Mathematisch-naturwissenschaftliche Fakultät
Geographisches Institut der Universität Zürich
Abteilung für Geographische Informationssysteme

Decentralized Event Detection in Geosensor Networks

Masterarbeit GEO 511

Autor: Raphael Renaud
Oberzelgstrasse 22
5612 Villmergen
E-Mail: raphael.renaud@gmx.ch
Matrikel-Nummer: 06-708-440

Betreuung: Dr. Patrick Laube

Fakultätsvertretung: Prof. Dr. Robert Weibel

Abgabedatum: 26. Januar 2011

Zusammenfassung

Umweltphänomene und -katastrophen wie Ölverschmutzungen in Gewässern, Waldbrände, Verbreitung invasiver Arten, Luftverschmutzung, Radioaktivität, etc. werden durch den Menschen überwacht, um möglichst früh Massnahmen zur Eindämmung negativer Folgen einleiten zu können. Neben herkömmlichen Messmethoden wie der Fernerkundung, lokaler Messstationen, visueller Überwachung etc., ermöglichte der technische Fortschritt in den letzten Jahren den Einsatz von sogenannten „Geosensor Networks“. Ziel solcher Sensornetzwerke ist es, selbstständig Informationen mit hoher räumlich-zeitlicher Auflösung in der Umwelt zu sammeln und diese zur Verfügung zu stellen. Doch speziell die knappen Energieressourcen der einzelnen Sensorknoten stellen noch eine hohe Hürde dar, bevor solche Netzwerke grossflächig sinnvoll eingesetzt werden können. Da jedoch die Rohdaten der Sensoren oft eine hohe Redundanz und dadurch einen kleinen Informationsgehalt besitzen, und durch den Transport von Daten ein hoher Energieverbrauch resultiert, wird der dezentralen (Vor-)Verarbeitung der Daten ein hohes Energiesparpotential und somit eine höhere Lebensdauer der Sensornetzwerke attestiert.

Diese Arbeit befasst sich mit der Detektion von Ereignissen (z.B. Waldbrände) in Geosensor-Netzwerken. Anhand eines neu entwickelten dezentralen Algorithmus wird versucht, das Potential dezentraler Ereignis-Detektion gegenüber der zentralen Datenverarbeitung in Bezug auf den Energieverbrauch umfassend zu ermitteln. Es wird nicht nur der durchschnittliche Energieverbrauch, welcher als proportional zur Netzwerkaktivität betrachtet wird, sondern auch die Verteilung der Belastung auf die einzelnen Knoten untersucht.

Die Erkenntnisse bezüglich der Performance von zentralen und dezentralen Methoden basieren auf den zwei spezifisch verwendeten Algorithmen, weshalb die allgemeinen Erkenntnisse und Aussagen eher als Tendenz zu verstehen sind. Die Ergebnisse zeigen jedoch, dass in dezentralen Algorithmen durchaus ein hohes Potential steckt, die Lebensdauer von Sensornetzwerken zu erhöhen. So scheinen dezentrale Methoden tendenziell gegenüber zentralen Methoden besser zu performen, je grösser das Untersuchungsgebiet und somit das Sensornetzwerk ist (Skalierbarkeit). Bei extrem niedriger und extrem hoher räumlicher Autokorrelation der Ereignisse im Verhältnis zur durchschnittlichen Distanz zwischen den einzelnen Sensoren (Sensordichte) scheinen dezentrale Algorithmen tendenziell ihre Vorteile gegenüber zentralen Methoden zu verlieren. Bei einem statischen Routing scheint der dezentrale Algorithmus jedoch praktisch unabhängig von verschiedenen Parametern wie Sensordichte, dem gewählten Graphen für die Kommunikation etc. und unabhängig vom durchschnittlichen Energieverbrauch die Belastung und somit den Energieverbrauch besser auf die einzelnen Sensorknoten zu verteilen, was insgesamt in einer längeren Lebensdauer des gesamtem Geosensor Netzwerk oder zumindest Teilen davon resultieren dürfte.

Da der durchschnittliche Energieverbrauch des dezentralen Algorithmus relativ gesehen zur zentralen Methode bei der Variation eines Parameters sich nicht unbedingt in die gleiche Richtung (z.B. bei der Variation der Sensordichte) oder zumindest sich nicht gleich stark verändert wie die Belastung der schon am meist belasteten Knoten, sollte eine Minimierung des durchschnittlichen Energieverbrauchs nicht als alleiniges Ziel dezentraler Algorithmen verstanden werden. Diese als Nebenprodukt aus dieser Arbeit entstandene Erkenntnis scheint sehr zentral zu sein, wird in vielen Arbeiten doch nur der durchschnittliche Energieverbrauch aller Sensorknoten als alleiniger Indikator für die Energieeffizienz von Algorithmen verwendet.

Summary

Environmental phenomena and disasters such as oil spills, forest fires, spread of invasive species, air pollution, radioactivity, etc. are increasingly monitored to take early actions to reduce possible negative consequences. In addition to traditional methods such as remote sensing, local metering stations, visual surveillance, etc., the technical progress in recent years made possible the use of so-called geosensor networks. The aim of such sensor networks is to automatically collect information at a high spatial and temporal resolution in the environment and make them available. But especially the scarce energy resource of each sensor node is still a big problem before the use of such networks over large areas makes sense. However, since the raw data from the sensors often have a high redundancy and thus contain a small amount of information, and the transportation of data uses much energy, distributed (pre-)processing of data is attested to have a high energy-saving potential and thus to increase the lifetime of such networks.

This thesis deals with the detection of events (e.g. forest fires) in geosensor networks. Using a newly developed decentralized algorithm, it tries to identify comprehensively the potential of distributed event detection over central data processing in terms of energy consumption. Not only the average energy consumption, which is considered to be proportional to the networks activity, but also the distribution of the network-load is examined.

The findings are based on the performance of two specific algorithms, the centralized and the decentralized, so the overall insights and statements are to be understood more as a tendency. The results show that in distributed algorithms lies a very high potential to increase the lifetime of sensor networks. Decentralized methods seem to overperform central methods the more, the larger the study area and the sensor networks is (scalability). At extremely low and extremely high spatial autocorrelation of the events relative to the average distance between the sensors (sensor density), decentralized algorithms seem to lose their benefits to central methods. In a static routing, it seems that the distributed algorithm balances the network-load better and thus distributes the energy consumption more evenly on the individual sensors which leads to a longer life of the entire network or at least parts of it, regardless of the average energy consumption. That seems to be nearly independent of various parameters such as sensor density, the selected graphs for the communication, etc.

Since the average energy consumption of the distributed algorithm in relative terms to the central methods does not change necessarily in the same direction (e.g. the variation of sensor density) or at least does not change equally strongly as the load on the already most loaded nodes by varying just one parameter, minimizing the average energy consumption should not be understood as the sole objective of decentralized algorithms. This seems to be a very important insight as a by-product of this thesis, looking at many papers which use the average energy consumption as the sole indicator of the energy efficiency of algorithms.

Danksagung

Ich möchte mich an dieser Stelle ganz herzlich bei allen bedanken, die mich im Verlaufe meines Geographiestudiums an der Universität Zürich und speziell während meiner Masterarbeit unterstützt haben.
Speziell möchte ich hier meinen Betreuer Dr. Patrick Laube erwähnen, der mich immer mit hilfreichen Tipps und auf eine motivierende Art und Weise durch die Arbeit begleitet hat. Auch ein herzliches Dankeschön geht an Dr. Matt Duckham von der Universität Melbourne, der sich bei seinem Aufenthalt in der Schweiz die Zeit genommen hat, mir einige Tipps sowie Anregungen für diese Masterarbeit auf den Weg zu geben.
Einen grossen Dank möchte ich auch an meine Familie sowie allen Angehörigen und Bekannten richten, die mich durch meine ganze Studienzeit sehr stark unterstützt haben.

Villmergen, 10.01.2011
Raphael Renaud

Inhalt

Abbildungen

Diagramme

Tabellen

1 Einleitung

1.1 Kontext und Motivation

Umweltphänomene und -katastrophen wie Ölverschmutzungen in Gewässern, Waldbrände, Verbreitung invasiver Arten, Luftverschmutzung, Radioaktivität, etc. werden durch den Menschen überwacht, um möglichst früh Massnahmen zur Eindämmung negativer Folgen einleiten zu können. Für die Beschaffung dieser Daten gibt es viele verschiedene Varianten wie die Fernerkundung, lokale Messstationen, visuelle Überwachung etc. All diese Methoden haben ihre Vor- und Nachteile. So müssen zum Beispiel bei der Satellitenüberwachung Kompromisse bezüglich der räumlichen und zeitlichen Auflösung eingegangen werden, sie ist jedoch oft die einzige vernünftige Möglichkeit, in unzugänglichen Gebieten Daten erheben zu können. Fortschritte in der Sensortechnik in den letzten Jahren brachten die „Geosensor Networks“ (vgl. Kapitel 2.1) hervor, welche eine neue Variante der Erhebung geographischer Informationen darstellen. Ziel solcher Sensornetzwerke ist es, selbstständig Informationen mit hoher räumlich-zeitlicher Auflösung in der Umwelt zu sammeln und zur Verfügung zu stellen. Doch nach wie vor gibt es einige Probleme zu bewältigen, bevor solche Netzwerke flächendeckend eingesetzt werden können: Neben den hohen Kosten der einzelnen Sensorknoten sind begrenzte Ressourcen nach wie vor ein zentrales Problem. Nicht nur sind die Rechen- und Speicherkapazität sowie die Sendeleistung solcher Sensorknoten begrenzt, sondern speziell auch die Energie. Traditionell sind solche Sensorknoten mit endlichen Energiequellen wie Batterien ausgestattet. Ist die Batterie leer, funktioniert der Knoten nicht mehr. Es gibt Anstrengungen, solche Sensorknoten zum Beispiel mit Solarpanels auszustatten, um die Lebensdauer theoretisch bis ins Unendliche verlängern zu können. Zum einen sind aber solche Lösungen mit erneuerbarer Energie nicht in jeder Situation möglich, zum anderen wird ab einer gewissen Grösse des Sensornetzwerkes die Netzwerkkapazität an ihre Grenzen stossen, wenn man davon ausgeht, dass alle gesammelten Daten innerhalb eines solchen Netzwerkes über die einzelnen Sensorknoten an einen zentralen Punkt gesendet werden müssen und die Energie nicht in unendlicher Menge bereitgestellt werden kann. Aus diesem Grund werden grosse Anstrengungen unternommen, um diesen Energieverbrauch zu minimieren und somit die Lebensdauer solcher Netzwerke zu maximieren.

Eine hohe räumliche Sensordichte sowie eine hohe zeitliche Auflösung der Messung wird unweigerlich zu einer hohen Redundanz an Daten führen, werden alle Rohdaten gesammelt. Hat man eine hohe Redundanz, ist jedoch automatisch der Informationsgehalt niedrig. Dementsprechend werden Lösungen gesucht, welche die Redundanz der Daten auf ein Minimum reduzieren, da davon ausgegangen werden kann, dass dadurch auch die Menge an zu übermittelnden Daten und somit auch die Netzwerkaktivität reduziert werden kann, was zu einer Erhöhung der Lebensdauer solcher Netzwerke führen soll. Ein Ansatz dafür ist, die Verarbeitung der Messdaten zu dezentralisieren.

Mit dem Thema „Decentralized Event Detection in Geosensor Networks“ versucht diese Arbeit, die Möglichkeiten und Grenzen von solch dezentralen Methoden besser zu verstehen.

1.2 Problemstellung und Ziele

Geosensor Networks (vgl. Kapitel 2.1) sind, wie schon in Kapitel 1.1 erwähnt, ressourcenseitig sehr beschränkt. So ist insbesondere die Energie der limitierende Faktor der Lebensdauer eines solchen Netzwerks. Zudem werden in Zukunft einzelne Sensoren voraussichtlich immer kleiner und billiger in der Produktion, was die Möglichkeit mit sich bringt, solche Netzwerke immer grossflächiger zu nutzen. Eine Vergrösserung solcher Netzwerke bringt jedoch mehrere Nebeneffekte mit sich: So wird die Netzwerklast immer höher, sollen alle Rohdaten von allen Sensoren zu einem zentralen Server

gesendet werden. Diese Tatsache wird wohl das Sensornetzwerk in der Grösse limitieren, zumal die Lebensdauer durch die hohe Netzwerklast und somit durch einen hohen Energieverbrauch verringert wird. Dies wirft die Frage nach alternativen Methoden auf, um diesen Problemen entgegenzuwirken. Ein Ansatz ist, die Lösung eines Problems, wie schon in Kapitel 1.1 erwähnt, zu dezentralisieren. Dies bringt mehrere potentielle Vorteile mit sich: Da jeder Sensor auch selber einen kleinen Prozessor besitzt, skaliert die gesamte Rechenkapazität linear mit der Grösse des Netzwerks. Diesem Argument kann man entgegensetzen, dass bei einem zentralen Server theoretisch von einer unendlichen Rechenleistung ausgegangen werden kann, da dieser nicht energiekritisch betrieben wird. Viel relevanter aber ist die Tatsache, dass bei einer dezentralen Berechnung nur relevante Informationen zum zentralen Server gemeldet werden müssen und somit der Energieverbrauch, welcher durch den Transport von Informationen verursacht wird, minimiert werden kann, macht man sich noch zusätzlich bewusst, dass der Energieverbrauch, um 1 Kilobyte Daten über 100m Distanz zu senden, etwa gleich gross ist wie der, um 3 Millionen einfache Instruktionen auf einem Prozessor durchzuführen (Shebli, Dayoub, & Rouvaen, 2007).
Das Ziel dieser Arbeit ist es, eine neue Methode zur dezentralen Ereignis-Detektion zu erarbeiten und anhand dieser zu untersuchen, welche Möglichkeiten dezentrale Algorithmen bei der Ereignis-Detektion unter Berücksichtigung der räumlichen Autokorrelation in Geosensor Networks mit sich bringen und wo deren Grenzen sein können. Das Wort „können“ bezieht sich darauf, dass anhand eines neu entwickelten Algorithmus nie behauptet werden kann, es gäbe nicht noch bessere Lösungen. Jedoch sollen Schwierigkeiten, tendenzielle Grenzen und definitive Möglichkeiten in dieser Arbeit aufgezeigt werden.

1.3 Forschungsfragen

Diese Arbeit ist angesiedelt in einem Gebiet, das sich mit folgender allgemeiner Forschungsfrage auseinandersetzt:

„Können bei der Ereignis-Detektion dezentrale Algorithmen den Energieverbrauch eines Geosensor Network verringern und wenn ja, unter welchen Bedingungen?“

Die Lebensdauer eines Geosensor Network hängt direkt mit dem Energieverbrauch der einzelnen Netzwerkknoten zusammen. Fällt ein Knoten aus, kann dieser weder eigene Informationen noch diejenigen von anderen Knoten weitersenden. Fällt einer oder mehrere Knoten aus, kann dies dazu führen, dass für Teile des Geosensor Networks keine Verbindung mehr zum sogenannten Sink-Node (der zentrale Knoten, welcher Daten an einen Server weiterleiten kann) vorhanden ist. Somit ist dieser Teil des Geosensor Network nicht mehr brauchbar, weshalb es von grossem Interesse ist, den Energieverbrauch einzelner Sensorknoten auf ein Minimum zu reduzieren. Aus oben genannter übergeordneter Forschungsfrage, welche den Kontext und auch den Fokus dieser Arbeit vorgibt, können aus geographischer Sicht folgende konkrete Forschungsfragen abgeleitet werden:

1. Inwiefern beeinflusst die räumliche Autokorrelation von Ereignissen die Effizienz dezentraler Ereignis-Detektion?
2. Welchen Einfluss hat die Grösse eines Geosensor Networks (=Anzahl Sensorknoten) auf die Effizienz dezentraler Algorithmen?

Es dürfte sehr interessant sein zu untersuchen, inwiefern die räumliche Autokorrelation eines Phänomens sowie die Grösse eines Geosensor Networks einen Einfluss auf die Effizienz der

dezentralen Ereignis-Detektion haben. Frage 1 kann in einem realen Feldversuch insofern nicht beeinflusst werden, als dass die zu detektierenden Ereignisse in der Regel vorgegeben sind, weshalb diese Frage von grossem Interesse sein dürfte, um potentielle Anwendungen dezentraler Ereignis-Detektion zu eruieren. Frage 2 zielt auf die Skalierbarkeit von dezentralen Algorithmen ab, da diese als zentral für die Einsatzmöglichkeiten solcher Netzwerke angesehen werden kann.
Diese Fragen sollen, wie schon in Kapitel 1.2 erwähnt, anhand eines neu zu entwickelnden dezentralen Algorithmus beantwortet werden. Aus diesem Grund kann schon jetzt festgehalten werden, dass die beiden Fragen sich wohl nicht abschliessend beantworten werden lassen, da dies nur möglich wäre, wenn alle möglichen dezentralen Methoden bekannt, implementiert und getestet werden könnten.

1.4 Aufbau der Arbeit

Die Gliederung der Arbeit orientiert sich stark am methodischen Vorgehen. Nach der Einleitung (Kapitel 1) wird auf die theoretischen Hintergründe eingegangen (Kapitel 2). Dort sollen die grundlegenden Theorien rund um das Thema „Decentralized Event Detection in Geosensor Networks“ erörtert werden, die im Kontext dieser Arbeit stehen. In der anschliessenden detaillierten Problemdefinition und Eingrenzung der Problemstellung (Kapitel 3) wird das in der Einleitung definierte Ziel der Arbeit konkretisiert, um die Anforderungen an den in Kapitel 4 zu entwickelnden Algorithmus abzustecken und anhand derer dieser Algorithmus dann in Kapitel 5 getestet und diskutiert werden kann.

2 Theoretischer Hintergrund

Dieses Kapitel führt in die theoretischen Hintergründe und den grösseren Kontext dieser Arbeit ein. In Kapitel 2.1 wird erklärt, was Geosensor Networks überhaupt sind. Kapitel 2.2 befasst sich mit der räumlichen Autokorrelation, um den theoretischen Hintergrund bezüglich der Forschungsfrage 1 zu erläutern. Kapitel 2.3 befasst sich mit der Frage, was unter dem Begriff „Ereignis-Detektion“ zu verstehen ist.

2.1 Geosensor Networks

Geosensor Networks sind laut Definition von Nittel et. al. (Report from the First Workshop on Geo Sensor Networks, 2004) Wireless Sensor Networks (WSN), welche Phänomene im geographischen Raum untersuchen und die aus sehr vielen einzelnen Sensorknoten bestehen (Alippi, Anastasi, Di Francesco, & Roveri, 2009). Wireless Sensor Networks selber werden als grosses, kabelloses, ad hoc, multi-hop, nicht-partitioniertes Netzwerk homogener, winziger, meist immobiler Knoten, welche zufällig im Interessensgebiet verteilt werden, definiert (Römer, 2005). Es gibt auch Ansätze, welche versuchen, eine optimale Platzierung solcher Knoten zu berechnen, wie zum Beispiel die Methode von Toumpis und Gupta (2005), weshalb in der Definition auch von „meist zufälliger Verteilung“ die Rede ist. In dieser Arbeit jedoch wird diese zufällige Verteilung (z.B. Abwurf auf einem Flugzeug) aus mehreren Gründen angenommen. Da davon ausgegangen wird, dass die einzelnen Sensorknoten immer kleiner und günstiger werden, geht man heute davon aus, dass die Netzwerke in Zukunft auch aus immer kleineren und immer mehr solcher Knoten bestehen werden, und dementsprechend auch eine grössere Fläche abdecken. Aus diesem Grund würden die Kosten einer Platzierung jedes einzelnen Knotens wohl zu hoch. Der zentrale Begriff hinter dieser Entwicklung nennt sich „Smart Dust“, welcher zum ersten Mal 1998 in der Literatur auftauchte und von einer Knotengrösse von nur noch einigen mm^3 ausgeht (Hsu, Kahn, & Pister, 1998; Römer, 2005). Ausserdem ist es sinnvoll, Methoden zu entwickeln, welche mit eine zufälligen Verteilung auskommen, da diese dann universeller einsetzbar sind.

Diese einzelnen (Sensor-)Knoten können als Mini-Computer angesehen werden. Jeder dieser Knoten besitzt einen Prozessor (CPU), Speicher, eine „Short-range Wireless Communication“, eine Energiequelle (z.B. Batterie) sowie einen oder mehreren Sensoren (Nittel, Duckham, & Kulik, 2004; Römer, 2005). Obwohl die exakten Eigenschaften dieser Sensorknoten stark variieren können, haben sie alle gemeinsam, dass sie in ihren Ressourcen beschränkt sind (Römer, 2005; Laube & Duckham, 2009; Krishnamachari & Iyngar, 2004; Carle & Simplot-Ryl, 2004). Die meisten Wireless Sensor Networks / Geosensor Networks gehören zu den „multi-hop ad hoc networks“, in welchen Sensorknoten nicht nur als Datenquelle agieren, sondern auch Mitteilungen / Informationen zu anderen Knoten weiterleiten. Da solche Sensorknoten meist nur eine relativ geringe Sendedistanz aufweisen, müssen die Knoten relativ dicht in der Umwelt verteilt werden, damit das gesamte Netzwerk als Ganzes funktionieren kann. Während einzelne Knoten in der Regel nur eine relativ limitierte Funktionalität aufweisen, kann das globale Verhalten solcher Sensornetzwerke relativ komplex sein. Der wahre Wert eines solchen Netzwerkes ist somit grösser als die Summe seiner Einzelteile (Römer, 2005).

Gemäss Bapat (1994) werden verteilte Systeme (Distributed Systems) als Informations-Prozessierungs-Systeme verstanden, welche eine Anzahl unabhängiger Computer beinhalten und die über ein Netzwerk miteinander kooperieren, um ein bestimmtes Ziel zu erreichen. Basierend auf dieser Definition können Geosensor Networks zu den verteilten Systemen gezählt werden (Römer,

2005). In manchen Systemen, welchen zugeschrieben wird, es handle sich um verteilte Systeme, nimmt jedes einzelne Element unterschiedliche Aufgaben wahr. So werden typischerweise in Client-Server-Systemen einzelne Aufgaben fix an die verschiedenen Einheiten verteilt (Laube & Duckham, 2009). Peer-to-peer (P2P) Systeme hingegen kennen diese Unterteilung von verschiedenen Aufgaben nicht und werden als selbst-organisierende dezentral verteilte Systeme mit symmetrischen Rollen definiert (Römer, 2005). Im Bereich der Sensor-Netzwerke werden beide dieser Varianten verfolgt. In dieser Arbeit wird jedoch von einer symmetrischen Rollenverteilung ausgegangen. Traditionell sind verteilte Systeme von der realen Umwelt entkoppelt. Dies ist der grosse Unterschied zu den Sensor-Netzwerken, welche direkt in die Umwelt integriert werden. Der Output eines solchen Geosensor Networks kann normalerweise als kontinuierlicher Fluss von Daten mit einem tiefen Informationsgehalt (Roh-Daten), hohem Datenvolumen, einer hohen Redundanz / hoher Korrelation sowie einem hohen Rauschlevel charakterisiert werden. Der gewollte Output ist aber typischerweise durch eine hohe Informationsdichte, ein niedriges Datenvolumen, tiefe Redundanz und hohe Genauigkeit charakterisiert (Römer, 2005). Hier lässt sich schon ein gewisser Vorteil durch die dezentrale (Vor-)Verarbeitung von Daten vermuten (Duckham & Reitsma, 2009; Römer, 2008).
Alle Informationen müssen schlussendlich an einem zentralen Ort gesammelt werden, damit sie dem Benutzer zur Verfügung stehen. Dies geschieht bei einen sogenannten Sink-Node, welcher in der Regel über Internet oder Satellit direkt mit dem Datencenter / Enduser verbunden ist. Dieser Sink-Node verfügt in der Regel über weitaus mehr, im Normalfall sogar „unendlich“ Ressourcen, weil er nicht auf eine endliche Stromversorgung angewiesen ist:

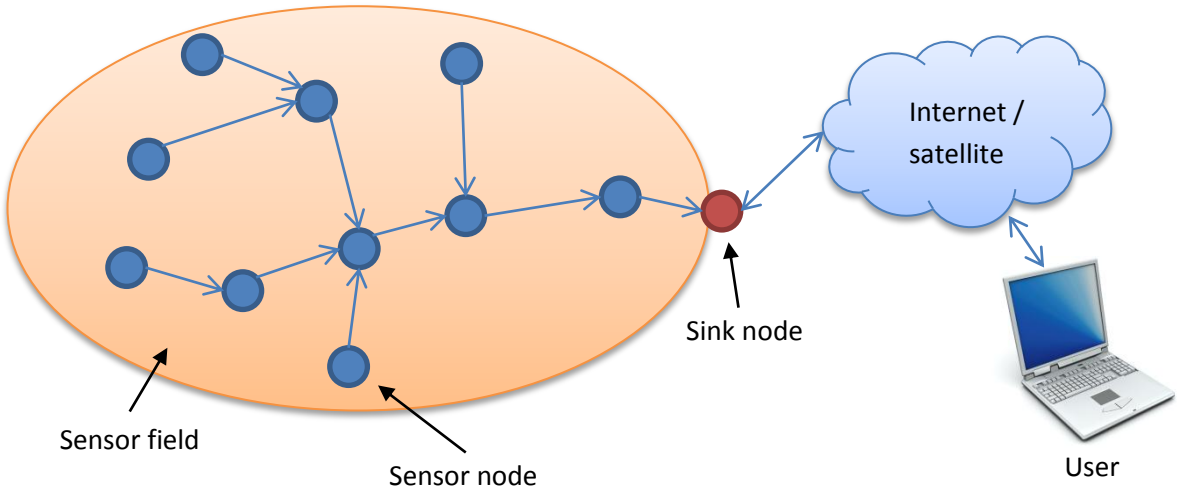


Abbildung 2.1: Typische Sensor-Netzwerkarchitektur
Typische Sensor-Netzwerkarchitektur in Anlehnung an Akyildiz, Su, Sankarasubramaniam & Cayirci (2002) und Alippi, Anastasi, Di Francesco & Roveri (2009).

Die Technologie der Sensornetzwerke ist noch relativ jung und teuer. Einer der wenigen kommerziellen Hersteller solcher Sensornetz-Systeme ist Libelium[1], welcher schon modulare Sensorknoten unter dem Namen „Waspmote“ anbietet, die den ZigBee-Standard[2] benutzen. Diese Knoten sind jedoch mit einer Fläche von fast 40cm^2 noch relativ gross und scheinen vom Ziel des „Smart Dust“ noch relativ weit entfernt. Welche Annahmen über die einzelnen Sensoren in dieser Arbeit verwendet werden, ist in Kapitel 3.1 zu finden.

[1] Libelium ist eine spanische Firma, welche sich auf Hardwareproduktion sowie die Implementierung von Kommunikationsprotokollen in Wireless Sensor Networks spezialisiert hat: http://www.libelium.com/
[2] ZigBee ist ein von der ZigBee-Allianz entwickeltes Funknetz-Protokoll für „low-cost, low-power wireless sensors“, welches sich als Standard etablieren soll: http://www.zigbee.org/

2.2 Räumliche Autokorrelation

Eine der fundamentalsten Annahmen in der statistischen Analyse ist, dass Stichproben unabhängig voneinander sind, wie zum Beispiel das Münzenwerfen oder das Würfeln. In der Analyse räumlicher Daten ist diese Annahme der Unabhängigkeit grundsätzlich falsch. Vielmehr tendieren räumliche Daten dazu, in höchstem Grade selbstkorrelierend zu sein. So ändern sich beispielsweise die natürlichen Ressourcen, die Vegetation, die Fauna sowie die Temperatur graduell über den Raum. Diese Tendenz lässt sich aber nicht nur in der Natur, sondern auch bei soziogeographischen Aspekten beobachten. So tendieren Leute mit ähnlichen Charakteren, sozialem Hintergrund und Einkommen dazu, sich in gleichen Gebieten niederzulassen, was in der Regel zu Clustern führt. Dieses Phänomen scheint so zentral zu sein, dass Geographinnen und Geographen es zum ersten Gesetz der Geographie gekürt haben (Shekhar, Zhang, Huang, & Vatsavai, 2003):

> *„Everything is related to everything else, but near things are more related than distant things."* (Tobler, 1970, S. 236)

Diese Eigenschaft nennt sich in der räumlichen Statistik „räumliche Autokorrelation". Demnach sind Dinge im Raum nicht zufällig verteilt, sondern haben eine gewisse Relation zueinander, abhängig von deren räumlichen Distanz. Folgende Abbildung zeigt den Vergleich zweier räumlicher Verteilungen mit und ohne räumliche Autokorrelation.

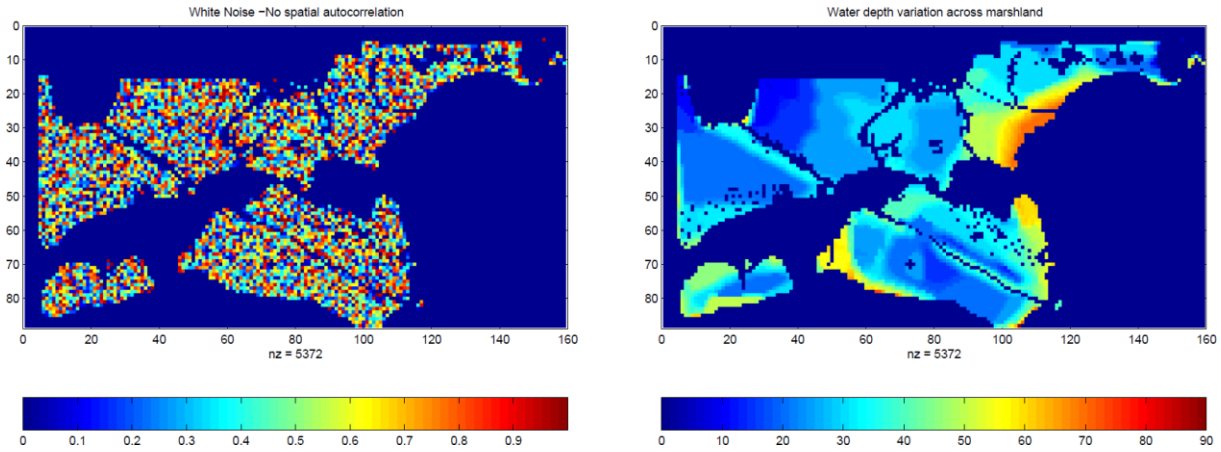


Abbildung 2.2: Autokorrelation
Weissrauschen mit unabhängiger Verteilung (links) und Wassertiefe mit räumlicher Autokorrelation (rechts) (Shekhar, Zhang, Huang, & Vatsavai, 2003)

Diese Korrelation, also die Abhängigkeit verschiedener Variablenwerte voneinander, kann anhand statistischer Verfahren berechnet werden. Nehmen wir ein Gitter von 6 x 6 binären Feldern, welche zum Beispiel das Vorhandensein einer bestimmten Pflanzenart (0 = nicht vorhanden, 1 = vorhanden) in einem Raster darstellen (Tabelle 2.1, nächste Seite), so ergeben sich je nach räumlicher Autokorrelation völlig andere Muster. Eine positive räumliche Autokorrelation, wie auch eine negative räumliche Autokorrelation, beruht meist auf vielen verschiedenen Faktoren, welche sich gegenseitig beeinflussen. Dass eine einzelne Pflanzenart zum Beispiel eine hohe räumliche Autokorrelation aufweist, hängt oft mit der lokalen Bestäubung zusammen, während der Wettkampf um Ressourcen die räumliche Autokorrelation in der Regel eher in eine negative Richtung beeinflusst.

Komplett geteiltes Muster						Gleichmässig verteiltes Muster						Zufälliges Muster					
1	1	1	0	0	0	1	0	1	0	1	0	0	0	1	1	0	1
1	1	1	0	0	0	0	1	0	1	0	1	0	1	1	0	1	0
1	1	1	0	0	0	1	0	1	0	1	0	1	0	1	1	0	0
1	1	1	0	0	0	0	1	0	1	0	1	0	0	0	0	0	1
1	1	1	0	0	0	1	0	1	0	1	0	1	1	1	1	0	0
1	1	1	0	0	0	0	1	0	1	0	1	1	0	1	1	0	1

Tabelle 2.1: Räumliche Verteilungsmuster
Unterschiedliche Verteilungsmuster einer binären Variable (z.B. das Vorhandensein einer Pflanzenart) in einem 6 x 6-Raster angelehnt an De Smith, Goodchild & Longley (2007). Das linke Muster weist eine starke positive räumliche Autokorrelation auf, da die Wahrscheinlichkeit des Vorhandenseins dieser Pflanzenart sehr hoch ist, wenn in den Nachbarzellen die Pflanzenart auch vorhanden ist. Das gegenteilige Bild, also eine negative räumliche Autokorrelation, zeigt das mittlere Raster, bei welchem die Pflanzenart nie auf zwei aneinandergrenzenden Flächen vorkommt. Das rechte Muster zeigt keine räumliche Autokorrelation, da die Verteilung komplett unabhängig von den Nachbarzellen ist.

Die zwei wohl bekanntesten Verfahren zur Berechnung der Autokorrelation sollen hier kurz erwähnt werden. Es handelt sich um den Moran's I und den Geary's C (De Smith, Goodchild, & Longley, 2007).

2.2.1 Moran's I

Das in Tabelle 2.1 dargestellte Beispiel kann intuitiv nachvollzogen werden. Jedoch ist dieses Beispiel auch stark idealisiert und wird in der Natur in dieser Art wohl eher selten anzutreffen sein. Aus diesem Grund braucht es statistische Verfahren, mit denen auch komplexere Daten verrechnet werden können. Dies kann zum einen sein, dass anstelle von nominalen Daten ordinal-, intervall- oder ratio-skalierte Daten stehen. Zum anderen kann es sich um ein Raster handeln, bei welchem die Nachbarschaftsdistanz und somit die Gewichtung der einzelnen Werte zueinander variieren kann (durch Hindernisse etc.). Somit können eine Menge von Werten $\{z_i\}$ sowie eine Menge Gewichte $\{w_{ij}\}$, welche die räumliche Distanz zwischen zwei einzelnen Werten wiedergeben, angenommen werden. Jetzt wird eine Funktion gesucht, welche folgende Kriterien erfüllt (De Smith, Goodchild, & Longley, 2007):

- Wenn zwei Werte (z_i, z_j) ähnlich sind, soll $f(z_i, z_j) > 0$ gelten.
- Wenn zwei Werte (z_i, z_j) unterschiedlich sind, soll $f(z_i, z_j) < 0$ gelten.
- Wenn beide Werte (z_i, z_j) gross sind, soll $f(z_i, z_j)$ gross sein.
- Die räumliche Distanz $\{w_{ij}\}$ soll in die Berechnung einbezogen werden.

Einer der einfachsten Wege, die ersten drei Kriterien zu erfüllen, ist, die Werte zu multiplizieren:

Formel 2.1
$$\sum_i \sum_j z_i z_j$$

Fügt man noch eine Korrektur des Mittelwertes sowie die Gewichtung hinzu, bekommt man folgendes:

Formel 2.2
$$\sum_i \sum_j w_{ij}(z_i - \overline{z})(z_j - \overline{z})$$

Angenommen, die räumliche Distanz w_{ij} kann nur die Werte 0 oder 1 annehmen, dann würde sie nur bestimmen, welche Elemente in die Berechnung einbezogen werden und welche nicht. Dieser Ausdruck scheint ähnlich der Kovarianz der selektierten Daten zu sein. Um die Anzahl einbezogener Werte dieser Summe anzugleichen und somit einen kovarianten Wert zu erhalten, muss durch die Summe der Gewichte dividiert werden:

Formel 2.3
$$\frac{\sum_i \sum_j w_{ij}(z_i - \overline{z})(z_j - \overline{z})}{\sum_i \sum_j w_{ij}}$$

Um diese Kovarianz zu standardisieren, dividieren wir diesen Ausdruck durch die Varianz der Daten, welche durch folgende Formel gegeben ist:

Formel 2.4
$$\frac{\sum_i (z_i - \overline{z})^2}{n}$$

Das Verhältnis von Formel 2.3 zu Formel 2.4 gibt uns den Moran's I (Formel 2.5), welcher typische Werte zwischen ungefähr +1 für komplett positive Autokorrelation und ungefähr -1 für komplette negative Autokorrelation aufweist (De Smith, Goodchild, & Longley, 2007):

Formel 2.5
$$I = \frac{1}{p} \frac{\sum_i \sum_j w_{ij}(z_i - \overline{z})(z_j - \overline{z})}{\sum_i (z_i - \overline{z})^2}$$, wobei

Formel 2.6
$$p = \sum_i \sum_j w_{ij} / n$$

Das obere und untere Limit dieses Indexes variiert je nach den Gewichten, die verwendet werden.
Der Erwartungswert von I bei bei räumlich unabhängigen Zufallswerten ist:

Formel 2.7
$$E(I) = -\frac{1}{N-1}$$

Für ein grosses n ist somit der Erwartungswert 0. Die Varianz von I ist unter diesen Umständen wie folgt definiert:

Formel 2.8
$$Var(I) = \frac{n^2(n-1)A - n(n-1)B - 2C^2}{(n+1)(n-1)C^2}$$, wobei

Formel 2.9
$$A = \frac{1}{2} \sum_i \sum_j (w_{ij} + w_{ji})^2, i \neq j$$

Formel 2.10
$$B = \sum_k \left(\sum_j w_{jk} + \sum_i w_{ik} \right)^2$$

Formel 2.11
$$C = \sum_i \sum_j w_{ij}, i \neq j$$

Der Moran's I ist in praktisch allen GIS-Paketen implementiert und wohl der verbreitetste Indikator, um die räumliche Autokorrelation zu quantifizieren. Er findet oft Verwendung in der univariaten Statistik sowie in Zeitreihenanalysen (De Smith, Goodchild, & Longley, 2007).

2.2.2 Geary's C

Eine Alternative zum Moran's I geht auf die Formel 2.1 zurück. Anstatt die Summen der Produkte miteinander zu verrechnen, könnte man auch die quadratische Differenz berechnen, da diese die gleichen Eigenschaften aufweist:

Formel 2.12
$$\sum_i \sum_j (z_i - z_j)^2$$

Analog zum Verfahren in Kapitel 2.2.1 kann auch hier durch eine Division der Varianz der Wert standardisiert werden, womit wir zu folgender Formel kommen, welche als Geary's C bekannt ist:

Formel 2.13
$$C = \frac{1}{p} \frac{\sum_i \sum_j w_{ij}(z_i - z_j)^2}{\sum_i (z_i - \overline{z})^2}$$, wobei

Formel 2.14
$$p = 2\frac{\sum_i \sum_j w_{ij}}{n-1}$$

Der Nachteil dieses Verfahrens gegenüber dem Moran's I ist, dass die Interpretation nicht mehr ganz so einfach ist. Der Wert von C variiert um 1, was eine nicht räumliche Autokorrelation bedeutet. Werte bei 0 zeigen eine positive Autokorrelation und Werte grösser als 1 eine negative Autokorrelation an (De Smith, Goodchild, & Longley, 2007).

2.2.3 Lokale räumliche Autokorrelation

Die zwei Indikatoren Moran's I und Geary's C sind sehr gute Indizien, um die räumliche Autokorrelation zu untersuchen, sie greifen jedoch auf die Gesamtheit aller Messwerte zurück und sind somit globale Indikatoren für die Messung räumlicher Autokorrelation. Für bestimmte Zwecke kann es jedoch interessanter sein, lokale Indikatoren zu benutzen, da nur die Korrelation innerhalb eines bestimmten Gebietes von Interesse ist. Für lokale Indikatoren hat sich der Begriff Local Indicators of Spatial Association (LISA) durchgesetzt. In diesem Bereich gibt es einige Literatur, jedoch basiert diese meist auf der Arbeit von Anselin (1995), welcher unter anderem den lokalen Moran- sowie Geary-Koeffizienten entwickelte.
Anselin (1995) definiert einen LISA als statistische Kenngrösse, welche folgende zwei Kriterien erfüllt:

- Der LISA gibt für jede Untersuchung eine Angabe über das Ausmass des signifikanten räumlichen Clusters ähnlicher Werte im Untersuchungsgebiet.
- Die Summe aller LISAs für alle Untersuchungen ist proportional zum globalen Indikator der räumlichen Grundgesamtheit.

Formal kann ein LISA der Variable y_i an Position i als eine Kenngrösser L_i so definiert werden, dass

Formel 2.15
$$L_i = f(y_i, y_j)$$

wobei y_i die untersuchten Werte in der Nachbarschaft J_i von i sind. Des Weiteren sollte L_i so definiert sein, dass es möglich ist, eine statistische Signifikanz der LISA an der Position i abzuleiten, so dass

Formel 2.16 $$Prob[L_i > \delta_i] \leq \propto_i$$

, wobei δ_i den Grenzwert und $\propto_i$ die gewählte Signifikanz darstellt.
Das zweite Kriterium kann formal als

Formel 2.17 $$\sum_i L_i = \gamma\Lambda$$

beschrieben werden, wobei Λ als globaler Indikator der Grundgesamtheit und γ als Skalierungsfaktor fungiert. Anders ausgedrückt ist die Summe der lokalen Indikatoren proportional zum globalen Indikator. Anselin (1995) definiert unter oben gegebener Definition den lokalen Moran einer Untersuchung i als

Formel 2.18 $$I_i = z_i \sum_j w_{ij} z_j$$

, wobei, analog zum globalen Moran's I, die Werte z_i und z_j analog zur Formel 2.2 um den Mittelwert korrigiert werden:

Formel 2.19 $$I_i = (z_i - \overline{z}) \sum_j w_{ij}(z_j - \overline{z})$$

Die Summe über j wird so gewählt, dass nur benachbarte Werte $j \epsilon J_i$ berücksichtigt werden.

Inwiefern obige Kennzahlen für die Ereignis-Detektion von Nutzen sind, wird sich noch herausstellen, jedoch lassen sich wichtige grundlegende Annahmen über die räumliche Autokorrelation ableiten, welche immer in Betracht gezogen werden sollten.
Die räumliche Autokorrelation...

- ...hängt von der Gewichtung zwischen den Messwerten ab, welche die räumliche Distanz der Messerwerte zueinander widerspiegelt.
- ...ist in einer lokalen Definition abhängig von den in die Nachbarschaft einbezogenen Werten.

Vuran et. al. (2004) sehen in der räumlichen Autokorrelation ein grosses Potential, energieeffiziente Protokolle für die Ereignis-Detektion zu entwickeln (vgl. Kapitel 2.3), wenn man sich dieses Wissen zu Nutze macht.

2.2.4 Modifiable Areal Unit Problem (MAUP)

Sobald Daten aggregiert werden müssen, um ein Phänomen zu untersuchen, wird das „modifiable areal unit problem“ (MAUP) omnipräsent. Abhängig vom Aggregationslevel und der Zonierung kann das Muster eines Phänomens variieren, was zu falschen Erkenntnissen und Entscheiden führen kann (Openshaw & Taylor, 1979). Folgende Tabelle veranschaulicht das MAUP bezüglich der Berechnung der räumlichen Autokorrelation:

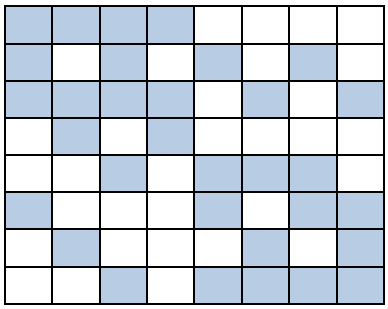
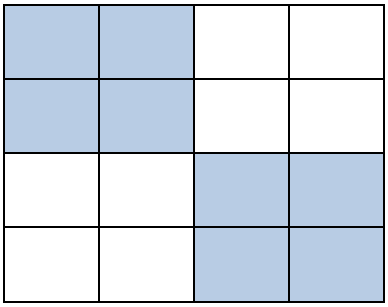
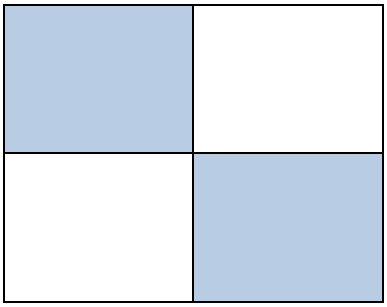
$0.25m^2$ Auflösung	auf $0.5m^2$ aggregiert	auf $1m^2$ aggregiert
		

Tabelle 2.2: MAUP bei der räumlichen Autokorrelation
Diese Tabelle stellt das MAUP im Zusammengang mit der räumlichen Autokorrelation dar. Während bei der Auflösung von $0.25m^2$ wohl eine leicht positive Autokorrelation festzustellen wäre, würde mit denselben Daten bei einer Aggregation auf $0.5m^2$ eine fast perfekt positive räumliche Autokorrelation suggeriert, während bei einer Aggregation auf $1m^2$ eine perfekte negative Autokorrelation festgestellt würde.

Betrachtet man obige Tabelle, so sieht man, dass es Situationen geben kann, bei denen mit denselben Daten zwei gegenteilige Schlussfolgerungen bezüglich räumlicher Autokorrelation gemacht werden können. Man muss sich auch immer bewusst sein, dass schon bei der Erhebung der Daten eine gewisse Aggregation vorgenommen wird, welche durch die technischen Möglichkeiten oder die zuvor definierte räumliche Auflösung gegeben ist. Es gibt zwar (noch) keine definitive Lösung des Problems, aber wenn man sich des Problems bewusst ist, ist es in den meisten Fällen möglich, Schlussfolgerungen richtig interpretieren zu können. Wichtig scheint jedoch zu sein, dass berechnete Werte nicht überinterpretiert werden. So macht es wahrscheinlich kaum Sinn, eine Korrelation auf zehn Nachkommastellen zu interpretieren, wenn bei einer minimalen Änderung der Auflösung der Daten schon in der ersten Nachkommastelle eine Veränderung eintreten würde.

2.3 Ereignis-Detektion

Da, wie weiter unten in diesem Kapitel noch zu sehen ist, ein Ereignis (Event) ein Spezialfall eines Ausreissers (Outlier) ist, müssen an dieser Stelle zuerst die dem Ausreisser zugrundeliegenden Konzepte erklärt werden.

Da das Gebiet der Ausreisser-Detektion von vielen verschiedenen Forschungsrichtungen wie Betrugsermittlung, Netzwerkintrusion, Performance-Analyse, Wettervorhersage, etc. abgedeckt wird (Chandola, Banerjee, & Kumar, 2007; Zhang, Meratnia, & Havinga, 2008), ist es nicht verwunderlich, dass in der Literatur viele Definitionen für Ausreisser zu finden sind. Zwei sollen hier genannt werden:

- Ein Ausreisser ist eine Beobachtung (oder eine Teilmenge von Beobachtungen), welche inkonsistent zum Rest des Datensatzes zu sein scheint (Barnett & Lewis, 1994).
- Ausreisser sind Muster in Daten, welche nicht in ein definiertes normales Verhalten oder zu einem definierten ausreisserischem Verhalten passen (Chandola, Banerjee, & Kumar, 2007).

Gemeinsam haben alle Definitionen, dass Ausreisser als abweichende (Mess-)Werte von der Grundgesamtheit zu sehen sind. Im räumlichen Bezug können räumliche Ausreisser (spatial outliers) somit als lokale Instabilität von Werten nicht-räumlicher Attribute oder als ein räumlich referenziertes Objekt, dessen nicht-räumliche(s) Attribut(e) sich extrem von denen seiner Nachbarn unterscheiden, auch wenn sie sich nicht signifikant von der Grundgesamtheit unterscheiden, definiert werden (Shekhar, Zhang, Huang, & Vatsavai, 2003; Shekar, Lu, & Zhang, 2003). Es sind also Werte, die vom ersten geographischen Gesetz der räumlichen Autokorrelation (vgl. Kapitel 2.2) abweichen.

Oben genannte Definitionen beschreiben Ausreisser auf eine sehr allgemeine und abstrakte Weise, indem diese sich vom normalen Verhalten unterscheiden. Eine logische Folgerung wäre, „normales“ Verhalten zu definieren, und alle anderen Werte als Ausreisser zu deklarieren, aber mehrere Faktoren erschweren diese scheinbar triviale Aufgabe (Chandola, Banerjee, & Kumar, 2007):

- Der „Normalzustand“ mit allen möglichen „normalen“ Verhalten ist sehr schwierig zu definieren.
- Oft genügt eine aktuelle Definition von einem „Normalzustand“ oder „normalem“ Verhalten nicht für die Zukunft.
- Die Grenze zwischen „normal“ und „abweichend“ ist meist unscharf.
- Die exakte Definition eines Ausreissers unterscheidet sich für verschiedene Applikationen.
- Oft enthalten Messwerte ein Rauschen, welches sehr ähnlich zur Definition von Ausreissern und darum schwierig zu unterscheiden ist.

