Tat schreiten sollen. Der Fortschritt wird dadurch auf $u+d$ verlangsamt. Um derartiges zu vermeiden, empfiehlt sich alternativ die Einführung eines globalen Kalenders und das Abstempeln der Briefe, so daß der Empfänger die Laufzeit der Post erkennen kann.

4 Unzuverlässige Komponenten und Kommunikation

Die Geschichte endet damit, daß Henrietta IV alle Untertaninnen noch einmal auf dem Marktplatz versammelt und mit diesem Tag den Mamajorcischen Kalender einführt. Ihre weiteren Bemühungen, die Kampagne der Vorfahren fortzuführen, scheiterten jedoch daran, daß ihre Untertaninnen inzwischen das Vertrauen in die Weisheit ihrer Herrinnen verloren hatten. Man konnte nicht mehr bei allen Frauen sicher sein, daß das neue Dekret peinlich genau befolgt werden würde. Nunmehr können selbst loyale Untertaninnen nicht mehr die Untreue ihres Ehemann als erwiesen ansehen, wenn andere Frauen bis zu einem gewissen Zeitpunkt nicht gehandelt haben. Damit waren fortan alle Bestrebungen zur Hebung der Moral in Mamajorca zum Scheitern verurteilt.

Dies zeigt, daß in einem System, wo einzelne Komponenten fehlerhaft arbeiten können, unbewußt oder beabsichtigt, die Synchronisationsproblematik noch erheblich zunimmt.[3] Selbst einfache Übertragungsfehler wie der Verlust von Nachrichten kann globale Konsistenz erschweren oder unmöglich machen.

Im Email-Nachrichtenverbund ist man es gewohnt, eine Fehlermeldung zu bekommen, wenn eine Nachricht den Empfänger nicht innerhalb einer gewissen Zeitspanne, etwa 3 Tage, erreicht hat. Leider funktioniert auch dieser Mechanismus nicht immer zuverlässig und man bekommt gelegentlich eine Nachricht erst nach Monaten oder überhaupt nicht als unzustellbar zurück. Auf den ersten Blick scheint es eine ganz simple Lösung mit wenig Overhead zu geben. Der Empfänger einer jeden Nachricht sendet eine Bestätigung zurück an den Absender. Was passiert jedoch, wenn diese verlorengeht. Der Empfänger besitzt zwar die beabsichtigte Information, der Absender aber weiß nicht, daß der Empfänger diese tatsächlich erhalten hat.

Der Absender könnte dann die Nachricht noch einmal senden. Günstigstenfalls wird dadurch nur das Nachrichtenaufkommen erhöht, oftmals können jedoch auch Nachrichtenduplikate Verwirrung stiften, wenn nicht geeignete Filtermechanismen eingesetzt werden. Erhält der Absender jedoch die Bestätigung, so kann sich der Empfänger nicht sicher sein, daß seine Bestätigung auch angekommen ist. Also könnte der Absender die Bestätigung des Empfängers mit einer Rückbestätigung quittieren, usw. Es sollte klar werden, daß auf diese Weise nie ein gleicher Wissenstand zwischen den beiden Partnern erreicht werden kann, in dem Sinne, daß

"A weiß, daß B weiß, daß A weiß, daß . . . daß die Nachricht angekommen ist."

Wir wollen im folgenden diese Konsistenzproblematik und die verschiedenen Systemausprägungen bezüglich Synchronisation und Fehlverhalten formalisieren und wichtige grundlegende Erkenntnisse vorstellen.

[3] Man überlege sich, was passiert wäre, wenn unter Henrietta I die einzige betrogene Ehefrau am ersten Abend wegen Krankheit handlungsunfähig gewesen wäre.

5 Consensus

5.1 Systemmodellierung

Wir betrachten Systeme von n autonomen Prozessoren $P_1, \ldots, P_n$, die physikalisch verbunden sind durch ein Kommunikationsnetzwerk. Dies kann ein Broadcast-Medium sein wie ein lokales Ethernet oder ein Token-Ring, bei dem zu jedem Zeitpunkt nur ein Prozessor eine Nachricht an einen oder mehrere andere Prozessoren versenden kann, oder ein Graph, bei dem die Prozessoren, die Knoten des Graphen, jeweils paarweise über separate Verbindungskanäle, die Kanten, miteinander kommunizieren können. Wir setzen voraus, daß der Absender einer Nachricht eindeutig identifiziert werden kann, d.h. Prozessoren besitzen eindeutige IDs und kennen den Partner am anderen Ende einer Verbindungsleitung. Denn anonyme Systeme erweisen sich als zu schwach für unsere angestrebten Ziele.

Die Topologie des Kommunikationsnetzes kann dabei ein vollständiger Graph sein (jeder ist mit jedem verbunden) oder bei großen Netzen typischerweise ein gradbeschränkter Graph – entweder mit einer unregelmäßigen Struktur wie beispielsweise das Internet, oder regelmäßig aufgebaut wie etwa mehrdimensionale Gitter oder Permutationsnetzwerke. Um die essentiellen Schwierigkeiten bei der Synchronisation solcher Systeme zu skizzieren, werden wir zunächst Netzwerke mit vollständigen paarweisen Verbindungen betrachten.

Bei Meinungsverschiedenheiten kann ein Konsens durch eine Mehrheitsentscheidung herbeigeführt werden. Man wählt die Alternative, die die meisten Stimmen erhalten hat. Bei binären Entscheidungen (nur zwei Alternativen) sind daher Systeme mit einer ungeraden Anzahl von Komponenten vorteilhaft, denn es können Pattsituationen nicht auftreten. Dies ist jedoch nicht entscheidend, denn andernfalls kann man für den Falle eines Patts eine Zusatzregel vorsehen – beispielsweise die erste Alternative auszuwählen im Sinne einer vorher festgelegten Ordnung.