Die meisten der aktuellen Ausreisser-Detektions-Techniken vereinfachen diese Umstände, indem sie auf ein sehr spezifisch abgegrenztes Gebiet anwendbar sind und meist auch vom theoretischen Hintergrund der Disziplin gelenkt sind, wie in Abbildung 2.3 auf der nächsten Seite zu sehen ist (Chandola, Banerjee, & Kumar, 2007).

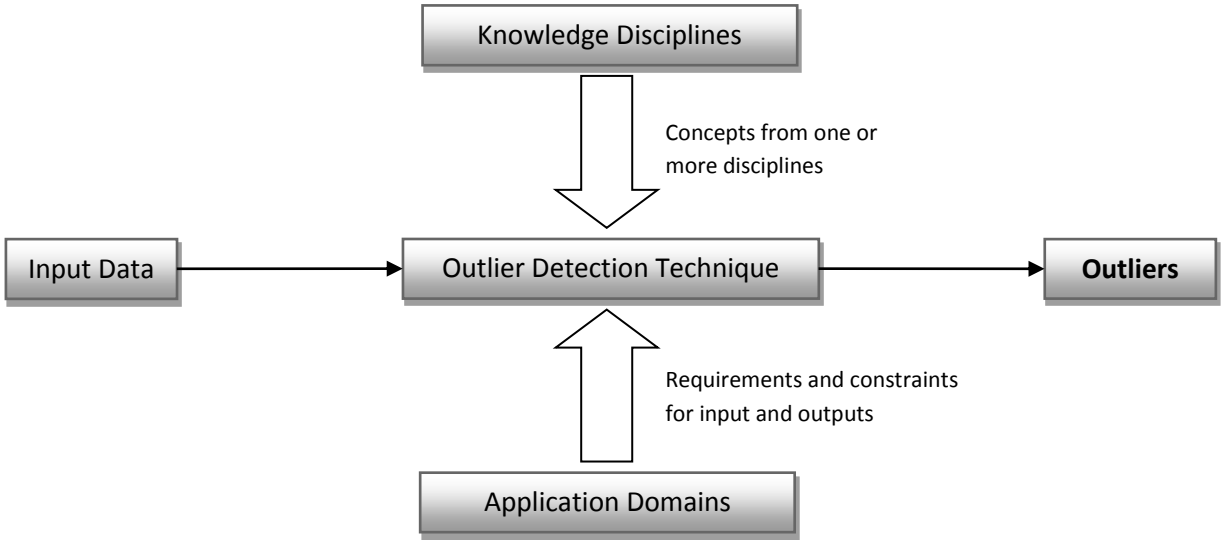


Abbildung 2.3: Generelles Design einer Ausreisser-Detektions-Technik
Eigene Abbildung in Anlehnung an Chandola, Banerjee & Kumar (2007)

Gemäss Abbildung 2.3 hängt jede Ausreisser-Detektions-Technik von folgenden drei Hauptelementen ab (Chandola, Banerjee, & Kumar, 2007):

1. Der Charakter der Daten, der Charakter der Ausreisser selber, sowie andere Eingrenzungen und Annahmen welche die Problemformulierung defnieren / eingrenzen.
2. Der Anwendungsbereich, in welchem die Technik eingesetzt werden soll. Einige Methoden werden zwar möglichst generisch entwickelt, beziehen sich jedoch fast immer auf einen eingegrenzten Anwendungsbereich.
3. Die Konzepte und Ideen von einer oder mehreren Disziplinen.

Ausreisser-Detektionen in Geosensor-Netzwerken können dabei grundsätzlich mehreren Motiven entspringen. Diese Arbeit untersucht die Ereignis-Detektion, wie in Abbildung 2.4 im mittleren Strang zu sehen ist:

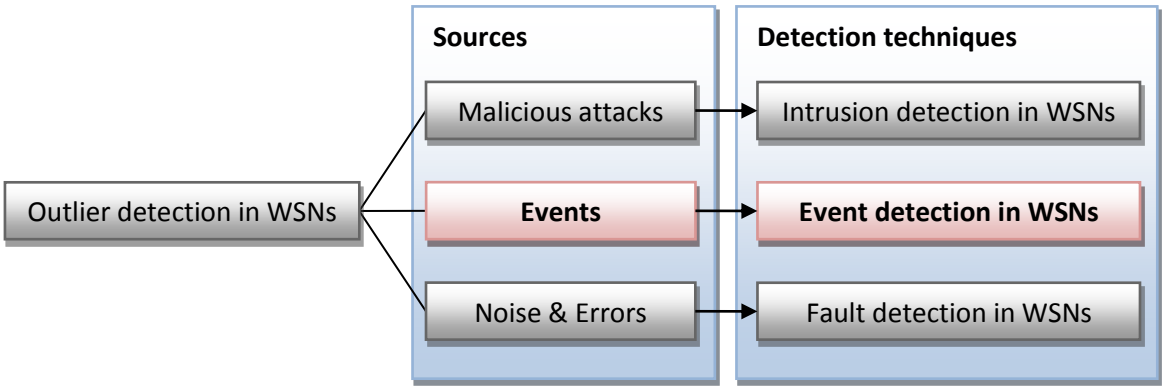


Abbildung 2.4: Drei Ausreisser-Quellen in „Wireless Sensor Networks“ und deren Detektions-Techniken
Eigene Abbildung in Anlehnung an Zhang, Meratnia & Havinga (2008)

Als Hauptkriterium, wie gut ein Ereignis-Detektions-Algorithmus performt, wird nebst dem Ressourcenverbrauch primär das Verhältnis der Detektionsrate zur Fehlalarmrate herangezogen. Normalerweise wird zur Visualisierung eine „Receiver Operating Characteristic (ROC)-Curve“ verwendet (Zhang, Meratnia, & Havinga, 2008), wie in Abbildung 2.5 auf der nächsten Seite zu sehen ist. Eine perfekte Kurve würde jedes Ereignis detektieren, ohne je einen Fehlalarm zu erzeugen. Wie

schon erwähnt, sind diese Ergebnisse jedoch immer auch in Zusammenhang mit dem Ressourcenverbrauch zu setzen. In diesem Spannungsfeld zwischen Genauigkeit der Detektion und Ressourcenverbrauch stehen alle Ereignis-Detektions-Techniken, weshalb es oft schwierig ist, solche Algorithmen adäquat zu vergleichen, da die Bedürfnisse sehr individuell vom Einsatzgebiet abhängen (Dziengel, Wittenburg, & Schiller, 2008). Sollte zum Beispiel Radioaktivität um ein Depot für Atommüll gemessen werden, kann davon ausgegangen werden, dass ein gewisser Energieverbrauch sowie eine hohe Fehlalarmrate in Kauf genommen wird, da ein nicht erkanntes Ereignis verheerende Folgen haben könnte.

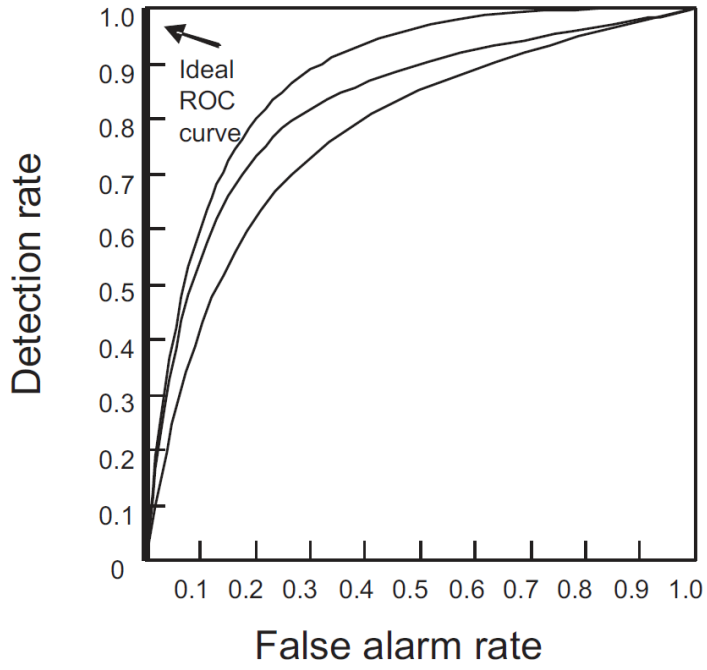


Abbildung 2.5: ROC-Kurve für verschiedene Ereignis-Detektions-Techniken
ROC-Kurve für verschiedene Ereignis-Detektions-Techniken nach Zhang, Meratnia & Havinga (2008)

Es kann schon hier festgehalten werden, dass in der Ereignis-Detektion viele verschiedene Ansätze verfolgt werden, welche von unterschiedlichen Annahmen ausgehen und/oder für unterschiedliche Zwecke Verwendung finden sollen. Wie in Kapitel 3 zu lesen ist, wird in dieser Arbeit auf Stufe Sensorknoten von einer idealen ROC-Kurve ausgegangen, da der Fokus in dieser Arbeit nicht auf der Definition und Erkennung von Ereignissen direkt liegt, sondern auf der energieeffizienten dezentralen Verarbeitung von Daten einzelner Sensorknoten, welche schon eindeutig aufgrund der Messwerte als einem Ereignis zugehörig eingestuft worden sind. So wird auch diese Arbeit von einer einfachen Definition eines Ausreissers in Form eines festen Grenzwertes eines Messwertes ausgehen (vgl. Kapitel 3 und 4). Genau hier definieren Zhang et. al. (2008) auch die Ereignis-Detektion (Event-Detection) als Spezialfall der Ausreisser-Detektion (Outlier-Detection), indem bei Ereignissen im Normalfall von einem vorhandenen Wissen über dessen Form und Ausprägung ausgegangen werden kann, während bei der Ausreisser-Detektion in der Regel kein Vorwissen darüber vorhanden sei.
Eine Einordnung dieser Arbeit zu in der Literatur beschriebenen Ansätzen zur Ereignis-Detektion wird jedoch nach hinten ins Kapitel 4.4 verschoben, nachdem in Kapitel 3 die Problemdefinition für diese Arbeit weiter eingegrenzt und in Kapitel 4.1 bis 4.3 der entwickelte dezentrale Algorithmus erläutert worden sind. Somit soll der in dieser Arbeit entwickelte dezentrale Algorithmus direkt in Zusammenhang mit in der Literatur vorgeschlagenen Lösungen gebracht und eingeordnet werden können.

3 Problemdefinition und Eingrenzung

Mit Blick auf das in Kapitel 1.2 definierte Ziel soll in diesem Kapitel die Problemdefinition konkretisiert und eingegrenzt werden. Es geht darum, die Rahmenbedingungen so zu definieren, dass der in Kapitel 4 zu entwickelnde Algorithmus sowie die in Kapitel 5 verwendete Simulation aufgrund zuvor festgelegter Kriterien erstellt werden kann. Dies ist insofern nötig, als dass speziell die Möglichkeiten und Eigenschaften der einzelnen Sensoren (Kapitel 3.1) und des Netzwerkes (Kapitel 3.2) sowie die Grundannahmen über die Umwelt (Kapitel 3.3) einen grossen Einfluss auf den Algorithmus und die Resultate in der anschliessenden Simulation haben können. Speziell werden in Kapitel 3.4 die genauen Anforderungen an den dezentralen Algorithmus und in Kapitel 3.5 die Beurteilungskriterien, an denen der Algorithmus gemessen wird, definiert. Weil es sich um eine Simulation und nicht um einen Feldversuch handelt, kann die Umwelt nur begrenzt realitätsnah abgebildet werden. Dafür besteht auf der anderen Seite die Möglichkeit, Netzwerke aufzubauen, welche vielleicht heute so technisch oder finanziell nicht umsetzbar sind, jedoch in einigen Jahren Realität sein könnten. Speziell im Fokus liegt hier beispielsweise die Skalierbarkeit von Netzwerken auf eine Grösse von mehreren tausend Sensoren. Der gesamte Quellcode des Algorithmus und der gesamten Simulation ist im Anhang zu finden (Kapitel 8), um die Nachvollziehbarkeit und Reproduzierbarkeit zu gewährleisten.

3.1 Sensorknoten

3.1.1 Normale Sensorknoten

Jeder Sensorknoten…

- …wird an einer zufälligen Position in die Umwelt gesetzt.
- …kennt seine absolute Position nicht.
- …hat eine eindeutige Identität.
- …hat einen kreisförmigen Senderadius mit Radius r.
- …ist immobil.
- …kann die Distanz d zu einem anderen Sensorknoten innerhalb des Senderadius r, jedoch nicht dessen Richtung ermitteln.
- …kann lokal aus den Messwerten ermitteln, ob er sich in einem Ereignis befindet.
- …misst exakt, ohne Rauschen und Fehler.
- …kann nicht ausfallen/zerstört werden.
- …kann jederzeit Anfragen anderer Sensorknoten empfangen.

Die meisten dieser Annahmen sind idealisiert. Jedoch soll es zum Beispiel nicht Teil dieser Arbeit sein, einen Algorithmus zu implementieren, welcher dynamisch die Topologie eines Netzwerkes nach einem Ausfall eines Knoten aktualisiert, auch wenn solche Probleme in einem Feldversuch mit einbezogen werden müssten. Die Eigenschaften der Knoten sind also so gewählt, dass das Problem der energieeffizienten Ereignis-Detektion möglichst isoliert von all den anderen Problemen untersucht werden kann, damit die Ergebnisse sich dann auch möglichst nur auf das untersuchte Thema beziehen. Einige der oben genannten Punkte sollen hier trotzdem noch kurz erklärt werden:
Eine zufällige Position der Knoten wurde aus folgender Überlegung heraus gewählt: Ein Algorithmus, welcher auf einer zufälligen Verteilung der Knoten basiert, wird auch mit einer vordefinierten Knotenverteilung umgehen können. Dass jedoch ein Algorithmus, welcher von einer definierten Verteilung ausgeht, auch bei zufällig verteilten Knoten funktioniert, muss nicht zwingend stimmen.

Insofern wurde hier versucht, eine Annahme zu treffen, die ein möglichst breites Spektrum an Verwendungszwecken abdecken kann und auch zukunftsorientiert ist.
Dass ein Knoten seine absolute Position nicht kennt, basiert auch auf mehreren Überlegungen. Zum einen greift wieder dasselbe Argument wie bei der zufälligen Positionierung: Je weniger Informationen vorausgesetzt werden, desto breiter und einfacher ist ein daraus resultierender Algorithmus einsetzbar. Des Weiteren müssten, sollte die Position bekannt sein, entweder die Knoten einzeln gesetzt werden, was der Annahme der zufälligen Position widerspricht, oder aber sie müssten mit einer Art GPS-Modul ausgestattet sein. Solche GPS-Module verbrauchen aber relativ viel Energie und werden das Gewicht, die Grösse und den Preis eines einzelnen Sensors negativ beeinflussen. Sollte der aktuelle Trend zu immer kleiner und günstigeren Sensoren anhalten, könnten Geosensor Networks theoretisch auch so engmaschig werden, dass die Genauigkeit von GPS-Daten sowieso zu gering würde.
Die Annahme, dass ein Sensorknoten innerhalb seines Senderadius r die Distanz d zu einem anderen Sensorknoten ermitteln kann, beruht darauf, dass dies heute schon mit relativ hoher Genauigkeit möglich ist (Römer, 2005). Auch ist die räumliche Autokorrelation nur dann verwendbar, wenn man irgendein Distanzmass zwischen den einzelnen Sensoren ermitteln kann. Ohne diese Annahme wäre die einzige räumliche Information, dass ein erreichbarer Knoten eine Distanz d kleiner als der Senderadius r haben muss ($d \leq r$).
Dass jeder Knoten lokal ermitteln kann, ob ein Ereignis vorhanden sei oder nicht, sollte nicht falsch verstanden werden. Wie in Kapitel 2.3 beschrieben, ist es schwierig, einen räumlichen Ausreisser und somit auch ein Ereignis zu definieren. Hier geht es vorerst auch nicht um die Grösse oder Stärke eines Ereignisses, sondern rein darum, ob der Sensor an seiner Position Werte misst, welche auf einen Ausreisser oder eben ein Ereignis schliessen lassen. Dafür müssen lokale Regeln definiert werden:

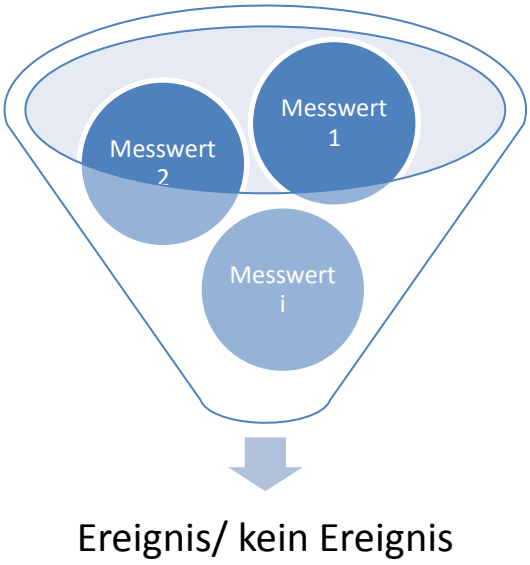


Abbildung 3.1: Ereignis-Detektion
Alle Messwerte [1, 2, …, i] verrechnet resultieren in einem eindeutigen Ergebnis Ereignis / kein Ereignis

Als Beispiel kann man sich, um nur einige wenige zu nennen, invasive Artenverbreitungen, Schadstoffausbreitungen, Waldbrände oder Ölteppiche im Meer vorstellen. All diese „Ereignisse“ könnten wohl aus einem oder mehreren Messwerten irgendwelcher Sensoren eruiert werden. So reicht wohl für die Detektion eines Waldbrandes ein einzelner Sensor, welcher die Temperatur misst, um festzustellen, ob ein gewisser Grenzwert überschritten wird, da hier von sehr hohen Differenzen

in den Messwerten ausgegangen werden kann. In anderen Anwendungen kann es je nach Definition des Ereignisses sein, dass mehrere Messwerte von verschiedenen Sensoren gebraucht werden, um dieses eindeutig zu detektieren.
Eine weitere Annahme, dass jeder Sensorknoten ohne Rauschen und Fehler misst, beruht wiederum darauf, dass in dieser Arbeit andere Themen im Vordergrund stehen, auch wenn dieses Problem in der Literatur oft angesprochen wird. Somit kann im in dieser Arbeit verwendeten Simulationsumfeld kein Fehlalarm auftreten, und darum wird es in dieser Arbeit auch keine Evaluation des Algorithmus an einer ROC-Kurve (vgl. Abbildung 2.5) geben.

3.1.2 Sink-Node

Der Sink-Node ist ein spezieller Knoten, welcher, wie in Kapitel 2.1 erläutert, als Verbindungsknoten zwischen dem Geosensor Network und der zu informierenden Stelle fungiert. Es wird angenommen, dass dieser über eine elektrische Anbindung und somit über unendliche Ressourcen verfügt. Weiter wird davon ausgegangen, dass es in einem Netzwerk genau einen Sink-Node gibt, welcher sich am Rande des Gebietes befindet. Dass der Knoten sich nicht in der Mitte befindet, hat zwei Gründe: Erstens wird es in der Realität in den meisten Fällen einfacher sein, einen solchen Sink-Node am Rand eines Gebietes zu versorgen als mittendrin. Zweitens kann so eine erhöhte Anforderung an die Algorithmen gestellt werden, da die Wege zum Sink-Node durchschnittlich grösser werden. Dies hat bezüglich Netzwerklast den gleichen Effekt, wie wenn für die Simulation ein grösseres Netzwerk verwendet würde (Kim, Seok, Choi, Choi, & Kwon, 2005).

3.2 Netzwerk

Bei den Simulationen soll von einem bereits ad-hoc initialisierten Netzwerk ausgegangen werden, da der Aufbau des Netzwerkes unter der Annahme, dass es sich um ein statisches Netzwerk handelt, nur einmal ausgeführt werden muss. Somit handelt es sich um einen konstanten Aufwand, der weiter keinen Einfluss auf die Performance des eigentlichen Algorithmus ausübt und somit vernachlässigt werden kann. Die Initialisierung des Netzwerks besteht aus folgenden drei Schritten:

1. Jeder Sensor ermittelt alle anderen Sensoren innerhalb seines Senderadius r
2. Das Netzwerk in Form eines ungerichteten Relative Neighborhood Graph (RNG) oder Gabriel Graph (GG) wird erstellt.
3. Vom Sink-Node aus wird ein Shortest-Path-Algorithmus ausgelöst. Dies führt dazu, dass jeder Knoten im Netzwerk den Knoten kennt, an welchen er Informationen per Multi-Hop in Richtung Sink-Node senden kann, wobei die Distanz über sämtliche Hops bis zum Sink-Node minimal ist.

Der erste Schritt, dass jeder Sensor all seine Sensoren innerhalb seines Senderadius r ermittelt ist relativ naheliegend, da dies jene Sensoren sind, mit welchen er potentiell kommunizieren kann. Würden all diese ermittelten Knoten miteinander verbunden, entstünde ein Unit Disk Graph (UDG) mit dem Radius r. Dieser ist dadurch definiert, dass ein Knoten sich mit jedem anderen Knoten innerhalb eines bestimmten Radius verbindet.
Der zweite Schritt befasst sich mit dem Aufbau des Netzwerkes. Aufgrund der Ergebnisse von Sadeq und Duckham (2008) sowie Laube und Duckham (2009) scheint es sinnvoll zu sein, den Gabriel Graph (GG) oder den Relative Neighborhood Graph (RNG) für die Untersuchungen zu verwenden. Für beide Graphen gibt es gemäss Sadeq und Duckham (2008) die Möglichkeit lokaler dezentraler Initialisierung. In dieser Arbeit wurde nur für den RNG ein funktionierender dezentraler Algorithmus implementiert (vgl. Kapitel 8.1.6). Der GG und der RNG sind wie folgt definiert:

Im GG werden zwei Nodes (A,B) mit einer Edge verbunden, solange sich innerhalb des Kreises mit dem Durchmesser AB zwischen den beiden Knoten keine weiteren Knoten befinden:

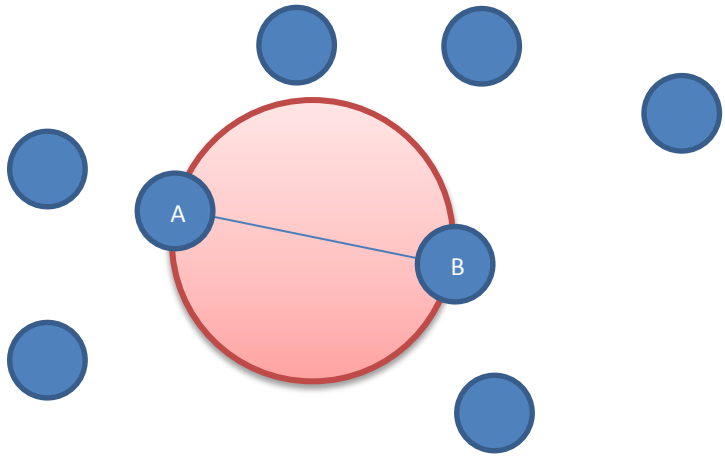


Abbildung 3.2: Gabriel Graph
Knoten A und B werden miteinander verbunden, da sich im Kreis mit dem Durchmesser AB (roter Kreis) keine weiteren Knoten befinden (GG).

Im RNG werden zwei Knoten (A, B) mit einer Edge verbunden, solange sich innerhalb der Schnittmenge der zwei Kreise um die beiden Knoten A und B mit dem Radius AB keine weiteren Knoten befinden. Der RNG ist somit ein Subgraph des GG:

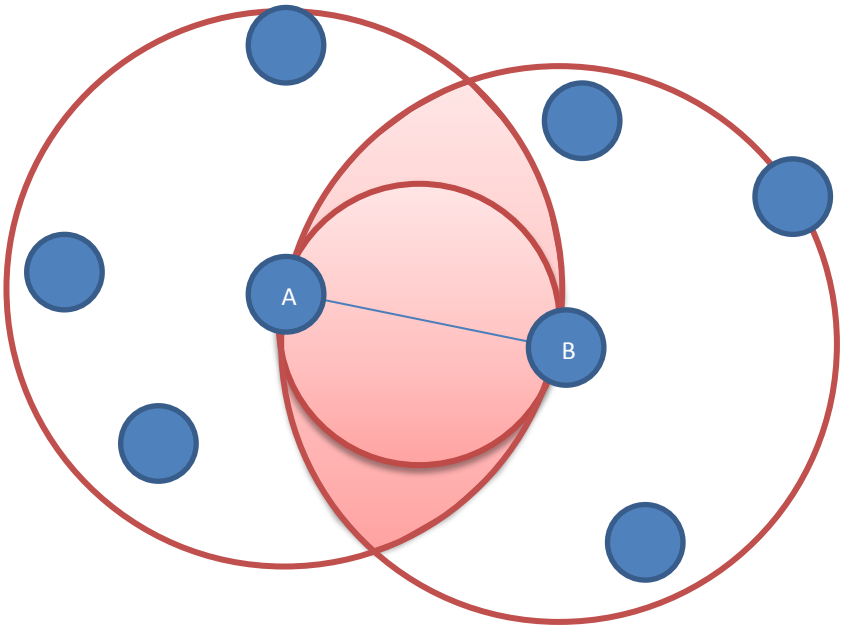


Abbildung 3.3: Relative Neigborhood Graph
Knoten A und B werden miteinander verbunden, da sich in der Schnittmenge der zwei Kreise um Knoten A und B mit Radius AB (roter Bereich) keine weiteren Knoten befinden (RNG). Der innere rote Kreis zeigt das Kriterium des GG, die beiden roten Spitzen den erweiterten Bereich des RNG zum GG.

Bei beiden, dem GG wie auch dem RNG, entsteht ein planarer Graph, was bei einem UDG nicht der Fall wäre.

Der dritte Schritt befasst sich mit dem Erstellen des Pfades für die Übermittlung von Informationen von einem Knoten X zum Sink-Node. Wie schon beschrieben, führt dieser Schritt dazu, dass jeder Knoten weiss, zu welchem anderen Knoten er Informationen senden muss, damit diese auf kürzestem Weg zum Sink-Node gelangen. Hier war das Kriterium nicht, möglichst wenige Hops zu verwenden, sondern die aufsummierte Distanz vom Knoten X zum Sink-Node zu minimieren, wobei der Unterschied zwischen diesen Varianten relativ gering ausfallen dürfte (Akyildiz, Su, Sankarasubramaniam, & Cayirci, 2002). Hierfür wurde eine eigene vereinfachte Variante eines Shortest-Path-Algorithmus in Anlehnung an den Dijkstra-Algorithmus (Dijkstra, 1959) gewählt. Welcher zugrundeliegende Graph im Netzwerk verwendet wird, ist nicht von Bedeutung, da jeder Sensor innerhalb des Senderadius r in diese Berechnung einbezogen wird, was dem UDG mit Radius r entspricht. Hierfür wurden weder der GG, noch der RNG verwendet, weil beide Untergraphen des UDG darstellen und somit das Ergebnis nur gleich oder schlechter ausfallen könnte.
Der zum Sink-Node führende Pfad wird als statisch angenommen. Dies heisst konkret, es werden keine dynamischen Routen berechnet, um die Netzwerklast auszugleichen. Dies ist vielleicht nicht mehr mit der heutigen Realität vergleichbar, jedoch gibt es einige gute Gründe, welche für eine Verwendung eines statischen Pfades sprechen. Einen dynamischen Pfad zu finden heisst grundsätzlich, dass Informationen (z.B. Energiestand) zwischen den Knoten ausgetauscht werden müssten, was einem dezentralen Informationsaustausch entspricht. Man müsste sich dann fragen, inwiefern dann ein zentraler Algorithmus noch zentral ist, wenn der Weg zum Sink-Node auf dezentralem Austausch von Informationen basiert. Als weiteres Argument kann die Skalierbarkeit eines Geosensor Networks genannt werden, denn ab einer bestimmten Grösse eines Geosensor Networks stösst auch eine dynamische Pfadführung zum Sink-Node hin an ihre Grenzen, da die Kreisfläche um den Sink-Node mit Senderadius r auf die Fläche von $A = \pi r^2$beschränkt ist, und diese Kreisfläche alle Informationen durchlaufen müssen. Als Veranschaulichung kann man sich zum Beispiel eine Stadt vorstellen: Wenn immer mehr Autos zur Stosszeit in die Stadt wollen, könnten diese durch intelligente Verkehrsleitsysteme zwar auf alle Strassen verteilt werden. Doch vergrössert man das Pendlereinzugsgebiet immer weiter, führt dies nur dazu, dass alle Strassen erst zu einem späteren Zeitpunkt überlastet werden. Wird nur eine Strasse benutzt, kann man denselben Stau beobachten, jedoch schon ab einem kleineren totalen Verkehrsaufkommen, was einem kleineren Einzugsgebiet entspricht. Dieser Umstand wird auch in die Grundüberlegungen zu den Zielen des dezentralen Algorithmus in Kapitel 4.1.1 einbezogen.

3.3 Umwelt

Die Umwelt möglichst umfassend abzubilden ist nicht das Ziel dieser Arbeit. Vielmehr soll eine Annäherung an eine Umwelt gewählt werden, welche die relevanten Aspekte adäquat abbildet. Im Falle eines Geosensor Network, welches zum Ziel hat, Ereignisse in der Umwelt zu detektieren, sollten somit die messbaren Parameter in der Umwelt vorhanden sein.
Als stellvertretendes Beispiel für viele verschiedene Verwendungszwecke wird der Waldbrand als Modell gewählt, in dem das Geosensor Network die Brände (=Ereignis) detektieren soll. Die Waldbrandsimulation wird anhand eines zweidimensionalen zellulären Automaten in Anlehnung an Quartieri et. al. (2010) erstellt. Zelluläre Automaten haben den Nachteil, dass ihr Verhalten so kompliziert ist, dass analytische Voraussagen kaum möglich sind (Czárán & Bartha, 1992), was in Simulationen, welche möglichst unabhängig und zufällig sein sollen jedoch auch als Vorteil gewertet werden kann. Hier wird erklärt, wie solche Waldbrände im Grundsatz simuliert werden. Der genaue Quellcode kann wiederum im Anhang (Kapitel 8) nachgelesen werden. Es gibt verschiedene Parameter, welche die Umwelt beeinflussen:

Parameter	Beschreibung
xDim	Breite des Untersuchungsgebietes (m)
yDim	Höhe des Untersuchungsgebietes (m)
eventSparkProbability	Wahrscheinlichkeit eines neuen Ereignisses pro Step (vgl. Kapitel 3.5)
eventSpreadingProbability	Grundausbreitungswahrscheinlichkeit des Ereignisses (in alle Richtungen gemäss Von-Neumann-Nachbarschaft)
windStrength	Windstärkekorrekturfaktor
initialEventValue	Initialtemperatur (°C)
maxEventValue	Maximaltemperatur (°C)
reduceEventValue	Temperaturrückgang pro Step nach dem Ereignis
initialFuel	Initialbrennstoff (t)
consumeFuel	Brennstoffverbrauch pro Step (t)
fuelGrow	Brennstoffwachstum pro Step (t)
maxFuel	Maximalbrennstoff (t/m^2)

Tabelle 3.1: Umweltparameter
Mit diesen Parametern kann die Umwelt in den Simulationen beeinflusst werden. In Kapitel 8 kann anhand des Quellcodes der Einfluss der einzelnen Parameter beurteilt werden.

Während der Simulation treten an zufälligen Orten innerhalb des Untersuchungsgebietes neue Feuer mit der Wahrscheinlichkeit „eventSparkProbability“ auf. Geht man von einem Blitzschlag als Ursache für ein Feuer aus, so kann dieser Wert somit als Wahrscheinlichkeit eines Blitzschlages an einem zufälligen Ort im Untersuchungsgebiet pro Step betrachtet werden, welcher ein Feuer auslöst. Diese Feuer breiten sich mit der Wahrscheinlichkeit „eventSpreadingProbability“ plus Korrekturfaktoren in die verschiedenen Himmelsrichtungen (Von-Neumann-Nachbarschaft) aus:

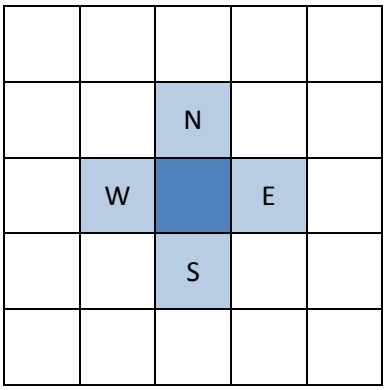


Abbildung 3.4: Die Von-Neumann-Nachbarschaft
Die Von-Neumann-Nachbarschaft umfasst alle Flächen (hellblau), welche mit der Basisfläche (dunkelblau) mindestens eine Kante gemeinsam haben (Quartieri, Mastorakis, Iannone, & Guarnaccia, 2010).

Die Korrekturfaktoren bestehen aus der Windrichtung, welche zufällig ist, sowie dem Windstärkekorrekturfaktor „windStrength“ und beeinflussen die Wahrscheinlichkeit der Ausbreitung richtungsabhängig. Die Windrichtung wird einen Zufallswert zwischen 0 und 2π erhalten, wobei 0 einem Ostwind, $\frac{\pi}{2}$ einem Nordwind, π einem Westwind, $\frac{3\pi}{2}$ einem Südwind entspricht, wobei die Umwelt als nach Norden ausgerichtet betrachtet werden kann. Die Windrichtung wird absichtlich als zufällig angenommen, da so eine gewisse Variabilität der simulierten Umwelt gewährleistet werden kann.
Berechnet werden die Ausbreitungswahrscheinlichkeiten in die vier Himmelsrichtungen anhand folgender Formeln, wobei sp = Grundausbreitungswahrscheinlichkeit, ws = Windstärke und wd = Windrichtung:

Ausbreitungsrichtung	Formel
Norden	$sp - ws\,(\sin wd)$
Osten	$sp - ws(\cos wd)$
Süden	$sp + ws\,(\sin wd)$
Westen	$sp + ws(\cos wd)$

Tabelle 3.2: Berechnungen der Ausbreitungswahrscheinlichkeit in die vier Himmelsrichtungen

Weiter verbrennt ein Feuer das sich auf der Zelle befindende Brennmaterial „initialFuel“ mit der Geschwindigkeit „consumeFuel“ pro Step. Der Brennstoff wächst nach dem Feuer mit der Menge „fuelGrow“ pro Step wieder nach, bis der Maximalwert „maxFuel“ erreicht ist.
Der in der Simulation zu messende Faktor ist die Temperatur. Ein Feuer verändert die Temperatur in seiner Umgebung, bis die maximale Temperatur „maxEventValue“ erreicht ist. Auch die Temperatur fällt nach einem Feuer wieder mit der Geschwindigkeit „reduceEventValue“ bis auf den Wert „initialEventValue“ zurück. Ein Ereignis kann folgendermassen aussehen:

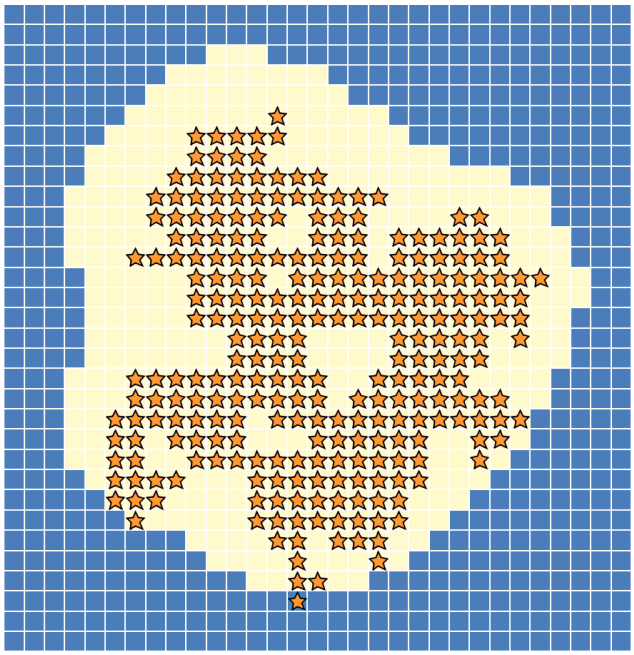


Abbildung 3.5: Beispiel eines simulierten Feuers
Die orangen „Sterne“ repräsentieren die Feuer. Die Umgebung bildet ein binärer Layer, wobei blau eine Temperatur unterhalb und gelb eine Temperatur oberhalb des Ereignis-Grenzwertes anzeigt. Dass die Temperaturgrenze im unteren Bereich innerhalb des Feuerperimeters ist, lässt auf eine Ausbreitung des Feuers von Norden nach Süden schliessen.

Dass die Temperatur nicht graduell sondern in einem binären Layer dargestellt wird, wurde aufgrund der in Abbildung 3.1 erklärten Anforderung der Knoten abgeleitet, welche aus den Messwerten zu einer Aussage Ereignis/kein Ereignis kommen müssen, welche selber auch nicht graduell ist. Somit stellt die gelbe Fläche die Fläche des Ereignisses dar, wie sie von den Sensorknoten wahrgenommen wird. Dafür muss ein einheitlicher Grenzwert für den Ereignis-Wert (hier Temperatur) festgelegt werden.

3.4 Anforderungen an den dezentralen Algorithmus

Damit in Kapitel 4 ein dezentraler Algorithmus entwickelt werden kann, muss definiert werden, welchen Zweck er erfüllen soll. Grundsätzlich soll der Algorithmus Ereignisse detektieren können. Unter der Annahme, dass jeder Sensorknoten lokal ermitteln kann, ob er ein Ereignis detektiert oder nicht (vgl. Kapitel 3.1), geht es also primär noch darum, diese Information intelligent dezentral zu verarbeiten. Von grossem Interesse dürfte es sein, …

1. …wie viele Ereignisse aktuell detektiert werden.
2. …wie gross die Flächen einzelner Ereignisse sind.

So kann man sich als Beispiele für interessante Ereignisse wiederum invasive Artenverbreitungen, Schadstoffausbreitungen, Ölteppiche im Meer oder eben die in dieser Arbeit verwendeten Waldbrände vorstellen. Bei allen Fällen sind wohl die Dimensionen sowie die Anzahl „Herde“ von Interesse. Je nach Art des zu detektierenden Ereignisses könnten auch noch zusätzliche Masse wie zum Beispiel die Form eines Ereignisses oder die genauen Messwerte wie Temperatur, Schadstoffbelastung usw. von Interesse sein, jedoch soll der zu entwickelnde Algorithmus als ein relativ einfacher Grundlagenalgorithmus zu werten sein, welcher für spezifische Anwendungen noch erweitert werden könnte.

Eine Anforderung an die Genauigkeit soll hier nicht fix vordefiniert werden. Vielmehr soll diese je nach Anwendungsbereich und Anforderung an den spezifisch zu detektierenden Event anpassbar sein. So kann man sich vorstellen, dass zum Beispiel bei einer Ölkatastrophe für die Einsatzleitung von Interesse ist, wie viele Ölteppiche mit welchem Ausmass sich auf dem Meer befinden. Es ist jedoch wahrscheinlich von geringem Interesse, wenn sich die Grösse eines Teppichs von 1000m^2 auf 1010m^2 erhöht, da dies für die einzuleitenden Massnahmen kaum relevant sein wird. Jedoch wird eine Verdoppelung der Fläche interessieren, da dies bedeuten könnte, dass ein weiteres Schiff zum Abpumpen des Öls in den Einsatz entsendet werden muss. Aus diesem Grund sollte der Algorithmus diese Flexibilität zulassen, damit das Verhältnis von Aufwand und Genauigkeit individuell an die Bedürfnisse angepasst werden kann.

Alle relevanten Informationen müssen schlussendlich zum Sink-Node übermittelt werden. Von dort aus können die Daten weitergeleitet und verarbeitet werden, was aber nicht mehr Inhalt dieser Arbeit ist.

Ein Geosensor Network kann wohl als partiell synchrones System angenommen werden (Lynch, 1996). Das heisst, einzelne Sensorknoten können zueinander nicht als komplett synchron betrachtet werden. Somit muss der Algorithmus auch funktionieren, egal in welcher Reihenfolge und Zeitverschiebung die einzelnen Knoten verschiedene Prozesse ausführen.

Wie in Kapitel 2.3 zu lesen ist, wird diese Arbeit die Probleme eines Fehlalarmes durch Annahme einer perfekten Ereignis-Definition und einer fehlerlosen Messung ausklammern, damit in der Evaluation ganz auf das Thema des Energieverbrauchs fokussiert werden kann. Anders als bei den meisten Detektions-Algorithmen, bei denen es um die Analyse von Patterns geht, damit entschieden werden kann, ob es sich um ein Ereignis handelt oder nicht, wird in dieser Arbeit davon ausgegangen, dass die Ereignis-Definition schon vollständig bekannt ist, da sich die Definition nicht durch eine gewisse Form des Ereignisses auszeichnet, sondern durch festgelegte Grenzwerte von Messwerten (vgl. Kapitel 3.1). Es wird also mit einer idealen ROC-Kurve gearbeitet und dies auf der Ebene der einzelnen Knoten. Dies wird aus dem Grund so angenommen, da diese Arbeit nicht untersuchen möchte, wie gut Ereignisse detektiert werden, sondern, wenn Ereignisse auf Stufe Sensorknoten detektiert worden sind, diese Informationen möglichst energieeffizient verarbeitet und dem User zur Verfügung gestellt werden können.

3.5 Beurteilungskriterien

Der in Kapitel 4.1 zu entwickelnde dezentrale Algorithmus soll mit einer zentralen Variante, welche in Kapitel 4.2 vorgestellt wird, in mehreren Simulationen verglichen werden. Mehrere Simulationen heisst, dass verschiedene Parameter variiert werden, um ein möglichst umfassendes Bild über das Verhalten der beiden Algorithmen zu erhalten. Diese Parameter sind:

1. Genauigkeit der Informationen
2. Grösse des Untersuchungsgebiets und des Netzwerkes
3. Sensordichte
4. Verwendeter Graph (RNG/GG)
5. Ereignisgrösse / räumliche Korrelation

Hierbei werden in Bezug auf die in Kapitel 1.3 spezifisch gestellten Forschungsfragen vor allem Punkt zwei und vier direkt Antworten liefern können.

Die Simulation selber wird in Steps unterteilt, wobei ein Step einer Zeiteinheit t entspricht. Diese Zeiteinheit ist relativ und könnte in der Realität einer beliebigen Zeitdauer entsprechen. Ein Step wird in zwei Sub-Steps unterteilt: Im ersten Sub-Step soll die gesamte Umgebung aktualisiert sowie ein Abgleich des gesamten Netzwerkes vollzogen werden. Dies heisst, alle Berechnungen werden getätigt und Daten weitergeleitet. Im zweiten Sub-Step sollen die Sensoren Messungen vornehmen. Diese zwei Steps werden so aufgeteilt, damit die Reihenfolge der Berechnungen der einzelnen Knoten keinen Einfluss auf das anschliessende Resultat hat. Dies kann bei einer Simulation geschehen, da im Gegensatz zur Realität die Sensorknoten ihre Berechnungen und Messungen streng sequentiell durchführen, was einer partiell synchronen Verarbeitung widerspricht (vgl. Kapitel 3.4).

Als Vergleichskriterium für die Effizienz wird die Anzahl gesendeter Mitteilungen der einzelnen Knoten herangezogen. Wie in Kapitel 1.2 beschrieben, verbraucht das Senden über Funk ein Vielfaches an Energie verglichen mit der lokalen Verarbeitung von Daten, weshalb in dieser Arbeit auch nur die Netzwerkaktivität als Energieverbraucher berücksichtigt wird, wie dies auch in anderen Arbeiten schon gemacht wurde (Deligiannakis, Kotidis, Stoumpos, & Delis, 2009). Es wird auf eine Bit-weise Abrechnung verzichtet, da schon eine Transformation von einer Gleitkommazahl (z.B. Double) zu einer diskreten Zahl (z.B. Integer) ein vollkommen anderes Bild ergeben könnte. Zudem müssten in diesem Fall das Übermittlungsprotokoll sowie mögliche Datenkomprimierungsalgorithmen und viele andere Faktoren mit einbezogen werden, was nicht mehr dem Ziel dieser Arbeit entspricht. Der Einfachheit halber wird definiert, dass das Senden von Anfragen sowie das Senden von Daten in Form von primitiven Datentypen als jeweils eine Mitteilung zählen. Die Anzahl versendeter Mitteilungen eines Knoten wird als proportional zum Energieverbrauch und somit umgekehrt proportional zur Lebensdauer eines Knotens angenommen. Weiter wird davon ausgegangen, dass jeder Sensor beim Senden von Informationen immer gleich viel Energie verbraucht, und zwar unabhängig von der Distanz zwischen den beiden Knoten. Würde der Sensorknoten die Stärke seines Signals der Distanz zum Empfängerknoten anpassen, so müssten auch zusätzliche signalhemmende Faktoren wie die Topographie oder die Vegetationsdichte mit einberechnet werden. Die Verteilung des Energieverbrauchs soll im Gegensatz zu vielen anderen Publikationen, welche sich auf den Gesamtenergieverbrauch beschränken (Dziengel, Wittenburg, & Schiller, 2008; Deligiannakis, Kotidis, Stoumpos, & Delis, 2009) umfassend unter verschiedenen Gesichtspunkten wie zum Beispiel „Load Balance“, Lebensdauer, Gesamtverbrauch etc. beurteilt und diskutiert werden.

4 Der Algorithmus

In diesem Kapitel geht es um die Entwicklung des dezentralen Algorithmus (Kapitel 4.1) sowie des zentralen Vergleichsalgorithmus (Kapitel 4.2). Es werden die Funktionsweisen erklärt, nicht jedoch den Quelltext dazu angegeben. Dieser ist in Anhang (Kapitel 8) zu finden. In Kapitel 4.3 wird noch auf die Unterschiede bezüglich der Informationsqualität der beiden Algorithmen eingegangen. Kapitel 4.4 befasst sich mit der Einordnung des dezentralen Algorithmus im Bezug zu bestehender Literatur.

4.1 Dezentraler Algorithmus

Beim Algorithmus geht es um die Ereignis-Detektion. Wie bereits definiert ist jeder Knoten selbstständig in der Lage zu identifizieren, ob es sich bei seinen Messwerten um ein Ereignis handelt oder nicht, womit die dezentrale Aufgabe des Algorithmus sich darauf beschränkt, auf intelligente Weise Informationen zu sammeln und diese aggregiert an den Sink-Node zu senden. Dieses Kapitel ist in vier Unterkapitel aufgeteilt: In Kapitel 4.1.1 werden Grundüberlegungen bezüglich des Ziels des dezentralen Algorithmus gemacht, dessen Funktionsweise in Kapitel 4.1.2 vorgestellt wird. Kapitel 4.1.3 befasst sich mit Ideen, die während der Entwicklung der Methode versucht wurden, jedoch schlussendlich keine Verwendung fanden. In Kapitel 4.1.4 werden mögliche Verbesserungen am Algorithmus vorgeschlagen, welche jedoch im Rahmen dieser Arbeit nicht implementiert wurden.

4.1.1 Grundüberlegungen zu den Zielen eines dezentralen Algorithmus

Die Entwicklung des dezentralen Algorithmus basiert auf den Vorgaben und Restriktionen des Kapitels 3, insbesondere an den gestellten Anforderungen, welche in Kapitel 3.4 beschrieben wurden. Das Ziel, an dem der Algorithmus schlussendlich auch gemessen wird, ist es, den Energieverbrauch einzelner Knoten durch die Dezentralisierung der Ereignis-Detektion zu minimieren. Wenn bei einer zentralen Ereignis-Detektion alle Informationen von jedem Knoten zum Sink-Node per Multi-Hop gesendet werden, wird schnell einmal klar, dass alle Informationen durch die wenigen Knoten müssen, welche direkt mit dem Sink-Node verbunden sind (vgl. Kapitel 3.2). Insofern liegt die Vermutung nahe, dass es sich bei diesen Knoten um solche handelt, die relativ viele Informationen weiterleiten müssen und somit einen hohen Energieverbrauch und eine kurze Lebensdauer aufweisen:

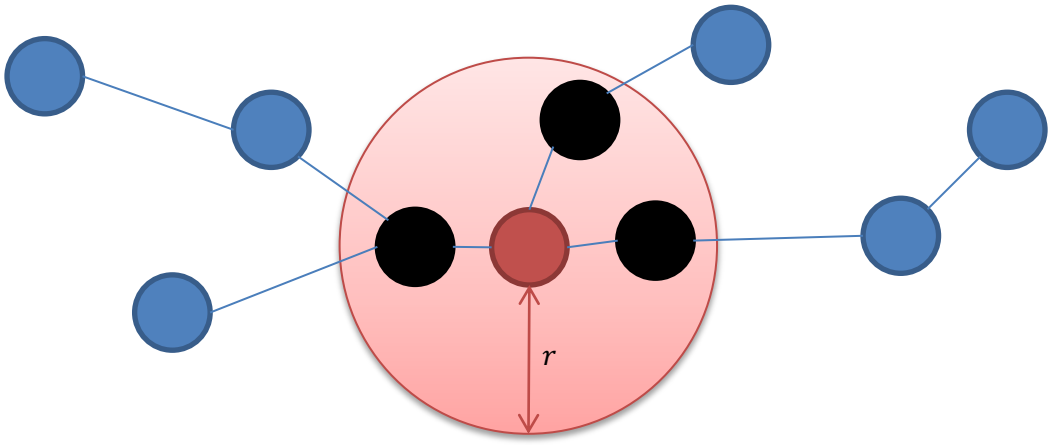


Abbildung 4.1: Kritische Knoten in einem Geosensor Network
Abgebildet sind sechs „normale“ Sensorknoten (blau), ein Sink-Node (rot) und drei kritische Knoten (schwarz) sowie der maximale Senderadius r eines Knotens

Die schwarzen Knoten sind kritisch für die Lebensdauer des gesamten Geosensor Network, denn fallen diese aus, ist es nicht mehr möglich Informationen an den Sink-Node zu senden. Das Ziel des dezentralen Algorithmus ist es, Informationen möglichst dezentral zusammenzutragen, um diese dann in kompakter Weise über die langen Sendewege des Shortest-Path (vgl. Kapitel 3.2) an den Sink-Node zu senden. Damit soll erreicht werden, dass der Energieverbrauch auch dezentralisiert wird und insbesondere die kritischen Knoten (schwarz) nahe dem Sink-Node massiv entlastet werden können.

4.1.2 Funktionsweise des Algorithmus

In diesem Kapitel wird die Grundfunktionsweise des entwickelten dezentralen Algorithmus vorgestellt. Das Kapitel wird dabei in mehrere Unterkapitel gegliedert: Kapitel 4.1.2.1 informiert über den Grundaufbau, Kapitel 4.1.2.2 bis Kapitel 4.1.2.4 befassen sich mit dem Aufbau eines neuen Graphen innerhalb eines Ereignisses, innerhalb dessen die Informationen entlang gesendet werden (Kapitel 4.1.2.5).

4.1.2.1 Grundaufbau

Der Algorithmus basiert darauf, dass sich die Knoten in verschiedenen Zuständen befinden können. Es macht Sinn, mit verschiedenen Zuständen zu arbeiten, da so das Verhalten des Knotens je nach Zustand angepasst werden kann:
Zustand 0: Normalzustand → Der Knoten ist in keinem Ereignis.
Zustand 1: Ereigniszustand → Der Knoten befindet sich in einem Ereignis.
Zustand 2: Ereigniszustand Head → Der Knoten befindet sich in einem Ereignis und ist ein „Head-Node“.
Zustand 0 ist wie beschrieben der Normalzustand. Das heisst, dass der Sensorknoten A sich aktuell in keinem Ereignis befindet, keine Netzwerkaktivitäten ausführt und in bestimmten Zeitintervallen Δt erneut untersucht, ob ein Ereignis gemessen wird. Auf Zustand 1 und 2 wird im nächsten Unterkapitel näher eingegangen. Es handelt sich beim vorgestellten Algorithmus gemäss Laube und Duckham (2009) um eine „local absorption“-Strategie, die per multi-hop Informationen aus der benachbarten Umgebung sammelt, um räumliche Informationen zu gewinnen.

4.1.2.2 Detektieren eines Ereignisses

Werden von einem Sensorknoten Werte gemessen, welche auf ein Ereignis schliessen lassen, kontaktiert dieser seine durch den Graphen direkt verbundenen Nachbarknoten, und informiert sich, ob ein solcher auch schon ein Ereignis gemessen hat. Findet er einen Knoten B, welcher sich in einem Ereignis befindet, verbindet sich Knoten A mit Knoten B und geht in Zustand 1 über. Findet Knoten A keinen Nachbarknoten, welcher schon in Zustand 1 oder 2 ist, wird dieser selber zu einem Head-Node und nimmt Zustand 2 an.
Ein Head-Node, also ein Knoten, welcher sich in Zustand 2 befindet, ist verantwortlich dafür, Informationen via shortest-Path (vgl. Kapitel 3.2) an den Sink-Node zu senden. Er ist zudem zuständig für die Filterung der Daten, und entscheidet anhand der geforderten Informationsgenauigkeit, ob Informationen an den Sink-Node weitergeleitet werden oder nicht. Dadurch, dass jeder Knoten, der sich neu im Ereignis befindet, sich mit einem Knoten in Zustand 1 oder 2 verbindet, hat dieser immer per Multi-Hop eine Verbindung zum Head-Node. Welche Informationen auf diesen Pfaden zum Head-Node gesendet werden, wird in Kapitel 4.1.2.5 erklärt. Anhand eines Beispiels soll oben beschriebener Sachverhalt verständlich gemacht werden:

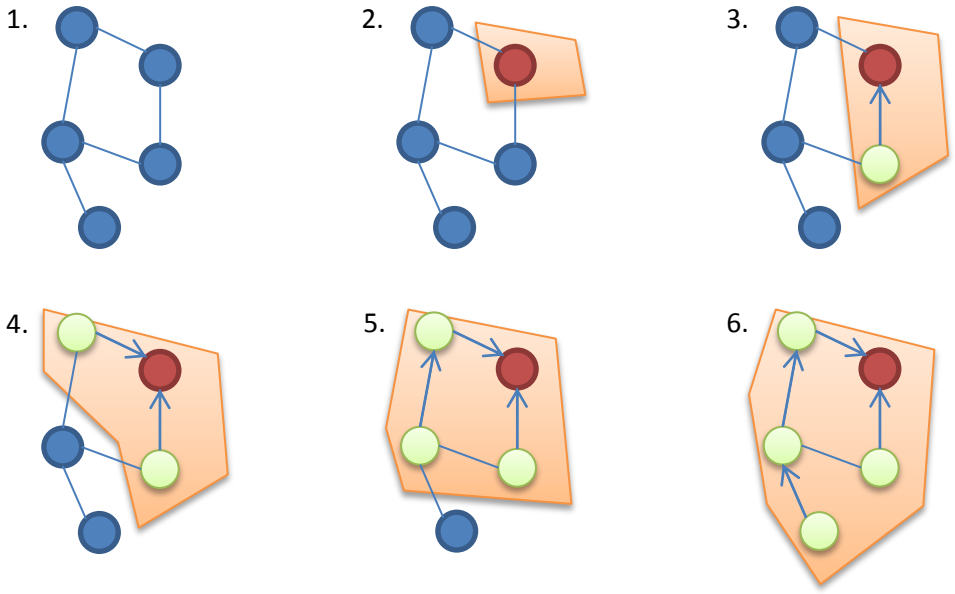


Abbildung 4.2: Auftauchen eines Ereignisses
Blaue Knoten = Zustand 0; grüne Knoten = Zustand 1; rote Knoten = Zustand 2; blaue Linien = Kanten des Graphen; blaue Pfeile = primäre Informationsflussrichtung zwischen den Knoten; orange Fläche = Ereignis/Feuer

In Abbildung 4.2 wird das Auftauchen eines Ereignisses in sechs Schritten exemplarisch gezeigt. Schritt 1 zeigt die Ausgangssituation, in welcher sich alle Knoten im Zustand 0 befinden. Taucht ein Ereignis auf (Schritt 2), sucht der Knoten, der dieses Ereignis detektiert, einen Nachbarknoten, welcher auch schon in diesem Ereignis ist. Da es in diesem Fall keinen solchen gibt, geht dieser Knoten in Zustand 2 über und wird zu einem Head-Node (roter Knoten). Im Schritt 3 breitet sich das Feuer aus, so dass es von einem zweiten Knoten erkannt wird. Dieser sucht wiederum bei seinen Nachbarknoten, ob das Ereignis schon detektiert wurde, und wird fündig. Somit verbindet sich der neue Knoten mit dem Head-Node und geht in Zustand 1 über. Dasselbe passiert in Schritt 4, 5 und 6 mit der Ausnahme, dass in Schritt 5 und 6 die Knoten nicht mehr direkt mit dem Head-Node verbunden sind, sondern mit einem anderen Knoten, der sich auch in Zustand 1 befindet. So können aber immer Informationen per Multi-Hop von jedem sich im Ereignis befindenden Knoten an den Head-Node gesendet werden. Es entsteht also ein eigenes kleines Subnetz innerhalb eines Ereignisses mit gerichteten Graphen, welche schlussendlich alle immer zum Head-Node zeigen.
Eine Frage, welche nicht beantwortet wurde, ist, wie beim Schritt 5 der neu das Ereignis detektierende Knoten entscheidet, zu welchem der beiden sich schon in Zustand 1 befindlichen Knoten er sich verbindet. Im hier verwendeten Algorithmus wird dies dem Zufall überlassen. Ob dies die beste Variante ist oder ob hier noch gewisses Potential zur Optimierung vorhanden ist, wird in Kapitel 4.1.3.1 diskutiert.

4.1.2.3 Verschmelzung zweier Ereignisse

Der Algorithmus muss neben dem „normalen“ Auftauchen eines Ereignisses, wie es in Kapitel 4.1.2.2 beschrieben ist, noch weitere Szenarien meistern können. Eine davon ist die Verschmelzung von zwei Ereignissen. Eine solche Verschmelzung kann grundsätzlich zwei Gründe haben:

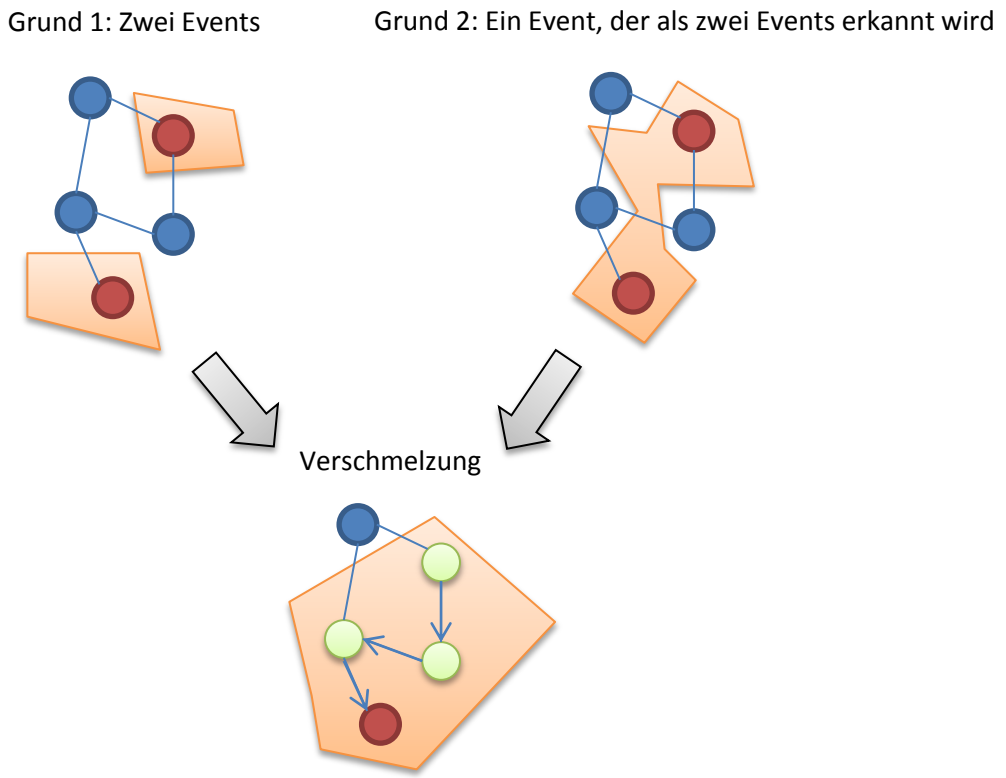


Abbildung 4.3: Gründe für die Verschmelzung zweier Ereignisse
Blaue Knoten = Zustand 0; grüne Knoten = Zustand 1; rote Knoten = Zustand 2; blaue Linien = Kanten des Graphen; blaue Pfeile = primäre Informationsflussrichtung zwischen den Knoten; orange Fläche = Ereignis/Feuer

Beim Grund 1 handelt es sich um zwei einzelne Ereignisse, welche sich ausbreiten, und dadurch zu einem Ereignis verschmelzen. Grund 2 muss mehr von der technischen Seite her betrachtet werden: Breitet sich ein Ereignis so aus, dass zwei Knoten, welche nicht über den Graphen miteinander verbunden sind, das selbe Ereignis entdecken, so wird dieses eine Ereignis als zwei unabhängige Ereignisse behandelt, da es für das Geosensor Network bei gegebener Sensordichte nicht möglich ist zwischen den beiden abgebildeten Situationen (Grund 1 und Grund 2) zu unterscheiden. Jedoch führen beide Ursachen dazu, dass bei einer weiteren Ausbreitung des/der Ereignis die beiden Ereignisse zu einem verschmelzen. Da es sich danach nur noch um ein grosses Ereignis/Waldbrand handelt, braucht es auch nur noch einen Head-Node, der die Informationen innerhalb eines Ereignisses sammelt. Doch wie wird erkannt, ob zwei Ereignisse verschmelzen?
Detektiert ein Knoten neu ein Ereignis, so sucht er wie in Kapitel 4.1.2.2 beschrieben Nachbarknoten, welche sich auch schon in einem Ereignis befinden. Dieser hört jedoch, hat er einen solchen Knoten gefunden (Zustand 1 oder Zustand 2), nicht mit dem Suchen auf, sondern schaut weiter, ob noch ein zweiter Nachbarsknoten nicht in Zustand 0 ist. Findet er weitere Knoten, so wird die Richtung des Pfades in einem Ereignis so gedreht, dass jeder Knoten in Richtung des neuen Head-Node zeigt. Zwei Dinge müssen jedoch noch berücksichtigt werden: Erstens muss jeder Knoten wissen, zu welchem Head-Node seine Informationen gesendet werden (der Head-Node fungiert als eindeutiges Identifikationsmerkmal eines Ereignisses), da, wie in Abbildung 4.2 bei Schritt 5 zu sehen ist, auch mehrere Nachbarsknoten in Zustand 1 oder 2 sein können, diese jedoch demselben Ereignis angehören und zum selben Head-Node senden. In diesem Fall soll natürlich der Pfad nicht erneut umgeleitet werden. In Kapitel 4.1.3.2 wird besprochen, ob es sich lohnen könnte, statt dass jeder Knoten in Zustand 1 seinen Head-Node kennt, diesen jedes Mal von neuem im Netz abzufragen. Zweitens muss jeder Knoten, obwohl es sich um einen gerichteten Graphen handelt, wissen, welche

Knoten Informationen an ihn senden. In der folgenden Abbildung kann von Schritt 2 zu Schritt 3 nachvollzogen werden, warum dies so ist: der Head-Node aus Schritt 2 oben rechts wird in Zustand 1 überführt, muss jedoch alle Knoten, welche Informationen zu ihm sendeten informieren können, dass der Head-Node gewechselt hat.

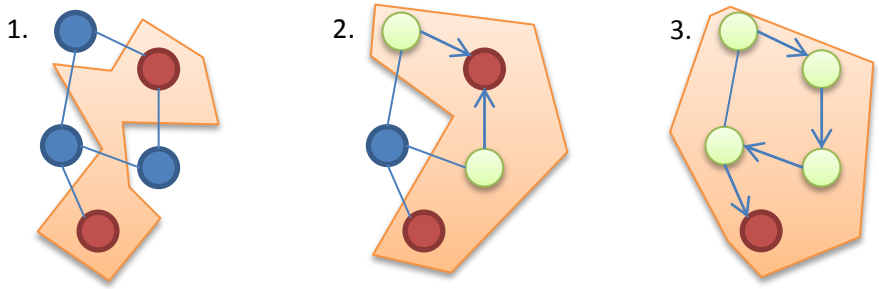


Abbildung 4.4: Verschmelzen zweier Events
Blaue Knoten = Zustand 0; grüne Knoten = Zustand 1; rote Knoten = Zustand 2; blaue Linien = Kanten des Graphen; blaue Pfeile = primäre Informationsflussrichtung zwischen den Knoten; orange Fläche = Event/Feuer

Auch hier wird es wiederum dem Zufall überlassen, welcher Head-Node nun in Zustand 2 bleibt und welcher in Zustand 1 wechseln muss. Es wird in Kapitel 4.1.3.1 besprochen, welche möglichen Optimierungen sich hier bieten.

4.1.2.4 Aufteilung und Schrumpfung eines Ereignisses

Neben der Verschmelzung kann natürlich auch das Gegenteil eintreffen: Ein Ereignis teilt sich in zwei neue Ereignisse auf. Dies wird direkt zusammen mit dem Schrumpfen eines Ereignisses behandelt, da das Verhalten gleich ist. Die Ausgangssituation dazu bildet ein Knoten, welcher in Zustand 1 war (wie sich ein Head-Node im Zustand 2 verhält, wird später erklärt) und kein Ereignis mehr detektiert:

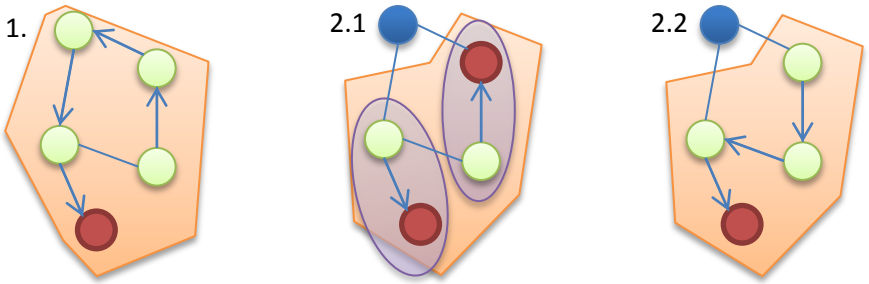


Abbildung 4.5: Ausscheiden eines Knotens aus einem Ereignis
Blaue Knoten = Zustand 0; grüne Knoten = Zustand 1; rote Knoten = Zustand 2; blaue Linien = Kanten des Graphen; blaue Pfeile = primäre Informationsflussrichtung zwischen den Knoten; orange Fläche = Ereignis/Feuer

In Schritt 2.1 kann man sehen, wie der Knoten oben links neu ausserhalb des Ereignisgebietes ist. Dieser setzt seinen Zustand wieder auf 0 und informiert den Knoten rechts von ihm, welcher bis anhin Informationen über den ausgeschiedenen Knoten an den Head-Node gesendet hat. Dieser wechselt selbständig in Zustand 2 und informiert rückwärts dem Pfad entlang alle Knoten, dessen Informationen über ihn an den alten Head-Node (unten) liefen, dass er der neue Head-Node sei. Wir haben somit temporär aus einem Ereignis virtuell zwei Ereignisse gebildet (violette Ovale). Als nächstes sucht jeder Knoten im neuen Subnetz, ob er einen Nachbarknoten findet, der einen anderen Head-Node hat. Wird ein solcher gefunden, passiert im Schritt 2.2 dasselbe wie bei der Verschmelzung von zwei Ereignissen (Kapitel 4.1.2.3), welche ja temporär virtuell auch so vorhanden

waren. Wenn nicht, sind automatisch schon zwei neue Ereignisse entstanden, was eine Aufteilung eines Ereignisses bedeutet.
Man kann sich jetzt fragen, warum zuerst zwei virtuelle Ereignisse gebildet werden und nicht direkt zuerst nach einem neuen Verbindungspunkt zum vorherigen Head-Node gesucht wird. Der Grund dafür ist jedoch relativ trivial: Da bei einer Trennung ein Knoten noch immer davon ausgeht, dass er zum alten Head-Node senden kann (der Head-Node fungiert wie schon erwähnt als eindeutiges Identifikationsmerkmal eines Ereignisses), könnte eine direkte Suche nach einem neuen Verbindungsknoten mit demselben Head-Node zu einem Loop führen (sich mit einem anderen Knoten verbinden, der aber auch über den weggefallenen Knoten Informationen an den Head-Node weiterleitete). Die vorübergehende Information, dass ein neuer Head-Node für das Ereignis zuständig ist, erfüllt also die gleiche Aufgabe, wie wenn all diese Knoten nur darüber informiert würden, dass sie nicht mehr zum alten Head-Node verbunden sind. Der Nachteil daran wäre aber, dass alle Knoten ein zweites Mal informiert werden müssten, wenn keine Verbindung zum alten Ereignis gefunden werden kann (Aufteilung des Gebietes), was nicht sehr effizient wäre.

4.1.2.5 Kommunikation innerhalb eines Ereignisses und Übermittlung an den Sink-Node

Nachdem also in den letzten Kapiteln ein Algorithmus erarbeitet wurde, der innerhalb eines Ereignisses immer einen Head-Node bestimmt, mit dem alle andere Knoten per Multi-Hop verbunden sind, stellt sich die Frage, was jetzt genau in diesem Netz gesendet werden soll. Zuerst sollte einmal erwähnt werden, welche Informationen ein Knoten X in einem solchen Netz speichert:

- Den Head-Node (=ID) des Ereignis-Subnetzes, an welchen Informationen gesendet werden müssen.
- Den nächsten Knoten, über welchen Informationen zum Head-Node gesendet werden können.
- Alle Knoten, welche mit diesem Knoten X verbunden sind und somit Informationen per Multi-Hop über ihn an den Head-Node senden.
- Die Detektionsfläche aller Knoten, welche über Knoten X Informationen zum Head-Node senden.

Diese Informationen müssen immer up-to-date sein, da es sonst zu Fehlern im Netzwerk kommen kann. Gibt es also zum Beispiel einen neuen Head-Node, müssen alle Knoten darüber informiert werden. Dies geschieht normalerweise in Gegenrichtung des Graphen. Aus diesem Grund müssen auch alle Knoten wissen, welche anderen Knoten Informationen an sie senden.
Der zentrale Teil ist jedoch die Informationssammlung über das Ereignis. In dieser Arbeit beschränkt sich dies auf die Übermittlung der Fläche eines Ereignisses. Jeder Knoten, der neu ins Ereignis-Netz eingegliedert wird, übermittelt die Ereignisfläche, die er detektiert, an den Head-Node. Woher weiss aber der Knoten, wie gross sein Zuständigkeitsgebiet ist? Es wird davon ausgegangen, dass jeder Knoten in der Lage ist, approximativ die Fläche des Gebietes zu ermitteln, für welche er zuständig ist. Hat man zum Beispiel ein Netzwerk, welches mit 100 Knoten eine Fläche von $100m^2$ bedeckt, dann ist ein Knoten im Durchschnitt für ca. $1m^2$ repräsentativ. Hat man jedoch nicht eine regelmässige Verteilung kann dieser Wert für jeden einzelnen Knoten variieren. Ein Voronoi-Diagramm kann hier eine gute Annäherung schaffen und ist gemäss Alsalih et. al. (2008) approximativ dezentral berechenbar. Der Einfachheit halber wurde jedoch in dieser Arbeit mit folgender Annäherungsformel gerechnet:

Formel 4.1

$$A = \frac{(2\overline{r})^2}{n}$$

, wobei A die Fläche, $\overline{r}$ die durchschnittliche Distanz zu den im Graphen verbundenen Knoten und n die Anzahl der im Graphen verbundenen Knoten sind. Auf die Formel wird hier nicht näher eingegangen, jedoch bestätigte sich bei mehreren Untersuchungen, dass der Fehler der aufsummierten Flächen aller Knoten mit den Parametern aller in Kapitel 5 gerechneten Simulationen maximal 5% von der Gesamtfläche des Untersuchungsgebietes abweicht. Diese Genauigkeit reicht hier, da diese Flächenberechnung nicht Teil des Algorithmus ist, sondern als bekannt angenommen wird. Jedoch ist diese nötig, um eine dezentrale Abschätzung der Relevanz neuer Informationen zu vollziehen, um die Effizienz des Algorithmus auf die erforderte Genauigkeit an Information anzupassen, wie es in Kapitel 3.4 gefordert und in Kapitel 5.3 untersucht wird.
Hier soll kurz veranschaulicht werden, wie diese Informationsverteilung der Ereignisgrösse vonstatten geht. Die Zahl innerhalb der Knoten zeigt die momentan ihnen bekannte Ereignisgrösse an. Der Einfachheit halber wird angenommen, dass jeder Knoten eine Fläche der Grösse 1 repräsentiert:

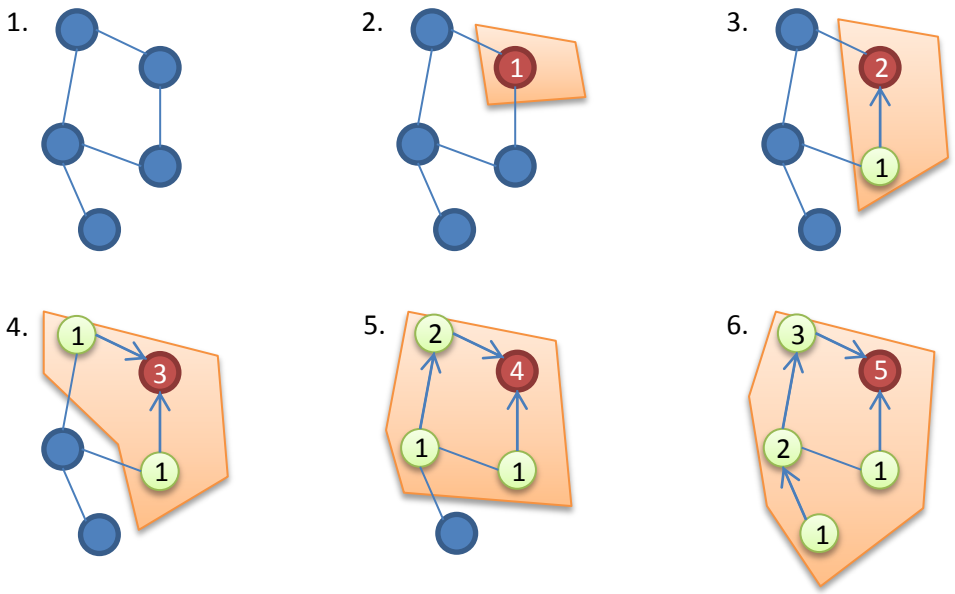


Abbildung 4.6: Informationsübermittlung innerhalb eines Ereignisses zum Head-Node
Blaue Knoten = Zustand 0; grüne Knoten = Zustand 1; rote Knoten = Zustand 2; blaue Linien = Kanten des Graphen; blaue Pfeile = primäre Informationsflussrichtung zwischen den Knoten; orange Fläche = Ereignis/Feuer; Zahlen innerhalb der Knoten = die dem Knoten bekannte Ereignisfläche

Wie man sieht, weiss der Head-Node zum jedem Zeitpunkt die Grösse des gesamten Ereignisses. Diese Informationsübermittlung muss selbstverständlich auch beim Ausscheiden eines Knoten aus einem Ereignis sowie dem Zusammenschluss von zwei Ereignissen funktionieren. Darauf soll hier jedoch nicht weiter eingegangen werden. Im Anhang (Kapitel 8) kann im Quellcode nachgelesen werden, wie dies dann in den verschiedenen Szenarien genau funktioniert.
Doch was macht der Head-Node nun genau mit dieser Information über die Ereignisgrösse? Hier setzt eigentlich der Hauptpunkt des dezentralen Algorithmus an. Dadurch, dass der Head-Node eines Ereignisses über alle Informationen innerhalb des Ereignisses verfügt, kann dieser die Daten schon vorverarbeiten. So kann verhindert werden, dass auch unwichtige Informationen über die langen Sendewege an den Sink-Node übermittelt werden. Am Beispiel in Abbildung 4.6 kann man sich

vorstellen, dass zum Beispiel für eine Einsatzleitung zur Waldbrandbekämpfung von Interesse ist, wenn ein Ereignis das erste Mal registriert wird (Schritt 2). Danach nehmen wir einmal an, dass nur jeweils eine Verdoppelung der Waldbrandgrösse von Interesse ist. Demnach würden nur noch in schritt 3 und 5 Informationen an den Sink-Node gesendet werden und die Zwischenschritte ausgelassen. Auch kann man sich anstelle eines relativen Ereigniswachstums vorstellen, dass jeweils ein absoluter Zuwachs um x m^2 von Interesse ist, wobei x hier jede beliebige Zahl sein könnte. Denkbar wäre auch, dass die Informationen nur in gewissen Zeitabständen aktualisiert werden. Die genaue Ausgestaltung wäre wohl von so vielen Faktoren abhängig, dass die Regeln, wann Informationen an den Sink-Node gesendet werden, für jeden Anwendungszweck individuell eingestellt werden müssten. Welche Möglichkeiten sich bieten, wird in Kapitel 4.1.4.2 noch genauer besprochen.

In dieser Arbeit wurden drei Variablen verwendet, welche darüber entscheiden, ob Informationen an den Sink-Node weitergeleitet werden oder nicht: Erstens eine minimale absolute Änderung der Ereignisgrösse, bevor die nächste Meldung gesendet wird. Die zweite Variable ist eine minimale relative Änderung der Ausbreitung des Ereignis im Bezug zur letzten gesendeten Meldung und die dritte Variable ist eine maximale absolute Grössenänderung, nach der in jedem Fall eine neue Meldung gesendet wird. Ein konkretes Beispiel soll den Einfluss dieser drei Variablen veranschaulichen:

- Minimale absolute Flächenänderung: 25m^2
- Minimale relative Flächenänderung: 1.5
- Maximale absolute Flächenänderung: 500m^2

Nun erkennt ein Knoten mit einer Ereignisfläche von 10m^2 ein Feuer. Diese Information wird weitergeleitet, da das erste Auftauchen eines neuen Ereignisses immer gemeldet wird. Ein weiterer Knoten mit 20m^2 kommt dazu. Diese Information wird nicht an den Sink-Node gesendet, weil die minimale absolute Flächenänderung von 25m^2 nicht erreicht wird. Dann schliesst sich ein weiterer Knoten mit 17m^2 Flächenzuständigkeit dem Head-Node an. Nun ist die minimale absolute Änderung von 25m^2 erreicht, da nun insgesamt die Fläche seit der letzten Meldung von 10m^2 auf 52m^2 angestiegen ist. Die nächste Information wird dann ab 78m^2 an den Sink-Node gesendet, da dann die minimale Änderung von 25m^2 sowie die minimale relative Flächenänderung von Faktor 1.5 erfüllt sind. Dies geht so weiter bis 1000m^2. Danach wird jeweils immer eine bei einer Differenz von 500m^2 informiert, da immer spätestens nach einer Änderung von 500m^2 gemeldet wird, unabhängig der minimalen relativen Flächenänderung von Faktor 1.5. Dieselben Regeln gelten natürlich auch, wenn sich ein Ereignis wieder verkleinert.

Wie erwähnt könnten auch zusätzliche Informationen beim Head-Node verarbeitet und an den Sink-Node weitergeleitet werden. So könnten Informationen wie Maximaltemperatur, Durchschnittstemperatur etc. durchaus interessant sein und könnten auch durch den aufgebauten Graphen übermittelt werden. Dies wird jedoch im Rahmen dieser Arbeit nicht weiter untersucht, da sich der Aufwand theoretisch nur um einen fixen Faktor verändern würde.

4.1.3 Nicht verwendete Ideen

Während der Programmierarbeit wurden noch viele weitere Ideen implementiert und getestet, die jedoch aus Effizienzgründen wieder verworfen wurden. Hier sollen die wichtigsten trotz allem kurz vorgestellt und erläutert werden:

4.1.3.1 Auswahl eines „optimalen“ Head-Nodes

Im vorgestellten Algorithmus ist der Knoten, welcher ein Ereignis als erster detektiert, automatisch der Head-Node. Kommt es zu einem Zusammenschluss zweier Ereignisse, wird es dem Zufall überlassen, welcher Knoten in Zustand 2 und somit Head-Node bleibt. Hier stellt sich automatisch die Frage, ob es nicht eine sinnvollere Möglichkeit als den Zufall gäbe, den Head-Node zu bestimmen. So kann man sich zum Beispiel vorstellen, dass ein Knoten, der sich bis anhin weniger lang im Ereignis befunden hat, wohl länger im Ereignis verbleiben wird. Denn wenn der Head-Node wechselt, müssen, wie in Kapitel 4.1.2.5 erwähnt, alle sich im Ereignis befindenden Knoten über ihren neuen Head-Node informiert werden, da sonst gewisse Abläufe nicht mehr stabil funktionieren. Es sollte also von Interesse sein, dass der Head-Node so selten wie möglich wechselt, da ein Wechsel immer auch mit Informationsübermittlung und somit mit Energieverbrauch verbunden ist.
Es hat sich jedoch herausgestellt, dass sich im Normalfall eine solche „intelligente“ Lösung nicht lohnt, und dies hat einen einfachen Grund: Um alle Knoten innerhalb eines Ereignis über einen neuen Head-Node zu informieren ist pro Knoten im Mittel eine Informationsübermittlung an einen anderen Knoten nötig, da jeder Knoten auch genau eine Information bekommt. Auch wenn die Wege innerhalb dieses Graphen relativ lang sein können, wird die Last doch sehr gut auf alle Knoten verteilt, was für jeden einzelnen Knoten eine sehr geringe Belastung bedeutet. Schon alleine die Suche nach einem „optimalen“ Knoten verbraucht in der Regel einiges mehr an Energie, da eine Informationsübermittlung alleine nicht genügt, weil Knoten untereinander auch verglichen werden müssten. Verschiedene Simulationen haben ergeben, dass ein Knoten, welcher spezifisch ausgewählt wurde, aber nicht so viel länger Head-Node bleibt, als dass sich der zusätzliche Aufwand für dessen Suche lohnen würde. Es wäre zwar vermessen zu behaupten, es gäbe a priori keine bessere Lösung, jedoch scheint eine solche, wenn es sie gäbe, erstens nicht ganz einfach zu implementieren und zweitens müsste wiederum untersucht werden, ob diese Lösung dann auch unabhängig von dem zu detektierenden Ereignis besser performt.

4.1.3.2 On-time-Abfrage des aktuellen Head-Nodes

Eine weitere Idee, welche untersucht wurde, ist, ob es nicht effizienter wäre, statt jeden Knoten beim Wechsel eines Head-Nodes über den neuen Head-Node zu informieren, nur bei Bedarf die Information über den Head-Node im Subnetz direkt abzufragen. Jedoch hat sich auch diese Idee als in der Simulation untauglich erwiesen. Wird der aktuelle Head-Node erst ermittelt, wenn die Information gebraucht wird, heisst dies, dass eine Meldung dem Pfad entlang zum Head-Node gesendet wird, und danach die Information über den aktuellen Head-Node zurück an den Absender gelangen muss. Diese Variante entlastet zwar Stränge im Netzwerk, welche diese Informationen wenig brauchen, führt jedoch insgesamt zu einer durchschnittlich stärkeren Belastung sowie zu einer einseitigen Belastung der Knoten in der Nähe des Head-Nodes, wie es auch in Bezug auf den Sink-Node in Kapitel 4.1.1 angesprochen wird. Insgesamt konnte keine positive Bilanz aus einer solchen Variante gezogen werden, besonders wenn man sieht, wie klein der Aufwand für jeden einzelnen Knoten ist, alle sich im Ereignis befindenden Knoten über einen neuen Head-Node zu informieren.

4.1.3.3 Dynamischer Netzaufbau in bestimmten Intervallen

Eine weitere Idee, welche untersucht wurde, ist ein dynamischer Netzaufbau innerhalb eines Ereignisses, der in einem gewissen Zeitintervall immer wieder neu erzeugt würde. Dies hätte zum einen den Vorteil (obwohl dieses Thema innerhalb dieser Arbeit nicht behandelt wird), dass bei einer Fehlfunktion eines Knotens dieser keinen Einfluss über eine längere Zeit hätte, da die dezentrale Informationsaggregation jedes Mal neu aufgebaut würde. Ein solcher Algorithmus funktioniert in einer Simulation auch weiter ohne Probleme und stellt sich auch als relativ performant heraus. Jedoch gäbe es in der Realität wohl grössere Probleme mit der Synchronisation eines Netzes, da irgendwie entschieden werden müsste, von welchem Knoten aus dieser Netzaufbau initialisiert würde, geht man von einer symmetrischen Rollenverteilung der einzelnen Knoten aus (vgl. Kapitel 2.1). Dies würde wiederum zu einem zusätzlichen Aufwand führen, der den Aufwand der Erhaltung eines permanenten Netzes wohl übersteigt. Auch hier kann nicht ausgeschlossen werden, dass eine solche Variante in der Realität durchaus auch erfolgreich umgesetzt werden könnte, jedoch konnte dies im Verlauf dieser Arbeit so nicht verifiziert werden.

4.1.4 Mögliche Verbesserungen

In diesem Kapitel werden mögliche Verbesserungen des Algorithmus angesprochen, welche jedoch im Verlauf dieser Arbeit nicht implementiert und getestet wurden.

4.1.4.1 Minimierung der Anzahl Hops innerhalb eines Ereignisses

In der vorgestellten Variante des dezentralen Algorithmus ist die Topologie des Graphen innerhalb eines Ereignisses durch den Verlauf des Ereignisses sowie die Abfragereihenfolge des Knoten beim Anschluss an das Ereignis vorgegeben. Der Graph wächst also historisch mit dem Verlauf des Ereignisses an. Es ist jedoch gut möglich, dass dies nicht immer einen optimalen Graphen gibt und oft sehr viel mehr Hops zum Head-Node benötigt werden, als unbedingt nötig sind. Gerade wenn die Ereignisse sehr gross sind, können sich so sehr lange Sendewege innerhalb eines Ereignisses bilden. Es könnte darum durchaus Sinn machen, diese Verbindungen von Zeit zu Zeit auf effizientere Varianten zu prüfen. Inwiefern sich dieser Mehraufwand gegenüber den Ersparnissen, welche durch die neue Topologie gemacht werden können, lohnt, müsste sich in den Simulationen noch beweisen. Auch wird dies sehr stark vom Verwendungszweck abhängen, da sich ja nicht alle Umweltereignisse in der gleichen Geschwindigkeit und der gleichen Form ausbreiten und auch wieder verschwinden.

4.1.4.2 Aggregationsvarianten

Der in dieser Arbeit entwickelte Algorithmus versucht Energie zu sparen, indem ein Head-Node dezentral eruiert, welche Informationen wichtig sind und welche Informationen es nicht wert sind, zum Sink-Node weitergeleitet zu werden. Diese Unterscheidung wird wie beschrieben durch drei Parameter vollzogen: Die minimale Ereignis-Flächenänderung (absolut und relativ) sowie die maximale Ereignis-Flächenänderung bilden den Rahmen dafür, welche Informationen als relevant erachtet werden. Es gäbe jedoch noch viele andere Varianten: So könnten fixe Grenzwerte gespeichert werden, bei dessen Überschreitung die Information weitergemeldet wird. Nimmt man den Waldbrand als Beispiel, so könnten diese Grenzen beispielsweise so gesetzt werden, dass Feuer erst ab der Grösse von $100m^2$ gemeldet werden, da kleinere Feuer oft von alleine verlöschen, ohne grössere Schäden anzurichten. Dann wird ab einer Grösse von $4000m^2$ wieder informiert, da ab dieser Grösse nicht nur Löschflugzeuge eingesetzt werden, sondern auch Feuerwehrmänner vor Ort sein müssen. Danach bräuchte es gar keine Meldungen mehr, da das Feuer von Personen lokal überwacht wird. Auch könnte man sich aber vorstellen, dass diese Informationen zeitlich

determiniert werden: So ist ein Feuer vielleicht erst von Interesse, wenn es mindestens schon eine halbe Stunde brennt, da erst länger brennende Feuer eine hohe Gefahr bergen, weil diese in der Regel nicht mehr von alleine verlöschen. Auch denkbar sind Kombinationen aus diesen Varianten, eventuell auch verbunden mit zusätzlichen Messwerten. So ist ein Feuer bei hoher Bodenfeuchtigkeit vielleicht nicht schon zum gleichen Zeitpunkt bedrohlich wie ein Feuer in einem Waldgebiet, das sich an einem extrem trockenen Standort befindet.
Man kann sich noch sehr viele verschiedene Szenarien ausdenken, jedoch ist hier, definiert man diese Regeln für einen ganz bestimmten Einsatzzweck, sehr viel Energiesparpotential vorhanden. Ein zusätzlich positiver Nebeneffekt davon ist, dass deren Relevanz zentral nicht nochmals beurteilt werden müsste, wenn nur relevante Informationen gesendet werden.

4.1.4.3 Mehrstufige Hierarchie

Im hier vorgestellten Algorithmus gibt es in einem Ereignis genau einen Head-Node, zu welchem alle Informationen gesendet werden. Es gibt also sozusagen insgesamt zwei Hierarchiestufen innerhalb eines Ereignisses: den Head-Node (Zustand 2) und die normalen Knoten innerhalb eines Ereignisses (Zustand 1). Man könnte sich jedoch überlegen, eine erweiterte Hierarchie einzubauen, indem auch Zwischenknoten zuerst Daten aggregieren und diese dann als Ganzes an den Head-Node weiterleiten. Somit könnte auf der Traffic innerhalb eines Ereignisses massiv reduziert werden, denn der dezentrale Algorithmus funktioniert bisher innerhalb eines Ereignisses mehr oder weniger gleich wie der zentrale Algorithmus innerhalb des gesamten Untersuchungsgebietes: beim zentralen Algorithmus sendet jeder Knoten seine Informationen unverzüglich per shortest-Path an den Sink-Node (vgl. Kapitel 4.2), beim dezentralen Algorithmus sendet jeder Knoten innerhalb eines Ereignis seine Informationen unverzüglich über den errichteten internen Graphen an den Head-Node. Man kann sich also eine Dezentralisierung innerhalb der Dezentralisierung vorstellen, was den Vorteil hätte, dass bei extrem grossen Ereignissen durch die wachsende Hierarchie theoretisch der Aufwand innerhalb des Ereignisses im besten Fall sogar konstant gehalten werden kann. Ob eine solche Implementation sinnvoll ist, hängt wiederum sehr stark von den Anforderungen an das Geosensor Network ab, denn wie schon in Kapitel 4.1.2.5 erläutert wurde, ist eine Dezentralisierung der Algorithmen auch mit einer Dezentralisierung der Informationsverarbeitung verbunden. Das heisst konkret, dass durch eine zu steile Hierarchie je nach Einstellung kaum mehr Informationen an den Sink-Node gelangen, was natürlich auch nicht dem Ziel eines Geosensor Network entspricht. Eine solche weitere Hierarchisierung müsste also parallel zu einem verbesserten internen Kommunikations- und Filterkonzept der Informationen erarbeitet und implementiert werden.

4.2 Zentraler Vergleichsalgorithmus

Damit der in Kapitel 4.1 entwickelte Algorithmus evaluiert werden kann, braucht es einen Vergleichsalgorithmus. Beim verwendeten Vergleichsalgorithmus handelt es sich um eine der einfachsten Varianten der Informationsweiterleitung. Auch hier wird wiederum mit verschiedenen Zuständen gearbeitet, jedoch nur mit zweien:

Zustand 0: Normalzustand → Der Knoten ist in keinem Ereignis.

Zustand 1: Ereigniszustand → Der Knoten befindet sich in einem Ereignis.

Da im zentralen Algorithmus keine dezentrale Verarbeitung der Daten stattfindet, braucht es auch keine Head-Nodes. Der Algorithmus besteht eigentlich nur daraus, bei einer Zustandsänderung diese an den Sink-Node zu melden. Somit weiss der Sink-Node zu jedem Zeitpunkt, welcher Knoten sich in einem Ereignis befindet, und welcher nicht. Diese Zustandsänderung wird per Shortest-Path, wie er in Kapitel 3.2 beschrieben ist und auch beim dezentralen Algorithmus verwendet wird, an den Sink-Node gesendet.

4.3 Unterschiede zwischen den Algorithmen bezüglich der Informationsqualität

Obwohl die beiden Algorithmen schlussendlich denselben Zweck erfüllen sollen, gibt es trotzdem noch Unterschiede bezüglich der Informationen, welche an den Sink-Node übermittelt werden. Wie schon erwähnt, basiert der dezentrale Algorithmus darauf, dass die Daten dezentral verarbeitet werden und gefiltert und aggregiert zum Sink-Node gelangen. Dies ist beim zentralen Algorithmus nicht der Fall, bei dem jede kleine Information einzeln an den Sink-Node übermittelt wird. Dies hat den Vorteil, dass die Daten in ihrer Roh-Form zentral verarbeitet werden können, da noch kein Datenverlust entstanden ist. Auf der anderen Seite sind in den Informationen des dezentralen Algorithmus implizit schon Informationen zur Netzwerktopologie vorhanden. Dadurch, dass die Aggregation zu einem Ereignis durch die Knoten-Nachbarschaft des Graphen gegeben ist, impliziert die dezentrale Information, dass es sich bei einem Ereignis um eine zusammenhängende Fläche handeln muss. Während beim zentralen Algorithmus also noch kein Datenverlust entstanden, dafür jedoch die Datenmenge sowie die Redundanz und Korrelation der Daten sehr gross und der Informationsgehalt relativ klein ist, wurde beim dezentralen Algorithmus das Gegenteil erreicht, was allgemein auch eher wünschenswert sein dürfte (vgl. Kapitel 2.1). Die Verarbeitung der Daten wurde also statt zentral schon zu einem grossen Teil dezentral im Sensor-Netzwerk selber durchgeführt. Geht man davon aus, dass die am Schluss verwendeten Informationen dieselben sind, sind beide Algorithmen bezüglich des resultierenden Informationsgehalts ebenbürtig. Jedoch gilt es zu bedenken, dass der durch die dezentrale Verarbeitung resultierte Verlust an Daten nicht mehr rückgängig gemacht werden kann, während mit den Rohdaten noch vertieftere Untersuchungen angestellt werden könnten. Dieser Umstand wurde schon in anderen Arbeiten wie jener von Yu et. al. (2005) angesprochen und untersucht und wird in Kapitel 5 in einigen Diskussionen nochmals aufgegriffen.