Um die Zuverlässigkeit von 1-Prozessorsystemen zu erhöhen, ist es naheliegend, Redundanz durch einfache Duplizierung zu erzielen. Eine Konfiguration, die einzelne Fehler kompensieren kann, sollte daher mindestens 3 autonome Komponenten besitzen. Dies ist eine gängige Vorgehensweise in der Praxis. Jedoch hat sich gezeigt, daß damit nicht alle Anforderungen erfüllt werden können. Denn dies simple Verfahren funktioniert nicht, wenn fehlerhafte Komponenten verschiedenartige Information versenden. Denn letztlich kann ein einzelner Prozessor nur auf der Grundlage seiner lokalen Information, d.h. den von den anderen empfangenen Nachrichten, entscheiden. In einem Verteilten System mit Fehlern kann man nie einen Zustand garantieren, in dem diese lokale Information für alle Komponenten identisch ist. Daher sind kritische Situationen nicht auszuschließen, wo eine marginale Informationsdifferenz zu unterschiedlichen individuellen Entscheidungen führt.

In einer diese Problematik motivierenden Arbeit skizzieren Lamport, Shostak und Pease [LSP82] die folgende Aufgabenstellung, die sie das *Problem der Byzantinischen Generäle* nennen.

> *Für eine Armee bestehend aus mehreren räumlich getrennten Divisionen jeweils angeführt durch einen General, soll ein Schlachtplan entworfen werden. Die Generäle können untereinander nur durch den Austausch von Botschaftern kommunizieren, da eine Zusammenkunft aller zur Beratung des Plans aus strategischen Gründen ausgeschlossen ist. Eine Entscheidung (etwa sofortiger Angriff oder noch Abwarten) muß daher von jedem General aus den Vorschlägen seiner Kollegen nach einem vorab festgelegten Verfahren vor Ort getroffen werden. Das Problem dabei ist, daß einige wenige eventuell nicht loyale Generäle, die heimlich mit dem Feind paktieren, durch Sabotage versuchen könnten, eine einmütige Entscheidung der übrigen zu verhindern.*

Zunächst überlegt man sich, daß es genügt, unabhängig für jeden General eine einmütige Festlegung zu treffen, was dieser für einen Vorschlag unterbreitet hat. Besitzt nämlich jeder denselben Vektor von Vorschlägen, so kann jeder loyale General lokal für sich eine vorher vereinbarte Entscheidungsfunktion wie etwa eine Mehrheitsregel auf diesen Vektor anwenden, und gelangt so in Einklang mit den anderen zu demselben Ergebnis.

Bei den Vorschlägen loyaler Generäle ist dies kein Problem, denn ein solcher hat die gleiche Botschaft an alle verschickt. Ein nichtloyaler General dagegen mag einigen "Angriff" und anderen "Rückzug" übersandt haben. Auf was sich die anderen letztendlich einigen, als dessen Vorschlag anzusehen, ist relativ gleichgültig, wichtig ist nur, daß sie zur gleichen Einschätzung kommen. Ohne diese Eigenschaft könnte eine kleine Gruppe illoyaler Generäle in einer Situation, wo die anderen loyalen Generäle zur Hälfte einen Angriff und zur anderen Hälfte ein Abwarten favorisieren, durch gezielte Falschinformation Folgendes erreichen: Einem Teil der loyalen Generäle gegenüber äußern diese Unterstützung für einen Angriff, was etwa bei einer Mehrheitsregel dann dazu führt, daß sich diese für Angriff entscheiden. Einem anderen Teil wird der Vorschlag Abwarten unterbreitet, so daß diese eine Mehrheitsmeinung für letztere Alternative vorfinden.

Somit zerfällt das ursprüngliche Problem in unabhängige gleichartige Teilprobleme, bei denen jeweils ein Konsens zu erzielen ist, als Voraussetzung aber nur ein einzelner Wert, nämlich der Vorschlag eines einzelnen Generals gegeben ist. Dies wollen wir nun folgendermaßen formalisieren.

Jeder Prozessor P_i $(i = 1, \ldots, n)$ startet mit einem Anfangswert x_i. Nach Austausch von Informationen mit seinen Nachbarn wählt P_i einen Entscheidungswert y_i. Der Ausgabevektor $Y = (y_1, \ldots, y_n)$ muß dabei in Abhängigkeit vom Eingabevektor $X = (x_1, \ldots, x_n)$ den folgenden Bedingungen genügen:

- **Terminierung (Wait Freeness):**
  Jeder korrekte Prozessor trifft nach endlich vielen Schritten eine Entscheidung. Durch das Verhalten und die Geschwindigkeit anderer Prozessoren kann diese daher nicht unendlich lange hinausgezögert werden. Manchmal wird die stärkere Bedingung erhoben, daß eine Entscheidung innerhalb einer fest vorgegebenen Zeitschranke fällt unabhängig von dem Verhalten anderer.
- **Übereinstimmung (Agreement):**
  Alle korrekten Prozessoren entscheiden sich für den gleichen Wert, d.h. $y_i = y_j$, falls P_i und P_j beide korrekt sind.
- **Sinnhaftigkeit (Validity):**
  Als Entscheidungswerte sind nur die aktuellen Eingabewerte zulässig, $y_i \in \{x_1, \ldots, x_n\}$. Dies bedeutet insbesondere, daß bei identischen Eingabewerten dieser Wert auch von jedem als Entscheidungswert gewählt werden muß.

  In manchen Fällen begnügt man sich mit dieser zweiten schwächeren Forderung und akzeptiert bei verschiedenartigen Eingabewerten einen beliebigen Ausgabewert. Eine noch weitergehende Abschwächung ist die folgende Bedingung, die
  Nichttrivialität:
  Die Menge der Entscheidungswerte, die sich ergeben, wenn man alle Anfangsverteilungen und alle möglichen Protokollabläufe betrachtet, umfaßt mehr als ein Element.