4.4 Einordnung des dezentralen Algorithmus

In diesem Kapitel sollen, wie bereits in Kapitel 2.3 erwähnt, in der Literatur beschriebene Methoden zur Ereignis-Detektion vorgestellt und in Bezug zur erarbeiteten Methode gebracht werden. Das Problem, weshalb es kaum möglich ist, verschiedene Algorithmen in Simulationen direkt zu vergleichen, ist zum einen, dass die meisten Source-Codes gar nicht oder nur schwer zugänglich sind, und zum anderen, dass praktisch jede Methode von anderen Voraussetzungen und Rahmenbedingungen ausgeht sowie andere Ziele verfolgt und oft nur das Oberthema der „Event Detection“ mit anderen Methoden gemeinsam hat. So beschreibt Zhang et. al. (2008), wie in Kapitel 2.3 bereits erwähnt, verschiedene Ausreisser-Detektions-Techniken und versucht diese anhand einer Taxonomie einzuordnen, damit ein Vergleich über die Eigenschaften der verschiedenen Methoden im weitesten Sinn möglich ist, ohne jedoch auf die Performance einzelner Algorithmen einzugehen.
Der in dieser Arbeit beschriebene dezentrale Algorithmus verwendet, wie schon in Kapitel 4.1.2.1 angesprochen, gemäss Laube und Duckham (2009) eine lokale Absorptions-Strategie, welche Informationen durch multi-hop weitersendet.
Obwohl es, wie bereits erwähnt, schwierig ist, Ansätze direkt zu vergleichen, findet man in der Literatur einige Lösungen, welche etwa die gleichen Absichten verfolgen oder zumindest ähnliche Elemente enthalten wie der Algorithmus in dieser Arbeit: So ist es auch das Ziel der Algorithmen von Chang und Tassiulas (2000), Singh et. al. (1998) oder Trigoni et. al. (2005), aufgrund eines optimalen Routing-Baumes Messwerte zu aggregieren. Jedoch zielen diese Arbeiten nicht auf die Ereignis-Detektion ab sondern viel mehr auf das permanente Monitoring, so dass eine Implementierung in die hier verwendete Simulationsumgebung zur Folge hätte, dass bei jedem Step (vgl. Kapitel 3.5) die gesamten Informationen gesammelt werden, obwohl sich der Zustand einzelner Knoten gar nicht geändert hat. Genau diesem Umstand haben Deligiannakis et. al. (2009) in ihrer Methode entgegengewirkt, indem nur „interessante“ Informationen weitergeleitet werden. Jedoch untersuchen all diese Arbeiten nur einen Teil des in dieser Arbeit gestellten Problems, gehen von kleinen Netzwerken aus und bauen die Bäume jeweils direkt zum Sink-Node auf. Insofern untersuchen diese Arbeiten genau den Teilaspekt dieser Arbeit, in dem es um den Aufbau eines Baumes innerhalb eines Ereignisses zum Head-Node geht, ohne jedoch dann die Information zum Sink-Node weiterzuleiten. So wurde als Beispiel in der Arbeit von Deligiannakis et. al. (2009) bei einer maximalen Sendedistanz der Sensoren von 90m mit einer Sensornetzgrösse von 36 bis 900 Sensoren gearbeitet, wohingegen in dieser Arbeit bis 40‘000 Sensoren zum Einsatz kommen, wie im nächsten Kapitel zu sehen ist.
Viele Methoden wie zum Beispiel jene von Dziengel et. al. (2008), Wittenburg et. al. (2010), Michaelides und Panayiotou (2007) oder Easwaran (2008) hingegen setzen den Fokus der Ereignis-Detektion darauf, wie Ereignisse definiert und anhand mehrerer Sensoren erkannt werden können, wobei die Methode stark von der Grunddefinition eines zu erwartenden Ereignisses abhängt, was, wie in Kapitel 2.3 erklärt, keine triviale Angelegenheit ist. Der Algorithmus in dieser Arbeit setzt den Fokus nicht sehr stark auf diese Frage, da, wie in Kapitel 3.1 beschrieben, angenommen wird, dass ein Ereignis von einem einzelnen Knoten eindeutig identifiziert werden kann, wobei die Ereignis-Definition nicht an das Ausmass sondern nur an Grenzwerte einer einzelnen Messung gekoppelt ist. Jedoch ist die Information, ob es sich um ein Ereignis handelt oder nicht, in Form der Ereignisgrösse, welche an den Sink-Node übermittelt wird, implizit in einem gewissen Mass vorhanden, so dass anhand der Ereignisgrösse entschieden werden könnte, ob es sich um ein „relevantes“ Ereignis handelt oder nicht. Dies heisst grundsätzlich, dass diese Informationsverarbeitung zu einem gewissen

Teil weiterhin zentral und nicht dezentral erfolgt, wobei dies wiederum davon abhängt, wie die in Kapitel 4.1.2.5 beschriebenen Grenzen für die Informationsübermittlung gesetzt werden.
Grenzt man das Thema auf die „Forest Fire Detection“ ein, obwohl es sich in dieser Arbeit um einen möglichst generischen Algorithmus in der räumlichen Ereignis-Detektion handelt (vgl. Kapitel 2.3, Abbildung 2.3), bei dem das zu detektierende Ereignis eine untergeordnete Rolle spielt, so findet man auch verschiedene Ansätze. Eine sehr ähnliche Methode wie die in dieser Arbeit verwendete beschreibt Yu et. al. (2005), in welcher neben der Ereignis-Detektion sogar eine Feuergefahrvorhersage implementiert wurde, dafür aber auch vorausgesetzt wird, dass über GPS die absolute Position einzelner Knoten bekannt ist. Da bei dieser Methode nicht zwingend ein Ereignis vorausgesetzt wird, werden die Head-Nodes auch nicht anhand auftretender Ereignisse ausgewählt sondern anhand der verbleibenden Energiereserven. Auch wurden wie im Algorithmus dieser Arbeit Grenzwerte eingebaut, welche verhindern, dass alle Informationen an den Sink-Node gesendet werden. Jedoch wird auch hier von kleinen Netzwerken bei einer niedrigen Sensordichte ausgegangen, weshalb die Autoren keine klaren Aussagen über die Skalierbarkeit des Algorithmus machen konnten. Lloret et. al. (2009) hingegen wählen einen Ansatz, bei welchem es zusätzlich neben der Detektion um die Verifikation mit Hilfe von Kameras geht. Aus diesem Grund wurde in dieser Methode auch kein dezentraler Algorithmus implementiert, da eingehende Informationen visuell verifiziert werden und die Erhöhung der Netzwerklebensdauer durch zusätzliche Verdichtung der Sensorknoten erreicht wird. Hefeeda und Bagheri (2009) wählen insofern einen ähnlichen Ansatz, als dass sie von einer sehr hohen Sensordichte ausgehen, weshalb sich die Sensoren im Aktiv- und Schlafmodus die Arbeit und somit den Energieverbrauch teilen. Ähnlich wie beim in dieser Arbeit vorgestellten Algorithmus gehen die Verfasser davon aus, dass die Sensorknoten ihre absolute Position nicht kennen, implementieren jedoch wie Yu et al. (2005) eine Feuergefahrvorhersage. Das Clustering- und Routing-Problem wird jedoch in der Arbeit von Hefeeda und Bagheri (2009) nicht weiter bearbeitet, da sie sich auf das „coverage problem“ beschränken, um die Aktiv-Schlaf-Zyklen der Sensoren so zu optimieren, dass eine möglichst gute Abdeckung bei geringem Energieverbrauch erzielt wird. Eine Methode von Zhang et. al. (2007) beruht auf dem Standard ZigBee (vgl. Kapitel 2.1), wobei der Fokus vor allem auf die Technik gelegt wurde, da die Algorithmen durch den Standard vorgegeben sind. Ihre Methode setzt aber im Gegensatz zum Algorithmus in dieser Arbeit auf heterogene Knoten und beurteilt den Energieverbrauch sowie die Lokalisierung als Probleme, welche weiterhin untersucht werden müssen.

5 Evaluation der Algorithmen

5.1 Allgemeine Bemerkungen und Vorgaben

In diesem Kapitel werden die in Kapitel 4 entwickelten Algorithmen in diversen Simulationen miteinander verglichen. Diese Vergleiche basieren auf den Vorgaben, welche in Kapitel 3.5 definiert wurden:

- In Kapitel 5.2 wird ein Standardsetting ausgearbeitet, welches als Vergleichsgrundlage für die weiteren Experimente dienen soll. Anhand mehrerer Untersuchungen mit diesem Standardsetting soll unter anderem herausgefunden werden, wie stabile Ergebnisse eine Simulation bietet.
- Kapitel 5.3 befasst sich mit der Variation der Informationsgenauigkeit beim dezentralen Algorithmus. Wie in Kapitel 4.1.2.5 erklärt, kann diese Informationsdichte anhand von Parametern an die jeweiligen Anforderungen angepasst werden.
- Kapitel 5.4 beschäftigt sich mit der Skalierbarkeit der Algorithmen, indem die Grösse des Untersuchungsgebietes und parallel dazu die des Geosensor Network variiert wird.
- In Kapitel 5.5 wird das Verhalten der Algorithmen auf eine Veränderung der Sensordichte untersucht.
- Kapitel 5.6 beschäftigt sich mit der Veränderung des Graphen und welchen Einfluss diese auf die Performance der Algorithmen ausübt.
- Zuletzt wird in Kapitel 5.7 untersucht, wie die beiden Algorithmen bei einer Veränderung der durchschnittlichen Ereignisgrösse und somit bei einer Veränderung der räumlichen Autokorrelation der Ereignisse reagieren.

Alle Untersuchungen werden in der agentenbasierten Simulationsumgebung Repast Simphony[3] und jeweils über einen Zeitraum von 20‘000 Steps (vgl. Kapitel 3.5) durchgeführt. Ausser in Kapitel 5.6 wird immer der RNG als Graph verwendet.

Wie in Kapitel 3.5 bereits definiert, werden der dezentrale Algorithmus sowie der zentrale Vergleichsalgorithmus anhand ihres Energieverbrauchs, der als proportional zur Anzahl gesendeter Mitteilungen/Informationen angenommen wird, verglichen. Als Einheit wird jeweils „Messages per Step“ angegeben. Dieser Wert pendelt sich mit der Dauer der Simulation ein und nähert sich somit mit der Laufzeit asymptotisch einem Grenzwert, der nach unendlich vielen Steps erreicht würde. Konkret werden jeweils der durchschnittliche Verbrauch der einzelnen Knoten sowie jener des jeweils maximal belasteten Knotens miteinander über die verschiedenen Simulationen verglichen, um Informationen über die Effizienz der beiden Algorithmen in unterschiedlichen Situationen zu gewinnen. Diese werden soweit möglich für jeden Algorithmus einzeln beurteilt. Das Hauptaugenmerk wird jedoch auf das Verhältnis zwischen den beiden Algorithmen gelegt, das einen direkten Vergleich der beiden Algorithmen unabhängig von der absoluten Performance der einzelnen Algorithmen beschreibt. Für jede Simulation wird zudem ein Histogramm erstellt, welches genauere Aufschlüsse über die Verteilung der Belastung oder eben des Energieverbrauchs geben soll. Es soll explizit darauf hingewiesen werden, dass bei diesen Untersuchungen nur die Effizienz der Algorithmen, nicht jedoch die daraus resultierende Datenqualität beurteilt wird. In den Kapiteln 4.1.2.5 und 4.3 sind die Gründe dafür beschrieben.

[3] Repast Simphony (http://repast.sourceforge.net/) ist eine kostenlose open-source Simulationsumgebung basierend auf Java und Groovy, die in die Entwicklungsumgebung Eclipse (http://www.eclipse.org/) integriert ist. Für diese Arbeit wurden Repast Simphony 1.2.0 und Java 6.0 benutzt.

5.2 Standardsetting (Vergleichsgrundlage)

5.2.1 Experiment

Um in den Kapiteln 5.3 bis 5.7 verschiedene Parameter variieren zu können und zu untersuchen, wie die beiden Algorithmen auf deren Veränderung reagieren, muss zuerst ein Standardsetting geschaffen werden, welches den Ausgangspunkt der Untersuchungen bildet. Um ein solches Setting auszuwählen, wurden vorgängig mehrere Testsimulationen mit unterschiedlichen Parameterwerten durchgeführt, worauf folgende Werte als Standard definiert wurden:

Parameter	Standard-wert	Beschreibung
xDim	2000	Breite des Untersuchungsgebietes (m)
yDim	100	Länge des Untersuchungsgebietes (m)
initialNumberOfSensorNodes	10000	Anzahl Sensor-Knoten
initialNumberOfSinkNodes	1	Anzahl Sink-Nodes (vgl. Kapitel 3.1.2)
sensorNodeSendingDistance	10	Sendedistanz r eines Knotens (m)
eventSparkProbability	0.05	Wahrscheinlichkeit eines neuen Ereignisses pro Step
eventSpreadingProbability	0.45	Grundausbreitungswahrscheinlichkeit des Ereignisses (in alle Richtungen gemäss Von-Neumann-Nachbarschaft)
windStrength	0.2	Windstärkekorrekturfaktor
initialEventValue	25	Initialtemperatur (°C)
maxEventValue	900	Maximaltemperatur (°C)
eventValueThreshold	750	Ereignis-Temperatur-Grenzwert (°C)
reduceEventValue	10	Temperaturrückgang pro Step nach dem Ereignis
initialFuel	1.5	Initialbrennstoff (t)
consumeFuel	0.01	Brennstoffverbrauch pro Step (t)
fuelGrow	0.002	Brennstoffwachstum pro Step (t)
maxFuel	1.5	Maximalbrennstoff (t/m^2)
minAbsoluteAreaDifferenceToReport	50	Minimale Flächenänderung für Meldung (m^2)
relativeAreaDifferenceToReport	2	Minimale relative Flächenänderung für Meldung
maxAbsoluteAreaDifferenceToReport	1000	Maximale Flächenänderung für Meldung (m^2)

Tabelle 5.1: Standardparameter für die Simulationen
In der linken Spalte sind die Inputparameter der Simulationen zu sehen. In Kapitel 8 kann anhand des Quellcodes der Einfluss der einzelnen Parameter beurteilt werden.

Einige der Variablenwerte aus obenstehender Tabelle sollen hier noch kurz erklärt werden. Wie in der Tabelle zu sehen ist, wurden Distanzen in Meter, Gewichte in Tonnen und Temperaturen in Grad Celsius angegeben. Dies wurde vor allem auf Grund der intuitiven Nachvollziehbarkeit so gewählt, die Zahlen könnten jedoch jede beliebige andere Einheit repräsentieren.
Bei „xDim“ und „yDim“ sieht man sehr unterschiedliche Werte von 2000 bzw. 100. Dies ergibt eine rechteckige Untersuchungsfläche, deren Breite zwanzig Mal grösser ist als die Länge. Dies hat den einfachen Grund, dass die Routingwege zum Sink-Node (Shortest-Path, vgl. Kapitel 3.2) möglichst lang werden. Dies sieht stark danach aus, als ob versucht wird, den dezentralen Algorithmus von Grund auf zu bevorteilen, jedoch stehen weitreichendere Überlegungen hinter diesen unterschiedlichen Werten: Wie in Kapitel 2.1 bereits erläutert, wird davon ausgegangen, dass durch die immer günstiger produzierbaren Sensor-Knoten die Netzwerke in einigen Jahren um einiges grösser sein könnten als heute. Aus diesem Grund ist es von Interesse, neue Algorithmen vor allem auch auf deren Skalierbarkeit zu testen, speziell weil sehr kleine Netzwerke in der Praxis kaum von grossem Interesse sein dürften, da die Einsatzgebiete von Geosensor Networks eher grossflächige

und unzugängliche Gebiete sind. Insofern kann durch die längliche Form des Testgebiets ein Stück weit das Verhalten in noch grösseren Gebieten simuliert werden, welche noch längere Sendewege zum Sink-Node haben dürften. Es kann davon ausgegangen werden, dass die Grösse eines solchen Netzwerkes in einigen Jahren ohne Probleme auch im realen Einsatz noch um das Vielfache anwachsen könnte, handelt es sich bei diesem Untersuchungsgebiet um eine Grösse von „nur“ 20 Hektaren.

Der Wert für die Anzahl der Sensor-Knoten wurde so gewählt, dass das Untersuchungsgebiet mit einer sinnvollen Dichte von einem Sensorknoten pro $20m^2$ abgedeckt wird. Dass nur ein Sink-Node verwendet wurde, beruht auf den Restriktionen in Kapitel 3.1.2 sowie auf der Tatsache, dass die Verdoppelung der Sink-Nodes nach der Initialisierung des Netzwerkes den gleichen Effekt hat wie eine Halbierung der Gebiets- und Netzwerkgrösse, sofern die Knoten sinnvoll im Gebiet verteilt sind. Die Veränderung dieser ersten vier Parameter wird in den Kapiteln 5.4 und 5.5 untersucht.

Auch die maximale Sendedistanz von 10m wurde aus mehreren Überlegungen heraus so gewählt: Erfahrungsgemäss kann der Graph (RNG) mit dieser maximalen Sendedistanz bei einer Dichte von einem Sensorknoten pro $20m^2$ und einer zufälligen Verteilung der Knoten vollständig aufgebaut werden. Auf der anderen Seite ist die Sendedistanz nicht übermässig lang, da in der Realität das Senden von Daten über längere Distanzen zu einer Erhöhung des Energieverbrauches führt, weil das Sendesignal verstärkt werden muss (Shebli, Dayoub, & Rouvaen, 2007). Dies wird jedoch in Kapitel 5.5 nochmals angesprochen.

Die verschiedenen Parameter zur Umwelt wurden schon in Kapitel 3.3 erläutert. Alle hier gewählten Werte bauen auf der Erfahrung aus etlichen Voruntersuchungen auf. Trotzdem basieren die Werte nicht auf empirischen Untersuchungen aus der Realität, was aufgrund der Simulation durch einen zellulären Automaten auch kaum möglich ist (vgl. Kapitel 3.3). Ausserdem müssten für ein noch realistischeres Feuermodell weitaus mehr Parameter wie zum Beispiel die Feuchtigkeit des Brennmaterials mitberücksichtigt werden. Jedoch scheinen für die gewählte Grösse des Szenarios diese Parameterwerte ein vernünftiges Feuerausbreitungsmodell zu liefern, welches für die Untersuchungen in dieser Arbeit ausreicht, da die Algorithmen grundsätzlich unabhängig vom gewählten Modell funktionieren. Die relative Wahrscheinlichkeit von 5% pro Step, dass ein neues Ereignis auftaucht, ist wohl nicht sehr realistisch. Jedoch kann so in einer vergleichsweise kurzen Zeit das Verhalten über einen langen Zeitraum simuliert werden. Denn ob sich alle Jahre ein Feuer ausbreitet, oder aber jede Minute ein neues Feuer auftaucht ist für die Simulation insofern irrelevant, als dass auch keine Informationen unter den Knoten ausgetauscht werden und somit der zentrale wie auch der dezentrale Algorithmus den gleichen Energieverbrauch von 0 aufweisen würden, solange kein Ereignis gemessen wird. Auch könnten Steps in der Realität für ganz unterschiedliche Zeitspannen stehen, je nachdem welche zeitliche Auflösung gebraucht wird. Es macht aber Sinn, Ereignisse, welche vielleicht über mehrere Jahre auftreten würden, innert kürzester Zeit auftreten zu lassen, um die Performance der beiden Algorithmen unter starker Belastung zu untersuchen. Dies führt insgesamt nur zu einer linearen Skalierung der Ergebnisse, ändert jedoch nichts an den Verhältnissen, da alle Ergebnisse einfach um einen Faktor X grösser/kleiner werden. In Kapitel 5.7 wird das Verhalten eines Geosensor Network auf Veränderungen der Umwelt, welche das Verhalten der Ereignisse/Waldbrände beeinflussen, untersucht.

Die letzten drei Parameter sind wichtige Werte des dezentralen Algorithmus und wurden auch schon in Kapitel 4.1.2.5 näher erläutert. Es ist klar, dass mit diesen Werten die Informationsgenauigkeit nicht auf demselben Level ist wie beim zentralen Algorithmus (vgl. Kapitel 4.3). Jedoch gilt es zu bedenken, dass die Stärke eines dezentralen Algorithmus darin liegt, dass Informationen „gebündelt“ und nicht als Rohdaten an den Sink-Node weitergeleitet werden, was den Informationsgehalt eines

einzelnen Datenpakets erhöht. Genau diese Konzentration auf relevante Information ist gewollt, wie in Kapitel 3.4 bei den Anforderungen an den dezentralen Algorithmus anhand eines Beispiels beschrieben wurde. Eine detaillierte Auswertung der Informationsgenauigkeit wird im Rahmen dieser Arbeit, wie auch schon in Kapitel 5.1 erwähnt, nicht durchgeführt, da diese im echten Einsatz spezifisch auf die Bedürfnisse angepasst werden müsste. In diesem Standardsetting wird die minimale absolute Änderung des Ereignis-Gebiets auf $50m^2$, die minimale relative Änderung des Ereignis-Gebiets auf den Faktor 2 und die maximale absolute Änderung, welche sicher kommuniziert wird, auf $1'000m^2$ gesetzt. Wie diese drei Parameter das Verhalten des Algorithmus beeinflussen, wurde in Kapitel 4.1.2.5 anhand eines Beispiels näher erläutert. In Kapitel 5.3 wird der Einfluss dieser drei Parameter auf die Performance des dezentralen Algorithmus untersucht.
In den folgenden Kapiteln gelten diese Parameter aus Tabelle 5.1 als Grundlage, deren Werte in verschiedenen Szenarien einzeln variiert werden.
Die Simulation mit den Standardwerten wird drei Mal unabhängig durchgeführt, um erstens stabile Vergleichswerte für die folgenden Experimente zu liefern und vor allem zu evaluieren, als wie stabil die Resultate einzelner Untersuchungen angenommen werden können.

5.2.2 Erwartungen

Es wird erwartet, dass der dezentrale Algorithmus bezüglich des durchschnittlichen Energieverbrauchs besser performt als die zentrale Variante. Geht man davon aus, dass in einem Gebiet, welches 2'000m breit ist, ein Ereignis im Durchschnitt ca. 1'000m vom Sink-Node, welcher sich am rechten Rand des Gebiets befindet, entfernt ist, sind bei einer Sendedistanz von maximal 10m im Durchschnitt mindestens 100 Hops nötig, um die Information vom Ereignis an den Sink-Node zu senden. Dies heisst konkret, dass für jeden zusätzlichen Knoten, der das Ereignis neu detektiert, auf alle Knoten verteilt insgesamt ca. 100 Mitteilungen versendet werden könnten, wovon hier nicht ausgegangen werden kann, wenn der Gesamtenergieverbrauch des dezentralen Algorithmus gleich dem des zentralen Algorithmus sein soll.
Viel wichtiger jedoch als der durchschnittliche Energieverbrauch ist die Verteilung der Belastung einzelner Sensorknoten. Man führe sich nochmals die Abbildung 4.1 vor Augen, welche aufzeigt, dass insbesondere die Knoten nahe am Sink-Node als zentral für die Lebensdauer des gesamten Netzwerkes betrachtet werden können. Auch hier wird erwartet, dass der dezentrale Algorithmus eine massiv bessere Verteilung der Belastung erreicht. Es wird zwar angenommen, dass bei der dezentralen Verarbeitung der Daten viele Knoten einen höheren Energieverbrauch aufweisen, da in einem Ereignis selber die Netzwerkaktivität höher ausfallen wird als bei der zentralen Variante. Diese zusätzliche Last wird aber vor allem allgemein schwach belastete Knoten betreffen, welche nicht als kritisch für die Lebensdauer des Gesamtnetzwerkes betrachtet werden. Dafür ist eine massive Entlastung der Knoten auf den Hauptachsen zum Sink-Node hin zu erwarten, weshalb die Knoten mit den höchsten Belastungen beim dezentralen Algorithmus eine weit tiefere Belastung aufweisen dürften.
Es wird weiter erwartet, dass die Ergebnisse bei den drei unabhängigen Simulationen bis auf eine geringe Streuung relativ geringe Differenzen aufweisen dürften, weil die Simulation über einen Zeitraum von 20'000 Steps läuft und dadurch bei einer relativen Häufigkeit von 0.05 Ereignissen/Step um die 1'000 Ereignisse simuliert werden, was in einer guten Stabilität der Ergebnisse resultieren dürfte.

5.2.3 Ergebnisse

5.2.3.1 Simulation 5.2-1: Standardsetting

Als Windrichtung wurde der Zufallswert 3.18 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.46
Osten: 0.65
Süden: 0.44
Westen: 0.25

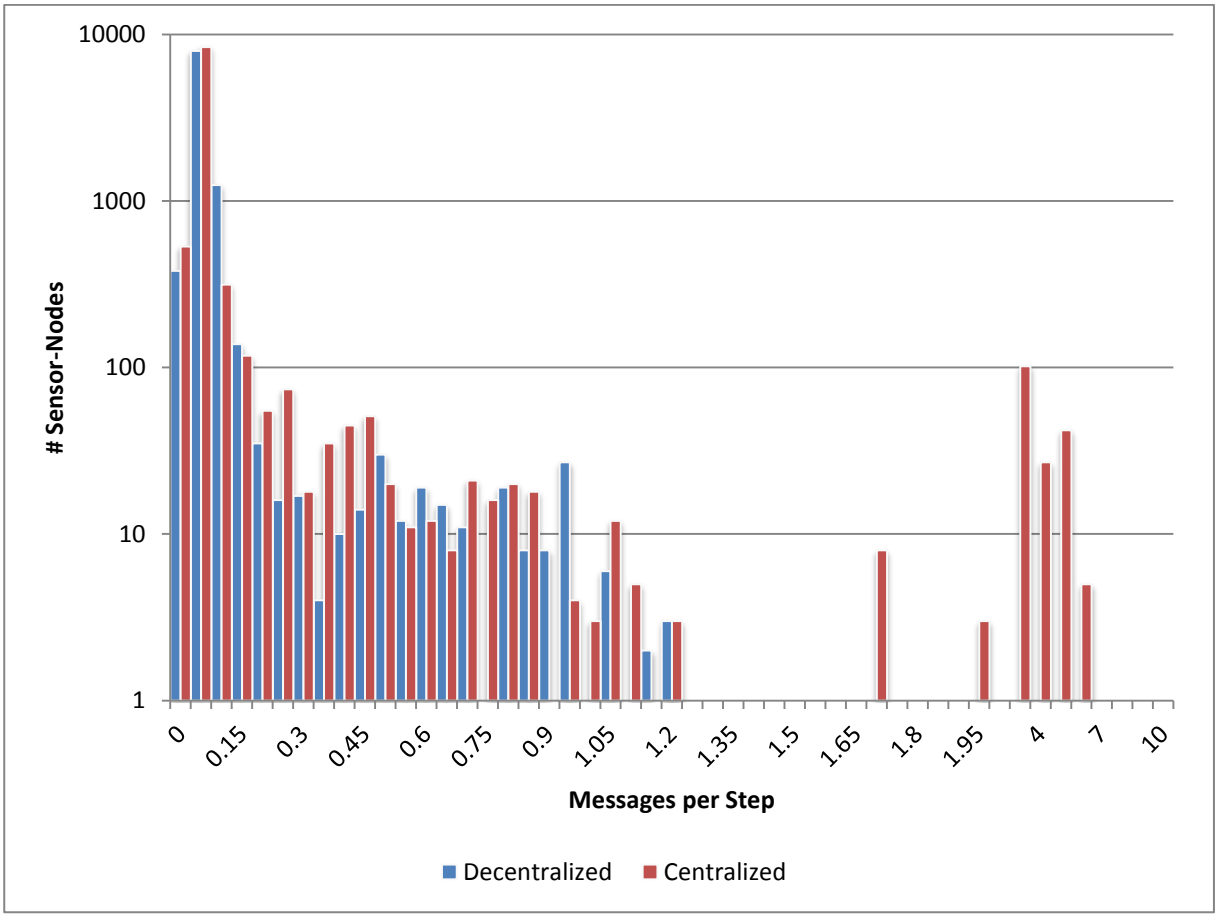


Diagramm 5.1: Simulation 5.2-1
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.038	0.084	0.457
Median	0.02	0.001	17.818
Std Dev	0.096	0.45	0.213
Max	1.171	5.768	0.203

Tabelle 5.2: Simulation 5.2-1
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.2.3.2 Simulation 5.1-2: Standardsetting

Als Windrichtung wurde der Zufallswert 0.21 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.41
Osten: 0.25
Süden: 0.49
Westen: 0.65

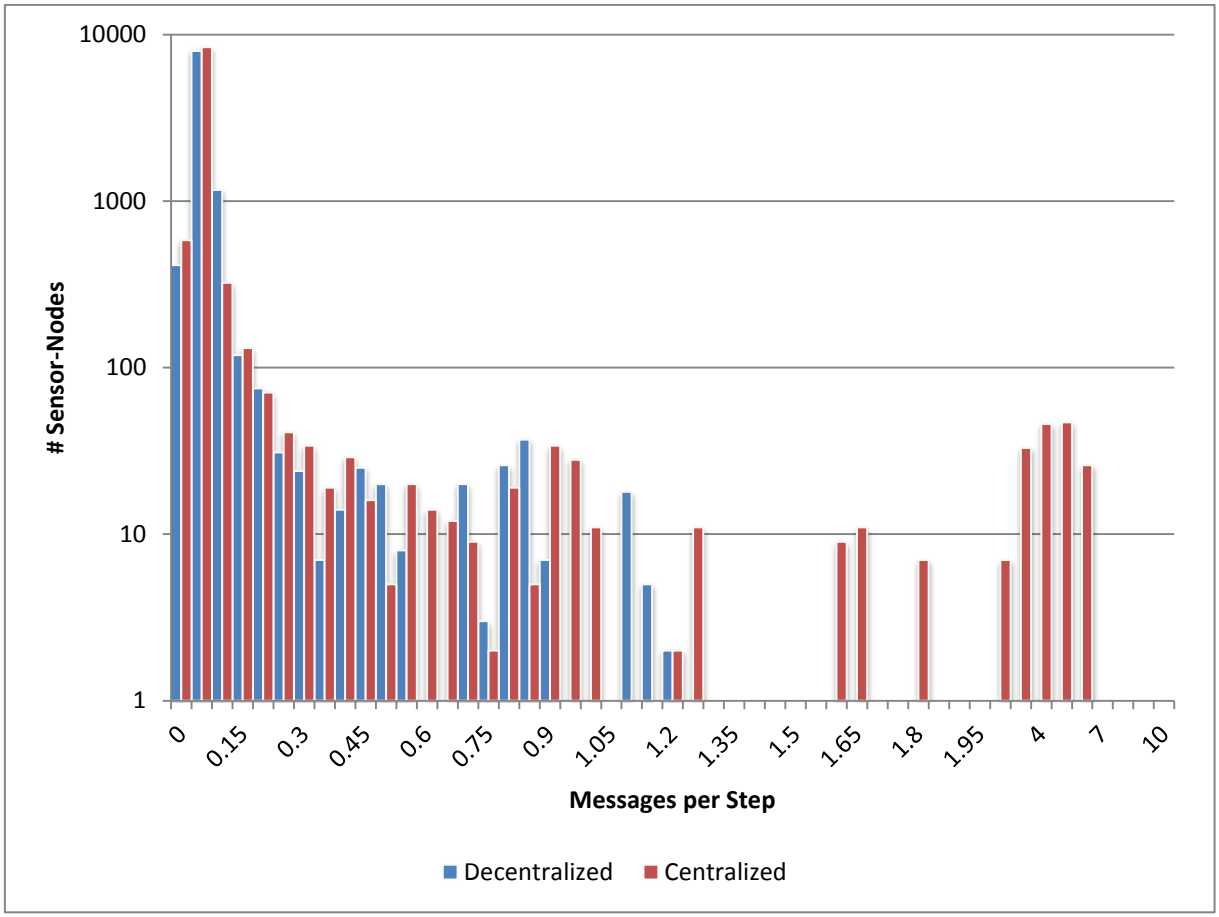


Diagramm 5.2: Simulation 5.2-2
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.04	0.09	0.443
Median	0.019	0.001	17.636
Std Dev	0.102	0.502	0.203
Max	1.199	5.982	0.2

Tabelle 5.3: Simulation 5.2-2
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.2.3.3 Simulation 5.1-3: Standardsetting

Als Windrichtung wurde der Zufallswert 5.63 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.57
Osten: 0.29
Süden: 0.33
Westen: 0.61

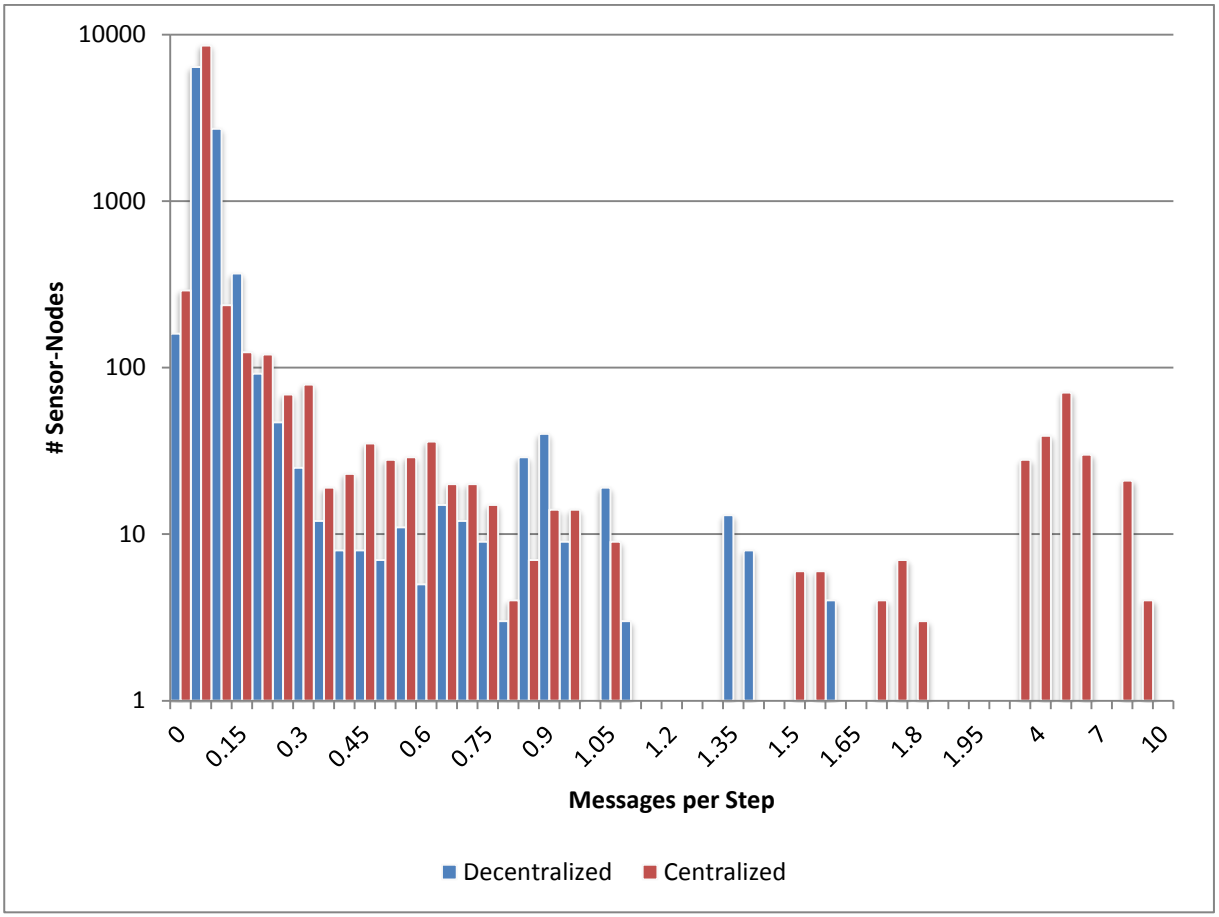


Diagramm 5.3: Simulation 5.2-3
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.057	0.123	0.463
Median	0.034	0.002	21.5
Std Dev	0.124	0.685	0.181
Max	1.589	8.823	0.18

Tabelle 5.4: Simulation 5.2-3
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.2.4 Diskussion

5.2.4.1 Vergleich der beiden Algorithmen

Aus den drei Tabellen (Tabelle 5.2 bis Tabelle 5.4) kann entnommen werden, dass der dezentrale Algorithmus bei diesem spezifischen Setting insgesamt einen tieferen Gesamtenergieverbrauch aufweist als die zentrale Variante. Die Werte des prozentualen Anteils des durchschnittlichen Energieverbrauches des dezentralen Algorithmus im Vergleich zur zentralen Lösung der einzelnen Knoten und somit auch des gesamten Netzwerkes schwankt zwischen 44.3% (Simulation 5.2-2) und 46.3% (Simulation 5.2-3). Es kann also rund die Hälfte an Energie gespart werden. Der Median-Verbrauch des dezentralen Algorithmus ist jedoch um das 17.6- bis 21.5-fache höher als derjenige der zentralen Variante. Vergleicht man innerhalb der beiden Algorithmen den Median mit dem Mittelwert, so zeigt sich, dass beim dezentralen Algorithmus mit einem rund 1.66- bis 2.06-fachen Mittelwert gegenüber dem Median ein massiv kleinerer Unterschied resultiert als beim zentralen Algorithmus, bei welchem der Mittelwert das 76.5- bis 81.8-fache des Median beträgt. Dies zeigt eine Tendenz auf, welche durch die ungefähr 5-mal höhere Standardabweichung beim zentralen Algorithmus untermauert wird: Beide Algorithmen haben sehr viele Knoten, welche schwach belastet werden, jedoch hat der zentrale Algorithmus mehr und/oder stärkere Ausreisser von Knoten, welche einen hohen Informationsdurchsatz und somit einen hohen Energieverbrauch aufweisen. Diese Tatsache wird durch den jeweiligen Knoten mit dem höchsten Energieverbrauch (unterste Spalte in der Tabelle oder der Balken ganz rechts im Histogramm) weiter verstärkt. In allen drei Untersuchungen weist der am stärksten belasteten Knoten im zentralen Algorithmus einen 5-mal höheren Energieverbrauch auf als jener im dezentralen Algorithmus. Es kann also behauptet werden, dass der dezentrale Algorithmus die Belastung und somit den Energieverbrauch regelmässiger auf die Knoten verteilt.

Die Histogramme (Diagramm 5.1 bis Diagramm 5.3) unterstreichen diese Vermutung zusätzlich. Bei allen kann die gleiche Tendenz ausgemacht werden, dass bei beiden Algorithmen die allermeisten Knoten einer Belastung von 0 bis ca. 0.2 Meldungen pro Step unterliegen. Hier sollte auch beachtet werden, dass es sich bei der Skala auf der y-Achse um eine logarithmische Skala handelt, wobei der Peak bei beiden Algorithmen und allen Histogrammen jeweils zwischen 0.05 und 0.10 Meldungen pro Step liegt, wo jeweils zwischen 63% und 85% aller Knoten enthalten sind. Hier soll auch erwähnt werden, dass auch die x-Achse nicht eine rein lineare Skala aufweist, sondern bis 2 Mitteilungen pro Step in 0.05er-Schritte unterteilt ist, und darüber in 1er-Schritten aggregiert wird. Beim zentralen Algorithmus sind die weitaus stärkeren Ausreisser zu beobachten als beim dezentralen Algorithmus. Während beim dezentralen Algorithmus nur 11 (Simulation 5.2-1) bis 47 (Simulation 5.2-3) Knoten eine Belastung von mehr als 1 Meldung pro Step aufweisen, sind es beim zentralen Algorithmus bei allen drei Versuchen über 200 oder mehr als 2% aller Knoten. Des Weiteren sind die Ausreisser wie schon erwähnt einiges stärker, was auf den stark belasteten Knoten auf den shortest-Paths zum Sink-Node hin zu begründen ist (vgl. Abbildung 5.1 auf der nächsten Seite). Inwiefern hier adaptives Routing das Bild verändern könnte, wurde bereits in Kapitel 3.2 angesprochen.

Aus den Ergebnissen heraus kann geschlossen werden, dass ein Geosensor Network, welches mit dem dezentralen Algorithmus arbeitet, eine wesentlich längere Lebensdauer haben dürfte als eines, welches den zentralen Algorithmus verwendet. Zum einen ist die Durchschnittsbelastung der Knoten tiefer und, was noch viel wichtiger ist, die Belastungsverteilung auf die verschiedenen Knoten viel ausgewogener. Es soll hier jedoch nochmals erwähnt werden, dass beim dezentralen Algorithmus auch die resultierende Datenmenge geringer ist, da die Informationen dezentral schon vorgefiltert werden. Insofern wird ein Teil der Energieersparnis mit einer total tieferen Informationsmenge

„erkauft“. Zu vergleichen ist dies mit einem digitalen Bild, bei dem die Auflösung reduziert wird, wobei individuell bezogen auf die jeweiligen Anforderungen ein Bild mit niedriger Auflösung durchaus denselben Zweck erfüllen kann. Die Erwartungen bezüglich der Performance der beiden Algorithmen konnten anhand der Ergebnisse alle bestätigt werden.

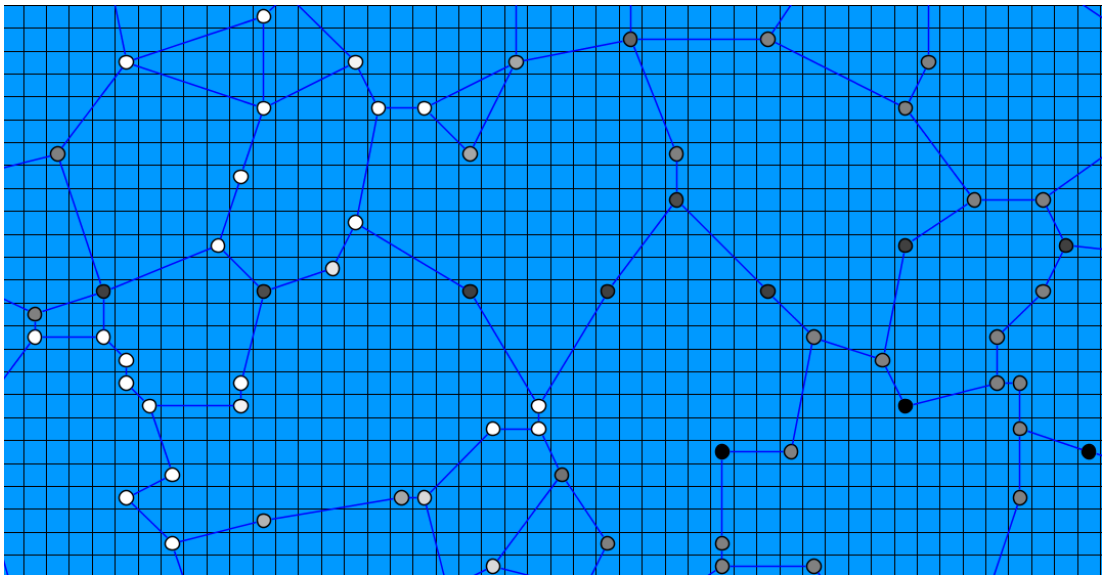


Abbildung 5.1: Belastungsverteilung der Sensorknoten
Dunkle Knoten werden durch den zentralen, helle Knoten durch den dezentralen Algorithmus stärker belastet. Hier wird nur eine relative Aussage gemacht, weshalb die absolute Belastung der Knoten nicht dargestellt wird. Jedoch zeigt sich eine starke Belastung durch den zentralen Algorithmus entlang der Shortest-Paths zum Sink-Node hin, wohingegen allgemein periphere Knoten durch den dezentralen Algorithmus einer grösseren Belastung ausgesetzt sind.

5.2.4.2 Stabilität der Ergebnisse

In diesem Experiment sollte aber auch die Stabilität der Resultate in den Simulationen getestet werden. Wenn man die Werte in den Tabellen und Histogrammen vergleicht, so kann man wohl behaupten, dass ein Test über 20‘000 Steps ausreicht, damit die Resultate eine hohe Stabilität aufweisen. Es gibt aber Unterschiede in der Gesamtbelastung zwischen den einzelnen Simulationen; so ist der Durchschnittsenergieverbrauch beim dritten Durchlauf (Simulation 5.2-3) beim zentralen wie auch beim dezentralen Algorithmus um ca. 30% höher. Warum dies so ist, ist relativ schwierig zu beurteilen, weil zelluläre Automaten, wie schon in Kapitel 3.3 erwähnt, nur sehr schwierig statistisch zu erfassen sind, da der Zufall einen relativ hohen Einfluss auf die Entwicklung eines Ereignisses und dadurch auch auf die Resultate hat. Jedoch ist dies in diesen Simulationen kein Problem, da sich beide Algorithmen an den gleichen Ereignissen messen. So zeigt sich auch, dass die relative Differenz des Energieverbrauchs zwischen den beiden Algorithmen stabil bleibt und nur sehr kleine Differenzen zu den anderen beiden Durchläufen aufweist, auch wenn im dritten Durchlauf (Simulation 5.2-3) allgemein höhere Belastungen der einzelnen Knoten gemessen wurden. Auch im Histogramm spiegelt sich dies wieder: Legt man das Histogramm des dritten Durchlaufs (Simulation 5.2-3) über eines der anderen beiden Durchläufe, so sehen diese vom Verlauf her sehr ähnlich aus, wobei nur eine Skalierung in x-Richtung stattfindet, was einer Multiplikation aller Werte um Faktor X entspricht. Des Weiteren konnte so auch gezeigt werden, dass die zufällige Windrichtung keinen wesentlichen Einfluss auf die Ergebnisse haben dürfte.
Da hier gezeigt werden konnte, dass die Ergebnisse über eine Simulationslänge von 20‘000 Steps als relativ robust betrachtet werden können, wird für die weiteren Untersuchungen nur noch jeweils eine Simulation pro Setting gerechnet.

5.3 Variation der Informationsgenauigkeit

5.3.1 Experiment

Dieses Kapitel widmet sich der Informationsqualität. Wie schon in Kapitel 5.2 beschrieben, sendet der dezentrale Algorithmus normalerweise nicht die gleiche Informationsdichte wie der zentrale Algorithmus, da genau bei der dezentralen Filterung und Verarbeitung der Daten das Energiesparpotential gegenüber der zentralen Variante steckt, weil nur möglichst relevante Informationen kommuniziert werden. Wie schon in Kapitel 4.1.4.2 beschrieben, gäbe es Alternativen zur hier verwendeten Aggregation der Daten, welche auf der Veränderung der Ereignisgrösse basiert. In diesem Experiment soll untersucht werden, inwiefern sich der Energieverbrauch unter Variation der Informationsdichte verändert. Es wird auch hier wiederum nicht die Qualität und Brauchbarkeit der Daten untersucht, da, wie schon in vorherigen Kapiteln erwähnt, diese sehr stark von den individuellen Bedürfnissen abhängt und somit kaum allgemeine Aussagen möglich sind. Grundsätzlich kann jedoch davon ausgegangen werden, dass im Zweifelsfall zu viele Informationen eher wünschenswert sind als zu wenige, da diese immer noch zentral rausgefiltert werden können. Die einzelnen Simulationen werden mit folgenden Parametern durchgeführt:

Simulation	minAbs	rel	maxAbs
5.3-1	0	1	1
5.3-2	25	1.5	500
5.3-3	50 (S)	2 (S)	100 (S)
5.3-4	100	4	2000
5.3-5	200	8	4000

Tabelle 5.5: Settings für die Variation der Informationsgenauigkeit
minAbs = minAbsoluteAreaDifferenceToReport
rel = relativeAreaDifferenceToReport
maxAbs = maxAbsoluteAreaDifferenceToReport
Das S in Klammern bedeutet, dass es sich hier um den in Kapitel 5.2 definierten Standardwert handelt.

Wie diese drei Parameter den dezentralen Algorithmus beeinflussen, kann in Kapitel 4.1.2.5 nachgelesen werden. Wichtig ist jedoch zu wissen, dass der zentrale Algorithmus durch diese Parameteränderung nicht beeinflusst wird.

5.3.2 Erwartungen

Da die Parameter den zentralen Algorithmus nicht beeinflussen, werden hier konstante Werte innerhalb der in Kapitel 5.2 registrierten Streuung erwartet. Er wird aber als Vergleich für die Interpretation der Ergebnisse des dezentralen Algorithmus helfen.

Bei der ersten Simulation wird davon ausgegangen, dass die durchschnittliche Belastung beim dezentralen Algorithmus über der des zentralen Algorithmus liegt, da eine gleiche Menge an Informationen an den Sink-Node gesendet wird, jedoch beim dezentralen Algorithmus noch zusätzlich die Belastung für die Erstellung und das Updaten des Graphen innerhalb des Ereignisses dazukommt. Die maximal belasteten Knoten werden bei der ersten Simulation beim dezentralen wie auch beim zentralen Algorithmus ziemlich genau der gleichen Belastung ausgesetzt sein, wobei beim dezentralen Algorithmus der Wert ein wenig höher sein dürfte, da nach einer Änderung des Head-Nodes dieselbe Information ein zweites Mal übermittelt wird. Es wird davon ausgegangen, dass mit abnehmender Informationsmenge, die an den Sink-Node gesendet wird, die durchschnittliche Belastung der Knoten beim dezentralen Algorithmus abnehmen wird. Die maximale Belastung der Knoten wird jedoch stärker abnehmen als die durchschnittliche Belastung aller Knoten, da der dezentrale Aufwand konstant bleibt, und nur die Informationsmenge, welche über den Shortest-Path zum Sink-Node gesendet wird, variiert. Es wird jedoch davon ausgegangen, dass, wenn die Informationsmenge immer weiter reduziert wird, die Energieersparnisse immer geringer werden. Dies hat sehr viel mit dem Verhältnis der Grösse einzelner Ereignisse und den verwendeten Parametern zu tun. Wird zum Beispiel die minimale absolute Grössendifferenz eines Ereignisses, welche zum Sink-Node rapportiert wird, höher angesetzt, als ein Ereignis jemals an Grösse erreicht, so wird für dieses Ereignis nur jeweils der erste Knoten, welcher das Ereignis detektiert, eine Information senden. Eine weitere Erhöhung dieses Wertes wird dann keine Energieersparnis mehr bringen, wobei ein solcher Wert in der Realität insofern keinen Sinn mehr machen würde, als dass der Informationsgewinn aus dem Geosensor Network nur noch sehr klein wäre.

5.3.3 Ergebnisse

5.3.3.1 Simulation 5.3-1: minAbs = 0, rel = 1, maxAbs = 1

Als Windrichtung wurde der Zufallswert 0.43 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.37
Osten: 0.27
Süden: 0.53
Westen: 0.63

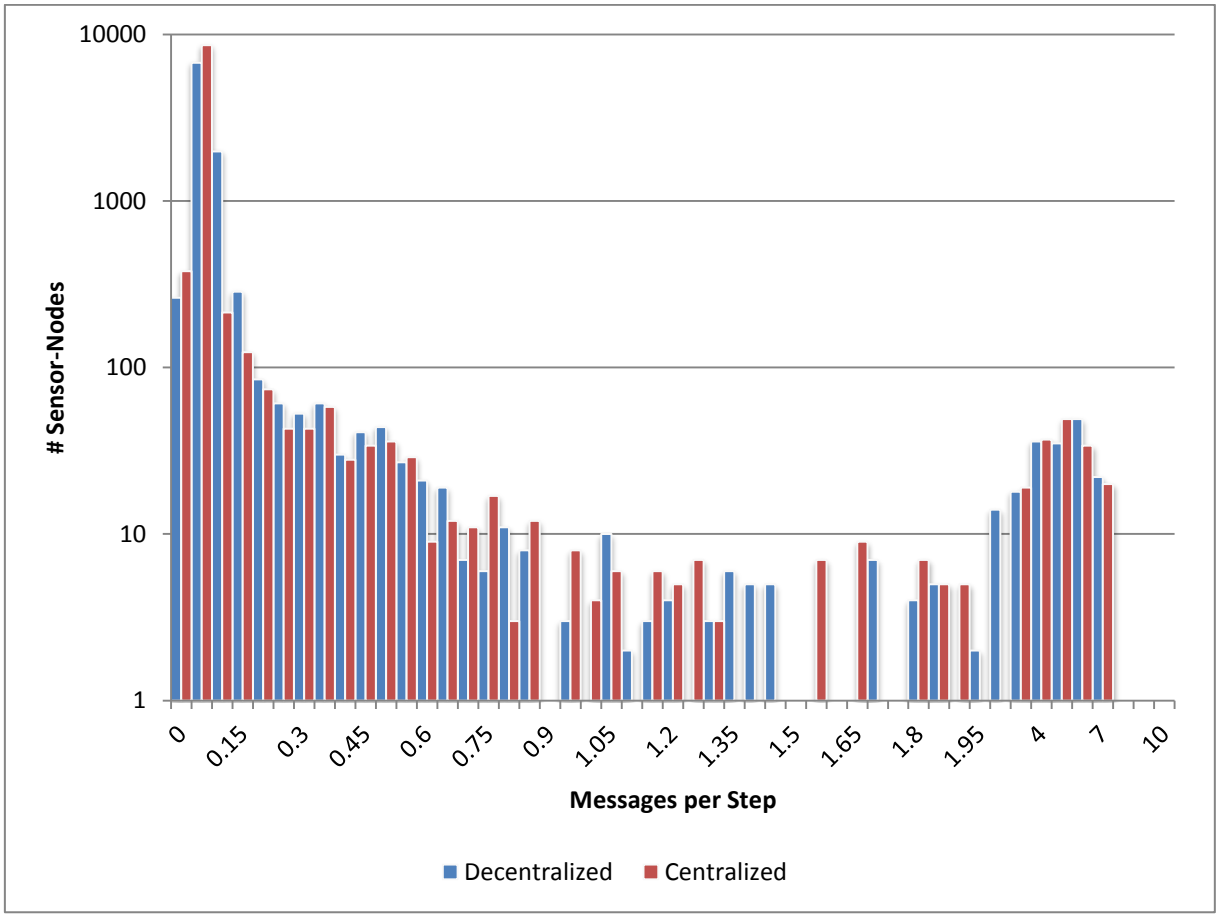


Diagramm 5.4: Simulation 5.3-1
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.134	0.105	1.281
Median	0.029	0.001	22.423
Std Dev	0.62	0.603	1.029
Max	6.875	6.752	1.018

Tabelle 5.6: Simulation 5.3-1
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.3.3.2 Simulation 5.3-2: minAbs = 25, rel = 1.5, maxAbs = 500

Als Windrichtung wurde der Zufallswert 3.61 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.54
Osten: 0.63
Süden: 0.34
Westen: 0.27

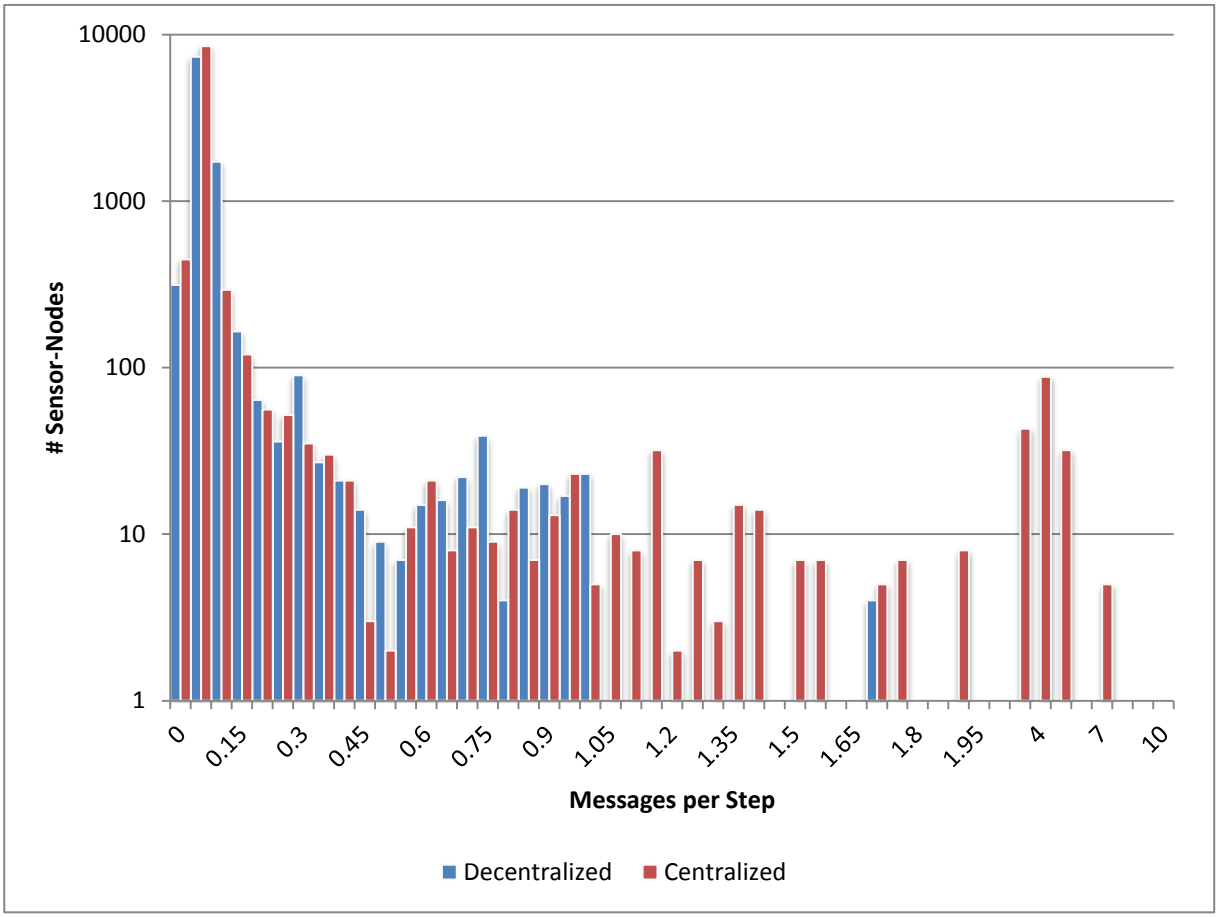


Diagramm 5.5: Simulation 5.3-2
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.05	0.097	0.519
Median	0.025	0.001	19.269
Std Dev	0.118	0.491	0.24
Max	1.718	7.018	0.245

Tabelle 5.7: Simulation 5.3-2
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.3.3.3 Simulation 5.3-3: minAbs = 50, rel = 2, maxAbs = 1000

Als Windrichtung wurde der Zufallswert 5.84 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.54
Osten: 0.27
Süden: 0.36
Westen: 0.63

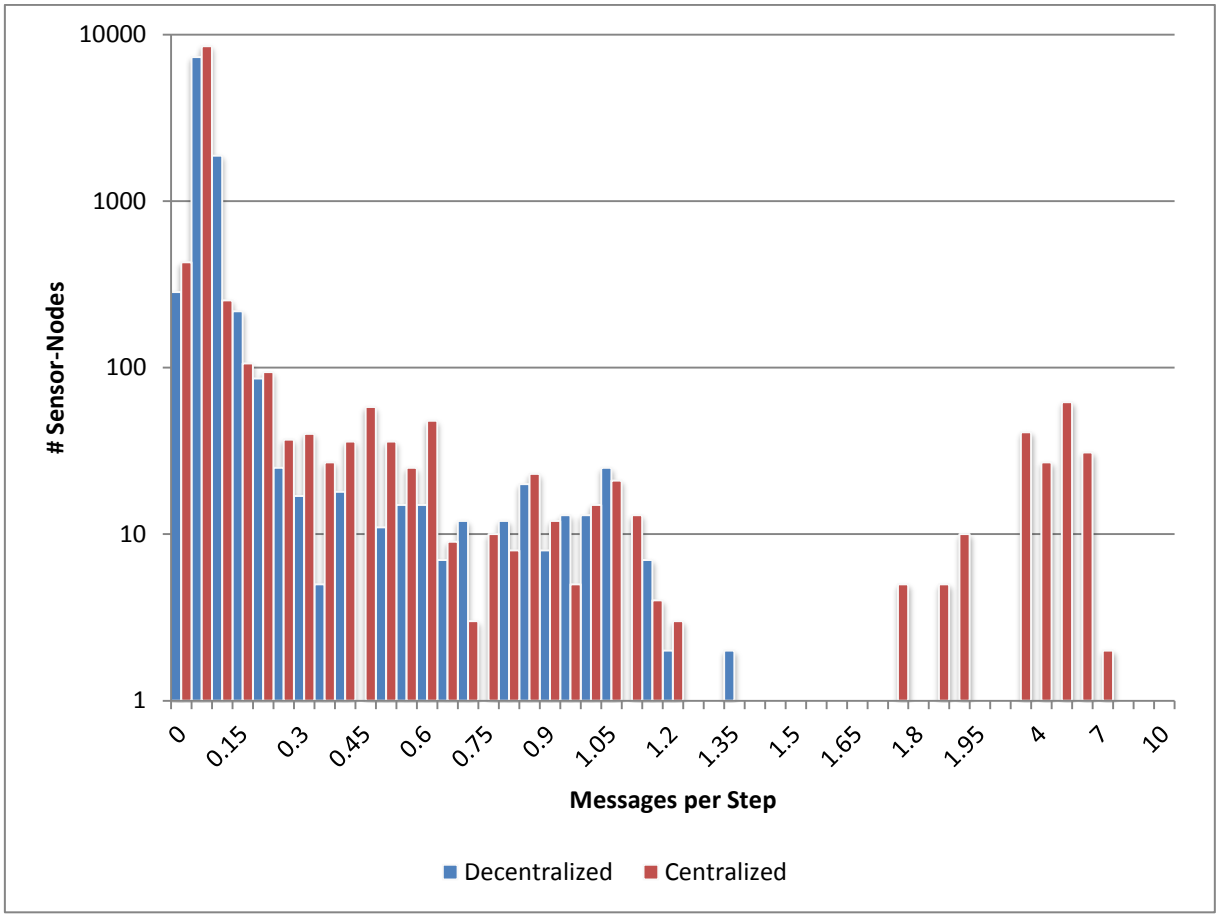


Diagramm 5.6: Simulation 5.3-3
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.046	0.102	0.45
Median	0.026	0.001	20.4
Std Dev	0.107	0.548	0.196
Max	1.314	6.912	0.19

Tabelle 5.8: Simulation 5.3-3
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.3.3.4 Simulation 5.3-4: minAbs = 100, rel = 4, maxAbs = 2000

Als Windrichtung wurde der Zufallswert 3.42 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.5
Osten: 0.64
Süden: 0.4
Westen: 0.26

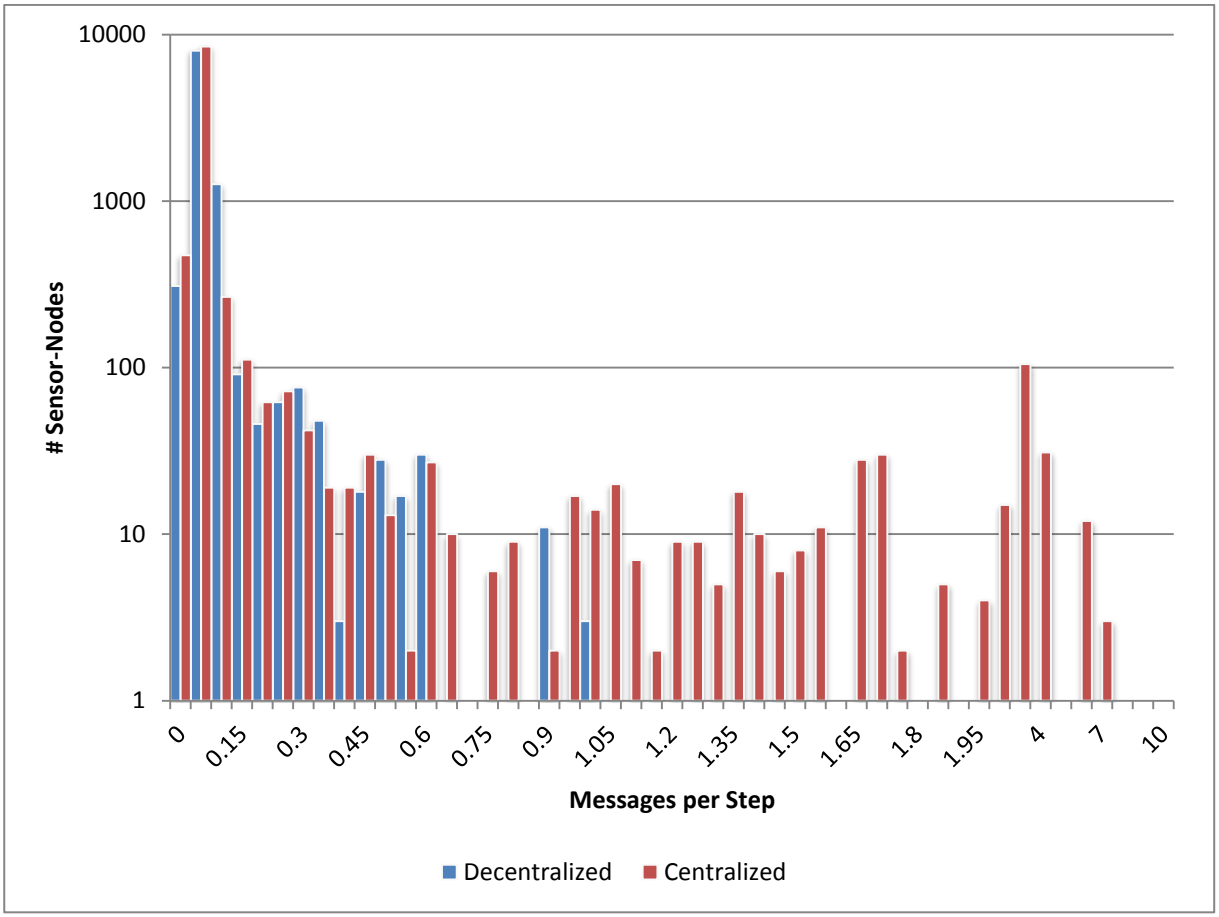


Diagramm 5.7: Simulation 5.3-4
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.037	0.092	0.399
Median	0.022	0.001	19.864
Std Dev	0.07	0.431	0.163
Max	0.968	6.312	0.153

Tabelle 5.9: Simulation 5.3-4
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.3.3.5 Simulation 5.3-5: minAbs = 200, rel = 8, maxAbs = 4000

Als Windrichtung wurde der Zufallswert 2.29 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.3
Osten: 0.58
Süden: 0.6
Westen: 0.32

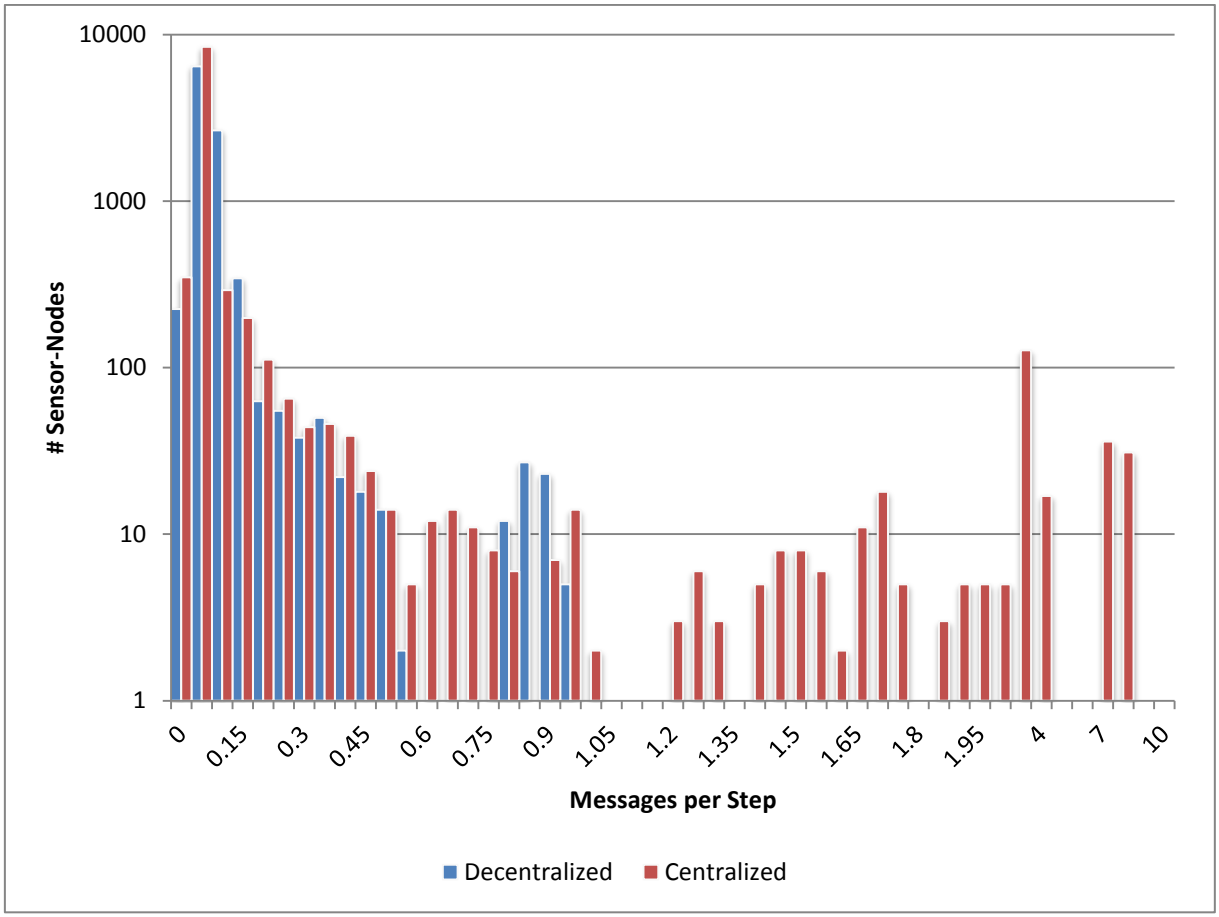
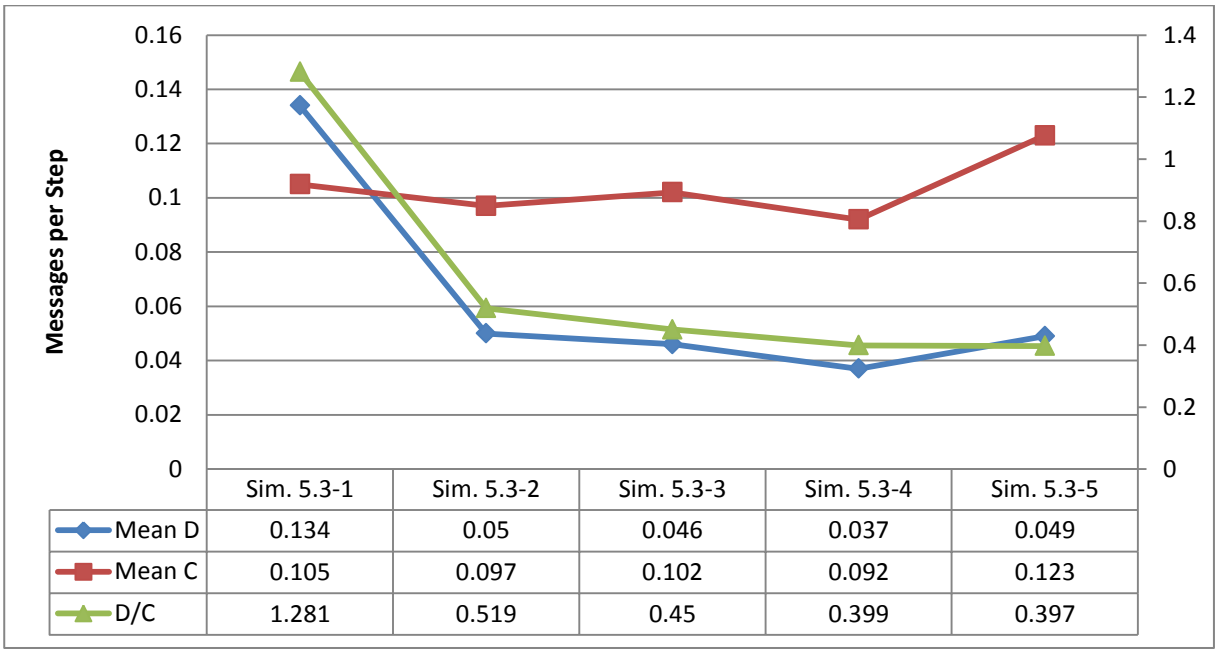


Diagramm 5.8: Simulation 5.3-5
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.049	0.123	0.397
Median	0.034	0.002	22.333
Std Dev	0.083	0.673	0.123
Max	0.915	7.765	0.118

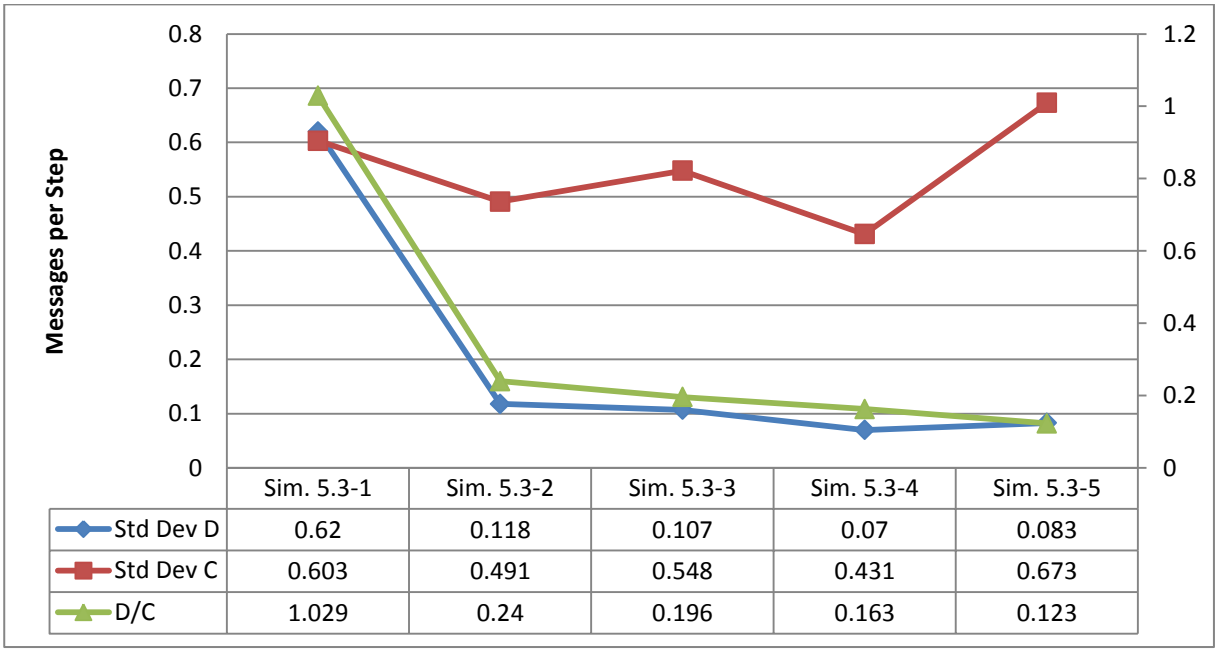
Tabelle 5.10: Simulation 5.3-5
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.3.3.6 Vergleich Simulation 5.3-1 bis Simulation 5.3-5



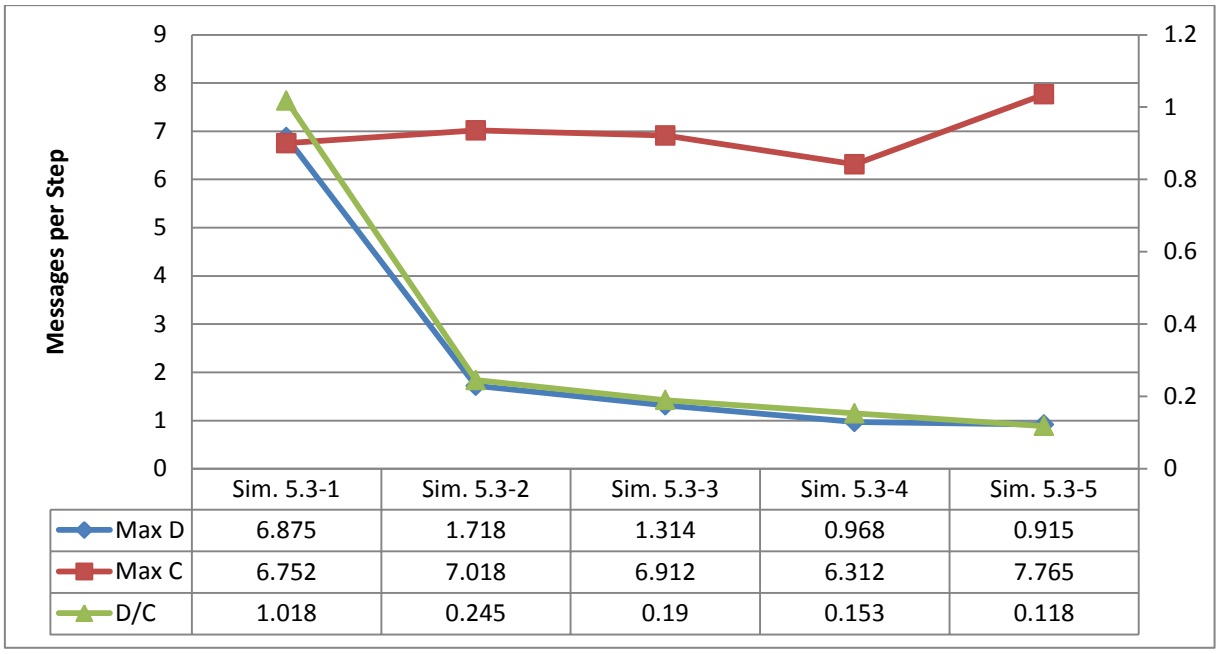
	Sim. 5.3-1	Sim. 5.3-2	Sim. 5.3-3	Sim. 5.3-4	Sim. 5.3-5
Mean D	0.134	0.05	0.046	0.037	0.049
Mean C	0.105	0.097	0.102	0.092	0.123
D/C	1.281	0.519	0.45	0.399	0.397

Diagramm 5.9: Durchschnittsverbrauch der Simulationen 5.3-1 bis 5.3-5 im Vergleich
Die Abkürzung D steht für den dezentralen Algorithmus, C für den zentralen; Auf der x-Achse ist die Nummer der Simulation angegeben. Von links nach rechts ist somit die Informationsdichte beim dezentralen Algorithmus abnehmend. Die blaue und rote Kurve sind in absoluten Werten angegeben (linke y-Achse), die grüne Kurve zeigt relative Werte (rechte y-Achse).



	Sim. 5.3-1	Sim. 5.3-2	Sim. 5.3-3	Sim. 5.3-4	Sim. 5.3-5
Std Dev D	0.62	0.118	0.107	0.07	0.083
Std Dev C	0.603	0.491	0.548	0.431	0.673
D/C	1.029	0.24	0.196	0.163	0.123

Diagramm 5.10: Standardabweichung der Simulationen 5.3-1 bis 5.3-5 im Vergleich
Die Abkürzung D steht für den dezentralen Algorithmus, C für den zentralen. Auf der x-Achse ist die Nummer der Simulation angegeben. Von links nach rechts ist somit die Informationsdichte beim dezentralen Algorithmus abnehmend. Die blaue und rote Kurve sind in absoluten Werten angegeben (linke y-Achse), die grüne Kurve zeigt relative Werte (rechte y-Achse).



	Sim. 5.3-1	Sim. 5.3-2	Sim. 5.3-3	Sim. 5.3-4	Sim. 5.3-5
Max D	6.875	1.718	1.314	0.968	0.915
Max C	6.752	7.018	6.912	6.312	7.765
D/C	1.018	0.245	0.19	0.153	0.118

Diagramm 5.11: Maximal belastete Knoten der Simulationen 5.3-1 bis 5.3-5 im Vergleich
Die Abkürzung D steht für den dezentralen Algorithmus, C für den zentralen. Auf der x-Achse ist die Nummer der Simulation angegeben. Von links nach rechts ist somit die Informationsdichte beim dezentralen Algorithmus abnehmend. Die blaue und rote Kurve sind in absoluten Werten angegeben (linke y-Achse), die grüne Kurve zeigt relative Werte (rechte y-Achse).

5.3.4 Diskussion

Die Ergebnisse entsprechen weitgehend den Erwartungen. Bei der Simulation 5.3-1, bei welcher auch beim dezentralen Algorithmus alle Informationen an den Sink-Node weitergeleitet werden, verbraucht der dezentrale Algorithmus im Durchschnitt ca. 28% mehr Energie als der zentrale Algorithmus. Diese 28% entsprechen somit ziemlich genau dem dezentralen Aufwand. Somit lässt sich auch ableiten, dass bei vollkommener Information der dezentrale Aufwand bei der gegebenen Netzwerks- und Untersuchungsgebietsgrösse rund einen Viertel des gesamten Aufwandes in Anspruch nimmt und dementsprechend drei Viertel der Aufwendungen für die Informationsübermittlung an den Sink-Node verwendet werden. Sehr interessant ist, dass sich der durchschnittliche Energieverbrauch von Simulation 5.3-1 zu Simulation 5.3-2 gegenüber dem zentralen Algorithmus von 128.1% auf 51.9% reduziert. In Kapitel 4.1.2.5 wird ein konkretes Beispiel mit denselben Werten von Simulation 5.3-2 beschrieben. Interessant ist dies vor allem dadurch, als dass bei dieser Informationsdichte noch nicht mit einer solch drastischen Reduktion des Energieverbrauchs gerechnet wurde. Wirft man aber einen Blick auf die Knoten mit dem maximalen Verbrauch, so zeigt sich, dass von Simulation 5.3-1 zu Simulation 5.3-2 der relative Verbrauch des dezentralen Algorithmus zum zentralen Algorithmus von 101.8% auf 24.5% abgenommen hat. Die 101.8% bestätigen die Vermutung, dass bei gleicher Informationsmenge der dezentrale Algorithmus ein bisschen schlechter abschneiden wird, da Informationen bei einer Änderung des Head-Nodes redundant übermittelt werden, was in dem Fall einem Mehraufwand von 1.8% entspricht. Aus den beiden Werten 101.8% und 24.5% kann aber die Schlussfolgerung gezogen werden, dass schon eine kleine Reduktion der Informationsmenge zu einer massiven Einsparung an Energie führen kann. Auch eine starke Reduktion der Standardabweichung zeigt an, dass die Belastung durch eine Informationsreduktion zu einer massiv besseren Belastungsverteilung führt, was auch die Vermutung bestätigt, dass die maximal belasteten Knoten durch diese Minderbelastung stärker entlastet werden als die durchschnittliche Belastung über alle Knoten gesehen.

Der weitere Verlauf der Kurven zeigt, dass zwar durch noch weiteres Reduzieren der Informationen noch geringfügig Energie eingespart werden kann, jedoch der Unterschied sehr gering wird. Von Simulation 5.3-2 zu Simulation 5.3-5 kann beim durchschnittlichen Verbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus gerade einmal von 51.9% auf 39.7% und der maximal belastete Knoten von 24.5% auf 11.8% reduziert werden. Gerade eine Reduktion von 24.5% auf 11.8% beim maximal belasteten Knoten lässt zwar in etwa eine Verdoppelung der Netzwerklebensdauer zu, ist jedoch weiter mit einem massiven Verlust an Informationen verbunden. In der Realität müsste wohl individuell auf das Problem zugeschnitten eine gute Balance zwischen Informationsdichte und Energieverbrauch gefunden werden. Weiter soll jedoch hier schon auf Kapitel 5.5 verwiesen werden, in welchem die Algorithmen auf eine Veränderung der Sensordichte untersucht werden, welche schlussendlich auch in einem Zusammenhang mit der Informationsdichte steht.

5.4 Variation der Grösse des Untersuchungsgebiets und des Netzwerks

5.4.1 Experiment

Bei folgendem Experiment wird die Grösse des Untersuchungsgebietes sowie des Netzwerkes variiert. Die Sensordichte wird jeweils konstant gehalten und somit die Grösse des Untersuchungsgebietes proportional zur Anzahl der Sensorknoten verändert. Mit Hilfe dieser Untersuchung soll herausgefunden werden, wie die beiden Algorithmen mit der Grösse des Netzwerks skalieren. Bei der Implementierung von Algorithmen ist die Skalierbarkeit einer der wichtigsten Faktoren, was auch jeweils eine der grössten Herausforderungen darstellt. Für die meisten Probleme ist es sehr einfach, einen Algorithmus zu erstellen, der das Problem lösen kann. Zumindest bei kleinen Datensätzen oder, in unserem Fall, mit wenig Sensoren. Doch was passiert, wenn sich die Datenmenge oder eben die Netzwerkgrösse verzehnfacht? Hat eine Methode einen quadratischen Aufwand $O(n^2)$, verhundertfacht sich der Aufwand. Bald einmal ist die Grenze erreicht, wo die Hardware, sei es die Prozessorleistung oder die Batteriekapazität einzelner Sensorknoten in einem Geosensor Network an Grenzen stossen, was eine weitere Vergrösserung des Netzwerkes nicht mehr zulässt (vgl. Kapitel 2.1). Schafft man es aber zum Beispiel, einen Algorithmus zu implementieren, welcher einen logarithmischen Aufwand $O(\log(n))$ oder sogar einen konstanten Aufwand $O(1)$ hat, so kann ein Netzwerk fast nach Belieben vergrössert werden, ohne einzelne Knoten kritisch zu belasten.

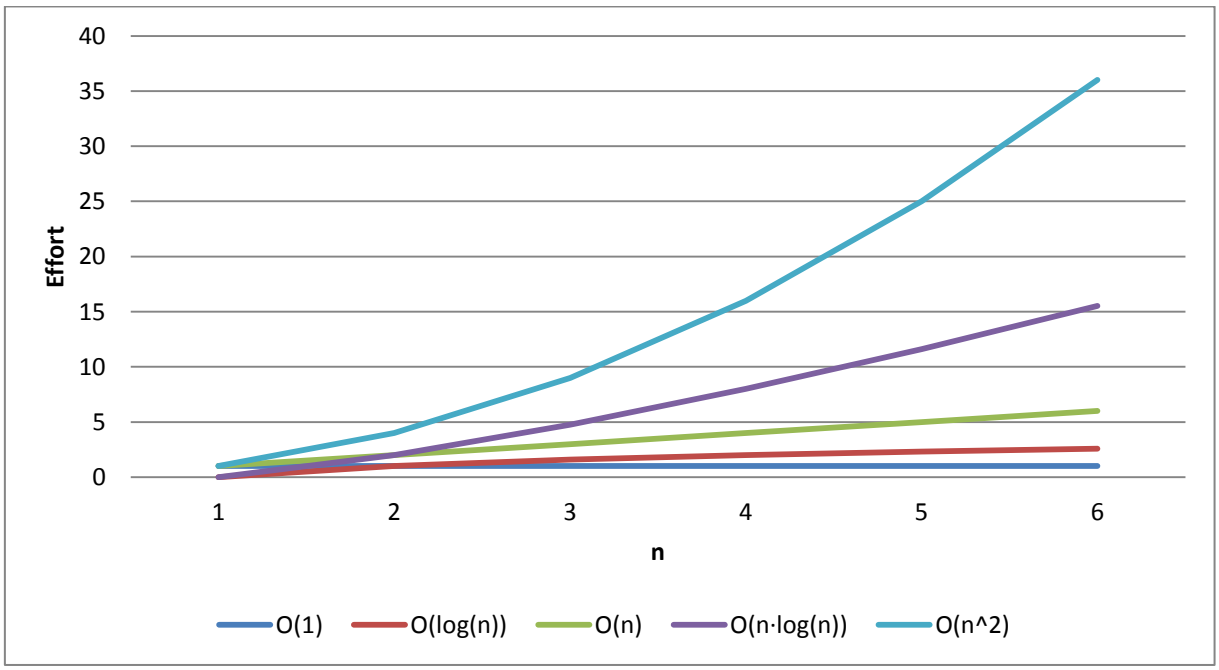


Abbildung 5.2: Aufwand verschiedener Algorithmen
Die Aufwände der verschiedenen Algorithmen (blau, rot, grün, violett, hellblau) nehmen von links nach rechts zu. Bei den Logarithmen wurde jeweils mit Basis 2 gerechnet.

Bei normalen Problemen, welche auf einem einzigen Prozessor gerechnet werden, bezieht sich dieser Aufwand normalerweise auf die Datenmenge, welche der Prozessor zu verarbeiten hat. In einem Geosensor Network kann dieser Aufwand jedoch unter vielen verschiedenen Gesichtspunkten betrachtet werden. So können die durchschnittliche Belastung auf die einzelnen Sensoren, oder aber auch die Veränderung der maximal belasteten Sensoren sowie weitere Indikatoren in die Betrachtung mit einbezogen werden. Die Durchschnittsbelastung sowie vor allem die

Maximalbelastung der Knoten dürften von grossem Interesse sein, da gerade die Maximalbelastung als kritische Grösse für die Gesamtlebensdauer eines Netzwerkes betrachtet werden kann.
Es werden Simulationen mit folgenden Parametern durchgeführt:

Simulation	xDim	yDim	initialNumberOfSensorNodes
5.4-1	250	100 (S)	1250
5.4-2	500	100 (S)	2500
5.4-3	1000	100 (S)	5000
5.4-4	2000 (S)	100 (S)	10000 (S)
5.4-5	4000	100 (S)	20000

Tabelle 5.11: Settings für die Variation der Grösse des Untersuchungsgebiets und des Geosensor Network
Das S in Klammern bedeutet, dass es sich hier um den in Kapitel 5.2 definierten Standardwert handelt.

5.4.2 Erwartungen

In diesem Kapitel werden die Erwartungen sowie die Diskussion für die beiden Algorithmen in einzelne Unterkapitel unterteilt, da die Veränderung der Parameter die beiden Algorithmen sehr unterschiedlich beeinflusst.

5.4.2.1 Dezentraler Algorithmus

In diesem Experiment wird erwartet, dass beim dezentralen Algorithmus die durchschnittliche Belastung aller Sensorknoten mit der Grösse des Netzwerkes und des Gebietes abnimmt, jedoch nicht ganz linear. Das hat den Grund, dass pro Knoten und Zeiteinheit bei einem doppelt so grossen Gebiet nur etwa die Hälfte der Ereignisse zu erwarten sind, da die Anzahl Ereignisse in etwa konstant bleibt, diese sich jedoch auf eine grössere Fläche verteilen. Das „etwa“ bezieht sich wiederum darauf, dass nicht genau vorhergesagt werden kann, wie sich die Ereignisse verhalten werden, was bei zellulären Automaten oft der Fall ist, wie in Kapitel 3.3 beschrieben. Ausserdem wurde auch schon in Kapitel 5.2 gezeigt, dass sogar beim gleichen Setting aus diesem Grund der Gesamtenergieverbrauch um rund 30% streuen kann. Aus zwei Gründen ist davon auszugehen, dass sich die Menge der von einem Ereignis betroffenen Knoten in der Tendenz auch ohne diese Streuung nicht ganz halbieren wird: Erstens haben die Ereignisse auch eine grössere Fläche, um sich auszubreiten, während sie in einem kleineren Gebiet schneller an die Gebietsgrenze stossen und verschwinden (am Waldrand kann sich das Feuer nicht mehr weiter ausbreiten). Dadurch wird das gleiche Ereignis tendenziell während seiner Lebensdauer von einer grösseren Anzahl Knoten detektiert. Zum Zweiten ist die Chance, dass ein neues Ereignis nicht entstehen kann, da sich schon ein Ereignis dort befindet, kleiner (ein Feuer, das schon brennt kann nicht mehr neu entstehen). Dass die Wege zum Sink-Node grösser werden, wird höchstens einen marginalen Einfluss auf den Energieverbrauch haben, da beim dezentralen Algorithmus die Informationsübermittlung zum Sink-Node einen verhältnismässig kleinen Einfluss auf den Gesamtenergieverbrauch hat. Auch bedeuten weitere Wege zum Sink-Node, dass die Belastung auf mehr Knoten verteilt wird, da automatisch die Gesamtanzahl der Knoten bei gleicher Sensordichte steigt. Der dezentrale Algorithmus dürfte sich aus genannten Gründen insgesamt in etwa parallel zur relativen (zur Gesamtuntersuchungsgebietsfläche) Ereignisfläche verändern.
Bei den am stärksten belasteten Knoten wird davon ausgegangen, dass der Energieverbrauch leicht zunehmen wird, da mit etwa der gleichen Anzahl Ereignisse gerechnet wird, jedoch aus den eben genannten Gründen mit einer leichten absoluten Steigerung der Ereignisgrösse zu rechnen ist. Dies wird zu einer leichten Mehrbelastung der wohl am meisten belasteten Sensor-Knoten in der Nähe des Sink-Nodes führen.

5.4.2.2 Zentraler Algorithmus

Beim zentralen Algorithmus ist davon auszugehen, dass die durchschnittliche Belastung der einzelnen Knoten bei einer Verdoppelung der Gebietsgrösse geringfügig zunehmen wird. Wäre eine gleich grosse Fläche von Waldbränden betroffen, könnte man von einer gleichbleibenden durchschnittlichen Belastung ausgehen, da sich der Energieverbrauch nur auf die Informationsübermittlung an den Sink-Node beschränkt. Bei einem doppelt so breiten Gebiet verdoppelt sich zwar die durchschnittliche Länge des Weges zum Sink-Node, jedoch verteilt sich die Belastung auch auf die doppelte Anzahl Knoten, weshalb bei gleichbleibender Ereignisgrösse von einem gleichbleibenden durchschnittlichen Verbrauch ausgegangen werden kann. Weshalb trotzdem mit einem leicht zunehmenden Durchschnittsenergieverbrauch gerechnet wird, liegt wiederum darin begründet, dass bei gleicher Anzahl aufkommender Ereignisse wohl aus schon in Kapitel 5.4.2.1 genannten Gründen die absolute Ereignis-Fläche zunehmen wird. Ginge man von einer gleich bleibenden relativ von einem Feuer betroffenen Fläche aus, würde somit der Durchschnittsverbrauch parallel zur betroffenen Fläche unabhängig von der Gesamtfläche des Untersuchungsgebiets steigen. Es kann also von einem linearen Zusammenhang zwischen der absoluten Ereignis-Fläche und dem durchschnittlichen Energieverbrauch ausgegangen werden.
Bei den maximal belasteten Knoten ist zu erwarten, dass deren Belastung leicht zunehmen wird. Dies wird jedoch auch ausschliesslich davon abhängen, inwiefern sich die absolute durchschnittliche Fläche, welche von einem Ereignis betroffen ist, verändert. Da hier mit einer Steigerung zu rechnen ist, wird auch mit einer parallelen Steigerung der Belastung der schon am meist belasteten Sensorknoten gerechnet.

5.4.2.3 Vergleich dezentraler und zentraler Algorithmus

Es wird erwartet, dass bei der durchschnittlichen Belastung der Knoten mit zunehmender Grösse des Geosensor Network der dezentrale Algorithmus gegenüber der zentralen Variante immer bessere Ergebnisse liefern wird, da voraussichtlich beim dezentralen Algorithmus die Performance sich parallel zur relativen Ereignisgrösse, beim zentralen jedoch zur absoluten Ereignisgrösse verändert. Es wäre also zu erwarten, dass der Aufwand des dezentralen Algorithmus etwa konstant bleibt, während die zentrale Variante ungefähr den doppelten Energieverbrauch haben dürfte, wenn die Grösse des Untersuchungsgebiets sich verdoppelt und sich die von Waldbrand betroffene Gesamtfläche auch verdoppelt (die relative Fläche somit gleich bleibt).
Bei den maximal beanspruchten Knoten dürften beide Algorithmen mit zunehmender Gebiets- und somit tendenziell zunehmender Ereignisgrösse ihre Performance parallel verschlechtern. Da jedoch beim dezentralen Algorithmus diese Grundbelastung grundsätzlich tiefer ist und sich die Belastung relativ zur Fläche ändern dürfte, wird die absolute Differenz zwischen den beiden Algorithmen mit wachsendem Untersuchungsgebiet auch zunehmen. Auch wird die Mehrbelastung beim dezentralen Algorithmus dadurch abgebremst, dass bei sehr grossen Ereignissen eine kleinere Datenmenge an den Sink-Node weitergeleitet werden muss (vgl. Kapitel 4.1.2.5).