Terminierung ist eine selbstverständliche, aber nicht zu vergessende Bedingung bei Algorithmen für Verteilte Systeme. Damit sollen Protokolle ausgeschlossen werden, die in ungünstigen Situationen, z.B. beim Ausfall gewisser Komponenten, das Problem unendlich lange vor sich herschieben, ohne zu einer Lösung zu gelangen.

Die zweite Bedingung der Übereinstimmung – zumindest der korrekten Prozessoren – ist unser eigentliches Ziel. Es ist klar, daß man keine Anforderungen an die Entscheidung inkorrekter Prozessoren stellen kann.

Die letzte Bedingung schließlich verlangt, daß das Ergebnis der Entscheidung die Ausgangslage – zumindest in eindeutigen Situationen – entsprechend widerspiegelt. Ein Entscheidungsverfahren mit diesen Eigenschaften wird als *Byzantinisches Agreement* bezeichnet. Die noch schwächere Forderung der Nichttrivialität ist das mindeste, was man verlangen sollte, nämlich um Verfahren auszuschließen, die immer einen vorab festgelegten Wert wählen. Diese Version heißt *nichttrivialer Consensus* oder einfach *Consensus*.

Die nachfolgenden negativen Resultate gelten schon für binäre Entscheidungsprobleme, d.h. die einzig möglichen Anfangswerte sind 0 und 1. Algorithmen sind in der Regel für beliebige Wertebereiche konzipiert. Man kann jedoch durch geeignete Maßnahmen erreichen, daß bei jedem Lauf nur eine kleine Anzahl von Werten zu betrachten ist. Wenn beispielsweise ein General zu Anfang viele verschiedene Werte an die anderen sendet, so kann dies relativ leicht erkannt werden und man kann derartige Falschmeldungen unbeachtet lassen.

5.2 Physikalische Synchronisation von Rechnernetzen

Durch die technische Ausführung besitzt ein Rechnernetz eine gewisse Synchronisation per se. Diesbezüglich ist es zweckmäßig, *synchrone und asynchrone* Modelle zu unterscheiden, und dies getrennt für die beiden wesentlichen Bereiche: zum einen die Kopplung der Prozessoren, zum anderen die Nachrichtenübertragung durch das Kommunikationsnetzwerk.

Kann die Hardware keine perfekte Synchronisation garantieren, so wäre die nächste Stufe eine *beschränkte Form von Asynchronität.* Es wird dann angenommen, daß die Schwankungen der relativen Geschwindigkeit der Prozessoren zueinander nach oben und unten durch feste bekannte Werte beschränkt sind. Für das Netzwerk werden garantierte maximale (und in manchen Fällen auch wichtig minimale) Übertragungszeiten für jede einzelne Nachricht gefordert.

Im *vollkommen asynchronen Modell* gibt es derartige Garantien nicht, d.h. eine Nachricht beispielsweise kann beliebig lange verzögert werden. Fehler können nicht durch einfache Timeout-Verfahren oder ähnliches behandelt werden. Denn der Verlust einer Nachricht ist prinzipiell nicht zu unterscheiden von einer großen Verzögerung bei der Übertragung, der Ausfall eines Prozessors nicht von einem Zustand, wo dieser durch starke Belastung extrem langsam reagiert. Man kann dies Problem auch nicht dadurch lösen, derartige systemkonforme Verzögerungen als virtuelle Fehler zu deklarieren. Dies würde nämlich überzogene und zum Teil nicht erfüllbare Anforderungen an die Fehlertoleranz stellen.

Nimmt man an, daß Prozessoren und Netzwerk in gleichem Maße synchronisiert sind, so ergeben sich die folgenden drei wesentlichen Modelle.

- *vollständig synchron:* getaktet in synchronen Runden wie oben am Beispiel des Firing-Squad-Problems beschrieben,
- *beschränkt asynchron,*
- *vollständig asynchron.*

5.3 Fehlertoleranz

Die Lösbarkeit der Consensus Probleme und die Komplexität der Algorithmen hängt wesentlich davon ab, welche Annahmen man über die Anzahl und Art der auftretenden Fehler macht. Oftmals wird dabei zur Vereinfachung angenommen, daß die Kommunikationsleitungen einwandfrei arbeiten und Fehler nur bei den Prozessoren auftreten. Mit anderen Worten, der Verlust einer Nachricht wird gewertet als ein Prozessorfehler, entweder beim Absender oder beim Empfänger. Dieses Vorgehen läßt sich auf Grund der nachfolgenden Ergebnisse rechtfertigen.

Die wichtigsten Typen von Fehlern, geordnet der Schwere nach sind die folgenden:

1. *Fail-Stop* oder Ausfall (Crash) eines Prozessors: Der Prozessor arbeitet bis zu einem gewissen Zeitpunkt einwandfrei und fällt dann vollständig aus, insbesondere sendet oder empfängt er keine Nachrichten mehr.

2. *Omission* oder partielle Unterbrechung: Einige Nachrichten gehen verloren, alle Nachrichten, die den Empfänger erreichen, sind jedoch korrekt.

3. *Timing:* Ausführen einer Aktion oder das Versenden von Nachrichten geschieht zu spät oder auch zu früh.

4. *Byzantinischer Fehler* oder Inkonsistenz: Gestörte Prozessoren verhalten sich beliebig inkorrekt, insbesondere können sie falsche Information versenden.

   Byzantinische Fehler werden sinnvollerweise noch in folgender Weise eingeschränkt:

   - *symmetrische Fehler:* Ein inkorrekter Prozessor verhält sich zu allen anderen in gleicher Weise inkorrekt.
   - *erkennbare Fehler:* Eine Nachricht, die falsch ist, kann vom Empfänger erkannt werden.
   - *beweisbare Fehler:* Ein Prozessor kann den anderen einen Beweis erbringen, wenn ihm ein Nachbar eine falsche Nachricht übersandt hat.