5.4.3 Ergebnisse

5.4.3.1 Simulation 5.4-1: Umweltgrösse 250m x 100m; 1250 Sensorknoten

Als Windrichtung wurde der Zufallswert 2.09 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.28
Osten: 0.55
Süden: 0.62
Westen: 0.35

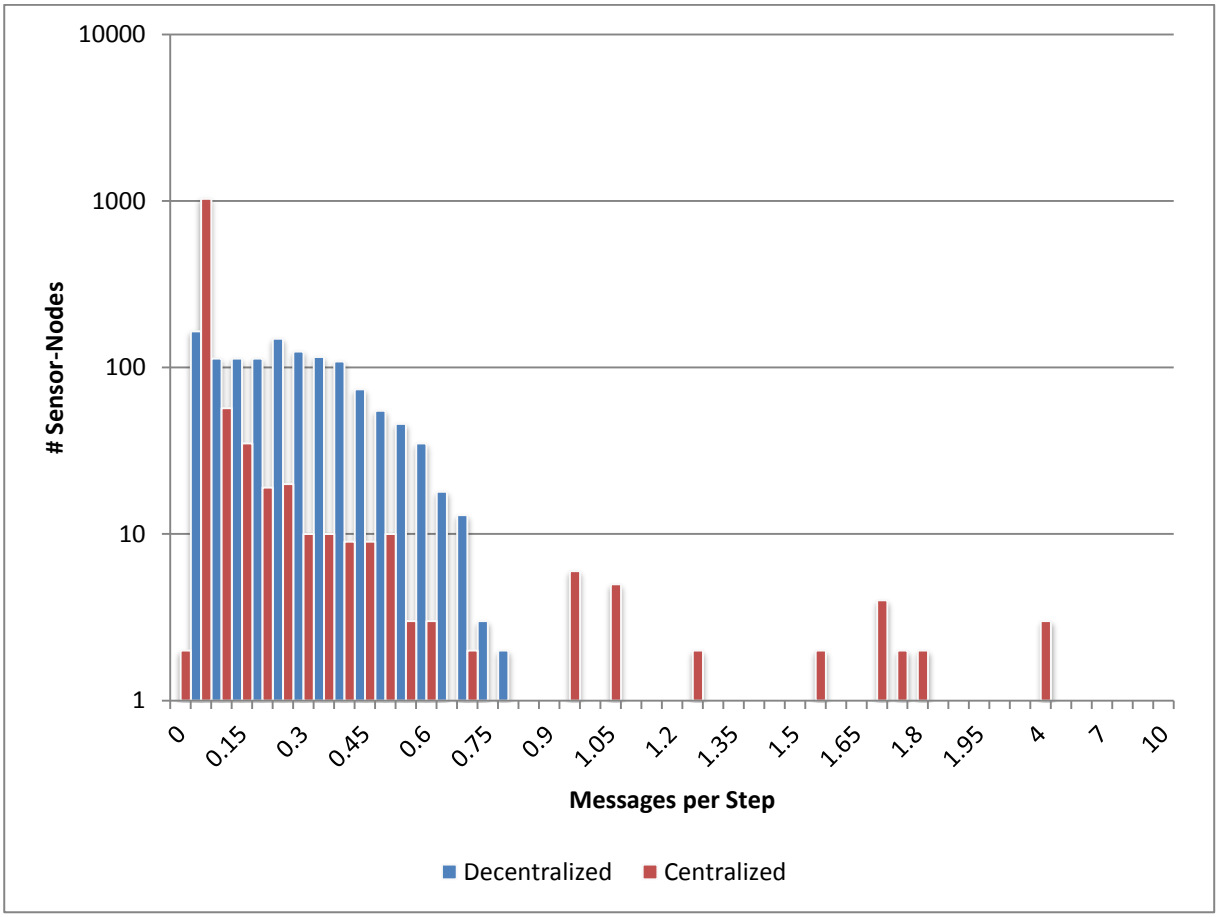


Diagramm 5.12: Simulation 5.4-1
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.256	0.077	3.343
Median	0.242	0.005	45.15
Std Dev	0.168	0.278	0.605
Max	0.839	3.761	0.223

Tabelle 5.12: Simulation 5.4-1
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.4.3.2 Simulation 5.4-2: Umweltgrösse 500m x 100m; 2500 Sensorknoten

Als Windrichtung wurde der Zufallswert 2.64 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.35
Osten: 0.63
Süden: 0.55
Westen: 0.27

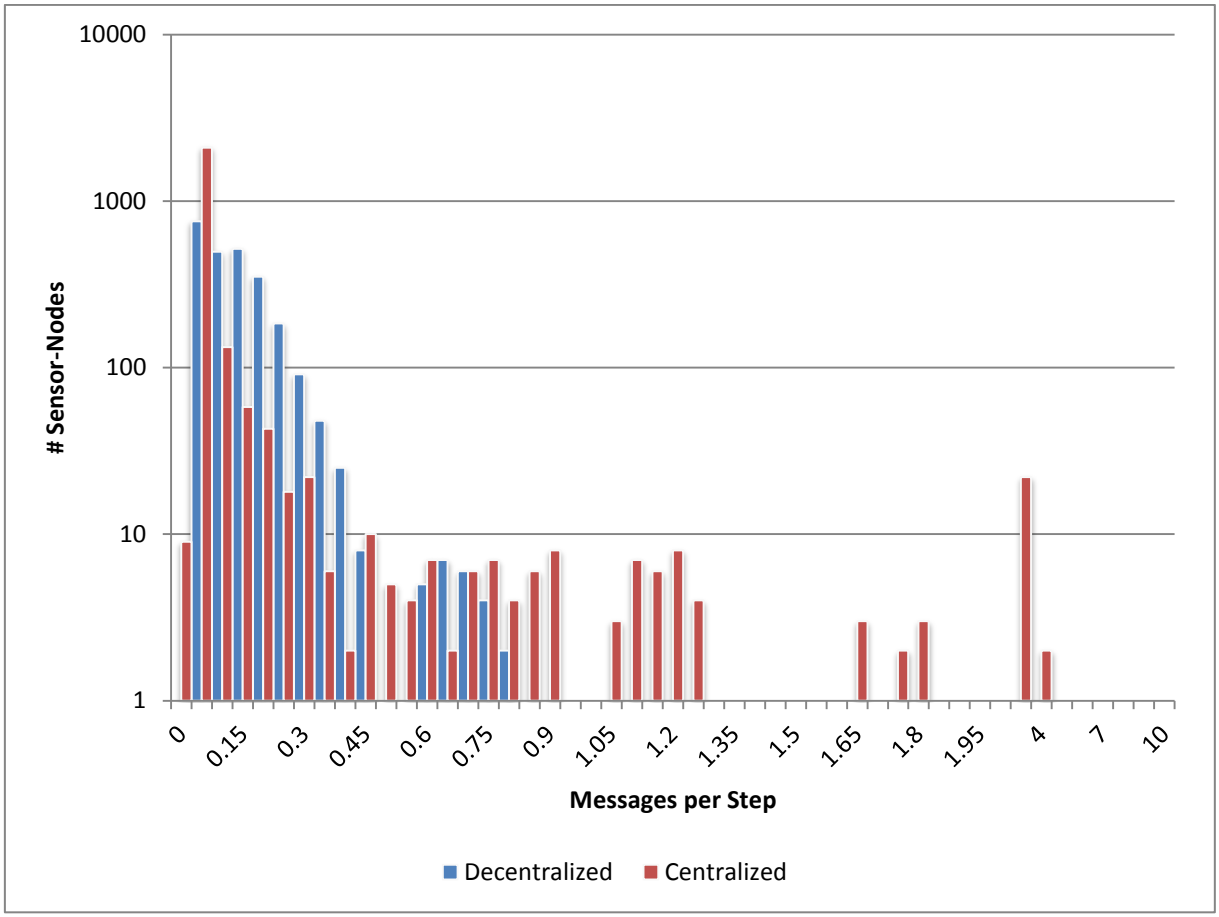


Diagramm 5.13: Simulation 5.4-2
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.116	0.08	1.444
Median	0.01	0.004	28.014
Std Dev	0.102	0.317	0.322
Max	1.04	5.233	0.199

Tabelle 5.13: Simulation 5.4-2
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.4.3.3 Simulation 5.3-3: Umweltgrösse 1000m x 100m; 5000 Sensorknoten

Als Windrichtung wurde der Zufallswert 2.86 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.39
Osten: 0.64
Süden: 0.51
Westen: 0.26

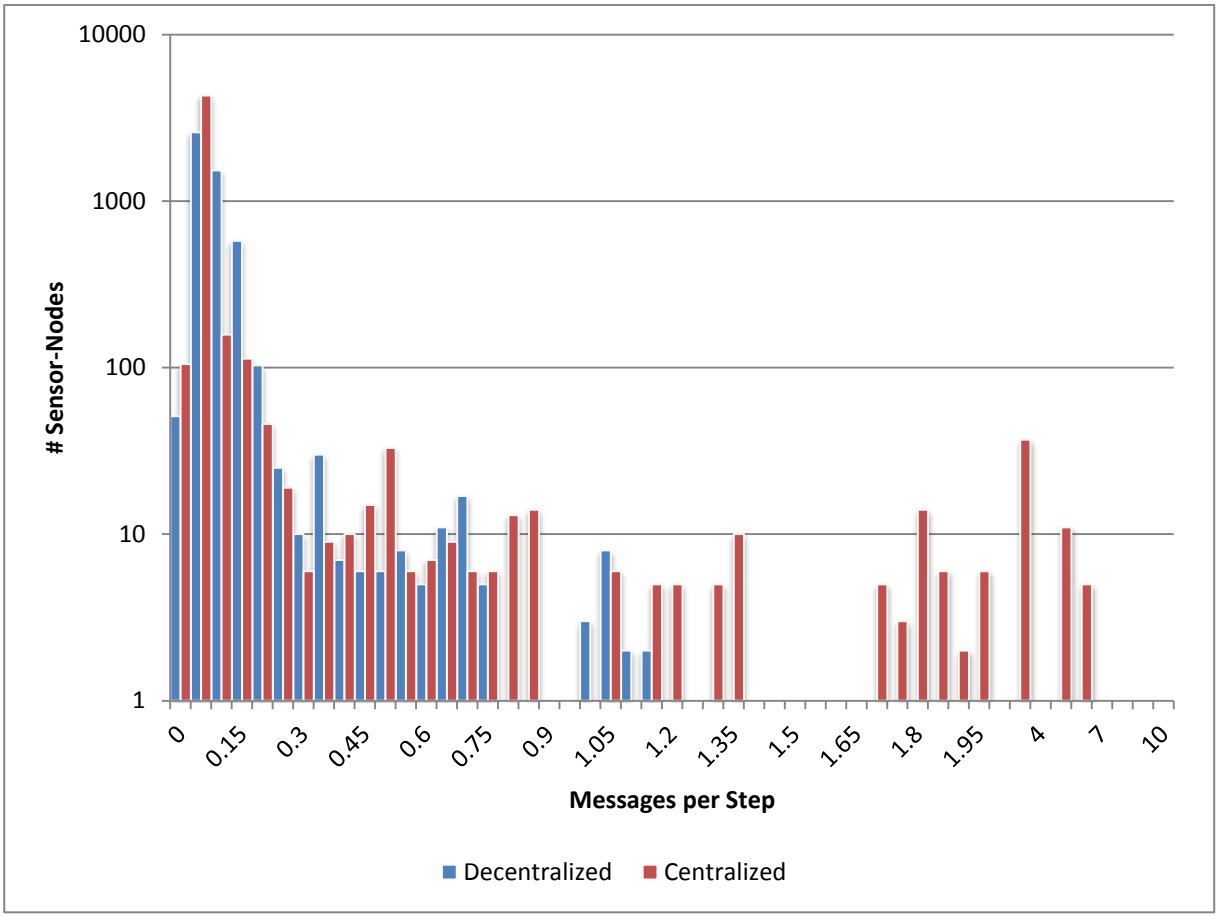


Diagramm 5.14: Simulation 5.4-3
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.065	0.082	0.79
Median	0.047	0.002	22.19
Std Dev	0.095	0.403	0.235
Max	1.153	5.626	0.205

Diagramm 5.15: Simulation 5.4-3
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.4.3.4 Simulation 5.4-4: Umweltgrösse 2000m x 100m; 10000 Sensorknoten

Als Windrichtung wurde der Zufallswert 6.01 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.5
Osten: 0.26
Süden: 0.4
Westen: 0.64

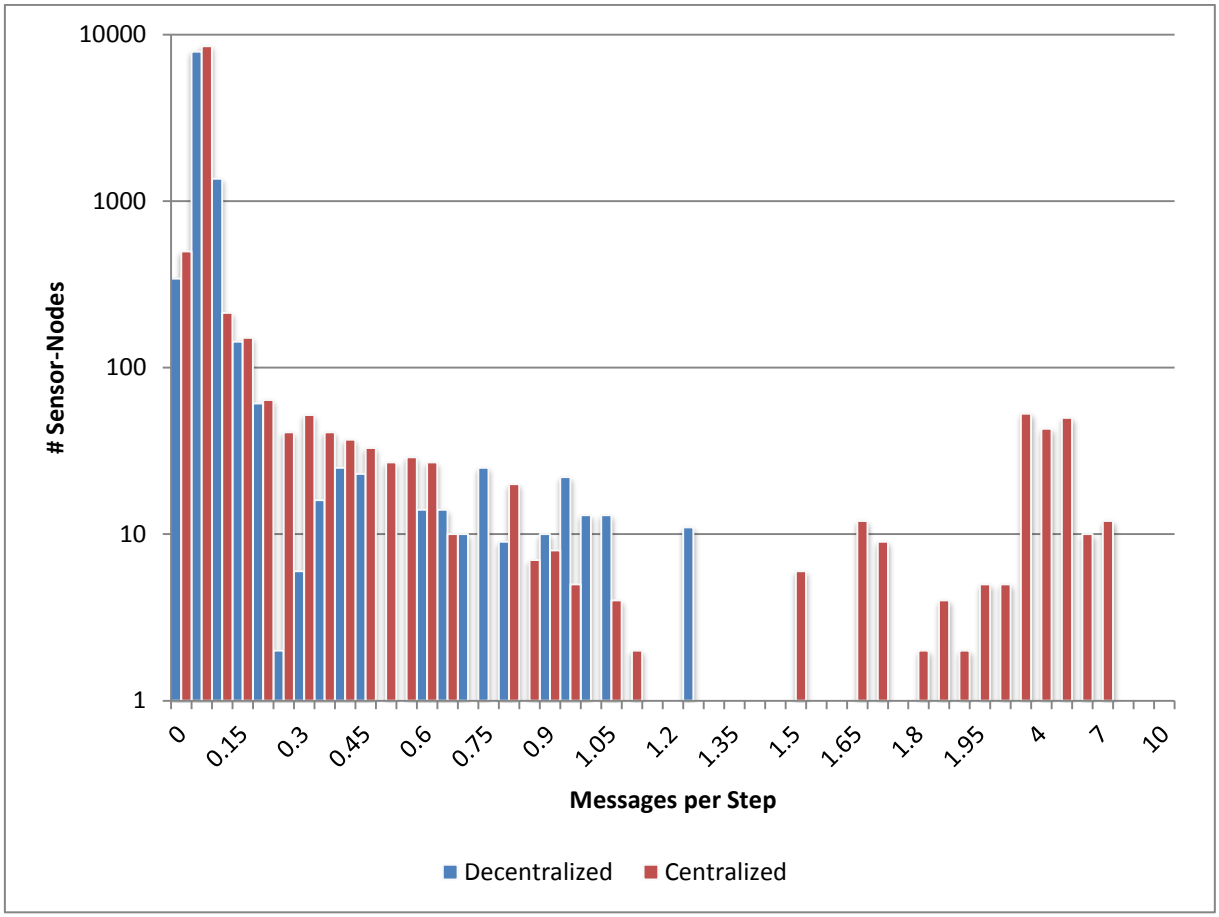


Diagramm 5.16: Simulation 5.4-4
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.042	0.093	0.448
Median	0.022	0.001	18.25
Std Dev	0.106	0.517	0.205
Max	1.247	6.166	0.202

Tabelle 5.14: Simulation 5.4-4
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.4.3.5 Simulation 5.4-5: Umweltgrösse 4000m x 100m; 20000 Sensorknoten

Als Windrichtung wurde der Zufallswert 0.31 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.39
Osten: 0.26
Süden: 0.51
Westen: 0.64

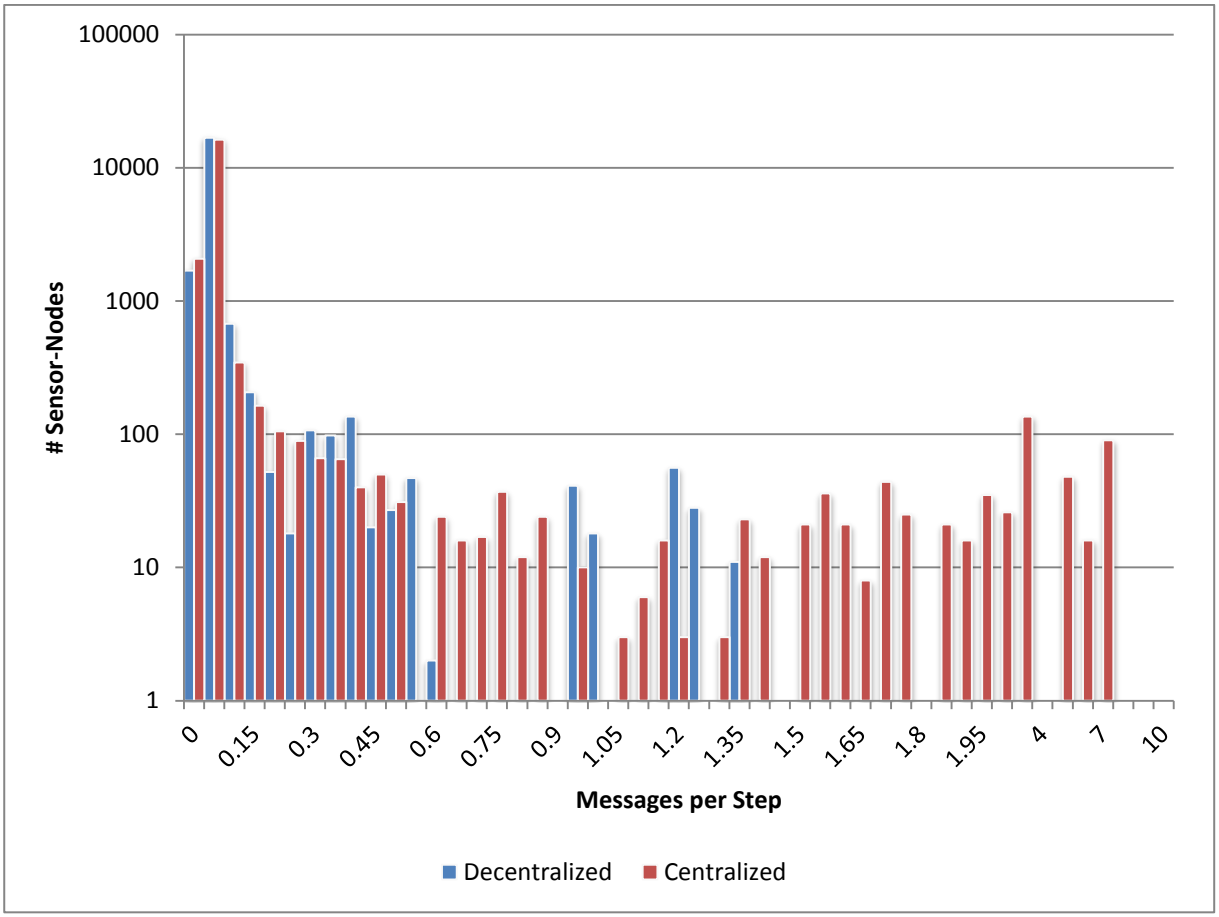
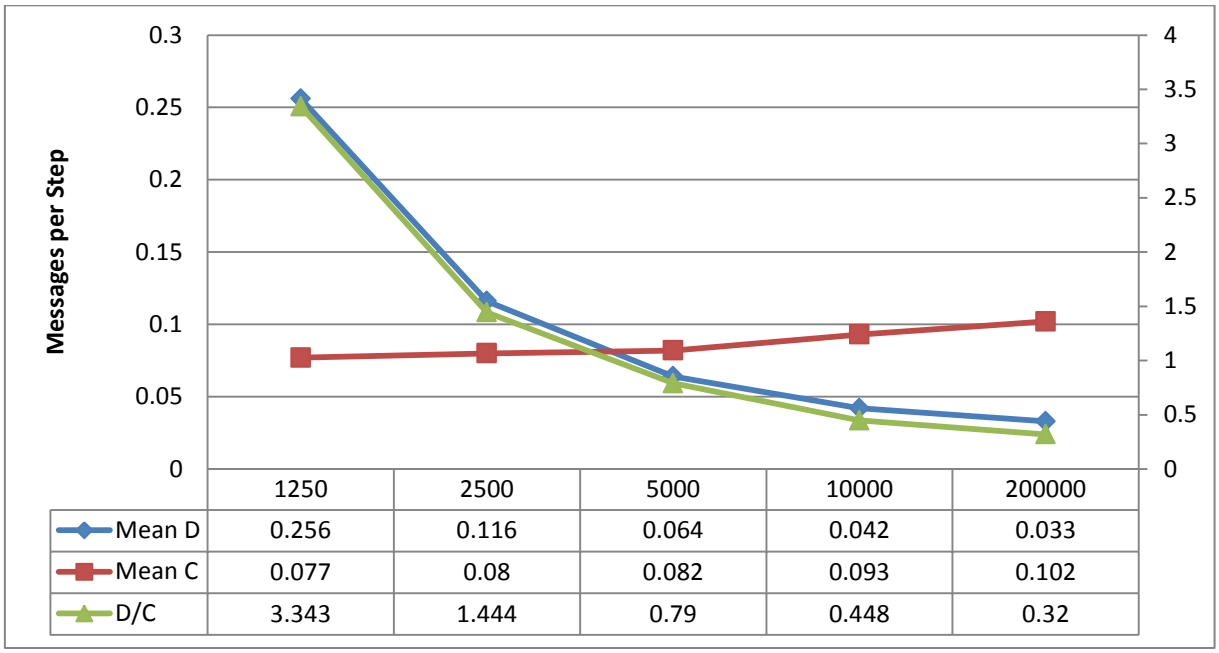


Diagramm 5.17: Simulation 5.4-5
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.033	0.102	0.32
Median	0.011	0.001	17.385
Std Dev	0.111	0.578	0.191
Max	1.342	6.947	0.193

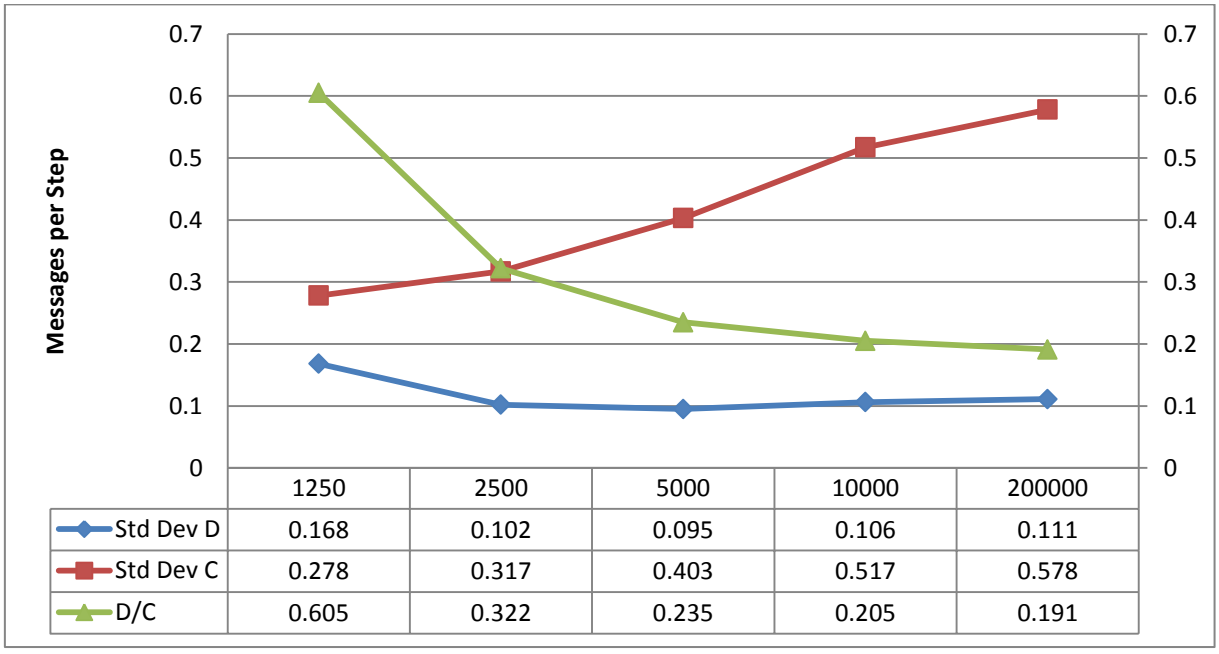
Tabelle 5.15: Simulation 5.4-5
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.4.3.6 Vergleich Simulation 5.4-1 bis Simulation 5.4-5



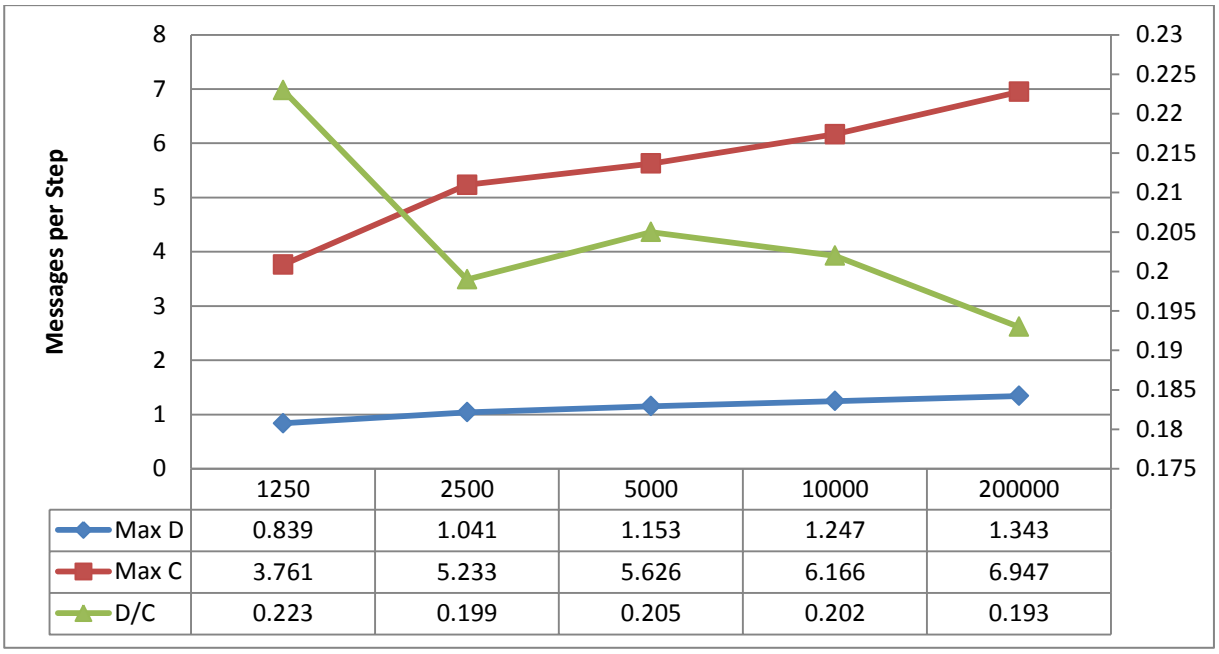
	1250	2500	5000	10000	200000
Mean D	0.256	0.116	0.064	0.042	0.033
Mean C	0.077	0.08	0.082	0.093	0.102
D/C	3.343	1.444	0.79	0.448	0.32

Diagramm 5.18: Durchschnittsverbrauch der Simulationen 5.4-1 bis 5.4-5 im Vergleich
Die Abkürzung D steht für den dezentralen Algorithmus, C für den zentralen; Auf der x-Achse ist die Netzwerkgrösse in Anzahl Knoten angegeben. Die blaue und rote Kurve sind in absoluten Werten angegeben (linke y-Achse), die grüne Kurve zeigt relative Werte (rechte y-Achse).



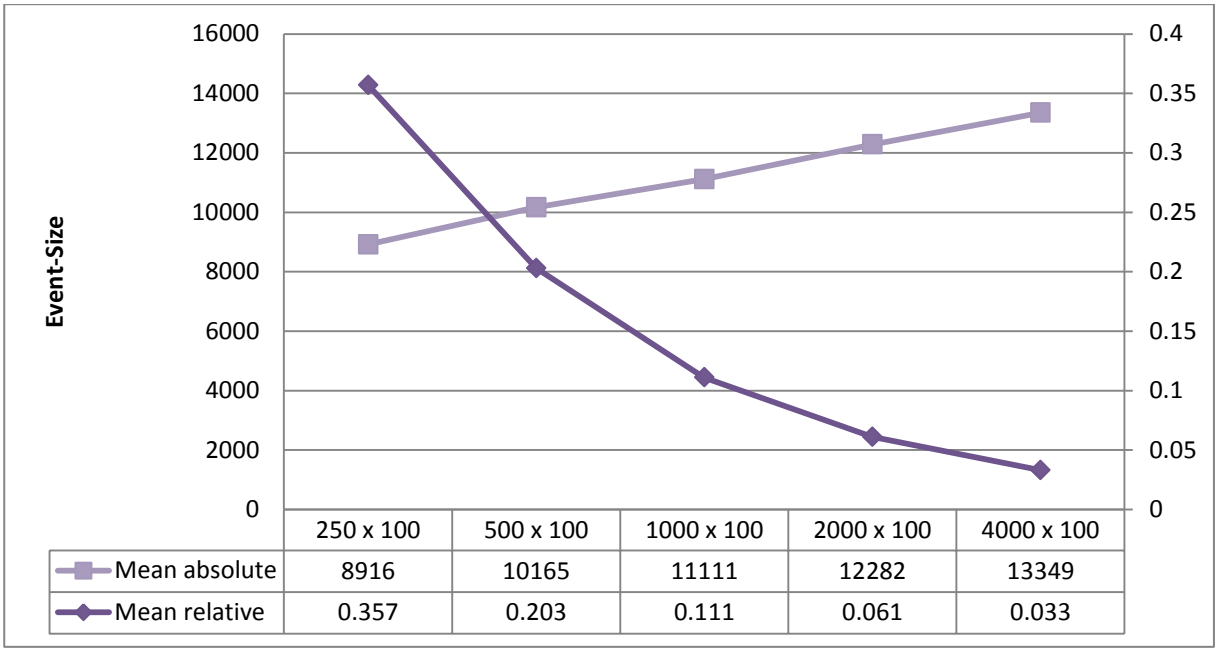
	1250	2500	5000	10000	200000
Std Dev D	0.168	0.102	0.095	0.106	0.111
Std Dev C	0.278	0.317	0.403	0.517	0.578
D/C	0.605	0.322	0.235	0.205	0.191

Diagramm 5.19: Standardabweichung der Simulationen 5.4-1 bis 5.4-5 im Vergleich
Die Abkürzung D steht für den dezentralen Algorithmus, C für den zentralen. Auf der x-Achse ist die Netzwerkgrösse in Anzahl Knoten angegeben. Die blaue und rote Kurve sind in absoluten Werten angegeben (linke y-Achse), die grüne Kurve zeigt relative Werte (rechte y-Achse).



	1250	2500	5000	10000	200000
Max D	0.839	1.041	1.153	1.247	1.343
Max C	3.761	5.233	5.626	6.166	6.947
D/C	0.223	0.199	0.205	0.202	0.193

Diagramm 5.20: Maximal belastete Knoten der Simulationen 5.4-1 bis 5.4-5 im Vergleich
Die Abkürzung D steht für den dezentralen Algorithmus, C für den zentralen. Auf der x-Achse ist die Netzwerkgrösse in Anzahl Knoten angegeben. Die blaue und rote Kurve sind in absoluten Werten angegeben (linke y-Achse), die grüne Kurve zeigt relative Werte (rechte y-Achse).



	250 x 100	500 x 100	1000 x 100	2000 x 100	4000 x 100
Mean absolute	8916	10165	11111	12282	13349
Mean relative	0.357	0.203	0.111	0.061	0.033

Diagramm 5.21: Durchschnittliche Grösse der Ereignisse
Die durchschnittlichen Grössen der Ereignisse sind absolut (hellviolett) und relativ (dunkelviolett) zur Gesamtfläche des Untersuchungsgebietes für die Simulationen 5.4-1 (ganz links) bis 5.4-5 (ganz rechts) angegeben. Die linke y-Achse gilt für die absoluten Werte, die rechte y-Achse für die relativen Werte.

5.4.4 Diskussion

5.4.4.1 Dezentraler Algorithmus

Wie erwartet nimmt die durchschnittliche Belastung der Sensorknoten mit zunehmender Grösse des Sensornetzes ab. Noch einmal soll erwähnt sein, dass die Entstehungswahrscheinlichkeit eines neuen Ereignisses konstant gehalten wurde und darum die prozentuale Fläche, welche von einem Waldbrand betroffen ist, tendenziell abnimmt. Es ist zu beobachten, dass wie erwartet die durchschnittliche Belastung („Mean D“ in Diagramm 5.18) sich parallel zur durchschnittlichen relativen Ereignisgrösse („Mean relative“ in Diagramm 5.21) verändert. Würde die relative durchschnittliche Brandfläche konstant bleiben, könnte hier von einer in etwa gleich bleibenden Belastung ausgegangen werden, wobei dann auch die durchschnittliche Grösse einzelner Ereignisse eine Rolle spielen würde. Und obwohl, wie schon in Kapitel 5.2 beschrieben, die Resultate bei gleichem Setting relativ stark streuen können, so ist hier trotzdem eine eindeutige Tendenz festzustellen.
Auch wie erwartet verläuft die Belastung der am meist frequentierten Knoten. Diese steigt mit der Grösse des Untersuchungsgebietes leicht an, jedoch schwächer als die absolute durchschnittliche Ereignisgrösse. Dies kann dadurch begründet werden, dass mehr Meldungen von einem Head-Node zum Sink-Node gesendet werden, solange ein Ereignis klein bleibt. Das heisst, dass die Belastung dieser Knoten langsamer zunimmt als die Grösse der Ereignisse. Diese Zusammenhänge werden in Kapitel 5.7 noch genauer untersucht. Ein weiteres Indiz, dass allgemein die Belastung der stark belasteten Knoten nicht extrem zunimmt, kann man den Histogrammen entnehmen, in welchen zu sehen ist, dass nur wenige Knoten jeweils bei den hohen Belastungen anzutreffen sind. Auch die Standardabweichung, welche ab einer Grösse des Sensornetzwerkes von 2‘500 Knoten praktisch konstant bleibt, zeigt, dass die Lastenverteilung im dezentralen Netzwerk gut auf alle Knoten verteilt wird.

5.4.4.2 Zentraler Algorithmus

Die durchschnittliche Belastung der Knoten nimmt in etwa parallel zur durchschnittlichen absoluten Gesamtereignisgrösse zu, was die Erwartung somit bestätigt. Dass jedoch die absolute Grösse der Ereignisse trotzdem prozentual betrachtet ein wenig schneller zunimmt als der durchschnittliche Verbrauch der Knoten, ist wohl dadurch zu begründen, dass ein Ereignis im Mittel eine längere Lebensdauer haben dürfte, da durch die relative Abnahme der Ereignisfläche das Brennmaterial zwischen zwei Ereignissen mehr Zeit hat, wieder nachzuwachsen.
Auch die Belastung der schon am meisten belasteten Knoten hat erwartungsgemäss mit steigender Grösse des Untersuchungsgebietes zugenommen, jedoch wie beim dezentralen Algorithmus ein wenig langsamer als die durchschnittliche absolute Ereignisfläche. Hier wird wieder derselbe Grund wie bei der durchschnittlichen Belastung der Knoten ausschlaggebend sein, dass die durchschnittliche Brenndauer eines Feuers auch steigt und dadurch, ohne zu einer Mehrbelastung zu führen, eine grössere durchschnittliche Gesamtereignisfläche registriert wird.

5.4.4.3 Vergleich dezentraler und zentraler Algorithmus

Beim Vergleich des dezentralen mit dem zentralen Algorithmus zeigt sich, dass beim durchschnittlichen Energieverbrauch der dezentrale Algorithmus mit zunehmender Grösse des Untersuchungsgebietes den Erwartungen entsprechend gegenüber dem zentralen Algorithmus immer besser performt. Bei einer Netzgrösse bis 2‘500 Sensorknoten zeigt hier der zentrale Algorithmus noch eine bessere Performance, danach scheint der dezentrale Algorithmus aber besser

zu sein. Dies hat damit zu tun, dass sich der Energieverbrauch des dezentralen Algorithmus parallel zur relativen Ereignisfläche, der des zentralen Algorithmus jedoch parallel zur absoluten Ereignisfläche verändert. Da mit zunehmender Grösse des Gebietes bei konstanter Ereignisfläche die relative Fläche stetig abnimmt, oder umgekehrt bei konstanter relativer Ereignisfläche die absolute Fläche permanent zunimmt, muss die Performance der beiden Algorithmen zwangsläufig auseinanderdriften, weshalb der dezentrale Algorithmus eine massiv bessere Skalierbarkeit bezüglich des durchschnittlichen Energieverbrauchs aufweist. Schon bei einer Grösse des Untersuchungsgebiets von $0.4km^2$ bei einer Sensordichte von einem Sensor pro $20m^2$ resultiert beim dezentralen Algorithmus eine Reduktion des Gesamtenergieverbrauchs von fast 70% gegenüber dem zentralen Algorithmus. Diese Differenz wird bei noch grösseren Netzwerken stetig zunehmen.
Beim Verbrauch der am meisten belasteten Knoten, welcher für die Lebensdauer eines Geosensor Network von grösserer Bedeutung ist als die durchschnittliche Belastung, zeigt sich nicht mehr ein so eindeutiges Bild. Hier kann man sehen, dass bei beiden Algorithmen die Belastung durch die Vergrösserung des Untersuchungsgebietes und somit der durchschnittlichen absoluten Ereignisgrösse zunimmt. Schaut man auf die Verhältnisse, so kann man in den Tabellen sehen, dass beim dezentralen Algorithmus die stark belasteten Knoten über alle fünf Untersuchungen hinweg immer ca. 20% des Energieverbrauchs der maximal belasteten Knoten des zentralen Algorithmus aufweisen. Es ist also unabhängig von der Netzwerkgrösse immer mit einer ca. 5-mal längeren Lebensdauer dieser Knoten zu rechnen, jedoch nimmt bei beiden Algorithmen die Lebensdauer mit der Grösse des Netzwerkes ab. Hier muss jedoch zusätzlich noch erwähnt werden, dass beim dezentralen Algorithmus je nach Einstellung der Informationsmenge (vgl. Kapitel 5.3), die an den Sink-Node übermittelt wird, durchaus Potential vorhanden ist, auch diesen Wert in etwa konstant halten zu können, während der zentrale Algorithmus keine Stellschrauben (geht man von identischer zeitlichen Auflösung aus) besitzt, mit welchen der Energieverbrauch gesenkt werden könnte. Wie gross dieses Potential jedoch ist, hängt von den individuellen Anforderungen an das Geosensor Network ab und kann somit nicht verallgemeinert werden.

5.5 Variation der Sensordichte

5.5.1 Experiment

In diesem Experiment werden die beiden Algorithmen auf deren Verhalten bei der Variation der Sensordichte getestet. Es muss jedoch in engem Zusammenhang mit Kapitel 5.4 betrachtet werden, denn es gibt insgesamt zwei Möglichkeiten, die Sensordichte zu variieren: Entweder man hält die Grösse des Untersuchungsgebietes konstant und ändert die Anzahl Sensoren, die sich darin befinden, oder man lässt die Anzahl Sensoren konstant und ändert die Grösse des Untersuchungsgebiets. Bei ersterem wird automatisch auch die Komponente der Netzwerksgrösse darin einfliessen, während bei der zweiten Variante sich wiederum das Verhältnis der Ereignisgrösse zur Gesamtfläche verändern wird. Geht man jedoch von einem realistischen Szenario aus, so ist die Umwelt fix gegeben und somit kann die Sensordichte nur über die Anzahl Sensoren variiert werden. Diese Variante wird hier mit folgenden Parametern getestet:

Simulation	initialNumberOfSensorNodes	sensorNodeSendingDistance
5.5-1	5000	20
5.5-2	7500	20
5.5-3	10000 (S)	20
5.5-4	15000	20
5.5-5	20000	20

Tabelle 5.16: Settings für die Variation der Sensordichte
Das S in Klammern bedeutet, dass es sich hier um den in Kapitel 5.2 definierten Standardwert handelt.

Wie der Tabelle zu entnehmen ist, wurde auch der Senderadius der einzelnen Sensorknoten von standardmässig 10m auf 20m verdoppelt. Dies hat natürlich auch einen direkten Einfluss auf die beiden Algorithmen, ist jedoch bei einer geringeren Dichte unabdingbar, da sonst eine Verbindung aller Knoten zum Sink-Node nicht mehr gewährleistet werden kann. Dieser höhere Senderadius wurde jedoch für alle Simulationen verwendet, da so innerhalb dieser fünf Simulationen mit vergleichbaren Werten gearbeitet werden kann.

5.5.2 Erwartungen

Grundsätzlich wird bei diesem Experiment erwartet, dass die Differenz des durchschnittlichen Energieverbrauchs der Knoten zwischen den beiden Algorithmen tendenziell tiefer sein wird, verglichen mit den anderen Experimenten, was auf die Erweiterung des Senderadius zurückzuführen ist. Wie in Kapitel 3.5 erwähnt, wird aus verschiedenen Gründen bei der Messung des Energieverbrauchs anhand der Anzahl gesendeten Mitteilungen die Distanz zwischen den Knoten nicht mit einberechnet. Durch die Verdoppelung des Senderadius wird die Anzahl Hops von einem Ereignis zum Sink-Node aber etwa um die Hälfte reduziert, was zur Folge hat, dass der durchschnittliche Verbrauch des zentralen Algorithmus um einiges tiefer sein dürfte, während beim dezentralen Algorithmus nur ein kleiner Teil der Energie eingespart werden kann, da der Aufbau des Graphen (RNG) nicht vom höheren Senderadius beeinflusst werden dürfte. Somit bleibt beim dezentralen Algorithmus der Aufwand innerhalb der Ereignisse gleich. Innerhalb dieses Experiments sind die Ergebnisse vergleichbar, da alle Simulationen mit demselben maximalen Senderadius durchgeführt werden. Quervergleiche zu anderen Experimenten sind jedoch schwierig, da dieser erhöhte Senderadius bei gleichem Energieverbrauch in der Realität einer verbesserten Technologie entsprechen würde.

Wie sich der durchschnittliche Energieverbrauch bei beiden Algorithmen bei einer Dichteänderung der Knoten genau verhalten wird, ist sehr schwierig vorherzusagen, weil durch eine Veränderung der Dichte zwei entgegengesetzte Tendenzen zu erwarten sind: Auf der einen Seite werden bei eine Verdoppelung der Sensordichte ungefähr doppelt so viele Sensoren das gleiche Ereignis messen. Beim zentralen Algorithmus führt dies voraussichtlich dazu, dass ungefähr doppelt so viele Informationen zum Sink-Node gemeldet werden. Auf der anderen Seite verteilt sich dieser Aufwand auch auf doppelt so viele Knoten. Beim dezentralen Algorithmus wird eine Verdoppelung der Sensordichte jedoch analog zur Folge haben, dass sich auch der dezentrale Kommunikationsaufwand innerhalb eines Ereignisses zwar massiv erhöhen wird, jedoch auch dieser Aufwand wiederum auf mehr Knoten verteilt wird. Ein weiterer Aspekt, der bei beiden Algorithmen hineinspielt, ist, dass der Shortest-Path zum Sink-Node tendenziell bei höherer Sensordichte aus weniger Hops besteht. Dies hat den einfachen Grund, dass bei einer sehr hohen Sensordichte der maximale Senderadius eher ausgenutzt werden kann als bei einer niedrigen Sensordichte und auch tendenziell ein direkterer Weg zum Sink-Node nahe an der Euklidischen Distanz verwendet werden kann. Dieser Umstand wird aber dementsprechend auch eher wieder dem zentralen Algorithmus entgegenkommen, welcher den gesamten Energieverbrauch für das Senden von Informationen an den Sink-Node aufwendet. Eine absolute Aussage, wie sich die beiden Algorithmen schliesslich über die fünf Simulationen verhalten werden, kann hier kaum gemacht werden, da der gesamte Vorgang zu komplex ist, um ihn umfassend erfassen und abschätzen zu können. Da mit zunehmender Sensordichte jedoch die durchschnittliche Sendedistanz einer Information beim dezentralen Algorithmus immer kleiner wird (der RNG wird dichter), während diese beim zentralen Algorithmus eher zunehmen dürfte, und wie schon erwähnt der Sendedistanz bei der Evaluation kein Einfluss auf den Energieverbrauch attestiert wird, dürfte eine höhere Sensordichte eher dem zentralen Algorithmus entgegenkommen.

Beim Energieverbrauch der maximal belasteten Knoten ist die Ausgangslage ein wenig klarer. Tendenziell kann gesagt werden, dass beim zentralen Algorithmus mit zunehmender Sensordichte mehr Informationen an den Sink-Node gesendet werden müssen, da wie schon oben erwähnt mehr Sensorknoten von einem Ereignis betroffen sind. Aus diesem Grund könnte davon ausgegangen werden, dass die Belastung der maximal belasteten Knoten mit der Erhöhung der Sensordichte zunehmen wird. Denkt man aber wieder an die Abbildung 4.1 zurück, so stimmt es zwar, dass mehr Informationen durch die Knoten, welche eine direkte Verbindung mit dem Sink-Node eingehen können, weitergeleitet werden müssen; jedoch kann die Belastung wiederum auch auf mehr Knoten verteilt werden. Dieser Einfluss ist schwierig abzuschätzen. Es ist jedoch aus diesem Grund nicht mit einem proportionalen Anstieg der maximalen Belastung zur Sensordichte zu rechnen. Beim dezentralen Algorithmus ist davon auszugehen, dass die Maximalbelastung mit der Erhöhung der Sensordichte ansteigt, jedoch nicht im selben Ausmass wie beim zentralen Algorithmus. Dies aus dem Grund, da die Menge an Informationen, welche zum Sink-Node gesendet wird, sehr konstant bleibt, da die gesendeten Informationen theoretisch unabhängig von der Sensordichte sind (vgl. Formel 4.1). Jedoch wird ein häufigerer Wechsel des Head-Nodes erwartet, was einen konstanten Aufwand über die verschiedenen Sensordichten sehr unwahrscheinlich macht.