Symmetrische Fehler erweisen sich noch als relativ gut kompensierbar. Beim Byzantinischen Agreement beispielsweise treten keine Probleme auf, wenn ein inkorrekter General allen anderen die gleiche falsche Meinung übermittelt, deren Übereinstimmung wird dadurch nicht gestört.

Falsche Nachrichten können bei der Generierung und bei der Übertragung entstehen. Übertragungsfehler im Netzwerk können durch geeignete Codierung der Nachrichten erkannt werden. Eine falsche Information eines Nachbarn kann jedoch unter Umständen mit Hilfe des lokalen Wissens enttarnt werden.

Schwierig wird es, einen Dritten davon zu überzeugen, daß Fehler im System aufgetreten sind. Erhält etwa ein Prozessor P_i eine falsche Nachricht von P_j, so kann er dies gegenüber P_k zwar behaupten. P_k steht jedoch vor der schwierigen Entscheidung, wem zu glauben, wenn P_j behauptet, diese Nachricht niemals an P_i gesendet zu haben.

Enorme Bedeutung erlangt diese Beweisbarkeit beispielsweise bei der Konzipierung elektronischer Zahlungssysteme. Wir wollen uns hier auf Authentisierungsverfahren in Form digitaler Unterschriften beschränken. Werden alle Nachrichten auf diese Weise authentisiert, so kann P_i einem Dritten gegenüber jederzeit den Beweis erbringen, daß P_j der Urheber gewesen ist.

Diese Klassifizierung und Annahmen über Fehler im System ist so zu verstehen, daß man nach Bewertung der möglichen Fehler in einem realen System und der Güte der gewünschten Fehlertoleranz ein entsprechendes Fehlermodell und dann für dieses Modell geeignete Verfahren wählt. Um ganz sicher zu gehen, sollten auch schwere Fehler wie Inkonsistenzen berücksichtigt werden, was jedoch in der Regel einen höheren Aufwand erfordert. Sind die Sicherheitsanforderungen nicht so hoch, kann man sich auf einfachere Fehlermodelle und damit effizientere Protokolle beschränken.

Für die weitere Analyse benötigen wir, wie sich herausstellen wird, noch eine obere Schranke für die maximale Anzahl inkorrekter Prozessoren. Diese Zahl bezeichnen wir im folgenden mit t. Ein Algorithmus oder Protokoll zur Lösung des Consensus Problems heißt t*–fehlertolerant* (t–resilient) bezüglich eines vorgegebenen Typs von Fehlern, falls für jede beliebige Verteilung von Anfangswerten und beliebiges fehlerhaftes Verhalten von maximal t Prozessoren die übrigen Prozessoren bei korrektem Verhalten den erwarteten Konsens erzielen.

Diese Art von Fehlertoleranz mit einer festen vorgebenen Schranke für die maximale Anzahl von Fehlern ist vielleicht nicht das, was man in der Praxis am liebsten hätte. Wünschenswerter wären Verfahren, die sich der Zahl der auftretenden Fehler anpassen und in ungünstigen Situationen entsprechend mehr Aufwand betreiben. Die Ergebnisse im folgenden werden zeigen, daß so etwas im allgemeinen nicht möglich ist. Konsens kann immer nur bis zu einer gewissen Fehlerschranke t, die als Parameter in diesen eingeht, garantiert werden. Man muß daher abwägen und entsprechend den Erfordernissen t genügend hoch ansetzen.

Neben deterministischen Protokollen sind auch probabilistische Protokolle betrachtet worden, die durch interne Randomisierung das Erreichen eines Konsens beschleunigen. Diese Algorithmen arbeiten fehlerfrei, d.h. korrekte Prozessoren entscheiden immer gleich. Durch Zufallsentscheidungen, teils global, teils lokal, wird dabei der (worst-case) Einfluß fehlerhafter Prozessoren gemindert. Im Rahmen dieser Einführung können wir diesen Punkt hier nicht weiter vertiefen.

6 Byzantinisches Agreement in synchronen Systemen

6.1 Fehlertoleranz

Wir betrachten zunächst synchrone, vollständig verbundene Rechnernetze. Die einfachste Konfiguration ist ein Ring aus 3 Prozessoren, von denen einer, etwa P_1 als vorschlagender General fungiert. Seine Botschaften an die P_i sind die Ausgangswerte x_i für das Consensus-Protokoll. Maximal einer der Generäle P_1, P_2, P_3 sei inkorrekt, d.h. $n = 3$ und $t = 1$. In dem Fall, daß alle Prozessoren bis auf einen defekt sind, ist Agreement offensichtlich trivialerweise erfüllt, deshalb sei im folgenden $t < n - 1$ vorausgesetzt.

Ist der General P_1 korrekt mit Anfangswert 0 und teilt dies P_2 und P_3 mit, so muß sich P_2 auf Grund der Validity Bedingung für den Wert 0 entscheiden. Dies gilt auch, wenn ihm P_3 inkorrekterweise mitteilt, er selber habe von P_1 den Wert 1 erhalten. Analoges gilt für einen korrekten General P_1 mit Wert 1 und die Entscheidung eines korrekten P_3 , unabhängig davon, daß P_2 , nun inkorrekt, behauptet, den Wert 0 empfangen zu haben. P_3 wird wie P_1 den Wert 1 wählen.

Man vergleiche dies nun mit einer dritten Situation, wo P_1 inkorrekt und P_2 und P_3 korrekt sind und P_1 die Ausgangswerte 0 an P_2 und 1 an P_3 sendet. Die beiden korrekten Generäle können sich zwar gegenseitig über diese Inkonsistenz unterrichten, sie aber nicht lösen. Die lokale Information ist für P_2 identisch zur ersten Situation, wo er sich für 0 entschied, und für P_3 identisch zur zweiten Situation mit Entscheidung 1. Daher müssen sich beide wie in den entsprechenden vorherigen Situationen entscheiden und verletzen damit die Einmütigkeit.

Dieses Argument läßt sich formalisieren und erweitern auf Netzwerke mit beliebig vielen Prozessoren, von denen bis zu ein Drittel fehlerhaft sein können. Gleichzeitig wurde in [PSL80] ein simples, aber extrem kommunikationsintensives Protokoll vorgestellt, welches das Problem für weniger als ein Drittel Fehler in $t + 1$ Runden löst. Ein Prozessor sendet in jeder Runde an alle anderen alle Information, die er bislang erhalten hat (man nennt derartiges auch ein *Full-Information-Protokoll*). Der Aufwand, genauer die Länge der Nachrichten dieses Protokolls, wächst exponentiell mit der Zahl der Runden ($O(n^{t+1})$). Die Entscheidungsregel und deren Korrektheitsbeweis sind die eigentliche Schwierigkeit und benutzen eine komplexe Induktion. Somit erhalten wir für den allgemeinsten Fall beliebiger Inkonsistenzen von Prozessoren das Ergebnis :

Theorem 2 *In synchronen Systemen der Größe n ist t-fehlertolerantes Byzantinisches Agreement genau dann zu erzielen, wenn $t < n/3$ gilt [PSL80].*

Dies positive Ergebnis für $t < n/3$ ist später unter Effizienzgesichtspunkten in einer langen Folge von Arbeiten verbessert worden. Zunächst wurde der Kommunikationsaufwand auf polynomielle Größe ($O(nt + t^3)$) reduziert, allerdings auf Kosten der Zeitkomplexität oder der Fehlertoleranz. Diese Entwicklung und den aktuellen Erkenntnisstand kann man in [GM93] nachlesen.

Halten wir damit fest: Ein redundantes System der Größe 3, wie es die simple Mehrheitsregel nahelegen mag, genügt nicht, um Agreement bei einem beliebigen Fehler zu garantieren. Man benötigt vielmehr 4 autonome Duplikate.[4]

[4] Die Relevanz dieser Erkenntnis mag man daran ablesen, daß in Fachkreisen Beispiele genannt wurden, wo dieser Trugschluß bezüglich 1-Fehlertoleranz bei Redundanz 3 in realen Systemen tatsächlich zu unerwarteten Problemen geführt haben soll.

6.2 Kommunikation mit Authentisierung

Bei Authentisierung durch signierte Nachrichten (digitale Unterschriften) ist obig skizziertes Szenario nicht möglich. In der ersten Situation etwa würde P_2 dem inkorrekten Prozessor P_3 nicht glauben, da ihm dieser nicht den Wert 1 mit der Unterschrift von P_1 präsentieren kann. In der letzten Situation dagegen können sich P_2 und P_3 gegenseitig den Nachweis erbringen, daß P_1 inkonsistente Information versendet hat, denn sie besitzen verschiedene von diesem signierte Werte.

Byzantinisches Agreement kann daher durch das folgende simple 2-Runden Protokoll erreicht werden. In der ersten Runde schickt der General seinen authentisierten Wert an die beiden anderen. Diese Nachrichten tauschen P_2 und P_3 in der zweiten Runde aus. Falls diese am Ende genau einen Wert mit der Unterschrift von P_1 besitzen, entscheiden sie auf diesen Wert. Andernfalls, d.h. wenn P_i keinen oder mehrere Werte erhalten hat, entscheidet er sich für einen vorab festgelegten Default-Wert (z.B. 0).

Ist P_1 korrekt, so wird nur sein Anfangswert in authentisierter Form vorliegen und jeder andere korrekte Prozessor entscheidet sich ebenfalls für diesen (Validity). Signiert P_1 dagegen verschiedene Werte, so werden sich die beiden anderen korrekten Prozessoren diese gegenseitig zusenden und daher beide den Default-Wert wählen (Agreement).

Bei größeren Systemen mit entsprechend größerer Fehlerschranke t reicht dies 2-Runden Protokoll nicht aus, um Agreement sicherzustellen. Denn neben dem General P_1 können noch andere fehlerhaft sein und bewirken, daß sich das lokale Wissen zweier korrekter Prozessoren auch nach der zweiten Runde unterscheidet. Dieser Aufwand läßt sich nicht vermeiden, auch wenn man die Fehlerarten noch weiter einschränkt, wie die folgenden Ergebnisse zeigen.

Theorem 3 *Bei Verwendung von Protokollen mit Authentisierung (beweisbare Fehler) oder für den einfacheren Fall von Omission-Fehlern ist in synchronen Systemen jede Fehlerschranke $t \leq n$ tolerierbar.*

In synchronen Systemen sind im ungünstigen Fall $t+1$ Runden Informationsaustausch für t-fehlertolerantes Byzantinisches Agreement notwendig und hinreichend, selbst wenn nur Fail-Stop-Fehler auftreten können [DS83].

6.3 Zeit- und Nachrichtenkomplexität

In der Praxis wird man aus Sicherheitsgründen die maximale Fehlerschranke relativ hoch ansetzen, im Normalfall jedoch keine oder nur wenige tatsächliche Fehler erwarten. Daher sind Protokolle von Interesse, die in solch einer gutartigen Situation das Agreement wesentlich schneller erreichen. Diese Eigenschaft nennt man *Early Stopping*. Die obigen Lösungen dagegen verwenden immer eine feste Rundenzahl $t+1$. Mit anderen Worten sind worst-case und best-case Zeitkomplexität identisch.

Unsere Untersuchungen haben ergeben, daß man dazu zwischen zwei Formen von Agreement unterscheiden muß.