5.5.3 Ergebnisse

5.5.3.1 Simulation 5.5-1: 5000 Sensorknoten = 1 Knoten pro 40m^2; Sendedistanz = 20m

Als Windrichtung wurde der Zufallswert 0.29 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.39
Osten: 0.26
Süden: 0.51
Westen: 0.64

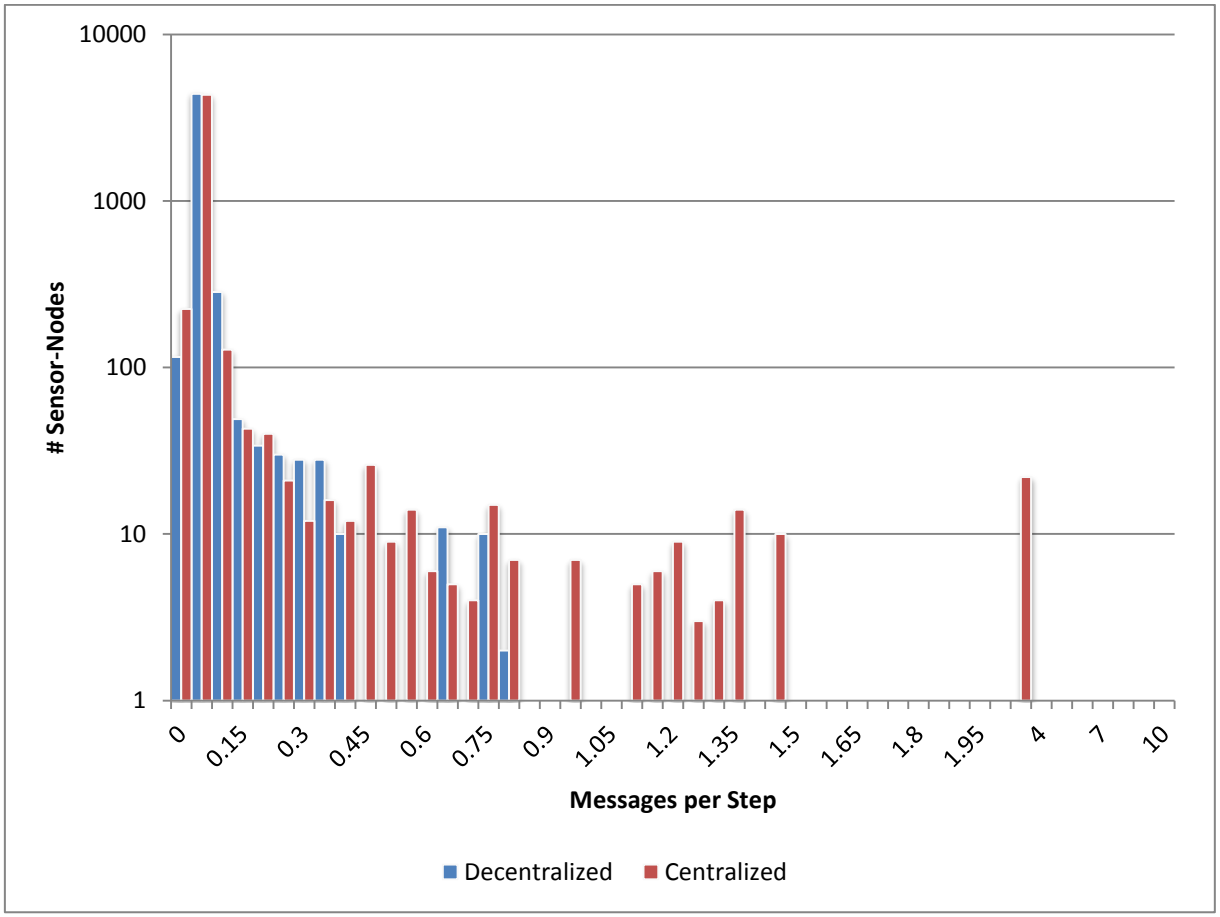


Diagramm 5.22: Simulation 5.5-1
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.03	0.047	0.627
Median	0.017	0.001	15
Std Dev	0.063	0.232	0.27
Max	0.799	3.045	0.262

Tabelle 5.17: Simulation 5.5-1
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.5.3.2 Simulation 5.5-2: 7500 Sensorknoten = 1 Knoten pro 26.7m^2; Sendedistanz = 20m

Als Windrichtung wurde der Zufallswert 4.29 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.63
Osten: 0.53
Süden: 0.27
Westen: 0.37

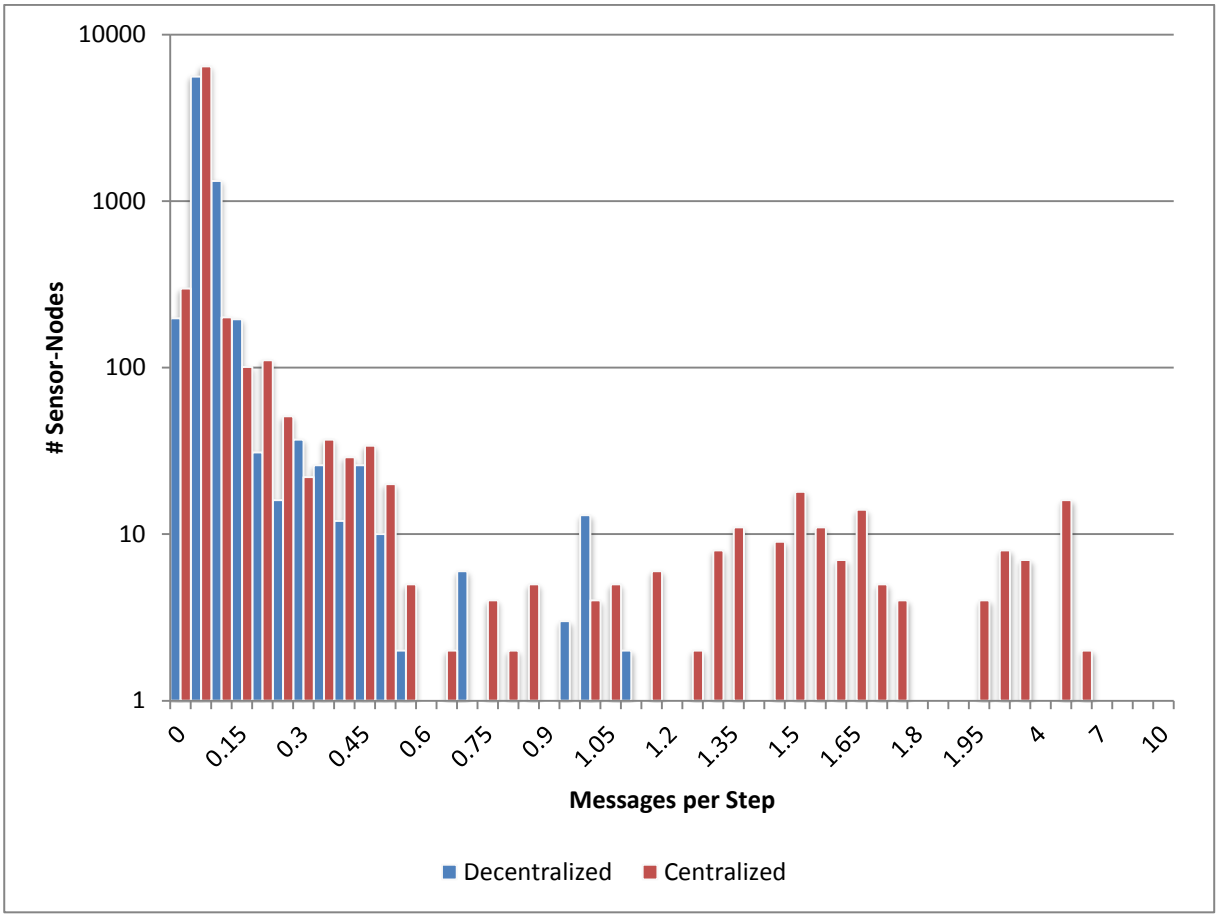


Diagramm 5.23: Simulation 5.5-2
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.039	0.057	0.693
Median	0.025	0.001	18.37
Std Dev	0.07	0.309	0.228
Max	1.07	5.066	0.211

Tabelle 5.18: Simulation 5.5-2
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.5.3.3 Simulation 5.4-3: 10000 Sensorknoten = 1 Knoten pro 20m^2; Sendedistanz = 20m

Als Windrichtung wurde der Zufallswert 2.81 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.39
Osten: 0.64
Süden: 0.51
Westen: 0.26

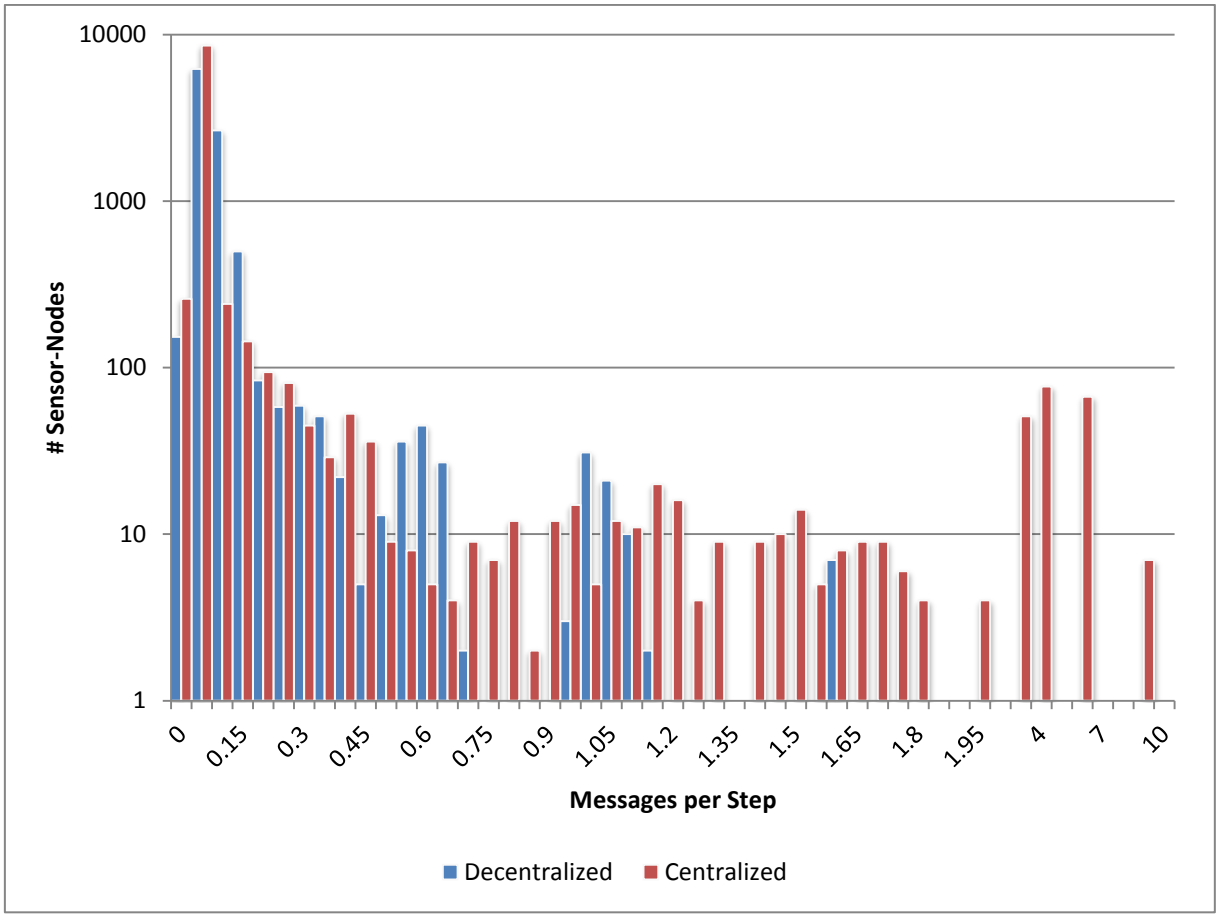


Diagramm 5.24: Simulation 5.5-3
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.036	0.048	0.737
Median	0.024	0.001	20.292
Std Dev	0.054	0.251	0.215
Max	1.056	5.095	0.207

Tabelle 5.19: Simulation 5.5-3
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.5.3.4 Simulation 5.5-4: 15000 Sensorknoten = 1 Knoten pro 13.3m^2; Sendedistanz = 20m

Als Windrichtung wurde der Zufallswert 6 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.51
Osten: 0.26
Süden: 0.4
Westen: 0.64

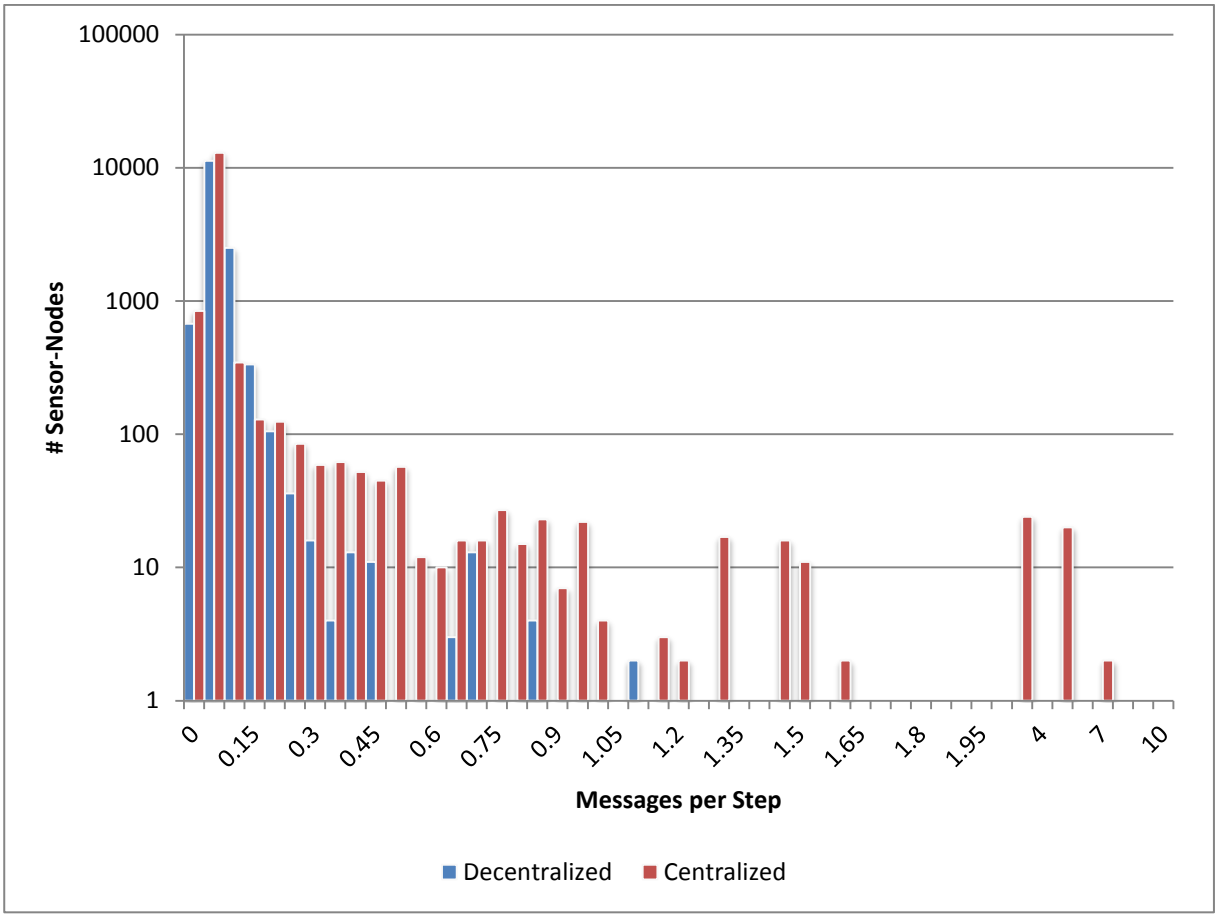


Diagramm 5.25: Simulation 5.5-4
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.032	0.039	0.823
Median	0.022	0.001	19.636
Std Dev	0.045	0.239	0.189
Max	1.1	6.937	0.159

Tabelle 5.20: Simulation 5.5-4
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.5.3.5 Simulation 5.4-5: 20000 Sensorknoten = 1 Knoten pro 10m^2; Sendedistanz = 20m

Als Windrichtung wurde der Zufallswert 6.24 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.46
Osten: 0.25
Süden: 0.44
Westen: 0.65

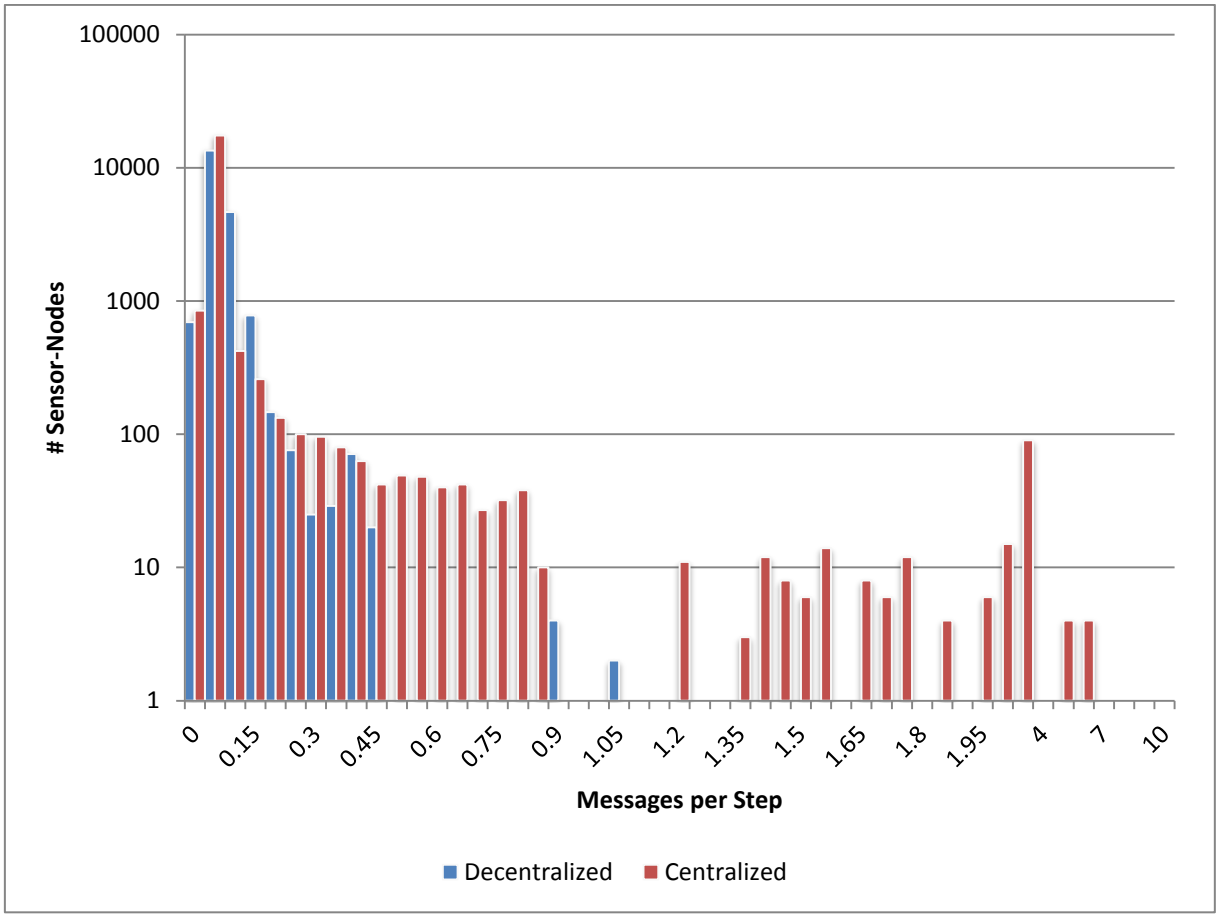
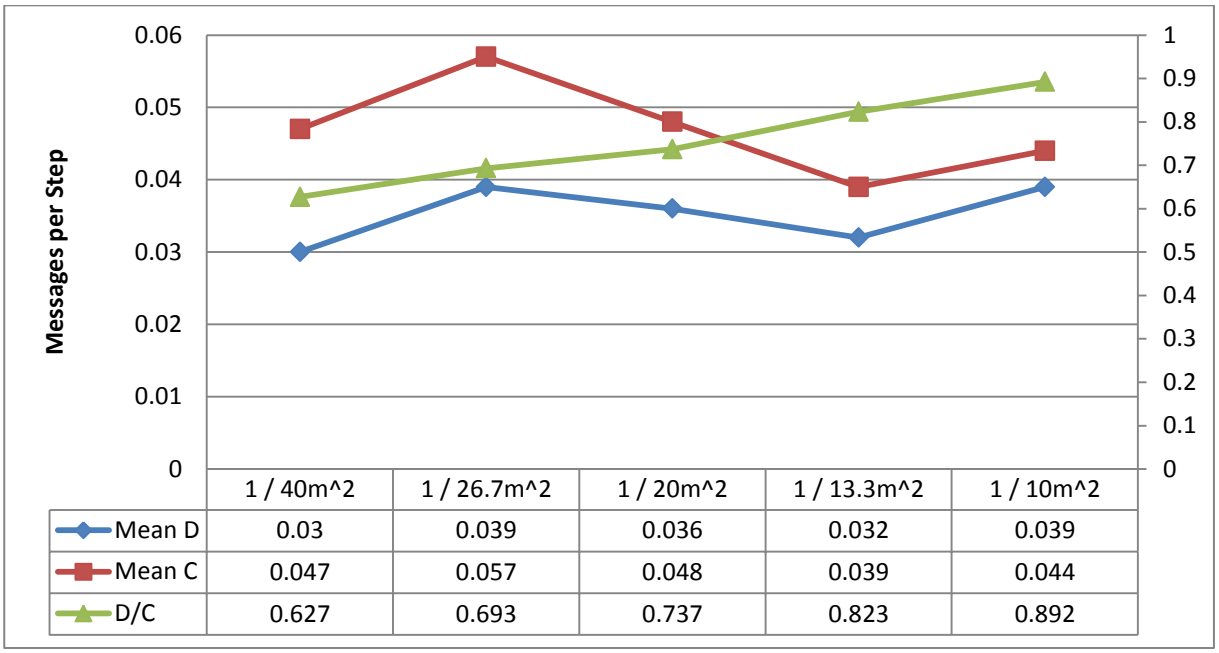


Diagramm 5.26: Simulation 5.5-5
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala. Beim zentralen Algorithmus sendet nur gerade ein Knoten mit einem Wert von 11.044 mehr als 10 Mitteilungen pro Step und ist durch die logarithmische Skala im obenstehenden Diagramm nicht ersichtlich.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.039	0.044	0.892
Median	0.028	0.001	25.591
Std Dev	0.049	0.249	0.197
Max	1.656	11.044	0.115

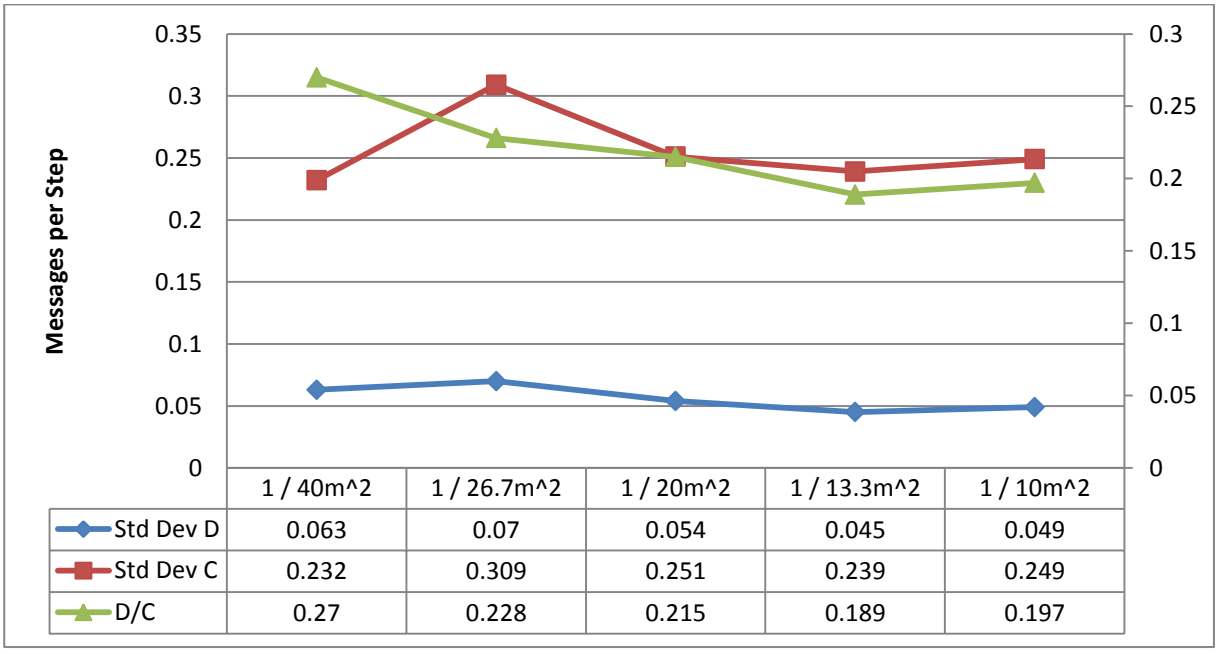
Tabelle 5.21: Simulation 5.5-5
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.5.3.6 Vergleich Simulation 5.5-1 bis Simulation 5.5-5



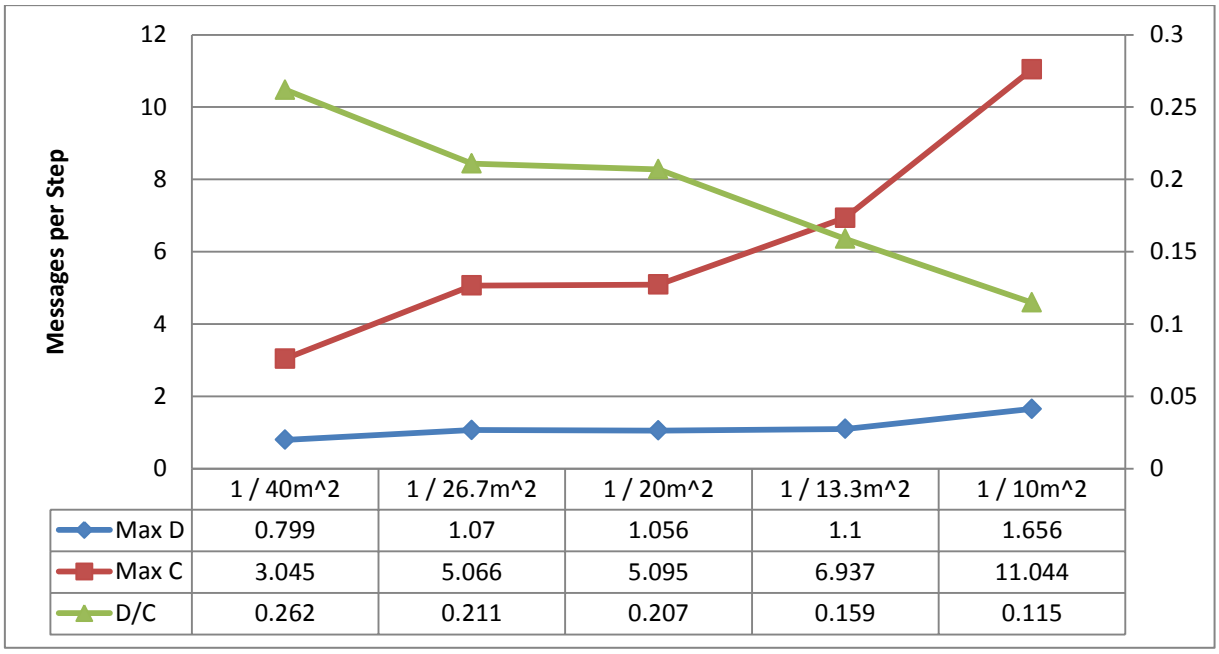
	1 / 40m^2	1 / 26.7m^2	1 / 20m^2	1 / 13.3m^2	1 / 10m^2
Mean D	0.03	0.039	0.036	0.032	0.039
Mean C	0.047	0.057	0.048	0.039	0.044
D/C	0.627	0.693	0.737	0.823	0.892

Diagramm 5.27: Durchschnittsverbrauch der Simulationen 5.5-1 bis 5.5-5 im Vergleich
Die Abkürzung D steht für den dezentralen Algorithmus, C für den zentralen; Auf der x-Achse ist die Sensordichte in Knoten pro m^2 angegeben. Die blaue und rote Kurve sind in absoluten Werten angegeben (linke y-Achse), die grüne Kurve zeigt relative Werte (rechte y-Achse).



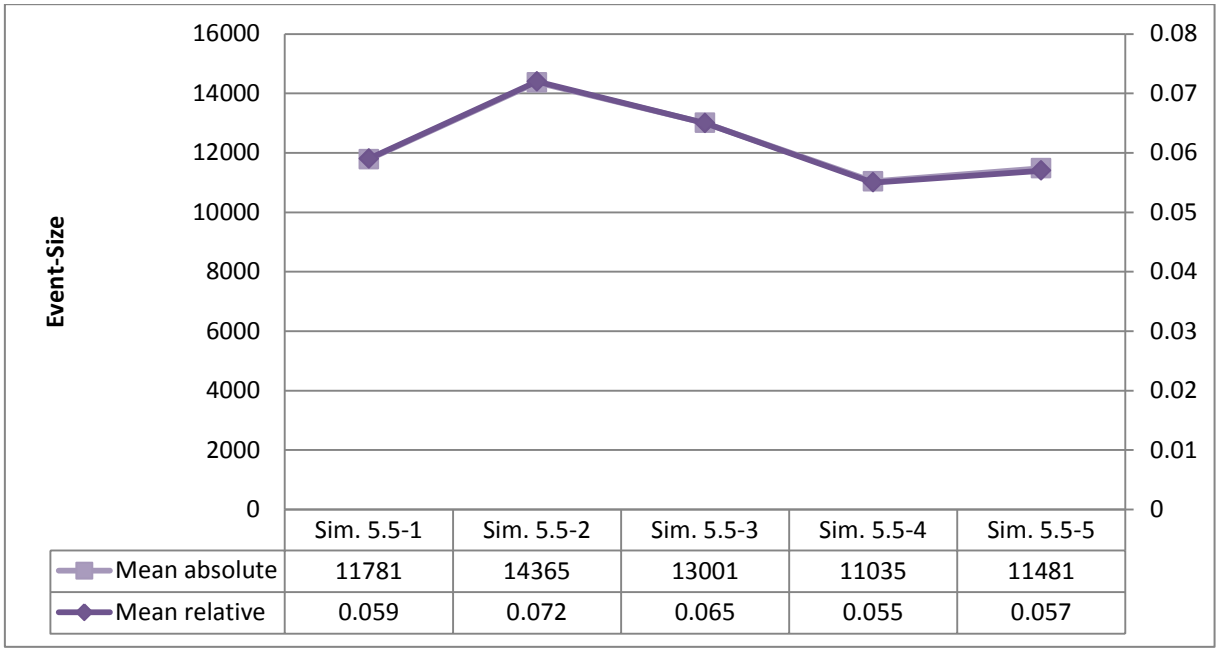
	1 / 40m^2	1 / 26.7m^2	1 / 20m^2	1 / 13.3m^2	1 / 10m^2
Std Dev D	0.063	0.07	0.054	0.045	0.049
Std Dev C	0.232	0.309	0.251	0.239	0.249
D/C	0.27	0.228	0.215	0.189	0.197

Diagramm 5.28: Standardabweichung der Simulationen 5.5-1 bis 5.5-5 im Vergleich
Die Abkürzung D steht für den dezentralen Algorithmus, C für den zentralen; Auf der x-Achse ist die Sensordichte in Knoten pro m^2 angegeben. Die blaue und rote Kurve sind in absoluten Werten angegeben (linke y-Achse), die grüne Kurve zeigt relative Werte (rechte y-Achse).



	1 / 40m^2	1 / 26.7m^2	1 / 20m^2	1 / 13.3m^2	1 / 10m^2
Max D	0.799	1.07	1.056	1.1	1.656
Max C	3.045	5.066	5.095	6.937	11.044
D/C	0.262	0.211	0.207	0.159	0.115

Diagramm 5.29: Maximal belastete Knoten der Simulationen 5.5-1 bis 5.5-5 im Vergleich
Die Abkürzung D steht für den dezentralen Algorithmus, C für den zentralen; Auf der x-Achse ist die Sensordichte in Knoten pro m^2 angegeben. Die blaue und rote Kurve sind in absoluten Werten angegeben (linke y-Achse), die grüne Kurve zeigt relative Werte (rechte y-Achse).



	Sim. 5.5-1	Sim. 5.5-2	Sim. 5.5-3	Sim. 5.5-4	Sim. 5.5-5
Mean absolute	11781	14365	13001	11035	11481
Mean relative	0.059	0.072	0.065	0.055	0.057

Diagramm 5.30: Durchschnittliche Grösse der Ereignisse
Die durchschnittlichen Grössen der Ereignisse sind absolut (hellviolett) und relativ (dunkelviolett) zur Gesamtfläche des Untersuchungsgebietes für die Simulationen 5.5-1 (ganz links) bis 5.5-5 (ganz rechts) angegeben. Die linke y-Achse gilt für die absoluten Werte, die rechte y-Achse für die relativen Werte. Da die Fläche des Untersuchungsgebietes immer gleich gross war, haben die beiden Kurven denselben Verlauf und wegen der Skalierung der y-Achsen sogar deckungsgleich.

5.5.4 Diskussion

Grundsätzlich bestätigt sich die Ausgangslage, dass durch den erhöhten Senderadius die Effizienz der beiden Algorithmen näher zusammenrückt. Während in Kapitel 5.2 der dezentrale Algorithmus beim durchschnittlichen Energieverbrauch rund 46% verglichen zum zentralen Algorithmus aufwenden musste, sind es hier bei gleichem Setting aufgrund der erweiterten Sendedistanz der Knoten in Simulation 5.5-3 74%.

Die Kurven der beiden Algorithmen in Diagramm 5.27 sehen sehr ungewöhnlich aus, da sich kein klarer Trend erkennen lässt. Sieht man sie jedoch in Zusammenhang mit Diagramm 5.21, in welchem die durchschnittliche Ereignisgrösse der verschiedenen Simulationen gezeigt wird, so lässt sich erkennen, dass sich der Durchschnittverbrauch bei beiden Algorithmen parallel zur durchschnittlichen Gesamtereignisgrösse verändert, weshalb die beiden Kurven so zu erklären sind und auf der Zufälligkeit der simulierten Umwelt beruhen. Weiter sieht man, dass bei Simulation 5.5-1 (1 Knoten pro $40m^2$) und Simulation 5.5-5 (1 Knoten pro $10m^2$) die durchschnittlichen Gesamtereignisgrössen weniger als 3% auseinanderliegen, weshalb diese Werte auch für jeden einzelnen separaten Algorithmus vergleichbar sind. Hier zeigt sich, dass beim zentralen Algorithmus die Sensordichte kaum einen Einfluss auf den durchschnittlichen Energieverbrauch hat, jedoch beim dezentralen Algorithmus mit zunehmender Sensordichte die Performance sinkt. Schaut man sich den relativen Verbrauch der beiden Algorithmen zueinander an, welcher unabhängig von der durchschnittlichen Grösse des Ereignisses als stabil betrachtet werden kann, so zeigt sich diese Tendenz auch eindeutig: Je dichter das Sensornetz ist, desto schlechter performt der dezentrale Algorithmus verglichen mit dem zentralen Algorithmus bezüglich des durchschnittlichen Verbrauchs der einzelnen Knoten.

Betrachtet man die Performance der beiden Algorithmen bezüglich der maximal belasteten Knoten, so fällt auf, dass hier das Bild ganz anders aussieht. Wie erwartet steigt die Belastung hier beim zentralen Algorithmus mit der Erhöhung der Sensordichte massiv an. Jedoch, wenn man Simulation 5.5-1 (1 Knoten pro $40m^2$) mit der von der durchschnittlichen Ereignisfläche her ähnlichen Simulation 5.4-5 (1 Knoten pro $10m^2$) vergleicht, steigt die Belastung nur um den Faktor 3.63 an, während sich die Dichte vervierfacht. Betrachtet man die Histogramme ein wenig genauer, so kann man erkennen, dass es sich beim am stärksten belasteten Knoten bei der Simulation 5.5-5 sogar mit mehr als 11 Mitteilungen pro Step beim zentralen Algorithmus um einen deutlichen Ausreisser handelt, weshalb diese Belastung in der Regel wohl noch ein bisschen tiefer ausfallen dürfte. Trotz allem ist eine deutliche Tendenz erkennbar. Auch beim dezentralen Algorithmus scheint die Belastung mit der Sensordichte anzusteigen, jedoch nicht im gleichen Ausmass. Hier erhöht sich die Belastung nur um rund den Faktor zwei, während sich die Dichte um den Faktor vier vergrössert, wobei auch hier wiederum derselbe Ausreisser wie beim zentralen Algorithmus auszumachen ist, weshalb dieser Faktor in den meisten Fällen auch ein bisschen tiefer sein dürfte. Es scheint hier also bei Simulation 5.5-5 bei der zufälligen Verteilung der Sensorknoten eine Konstellation aufgetreten zu sein, bei welcher ein grosser Teil der Shortest-Path zum Sink-Node über denselben Knoten führt. Die wichtigste Erkenntnis jedoch ist, dass die Belastung beim zentralen Algorithmus verglichen mit dem dezentralen Algorithmus mit zunehmender Sensordichte um einiges stärker ansteigt, was an der grünen Kurve in Diagramm 5.29 relativ eindeutig zu erkennen ist. Somit kann behauptet werden, dass der dezentrale Algorithmus bezüglich der maximal belasteten Knoten besser mit hohen Sensordichten umgehen kann.

In einer Gesamtbetrachtung sieht man, dass bei den beiden Disziplinen Durchschnittsverbrauch und Maximalverbrauch nicht derselbe Algorithmus besser mit der Erhöhung der Sensordichte zurechtkommt. Geht man aber wiederum davon aus, dass die maximal belasteten Knoten für die

Lebensdauer eines Netzwerkes entscheidender sind, schneidet auch hier der dezentrale Algorithmus bei einer Erhöhung der Sensordichte insgesamt wohl besser ab. Auch bei diesem Experiment soll erwähnt werden, dass ein dichteres Sensornetzwerk auch genauere Informationen liefern kann. Dadurch müsste wiederum auch bezüglich der Sensordichte in einem realen Szenario ein Kompromiss zwischen Sensordichte und „Energieverbrauch“ gefunden werden. Ein Aspekt, welcher hier auch nicht in die Betrachtung einbezogen wurde, ist die Tatsache, dass durch eine höhere Sensordichte potentiell Energie gespart werden kann, indem der Senderadius der Knoten reduziert wird. Aus diesem Grund sind die Ergebnisse dieses Experimentes in der Praxis wohl eher in Bezug auf die Netzwerklast einzelner Knoten und nicht bezüglich des realen Energieverbrauchs zu interpretieren, denn hier wird jeweils von einem fixen Energieverbrauch für das Senden von Informationen an einen anderen Knoten ausgegangen.

5.6 Variation des Graphen (RNG / GG)

5.6.1 Experiment

In diesem Kapitel wird untersucht, wie sich eine Veränderung des Graphen auf die Algorithmen auswirkt. In Kapitel 3.2 wurden die beiden Graphen, der Gabriel Graph (GG) und der Relative Neighborhood Graph (RNG) vorgestellt. Es werden also im Folgenden zwei Simulationen durchgeführt: Bei der Simulation 5.6-1 handelt es sich um das Standardsetting mit dem RNG, bei der Simulation 5.6-2 wird der GG verwendet. Weitere Graphen wurden nicht getestet.

5.6.2 Erwartungen

Wie in Kapitel 3.2 schon erklärt wurde, handelt es sich beim RNG um einen Subgraphen des GG. Das heisst, im GG sind alle Kanten vorhanden, welche auch im RNG vorkommen. Insofern ist die durchschnittliche Anzahl Verbindungen pro Knoten beim GG grösser oder mindestens gleich gross wie beim RNG. Beim dezentralen Algorithmus hat dies grundsätzlich einmal folgenden Effekt: Jedes Mal, wenn ein Knoten all seine Nachbarknoten abfragen muss (vgl. Kapitel 4.1), erhöht sich der Aufwand, da mehr Verbindungen vorhanden sind. Hat der Knoten seine Verbindungen aufgebaut, ist davon auszugehen, dass der durchschnittliche Aufwand bei beiden Graphen wieder der gleiche ist, da innerhalb eines Ereignisses jeder Knoten genau zu einem anderen Knoten sendet, unabhängig vom Graphen. Des Weiteren ist davon auszugehen, dass durch die höhere Verbindungsdichte beim GG der in Kapitel 4.1.2.3 angesprochene Umstand, dass ein Sensornetzwerk ein Ereignis als zwei Ereignisse deklariert, verkleinert wird. Dadurch sollte sich auch die Anzahl an Verschmelzungen zweier Ereignisse zu einem neuen reduzieren, was den Energieverbrauch positiv beeinflussen dürfte. Dasselbe wird bei der Aufteilung eines Ereignisses in zwei kleinere Ereignisse erwartet. Da die Ausbreitungsart und die Grösse sowie die Form von Ereignissen hier jedoch einen sehr hohen Einfluss auf diesen Umstand haben, ist eine Abschätzung der Effizienz beider Graphen im dezentralen Algorithmus nur sehr schwierig möglich. Es kann jedoch davon ausgegangen werden, dass die Unterschiede beim durchschnittlichen Verbrauch nicht allzu gross ausfallen dürften. Bei den maximal belasteten Knoten ist davon auszugehen, dass die Effizienz in etwa gleich bleiben wird, da die gesendeten Informationen bis auf die erwähnten Verschmelzungen und Aufteilungen von Ereignissen dieselben sein dürften.

De zentrale Algorithmus ist nicht vom Graphen betroffen, da, wie schon in Kapitel 3.2 beschrieben, die Informationsübermittlung an den Sink-Node sowieso über einen UDG geschieht, um die Anzahl Hops sowie die gesamte Sendedistanz möglichst gering zu halten. Insofern wird aber der zentrale Algorithmus einen guten Vergleich bieten, da die relative Belastung zueinander auch bei einer möglichen Streuung der absoluten Werte relativ robust bleibt (vgl. Kapitel 5.2).

5.6.3 Ergebnisse

5.6.3.1 Simulation 5.6-1: Relative Neighborhood Graph (RNG)

Als Windrichtung wurde der Zufallswert 2.83 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.39
Osten: 0.64
Süden: 0.51
Westen: 0.26

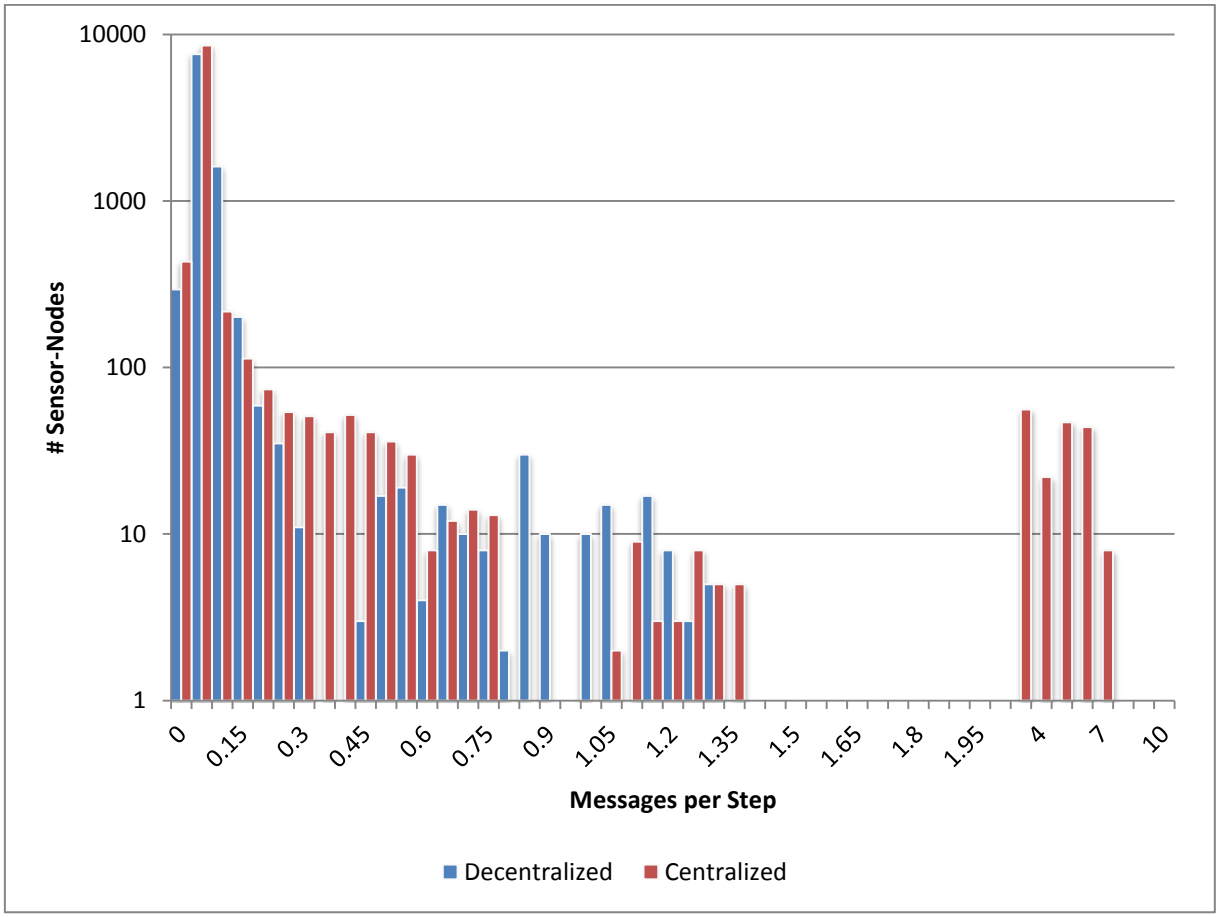


Diagramm 5.31: Simulation 5.6-1
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.045	0.097	0.464
Median	0.025	0.001	20.667
Std Dev	0.112	0.566	0.198
Max	1.261	6.46	0.195

Tabelle 5.22: Simulation 5.6-1
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.6.3.2 Simulation 5.6-2: Gabriel Graph (GG)

Als Windrichtung wurde der Zufallswert 2.40 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.31
Osten: 0.6
Süden: 0.59
Westen: 0.3

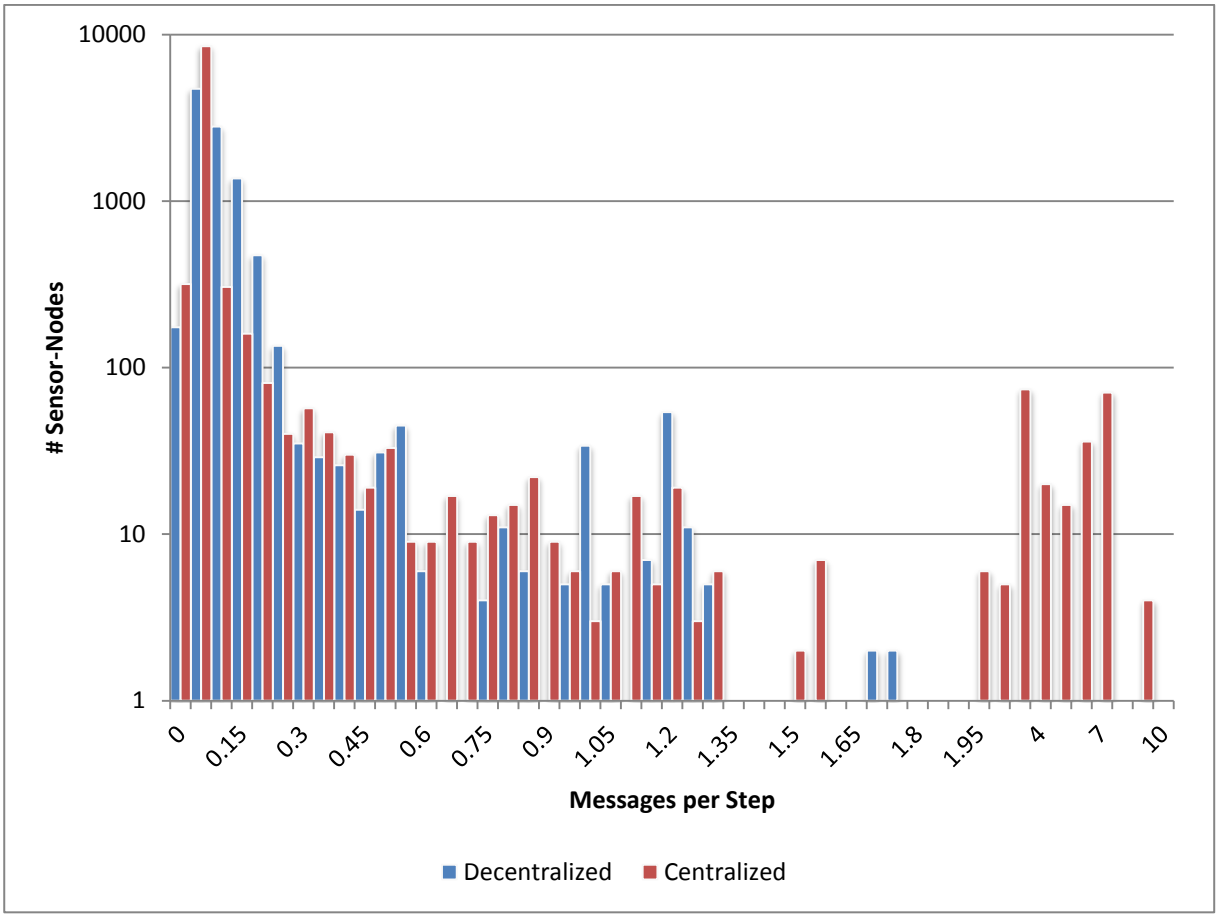


Diagramm 5.32: Simulation 5.6-2
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.081	0.136	0.596
Median	0.051	0.002	30.265
Std Dev	0.142	0.729	0.194
Max	1.720	8.954	0.192

Tabelle 5.23: Simulation 5.6-2
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.6.4 Diskussion

Beim Durchschnittsverbrauch kann man vom RNG zum GG beim dezentralen Algorithmus einen deutlichen Anstieg feststellen. Nimmt man die zentrale Variante, welche ja von keinen Parameteränderungen betroffen ist, sieht man, dass bei der Simulation 5.6-2 der Verbrauch aber auch gestiegen ist. Vergleicht man aber die relativen Werte zum zentralen Algorithmus, welche ja gemäss Kapitel 5.2 als relativ stabil zu betrachten sind, so zeigt sich eine Steigerung des Energieverbrauches von 46.4% auf 59.6%, was einem Mehrverbrauch von ca. 28.5% entspricht. Bei der Standardabweichung sowie den am stärksten belasteten Knoten scheint das Verhältnis wie erwartet jedoch praktisch identisch zu bleiben, was darauf schliessen lässt, dass beim RNG vor allem die dezentrale Aggregation der Daten effizienter abläuft als beim GG. Worauf dies genau zurückzuführen ist, ist sehr schwierig zu beurteilen, da es von sehr vielen Faktoren, speziell auch vom verwendeten Umweltszenario, abhängt. Ein Punkt, der schon angesprochen wurde, ist, dass beim GG im Durchschnitt mehr Verbindungen bestehen, was automatisch zu einem Mehrverbrauch führt. Wie gross dieser jedoch ist, kann aus diesen Daten nicht beurteilt werden, da nur die Anzahl der gesendeten Informationen, nicht aber die Art der Informationen aufgezeichnet wird. Hier gibt es bestimmt noch Potential für eine differenziertere Untersuchung des Algorithmus, was im Ausblick (Kapitel 6) noch angesprochen wird. Bezüglich der Qualität der Informationen ist es so, dass beim GG wohl dezentral die Daten besser oder gleich gut bearbeitet werden. Dies hat den Grund, dass beim GG durch die höhere Anzahl Verbindungen weniger Ereignisse vermeintlich als zwei Ereignisse erkannt werden und somit auch als eine anstelle von zwei Informationen an den Sink-Node gesendet werden. Umgekehrt ist auch denkbar, dass zwei Ereignisse als ein grosses gerechnet werden, jedoch scheint diese Problematik kleiner, da sie durch die maximale Sendedistanz der Knoten beschränkt ist. Beide Fehler sind jedoch ausser vom Graphen auch stark abhängig von der Sensordichte und werden mit derer Erhöhung oder Verminderung gegenläufig mehr beziehungsweise weniger Fehler produzieren. Inwiefern hier jedoch in der Realität ein Unterschied resultieren würde, lässt sich so nicht abschätzen, da dann auch die zentrale Datenprozessierung ausserhalb des Geosensor Network einen Einfluss darauf hätte, was auch auf den zentralen Algorithmus zutrifft. Ausserdem müsste dann auch genau definiert werden, ab wann man von einem beziehungsweise zwei Ereignissen spricht.
Eine eindeutige Schlussfolgerung, welcher Graph sinnvoller ist, kann hier so nicht getroffen werden. Aufgrund des Energieverbrauches performt der RNG besser bezüglich des Durchschnittsverbrauchs. Bei den stark belasteten Knoten jedoch, welche für die Gesamtlebensdauer des Geosensor Network eher relevant sind, lässt sich zwischen den beiden Graphen kein Unterschied feststellen. Es könnte in einem realen Versuch sogar so sein, dass bei verschiedenen Szenarien (Waldbrand, auslaufendes Öl, Luftschadstoffmessung etc.) und unterschiedlichen Voraussetzungen bezüglich Sensordichte, Sendedistanz der Sensoren etc. nicht bei jedem Szenario der gleiche Graph die besseren Resultate liefert.