- *Immediate Agreement*: Alle korrekten Prozessoren entscheiden sich in der gleichen Runde; Anwendungsbeispiel wäre eine simultane Aktion wie beim Firing-Squad-Problem.
- *Eventual Agreement*: Korrekte Prozessoren wählen denselben Wert, jedoch möglicherweise zu verschiedenen Zeitpunkten; Anwendungsbeispiel etwa Transaktionen in verteilten Datenbanksystemen.

Early Stopping ist unter der *Immediate*-Bedingung nicht möglich. Im anderen Fall kann man diese Eigenschaften erreichen, es bedarf allerdings einer zusätzlichen Runde.

Theorem 4 *Um Immediate Agreement zu erzielen, sind immer* $t+1$ *Runden notwendig, selbst wenn in der konkreten Situation überhaupt keine Fehler auftreten.*

Eventual Agreement kann bei f *tatsächlichen Fehlern bereits in* $f+2$ *Runden erzielt werden. Diese Schranke ist bestmöglich [DRS90].*

Wir hatten oben bereits kurz die Nachrichtenkomplexität angesprochen. Zur Vereinfachung wollen wir bei Protokollen ohne Authentisierung die Anzahl der Nachrichten zählen, während bei Protokollen mit Authentisierung auch die Anzahl der Unterschriften, die in Nachrichten enthalten sind, von Bedeutung ist. Der notwendige Aufwand kann relativ gut abgeschätzt werden. Dazu folgendes Ergebnis:

Theorem 5 *Ein* t *-fehlertolerantes Protokoll für Byzantinisches Agreement benötigt im worst case ohne Authentisierung mindestens* $\Omega(nt)$ *Nachrichten und mit Authentisierung mindestens* $\Omega(n+t^2)$ *Nachrichten mit insgesamt mindestens* $\Omega(nt)$ *Unterschriften [DR85].*

Diese Schranken sind asymptotisch bestmöglich, da sie von geeigneten Protokollen erreicht werden. Bei der ersten gilt dies zumindest für kleine t.

6.4 Allgemeine Netzwerke

Die bislang betrachteten Algorithmen setzten voraus, daß jeder Prozessor an jeden anderen auf direktem Wege Nachrichten senden kann. Für große Systeme sind vollständige Netzwerke unrealistisch. In weniger dichten Netzwerken funktionieren modifizierte Verfahren, wenn man den Durchmesser des Graphen berücksichtigt. Bei den einfachen Fehlertypen reicht es, wenn der Subgraph aus korrekten Prozessoren zusammenhängend ist. Bezeichnet k die Zusammenhangszahl des Netzwerkes, so ist dafür hinreichend $k > t$. Bei Byzantinischen Fehlern gilt dagegen

Theorem 6 *Für* t *–fehlertolerantes Byzantinisches Agreement bei Inkonsistenz ohne Authentisierung ist in einem Netzwerk notwendig und hinreichend* $t < k/2$ *[Do82].*

Die Notwendigkeit folgt aus der Eigenschaft, daß in einem Netzwerk mit Zusammenhangszahl $k \leq 2t$ zwei Teilmengen von Prozessoren der Mächtigkeit jeweils maximal t die übrigen Prozessoren trennen können. Verhalten sich alternativ alle Prozessoren in einer dieser beiden Teilmengen hinreichend inkonsistent und verfälschen die Nachrichten zwischen den Teilen des getrennten Netzwerkes, so haben die Prozessoren in den verschiedenen Teilen keine Möglichkeit, die beiden Situationen zu unterscheiden.

Andererseits kann man in einem Netzwerk mit Zusammenhang mindestens $2t+1$ die Kommunikation in einem vollständigen Netzwerk fehlerfrei simulieren. Eine Nachricht wird vom Absender dupliziert und über $2t+1$ disjunkte Wege an den Empfänger versandt. Da mindestens eine Mehrheit von $t+1$ Wegen nur korrekte Prozessoren enthält, gelangen die Nachrichten über diese unverfälscht ans Ziel.

Damit sind durch n und k die maximale Fehlertoleranz t eines beliebigen Netzwerkes für Byzantinisches Agreement eindeutig bestimmt: $t < \min\{n/3, k/2\}$. In [FLM86] wird ein elegantes Argument mit Hilfe von Überdeckungsgraphen entwickelt, um diese Schranke nicht nur für das Byzantinische Agreement, sondern auch für ähnliche Konsens-Probleme zu beweisen.

7 Consensus in asynchronen Systemen

Betrachten wir nun den Fall eines (vollkommen) asynchronen Netzwerkes. Es hat sich gezeigt, daß nicht die unterschiedliche Arbeitsgeschwindigkeit der Prozessoren die entscheidende Schwierigkeit ist – denn dies läßt sich bis zu einem gewissen Grad synchronisieren – sondern die Nachrichtenverzögerung. Das erste und grundlegende Resultat besagt, daß im asynchronen Fall überhaupt kein Konsens erreicht werden kann.

Theorem 7 *In vollkommen asynchronen Systemen ist nichttrivialer Consensus unmöglich, selbst für den einfachsten Fall* $t = 1$ *und Ausfall-Fehler [FLP85].*

Die Beweisidee ist wie folgt: Eine Konfiguration C des Systems heiße *bivalent*, wenn es noch möglich ist, beide Entscheidungswerte zu erreichen. Es gibt mit anderen Worten zwei verschiedene Fortsetzungen von C, so daß die korrekten Prozessoren einmal auf 0 und einmal auf 1 entscheiden. Andernfalls heiße C univalent (genauer 0-valent oder 1-valent). Wann immer nun ein Protokoll in eine entscheidende Phase tritt, d.h. von einer bivalenten in eine univalente Konfiguration übergeht, und dies muß durch die Aktion eines einzelnen Prozessors geschehen, besteht die Gefahr, daß das System bei Ausfall oder extremer Verlangsamung dieses Prozessors nie zu einer Entscheidung gelangt.

Da nichttrivialer Consensus eine sehr schwache Anforderung darstellt, hat dies Ergebnis in Fachkreisen zunächst zu der Auffassung geführt, daß in einem asynchronen System keinerlei sinnvolle fehlertolerante Kooperation möglich ist. Das folgende Problem einer kompakten Namensvergabe zeigt jedoch, daß man auch im vollkommen asynchronen Fall etwas erreichen kann. Ziel des *Renaming-Problems* ist jedoch gerade das Gegenteil von Einmütigkeit.

> *Jeder Prozessor* P_i *besitzt als Eingabe einen eindeutigen Namen* x_i, *der aus einem großen Universum stammt.* P_i *soll aus einer Menge* W *einen neuen Namen* y_i *wählen, so daß* $y_i \neq y_j$ *für alle* $i \neq j$. *Dabei wähle man* w, *die Mächtigkeit der Menge* W, *in Abhängigkeit von* n *und* t *möglichst klein.*

Bei einer nichtdistributiven Lösung mit einer zentralen Namensvergabestelle, etwa Prozessor P_1, würde $w = n$ genügen; diese weist allerdings keinerlei Fehlertoleranz auf. Die naheliegende Idee, P_1 im Falle des Ausfalls durch einen Vertreter P_2 zu ersetzen, scheitert daran, daß dazu ein Konsens über den Ausfall von P_1 notwendig ist. Dies ist jedoch unmöglich, wie wir oben gesehen haben. Erhöht man jedoch den Entscheidungsspielraum, so wird das Problem distributiv und fehlertolerant lösbar.

Theorem 8 *Renaming ist in asynchronen Systemen* t*-fehlertolerant mit* n *Namensalternativen nicht lösbar, selbst für* $t = 1$.
Bei $n + t$ *Alternativen gibt es dagegen geeignete Protokolle, die dies erreichen [ABD90].*

Da Fehlererkennung ein grundsätzliches Problem darstellt, im asynchronen Fall jedoch nicht garantiert werden kann, wurde folgende Abschwächung unter dem Namen *kollektive Konsistenz* betrachtet, ein Consensus ohne garantierte Terminierung.

> *Fehlerhafte Prozessoren sollen erkannt und isoliert werden. Jeder Prozessor* P_i *bestimmt zu diesem Zweck eine Menge* V_i *von Prozessoren, die er als fehlerhaft ansieht. Falls zwei Prozesooren* P_i, P_j *eine Entscheidung treffen und* $P_j \notin V_i$, *so wird verlangt, daß* $V_i = V_j$.

Auf Grund des obigen Unmöglichkeitsergebnisses kann kollektive Konsistenz nicht in allen Situationen "wait-free" erreicht werden. Genauer gilt:

Theorem 9 *Jedes nichttriviale Protokoll für kollektive Konsistenz in einem asynchronen System besitzt entweder einen Lauf ohne Fehler, in dem kein Prozessor jemals eine Entscheidung trifft, oder einen Lauf, bei dem alle Prozessoren auf einen Prozessor warten [DHS95].*

8 Zeit-Synchronisation

Zum Abschluß wollen wir noch kurz auf das Problem der zeitlichen Synchronisation eingehen. Jeder Knoten in einem Rechnernetz besitzt heutzutage eine interne Uhr, seine lokale Hardware-Clock, die sich bezüglich einer absoluten Referenzuhr mit einer gewissen Abweichung, der sogenannten Drift-Rate, bewegt[5]. Selbst in einem lokalen Cluster können sich so erheblich Zeitunterschiede einstellen, wenn nicht manuell oder periodisch auf Betriebssystemebene (typsicherweise in einer Master-Slave-Form) eine Synchronisation durchgeführt wird.

Eine möglichst genaue Synchronisation der Uhren ist aus vielerlei Gründen wichtig. In synchron arbeitenden Systemen etwa können nur so scharfe Zeitpunkte für Timeouts definiert werden. In Realzeit-Systemen ist eine gute Synchronisation Grundvoraussetzung. Sogenannte *Time-Stamps* zur Ordnung von Ereignissen oder die Festlegung von Zeitintervallen – etwa für die Gültigkeit von Schlüsseln beim Einsatz kryptografischer Verfahren in Rechnerverbänden – sind ebenfalls auf synchronisierte Uhren angewiesen. Lamport hat diese Problematik als erster eingehend diskutiert (siehe hierzu [La78]). Inzwischen sind eine Vielzahl von Protokollen zur Uhrensynchronisation mit unterschiedlicher Fehlertoleranz und Güte entwickelt worden (beispielhaft genannt seien [LL84, DHS86], einen guten Überblick über die älteren Verfahren gibt [Sc87]).

Man unterscheidet dabei zwischen *interner Synchronisation* und nennt die maximale Abweichung zwischen zwei Uhren im Netz die *Präzision* und *externer Synchronisation* zur "realen Zeit" (sprich einer externen sehr genauen (Atom-)Referenzuhr). Die letztere Abweichung bezeichnet man als *Akkuratheit*. Für viele Anwendungen ist eine gute Präzision ausreichend. Die Akkuratheit hängt davon ab, mit welcher Genauigkeit die externe Zeit im Netz gelesen werden kann, während die erreichbare Präzision im wesentlichen durch die Unsicherheit bei der Nachrichtenübertragung bestimmt wird. Der Leser möge sich an das zu Anfang diskutierte Problem des Mamajorcischen Postdienstes erinnern.

Mills hat ein relativ einfaches Verfahren, ein sogenanntes Round-Trip-Protokoll, in die Praxis umgesetzt und als das *Network-Time-Protokoll* NTP implementiert, mit dessen Hilfe Rechner weltweit über das Internet im Millisekundenbereich synchronisiert werden können [Mi91]. Ausgewählte Time-Server-Knoten bieten hierbei anderen Rechnern die Möglichkeit, die Zeit abzufragen. Je schneller diese Kommunikation bestehend aus Anfrage und Antwort abgewickelt wird, desto präziser kann ein Knoten seine interne Uhr anschließend adjustieren. Dazu wird eine logische Uhr kreiert, die aus der Hardware-Clock und einem geeigneten Offset besteht. In Abhängigkeit von der maximalen Drift-Rate kann durch periodische Synchronisation jedes Knotens mit einem Time-Server eine gute Präzision langfristig sichergestellt werden.

Kommen wir damit zum Ende unseres Überblicks über logische und zeitliche Synchronisationsprobleme in Verteilten Systemen. Die Fragen,

was ist Zeit, gibt es eine "reale Zeit", wie verändert sich die Zeit, wie ist sie meßbar?

sind unter anderem von Kant und Einstein unter dem Blickwinkel der Philosophie und der Physik untersucht worden. In modernen, räumlich getrennten Rechnersystemen bekommt dieser Begriff auf Grund der Synchronisationsproblematik eine zusätzliche Bedeutung. Überlegungen zum Thema Zeit aus Sicht der Informatik kann man unter anderem in [PR94, DRS94] finden.

[5] Für moderne Workstations liegt diese typischerweise im Bereich von $10^{-5} \ldots 10^{-6}$, d.h. Abweichungen von bis zu einer Sekunde pro Tag sind üblich.

Literatur

[ABD90] H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, R. Reischuk, Renaming in an Asynchronous Environment, J.ACM 37, 1990, 524-548.

[CM86] K. Chandy, J. Misra, How Processes Learn, Distributed Computing 1, 1986, 40-52.

[Do82] D. Dolev, The Byzantine Generals Strike Again, J. Algorithms 3, 1982, 14-30.

[DHS86] D. Dolev, J. Halpern, R. Strong, On the Possibility and Impossibility of Achieving Clock Synchronization, J. Computer and System Sciences 32, 1986, 230-250.

[DHS95] C. Dwork, C. Ho, R. Strong, Collective Consistency, Technical Report, IBM Almaden Research, 1995.

[DR85] D. Dolev, R. Reischuk, Bounds on Information Exchange for Byzantine Agreement, J.ACM 32, 1985, 191-204.

[DRS90] D. Dolev, R. Reischuk, R. Strong, Early Stopping in Byzantine Agreement, J.ACM 37, 1990, 720-741.

[DRS94] D. Dolev, R. Reischuk, R. Strong, Observable Clock Synchronisation, Proc. 14. ACM Symposium on Principles of Distributed Computing, 1994, 284-293.

[DS83] D. Dolev, R. Strong, Authenticated Algorithms for Byzantine Agreement, SIAM J. Computing 12, 1983, 656-666.

[FHV92] R. Fagin, J. Halpern, M. Vardi, What Can Machines Know? On the Properties of Knowledge in Distributed Systems, J.ACM 39, 1992, 328-376.

[FLM86] M. Fischer, N. Lynch, M. Merritt, Easy Impossibility Proofs for Distributed Consensus Problems, Distributed Computing 1, 1986, 26-39.

[FLP85] M. Fischer, N. Lynch, M. Paterson, Impossibility of Distributed Consensus with 1 Faulty Process, J.ACM 32, 1985, 374-382.

[GM93] J. Garay, Y. Moses, Fully Polynomial Byzantine Agreement in $t+1$ Rounds, Proc. 25. ACM Symposium on the Theory of Computing, 1993, 31-41.

[HF89] J. Halpern, R. Fagin, Modelling Knowledge and Action in Distributed Systems, Distributed Computing 3, 1989, 159-177.

[HM90] J. Halpern, Y. Moses, Knowledge and Common Knowledge in a Distributed Environment, J.ACM 37, 1990, 549-587.

[La78] L. Lamport, Time, Clocks and the Ordering of Events in Distributed Systems, Communications of the ACM 21, 1978, 558-565.

[La84] L. Lamport, Using Time instead of Timeout for Fault-Tolerant Distributed Systems, ACM Tr. on Progr. Lang. and Systems 6, 1984, 254-280.

[Le84] D. Lehmann, Knowledge and Common Knowledge and Related Puzzles, Proc. 3. ACM Symposium on Principles of Distributed Computing, 1984, 62-67.

[LL84] J. Lundelius, N. Lynch, An Upper and Lower Bound for Clock Synchronization, Information & Control 62, 1984, 190-204.

[LSP82] L. Lamport, R. Shostak, M. Pease, The Byzantine Generals Problem, ACM Tr. on Progr. Lang. and Systems 4, 1982, 382-401.

[Mi91] D. Mills, Internet Time Synchronization: The Network Time Protocol, IEEE Tr. Computers 39, 1991, 1482-1493.

[MB94] Y. Moses, B. Bloom, Knowledge, Timed Precedence and Clocks, Proc. 13. ACM Symposium on Principles of Distributed Computing, 1994, 294-303.

[MDH86] Y. Moses, D. Dolev, J. Halpern, Cheating Husbands and other Stories, Distributed Computing 1, 1986, 167-176.

[MT88] Y. Moses, M. Tuttle, Programming Simultaneous Actions Using Common Knowledge, Algorithmica 3, 1988, 121-170.

[PR94] B. Patt-Shamir, S. Rajsbaum, A Theory of Clock Synchronization, Proc. 26. ACM Symposium on the Theory of Computing, 1994, 810-819.

[PSL80] M. Pease, R. Shostak, L. Lamport, Reaching Agreement in the Presence of Faults, J.ACM 27, 1980, 228-234.

[Ru94] J. Rushby, A Formally Verified Algorithm for Clock Synchronization Under a Hybrid Fault Model, Proc. 13. ACM Symposium on Principles of Distributed Computing, 1994, 304-313.

[Sc87] F. Schneider, Understanding Protocols for Byzantine Clock Synchronization, Technical Report, Cornell University, 1987.