5.7 Variation der Ereignisgrösse / der räumlichen Korrelation

5.7.1 Experiment

In diesem Experiment soll untersucht werden, wie sich die beiden Algorithmen bezüglich der Ereignisgrösse beziehungsweise der räumlichen Autokorrelation verhalten. In der Umwelt hat jedes Phänomen einen Raumbezug, so auch Waldbrände. Wie in Kapitel 2.2 beschrieben, unterliegen sehr viele solcher Phänomene der räumlichen Autokorrelation. Diese kann anhand von verschiedenen Parametern gemessen und quantifiziert werden. Würden Ereignisse, welche durch ein Geosensor Network erkannt werden, keine positive räumliche Autokorrelation aufweisen, wäre eine dezentrale Vermessung eines Ereignisses praktisch nicht umsetzbar, da es keine statistischen Anhaltspunkte gäbe, welche Knoten von einem Ereignis betroffen werden könnten. Gerade aber diese starke räumliche Autokorrelation nimmt sich der in Kapitel 4.1 vorgestellte dezentrale Algorithmus zu Hilfe, indem er davon ausgeht, dass die Chance gross ist, dass der Nachbarknoten demselben Ereignis auch angehört, wenn ein Knoten ein Ereignis misst. Nur mit dieser Grundvoraussetzung ist es diesem Algorithmus überhaupt möglich, ein zusammenhängendes Ereignis zu detektieren. Doch was passiert, wenn diese Ereignisse grösser oder vor allem kleiner werden? Wie performt der Algorithmus, wenn ein Ereignis so klein wird, dass er nur noch von einigen wenigen Knoten registriert wird? Oder was passiert, wenn ein Ereignis extrem gross wird? Diese Fragen sollen in diesem Kapitel beantwortet werden. Es sollen die Umweltparameter so variiert werden, dass die Ereignisse im Durchschnitt grösser beziehungsweise kleiner werden. Hierfür gibt es in der vorgestellten Simulationsumgebung verschiedene Parameter, mit denen dies erreicht werden kann. Zum einen könnte die Brennmaterialmenge variiert werden, oder analog dazu das Feuer in derselben Zeiteinheit mehr oder weniger Brennmaterial verbrennen. Auch könnte die Ausbreitungswahrscheinlichkeit verändert werden. Für dieses Experiment entschied sich der Autor jedoch dafür, die Brenngeschwindigkeit zu variieren. Unabhängig davon, ob dies ein realistisches Szenario darstellt, ist es eine einfache Möglichkeit anhand der Veränderung eines einzigen Parameters die Ausbreitung von Feuer zu vergrössern beziehungsweise zu verkleinern. Verbrennt ein Feuer in der gleichen Zeit das doppelte an Brennmaterial, wird es dementsprechend an dieser Stelle nur halb so lange brennen und wird demzufolge bei gleicher Ausbreitungsgeschwindigkeit durchschnittlich zu einem beliebigen Zeitpunkt eine kleinere Ausdehnung aufweisen. Das Gegenteil passiert, wenn das Feuer weniger Brennmaterial pro Zeiteinheit verbraucht, wodurch es eine längere Brennzeit aufweist und somit eine grössere Ausdehnung erreichen kann. Es werden folgende Experimente durchgeführt:

Simulation	consumeFuel
5.7-1	0.08
5.7-2	0.04
5.7-3	0.02
5.7-4	0.01 (S)
5.7-5	0.005

Tabelle 5.24: Settings für die Variation der Ereignisgrösse / räumliche Autokorrelation
Das S in Klammern bedeutet, dass es sich hier um den in Kapitel 5.2 definierten Standardwert handelt. Die Kolumne „consumeFuel“ beschreibt, wie schnell ein Feuer das Brennmaterial verbrennt. Bei Simulation 5.6-4 entspricht der Wert dem Standardwert aus Kapitel 5.2. Ein Wert von 0.01 (t/Step) zum Beispiel bedeutet, dass ein Feuer, welches auf einem Feld mit 1.5t Brennmaterial anfängt zu brennen, 150 Steps lang brennt, bevor es verlöscht und das Brennmaterial wieder anfängt nachzuwachsen.

Der Einfluss dieser Veränderung der Variable „consumeFuel“ muss jedoch auch gemessen und quantifiziert werden können, damit Aussagen über die neue Umweltsituation gemacht werden können. Im Kapitel 2.2 wurden einige solche globale wie auch lokale Verfahren erläutert. Aus Performancegründen ist es in einer sinnvollen Zeit nicht möglich, globale Masse wie den Moran's I oder den Geary's C zu berechnen, was daran liegt, dass diese Verfahren durch die Verrechnung von jedem Feld mit jedem anderen Feld einen quadratischen Aufwand $O(n^2)$ haben, und dies bei 200'000m^2 bedeutet, dass bei einer Auflösung von 1m^2 pro Steps $4 \cdot 10^{10}$ Rechenschritte nötig wären. Um diesen Aufwand reduzieren zu können, wurde ein lokales Mass ähnlich dem lokalen Moran (vgl. Kapitel 2.2.3) gewählt, welches über die Fläche und alle Steps gemittelt wird. Für die Berechnung der räumlichen Autokorrelation wird folgende Formel verwendet:

Formel 5.1
$$I = \frac{\sum_{t=1}^{n} \frac{\sum_{i=1}^{m} z_{ti} \sum_j w_{tij} z_{tj}}{\sum_i z_i}}{n}$$

Diese Formel sieht vordergründig kompliziert aus, sollte jedoch klarer werden, wenn man die relativ starke Verschachtelung auseinander nimmt:

Formel 5.2
$$I = \frac{\sum_{t=1}^{n} G_t}{n}$$

Formel 5.3
$$G_t = \frac{\sum_i L_i}{\sum_i z_i}$$

Formel 5.4
$$L_i = z_i \sum_j w_{ij} z_j$$

Betrachtet man Formel 5.4, so entspricht dies dem lokalen Moran, wie er in Formel 2.18 zu finden ist. Dieser wird über alle Untersuchungen ermittelt und durch die Summe aller Werte z_i dividiert (Formel 5.3), was zu einem globalen Durchschnittswert für einen einzelnen Step führt. Diese Mittelwerte werden wiederum über alle Steps gemittelt (Formel 5.2), womit man einen Gesamtdurchschnitt über die Zeit und das gesamte Untersuchungsgebiet erhält. Weiter muss, wie in Kapitel 2.2.3 beschrieben, definiert werden, welche Nachbarschaftswerte bei einer Definition eines lokalen Korrelationsmasses mit einbezogen werden. In diesen Untersuchungen werden dafür die Nachbarzellen mit einer Distanz kleiner als 1m zum Zentrum einbezogen. Dies ergibt gerade wieder die Von-Neumann-Nachbarschaft, wie sie in Abbildung 3.4 dargestellt wird und somit folgende standardisierte Gewichtsmatrix w:

0	0	0	0	0
0	0	0.25	0	0
0	0.25	L_i	0.25	0
0	0	0.25	0	0
0	0	0	0	0

Tabelle 5.25: Gewichtsmatrix w

Die vier Von-Neumann-Nachbarn der zu berechnenden Zelle wurden mit Gewichten von je 0.25 angegeben. Dies ist wie beschrieben die schon standardisierte Gewichtung, welche aus der folgenden Formel abgeleitet wurde, geht man von einem Standardgewicht von 1 aus:

Formel 5.5

$$\frac{1}{\sum w} = \frac{1}{4}$$

Da es sich beim Ereignislayer, wie in Abbildung 3.5 zu sehen ist, um einen binären Layer handelt (0 = kein Ereignis/Feuer, 1 = Ereignis/Feuer), entspricht das lokale Mass L_i gerade der durchschnittlichen Wahrscheinlichkeit, dass sich auf einer Von-Neumann-Nachbarzelle einer Zelle mit einem Ereignis auch ein Ereignis befindet:

0	0	0	0	0
0	0	0.25	0	0
0	0.25	L_i	0.25	0
0	0	0.25	0	0
0	0	0	0	0

Tabelle 5.26: Beispiel eines lokalen Korrelationsmasses L_i
Die orangen Flächen bedeuten Ereignis/Feuer (=1), weisse Flächen sind ausserhalb des Ereignisses (=0). Die Zahlen sind wiederum die Gewichte w aus Tabelle 5.25.

Setzt man die Zahlen in die Formel 5.4 ein, erhält man einen Wert von 0.75. Wie schon beschrieben, entspricht dies genau der Wahrscheinlichkeit eines Ereignisses auf einer Von-Neumann-Nachbarzelle. Macht man diese Berechnungen für alle Ereigniszellen, so bekommt man folgende L_i für die einzelnen Zellen:

0	0	0	0	0
0	0	0	0.25	0
0	0.5	0.75	0.75	0
0	0.5	0.75	0.5	0
0	0	0	0	0

Tabelle 5.27: Beispiel mehrerer lokaler Korrelationsmasse L_i in einem Ereignis
Die orangen Flächen bedeuten Ereignis/Feuer (=1), weisse Flächen sind ausserhalb des Ereignisses (=0). Die Zahlen repräsentieren die berechnete lokale Autokorrelation L_i.

Fügt man diese Werte in die Formel 5.3 ein, kommt man auf einen Wert von 0.571. Dies ist die Wahrscheinlichkeit, dass sich für eine beliebige Zelle im Ereignis die Nachbarszelle auch in einem Ereignis befindet. Dieser Wert, ein globales Korrelationsmass, kann zwischen 0 und 1 variieren, wobei 0 einer totalen negativen Autokorrelation und 1 einer totalen positiven Autokorrelation der

Ereignisse entsprechen würde. Ein Wert von 0.5 würde bei keiner räumlichen Autokorrelation oder eben einer zufälligen Verteilung der Ereignisflächen erwartet. Ein sehr interessanter Nebeneffekt, der durch den binären Layer (0/1) entsteht, ist, dass nur die räumliche Autokorrelation von Ereignisflächen unabhängig von den nicht von einem Ereignis betroffenen Zellen berechnet wird (eine Multiplikation mit 0 gibt immer 0). Dies ist insofern für die Simulationen sinnvoll, da diese Flächen für die räumliche Autokorrelation der Ereignisse nicht von unmittelbarem Interesse in Bezug auf die Performance des Geosensor Network sind.
Fügt man am Schluss die globalen Korrelationsmasse G_t in Formel 5.2 ein, erhält man die durchschnittliche globale Korrelation der Ereigniszellen über die gesamte Dauer einer Simulation. Diese Werte stimmen so für die gewählte Auflösung von $1m^2$. Wie in Kapitel 2.2 bezüglich des MAUP beschrieben wurde, kann eine Variation dieser Auflösung diesen Korrelationswert erheblich verändern. Obwohl die verwendeten Daten hier in der Simulation bei der Berechnung der Korrelation nicht vorher aggregiert wurden, ist eine Auflösung von $1m^2$ schon bei der Berechnung der Umgebung als „zufällig“ zu interpretieren, könnte die gleiche Simulation doch auch mit $0.5m^2$ oder $0.1m^2$ Auflösung durchgeführt werden. Darum soll dieses Mass auch nicht als ratio-skalierte sondern vielmehr als ordinal skalierte Werte interpretiert werden.

5.7.2 Erwartungen

5.7.2.1 Veränderung der Umweltsituation

Um die Erwartungen bezüglich des Verhaltens der beiden Algorithmen formulieren zu können, muss zuerst das Verhalten der Umwelt auf die Veränderung des Parameters „consumeFuel“ überlegt werden. Dadurch, dass bei einer Verdoppelung des Wertes ein Waldstück doppelt so schnell abbrennt, wird das Feuer auch nur halb so lange brennen. Aus diesem Grund kann davon ausgegangen werden, dass sich bei einer Erhöhung des Wertes die Gesamtfläche der Feuer verkleinern wird. Es kann jedoch nicht davon ausgegangen werden, dass sich bei einer Verdoppelung des Wertes die Fläche halbieren wird und umgekehrt. Die Ursache dafür, wie schon in Kapitel 5.4 erwähnt, ist, dass die Wahrscheinlichkeit, dass ein neues Feuer entsteht, grösser ist, wenn die Gesamtfeuerfläche kleiner ist (wenn ein Blitz in ein Feuer schlägt, wird dadurch kein neues Feuer entfacht). Gleichzeitig wird sich mit der Verkleinerung der Gesamtereignisfläche wie auch jedes einzelnen Ereignisses die räumliche Autokorrelation verändern. Werden die Ereignisse kleiner, wird der im vorherigen Kapitel beschriebene Wert für die räumliche Autokorrelation kleiner, bei einer Vergrösserung der Ereignisse dementsprechend grösser. Es kann jedoch davon ausgegangen werden, dass, weil im Verhältnis zur Gesamtgrösse eines Ereignisses die gewählte Auflösung von $1m^2$ sehr hoch ist, die Werte der räumlichen Autokorrelation allgemein sehr hoch werden dürften. So ergibt sich beim Beispiel aus Tabelle 5.27 schon bei einem Ereignis von $7m^2$ Grösse ein Korrelationswert von 0.571. Jedoch wurde schon im vorherigen Kapitel beschrieben, dass es wohl sinnvoller ist, diese Werte als ordinal skaliert zu betrachten, damit das MAUP reduziert werden kann. Weiter muss jedoch auch noch ein anderer Aspekt mit einberechnet werden: Durch eine Veränderung der Lebensdauer eines Feuers wird sich auch zeitliche Dynamik der Ereignisse verändern.

5.7.2.2 Vergleich des durchschnittlichen Energieverbrauchs der beiden Algorithmen

Wie in Kapitel 5.4 herausgefunden wurde, variiert die durchschnittliche Belastung der Knoten bei beiden Algorithmen mit der durchschnittlichen Ereignisgrösse. Beim dezentralen Algorithmus ist die relative Ereignisfläche zur Gesamtfläche hauptsächlich ausschlaggebend für die Performance, beim zentralen Algorithmus ist es die absolute Ereignisfläche. Da sich jedoch die relative wie auch die

absolute Ereignisfläche bei einer fixen Umweltgrösse parallel zueinander verändern, bleiben die beiden Werte vergleichbar.
Beim zentralen Algorithmus ist zu erwarten, dass die räumliche Autokorrelation oder eben die Grösse der einzelnen Ereignisse selber keinen Einfluss auf die Performance haben wird. Dies ist dadurch zu begründen, dass beim zentralen Algorithmus alle Informationen unabhängig von den Nachbarknoten gesendet werden und somit räumliche Beziehungen innerhalb des Netzwerkes keine Rolle spielen dürften. Da jedoch durch die Ereignisdauer auch die zeitliche Komponente betroffen ist, kann davon ausgegangen werden, dass bei tiefer räumlicher Autokorrelation durch die verkürzte Dauer eines Ereignisses und die daraus höhere Zustandsänderungsrate der Energieverbrauch sich trotzdem erhöhen könnte. Wie gross dieser Einfluss sein wird, ist schwierig abzuschätzen, da zu viele zufällige Faktoren in die Umwelt hineinspielen, als dass diese umfassend erfasst werden könnten. Da jedoch vor allem die relative Performance der beiden Algorithmen zueinander interessiert, wird hier keine Vorhersage für das Verhalten des zentralen Algorithmus bezüglich des durchschnittlichen Verbrauchs über die verschiedenen Simulationen gemacht.
Beim dezentralen Algorithmus scheinen die Sachverhalte noch komplexer zu sein. Natürlich wird sich die mittlere Belastung hier auch der durchschnittlichen Gesamtereignisgrösse anpassen, jedoch wird die Grösse der einzelnen Ereignisses und somit die räumliche Autokorrelation wohl zusätzlich einen entscheidenden Einfluss auf die Performance ausüben. Sind die einzelnen Ereignisse sehr klein (niedrige positive Autokorrelation), dürften die Vorteile des dezentralen Algorithmus nicht allzu gross ausfallen, da so nur relativ wenige Knoten miteinander verbunden sind. Auch würde es zu einer Häufung von Zusammenschlüssen und Teilungen von Ereignissen kommen, was wiederum mit zusätzlichem Energieverbrauch verbunden ist. Im extremsten Fall würden sehr kleine Ereignisse bedeuten, dass diese jeweils nur von einem einzigen Knoten erkannt würden. Dieser wäre dann automatisch ein Head-Node, und es könnte keine Aggregation von Daten stattfinden, womit der dezentrale Algorithmus schlussendlich gleich funktionieren würde wie der zentrale Algorithmus, wobei durch den Kontakt zu den Nachbarsknoten sogar noch ein Mehraufwand bei der dezentralen Variante resultieren würde. Geht man ins andere Extrem, dass ein Ereignis so gross wird, dass eine perfekte positive räumliche Autokorrelation vorherrscht und das Ereignis die gesamte Untersuchungsfläche ausfüllt, wären die Vorteile des dezentralen Algorithmus auch wieder zunichte, denn dann würde der Head-Node sozusagen die gleiche Funktion übernehmen wie beim zentralen Algorithmus der Sink-Node, jedoch vielleicht mit einer zufällig besseren Platzierung näher am Zentrum des Ereignisses. Trotzdem würden die Informationswege innerhalb des Ereignisses sehr weit, und da sogar mit einem engmaschigeren Graphen gearbeitet wird (RNG statt UDG), dürfte der dezentrale Algorithmus den Aufwand des zentralen Algorithmus sogar übertreffen, wobei der Informationsaustausch für die Erhaltung des lokalen Netzwerkes innerhalb des Ereignisses noch gar nicht dazugerechnet ist.
Es ist also zu erwarten, dass der dezentrale Algorithmus bei mittlerer räumlicher Autokorrelation gegenüber dem zentralen Algorithmus bezüglich des durchschnittlichen Verbrauchs am besten performen dürfte. Bei sehr hoher (grosse Ereignisse) beziehungsweise sehr niedriger (kleine Ereignisse) räumlicher Autokorrelation dürfte der dezentrale Algorithmus aus oben genannten Gründen schlechter abschneiden als der zentrale, wobei bei der hier getesteten Variabilität der räumlichen Autokorrelation dieses Szenario noch nicht eintreten dürfte. Ob diese Kurve mit einem relativen Optimum des dezentralen Algorithmus bei einer mittleren Autokorrelation (Parabel mit Öffnung nach oben) schlussendlich so zu sehen ist, kann auch nur sehr schwierig vorausgesagt werden. Hier spielen zusätzlich sehr viele Faktoren wie die Sensordichte im Verhältnis zur Ereignisgrösse oder der verwendete Graph voraussichtlich eine zu grosse Rolle, so dass eine genaue

Vorhersage sehr schwierig ist. Auch eine isolierte Aussage zum dezentralen Algorithmus bezüglich des Verhaltens auf die veränderte Umweltsituation ist wie beim zentralen Algorithmus kaum möglich, da wiederum das genaue Verhalten der Umwelt nur sehr schwierig abgeschätzt werden kann und zu viele Einflussfaktoren den Energieverbrauch schlussendlich beeinflussen.

5.7.2.3 Vergleich des maximalen Energieverbrauchs der beiden Algorithmen

Beim maximalen Energieverbrauch ist es aus denselben Gründen wie beim durchschnittlichen Verbrauch sehr schwierig, Vorhersagen zum Verhalten der einzelnen Algorithmen zu machen. Für den zentralen Algorithmus wird jedoch erwartet, dass er sich analog zum durchschnittlichen Energieverbrauch verhalten wird. Ist also der Durchschnittsverbrauch hoch, wird auch der Energieverbrauch des maximal belasteten Knotens hoch sein, da in diesem Experiment die Anzahl Knoten wie auch die Dichte des Geosensor Network konstant bleiben.
Beim dezentralen Algorithmus sieht es ein bisschen anders aus. Voraussichtlich werden bei grossen Ereignissen mit einer hohen durchschnittlichen räumlichen Autokorrelation die Head-Nodes länger dieselben bleiben sowie die Informationsmenge konstanter auf einem tiefen Niveau verweilen, so dass weniger Informationen an den Sink-Node übermittelt werden. Sollte sich jedoch die durchschnittliche Gesamtereignisgrösse bei durchschnittlich grösseren Ereignissen massiv erhöhen, könnte dies entgegenwirkend den positiven Effekt kompensieren oder im schlechtesten Fall sogar überkompensieren, da sich, wie in Kapitel 5.4 beschrieben, die Belastung der maximal belasteten Knoten beim dezentralen Algorithmus bei in etwa gleichbleibender räumlicher Autokorrelation parallel zur absoluten Ereignisgrösse verändert. Insofern wird wieder viel davon abhängen, wie sich die Umwelt verhält. Würde die durchschnittliche Gesamtereignisgrösse konstant gehalten und nur die räumliche Autokorrelation verändert werden, wäre die Tendenz zu tieferer Belastung bei grösseren Ereignissen wohl um einiges klarer.
Beim Vergleich der beiden Algorithmen, um den es hauptsächlich bei dieser Untersuchung geht, kann jedoch davon ausgegangen werden, dass der dezentrale Algorithmus bei den maximal belasteten Knoten tendenziell mit zunehmender durchschnittlicher Grösse, also höherer Autokorrelation, gegenüber dem zentralen Algorithmus immer besser abschneiden wird, wobei eine maximale relative Effizienzdifferenz zu erwarten ist, ab der eine weitere Erhöhung der räumlichen Autokorrelation keine weitere Verbesserung des dezentralen gegenüber dem zentralen Algorithmus mehr zu erwarten ist.

5.7.3 Ergebnisse

5.7.3.1 Simulation 5.7-1: Verbrennung von 0.08t Brennmaterial pro Step

Als Windrichtung wurde der Zufallswert 4.05 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.61
Osten: 0.57
Süden: 0.29
Westen: 0.33

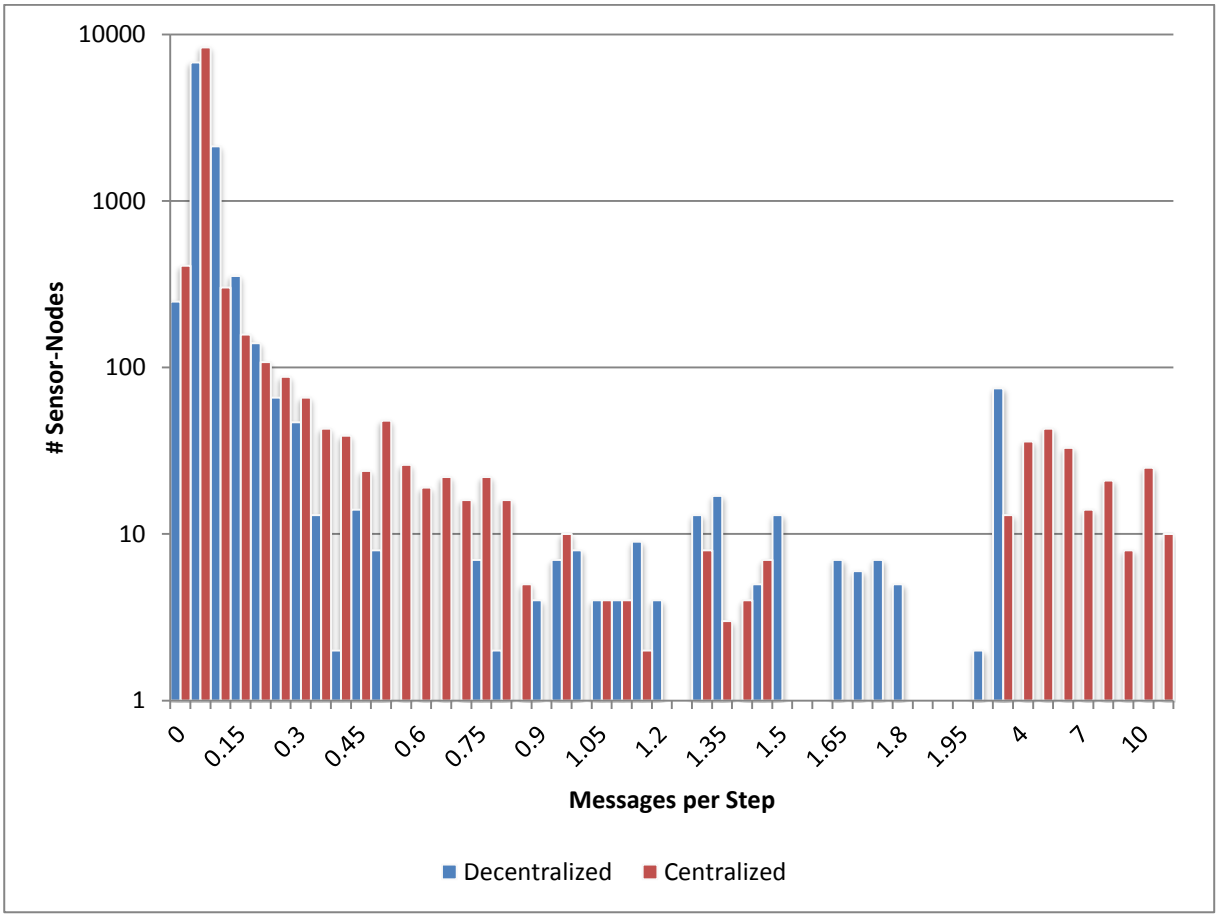


Diagramm 5.33: Simulation 5.7-1
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.075	0.153	0.488
Median	0.031	0.002	14.619
Std Dev	0.26	0.897	0.29
Max	3.009	10.342	0.291

Tabelle 5.28: Simulation 5.7-1
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.7.3.2 Simulation 5.7-2: Verbrennung von 0.04t Brennmaterial pro Step

Als Windrichtung wurde der Zufallswert 5.41 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.6
Osten: 0.32
Süden: 0.3
Westen: 0.58

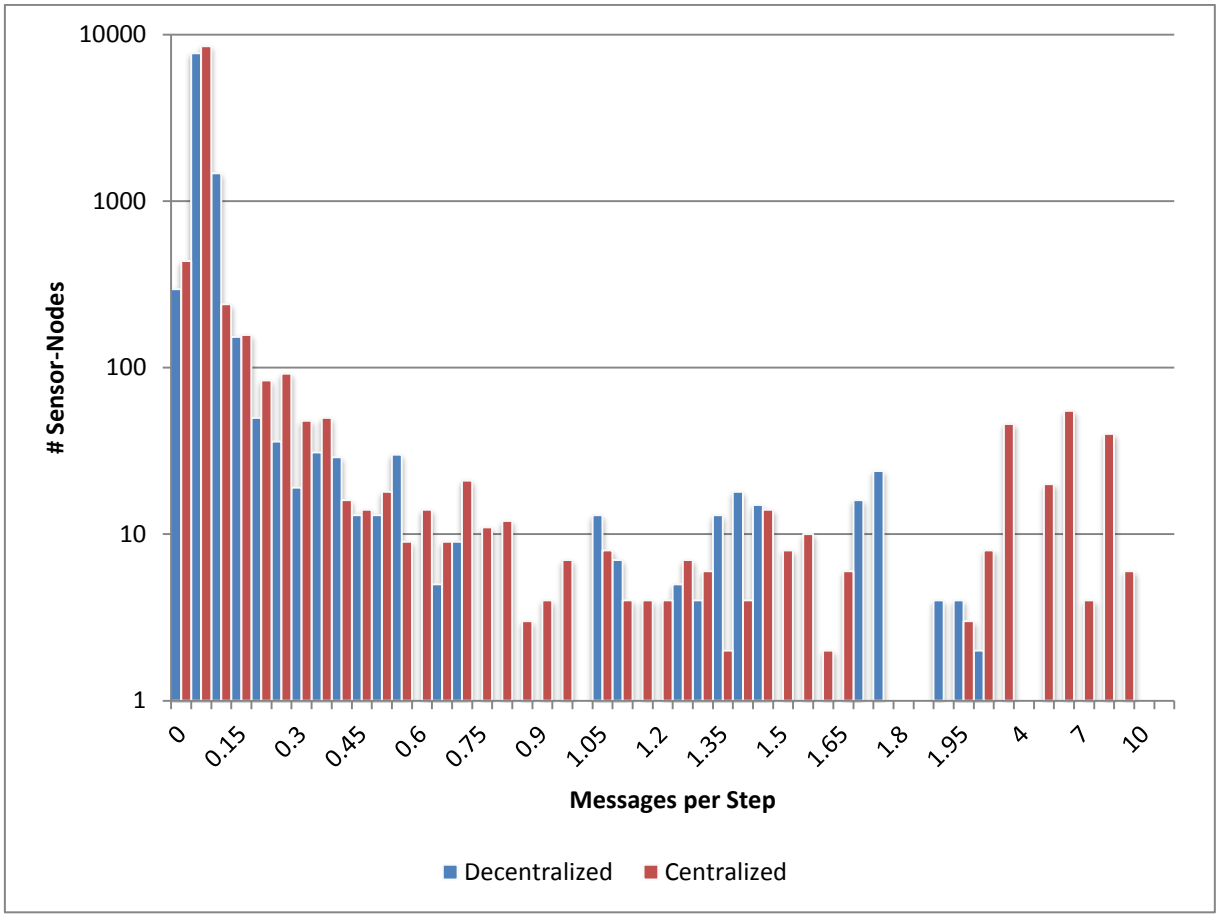


Diagramm 5.34: Simulation 5.7-2
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.054	0.121	0.441
Median	0.025	0.002	16.7
Std Dev	0.172	0.717	0.239
Max	1.966	8.137	0.242

Tabelle 5.29: Simulation 5.7-2
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.7.3.3 Simulation 5.7-3: Verbrennung von 0.02t Brennmaterial pro Step

Als Windrichtung wurde der Zufallswert 5.93 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.52
Osten: 0.26
Süden: 0.38
Westen: 0.64

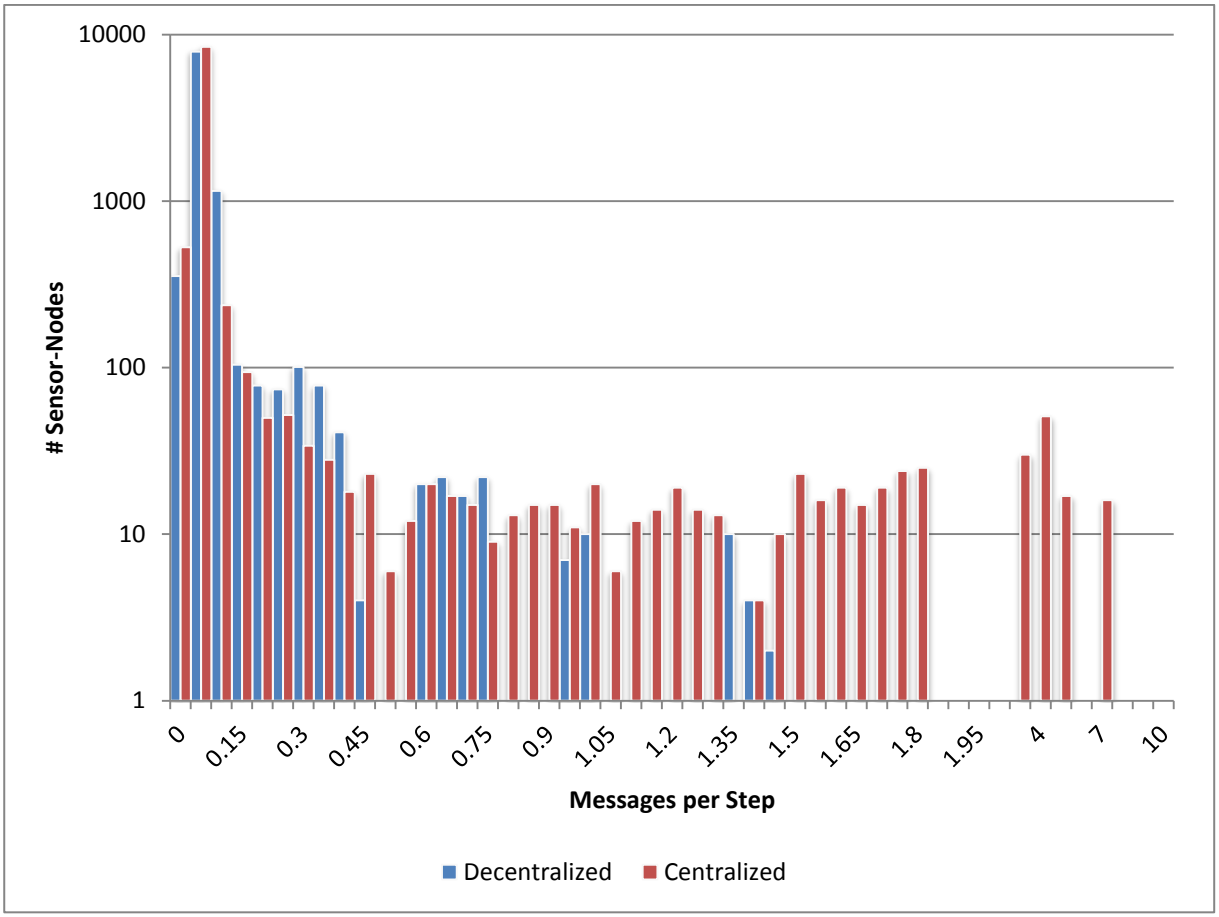


Diagramm 5.35: Simulation 5.7-3
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.043	0.101	0.425
Median	0.021	0.001	17.875
Std Dev	0.101	0.484	0.208
Max	1.429	6.962	0.205

Tabelle 5.30: Simulation 5.7-3
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.7.3.4 Simulation 5.7-4: Verbrennung von 0.01t Brennmaterial pro Step

Als Windrichtung wurde der Zufallswert 3.2 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.46
Osten: 0.65
Süden: 0.44
Westen: 0.25

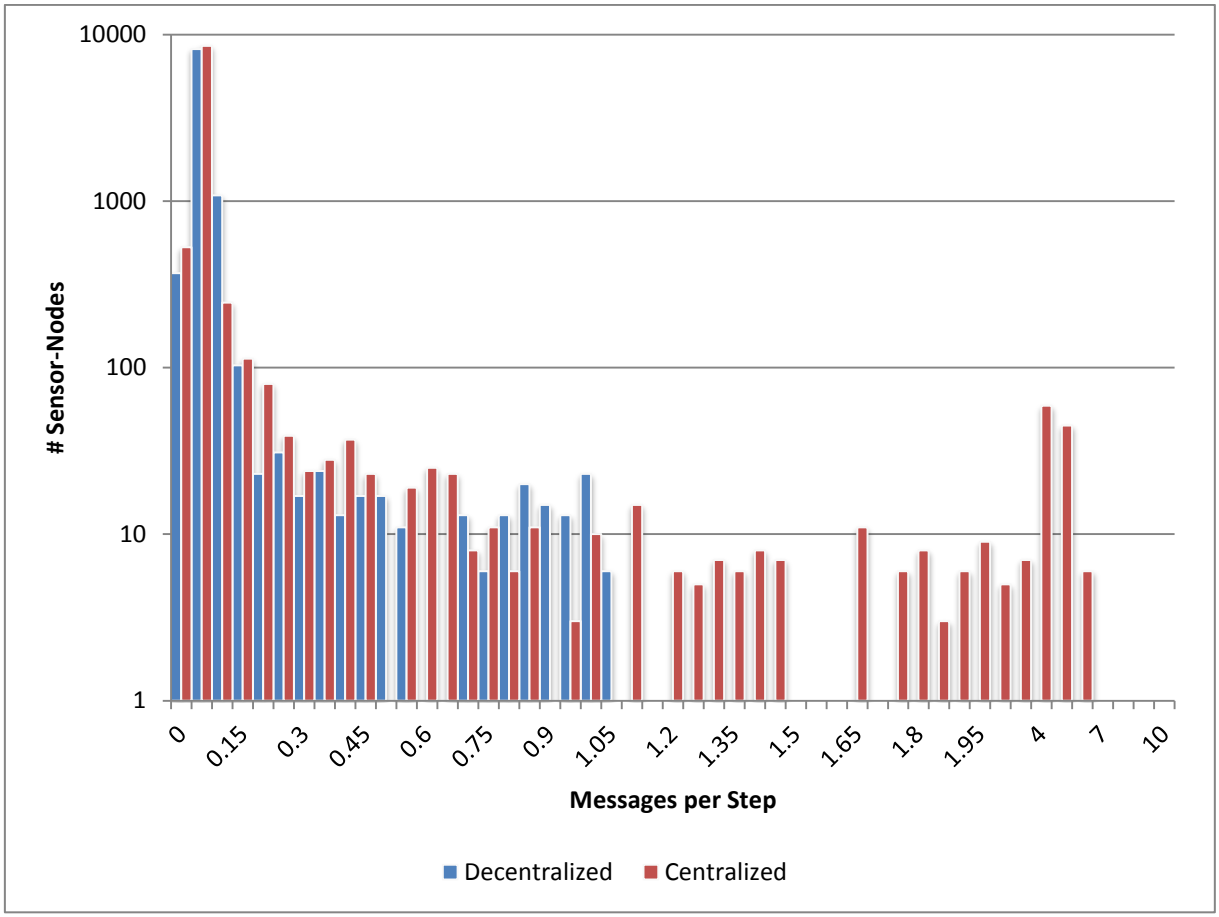


Diagramm 5.36: Simulation 5.7-4
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.037	0.082	0.456
Median	0.019	0.001	17.591
Std Dev	0.098	0.463	0.212
Max	1.054	5.07	0.208

Tabelle 5.31: Simulation 5.7-4
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.7.3.5 Simulation 5.7-5: Verbrennung von 0.005t Brennmaterial pro Step

Als Windrichtung wurde der Zufallswert 3.45 berechnet, wobei folgende Ausbreitungswahrscheinlichkeiten eines Feuers in die verschiedenen Himmelsrichtungen resultieren (Berechnung zu finden in Kapitel 3.3):

Norden: 0.51
Osten: 0.64
Süden: 0.39
Westen: 0.26

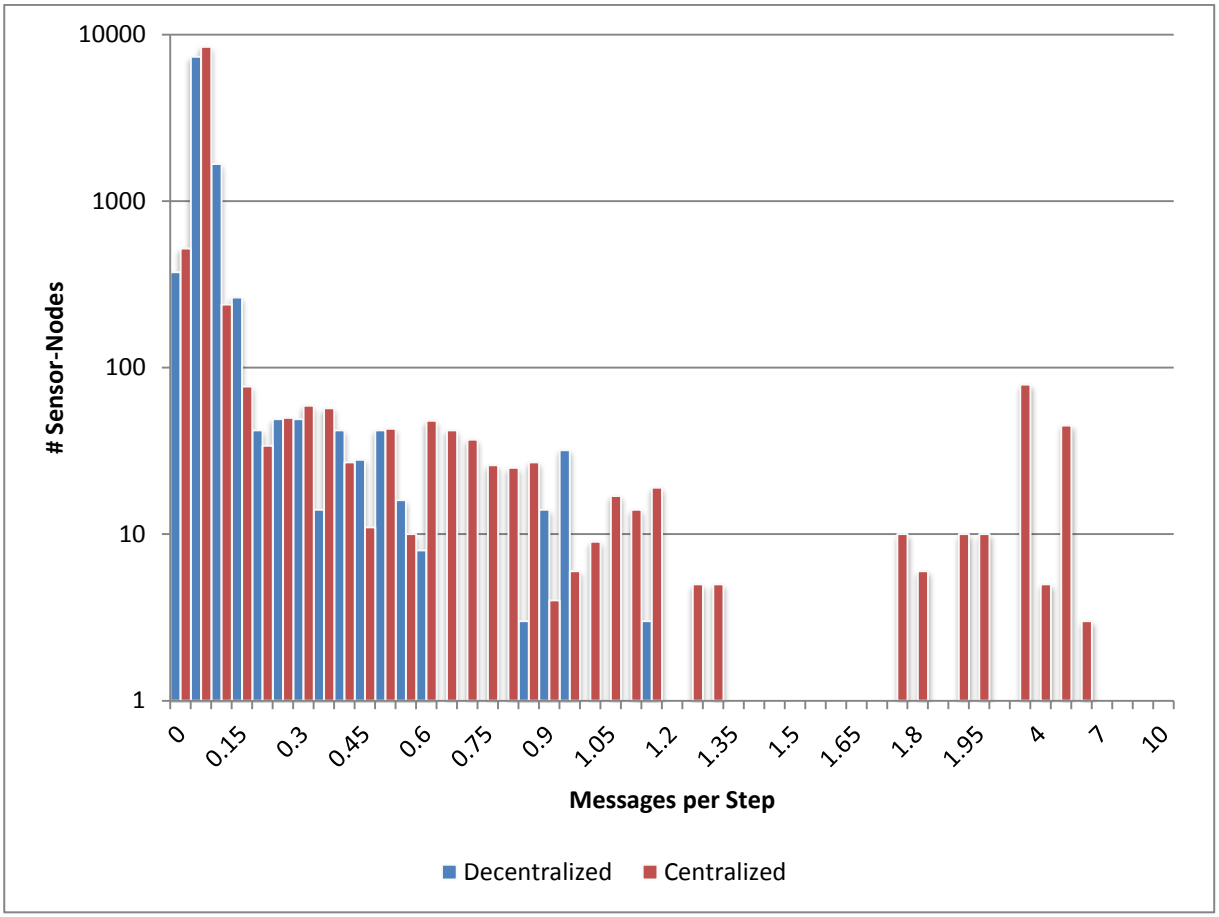
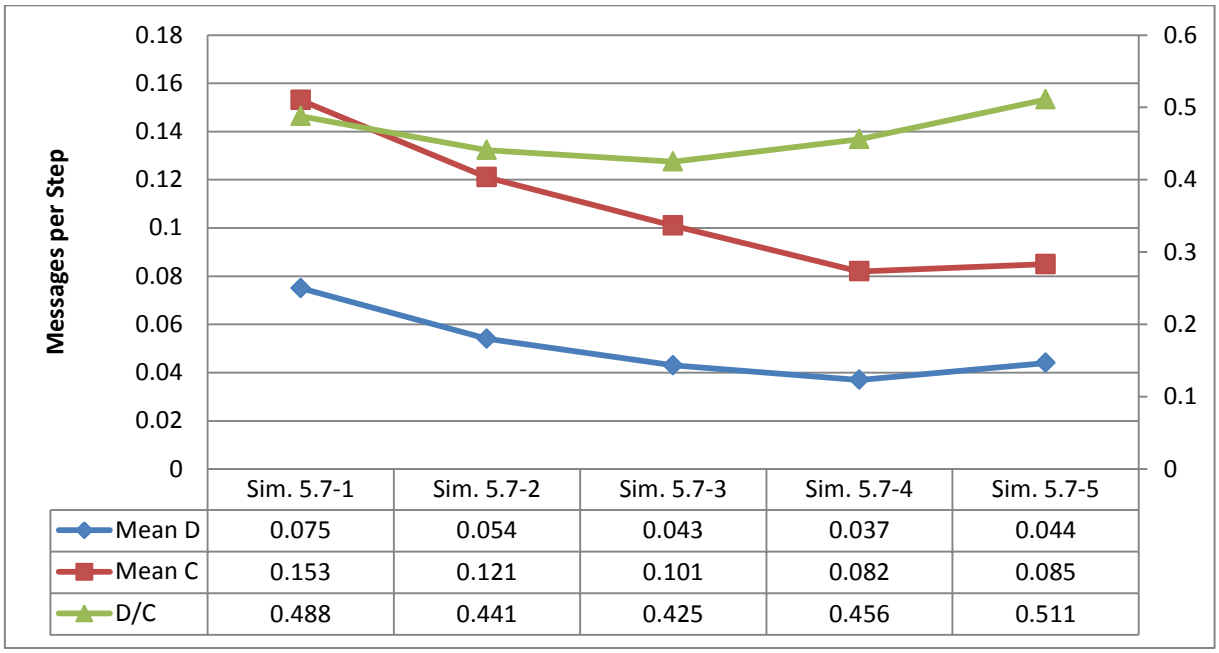


Diagramm 5.37: Simulation 5.7-5
Die Werte auf der x-Achse zeigen die oberen Klassengrenzen der einzelnen Balken an. Die y-Achse zeigt eine logarithmische Skala.

Messages per Step	Decentralized	Centralized	D/C
Mean	0.044	0.085	0.511
Median	0.025	0.001	23.136
Std Dev	0.088	0.411	0.214
Max	1.108	5.543	0.2

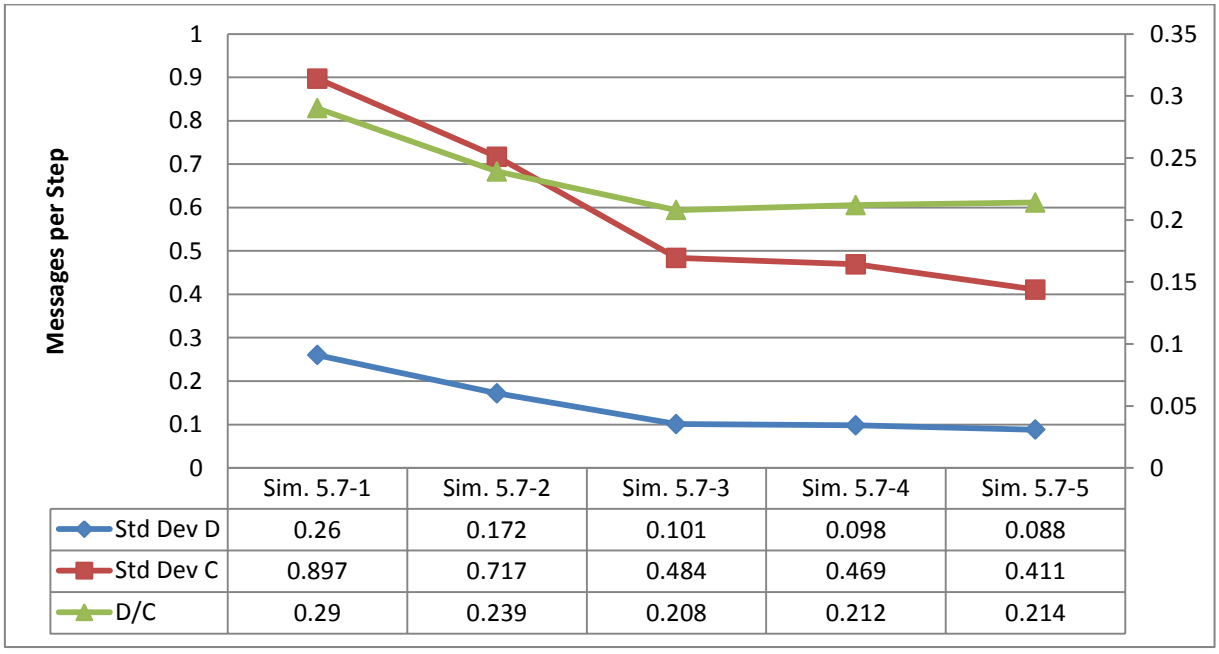
Tabelle 5.32: Simulation 5.7-5
Die Tabelle zeigt die vier aus der Simulation resultierenden Werte Mean (=Mittelwert), Median, Std Dev (=Standardabweichung) und Max (maximal belasteter Knoten) der beiden Algorithmen. In der letzten Spalte (D/C) werden die Werte von der Spalte „decentralized“ durch die Werte der Spalte „centralized“ dividiert, womit der relative Energieverbrauch des dezentralen Algorithmus im Verhältnis zum zentralen Algorithmus angegeben wird.

5.7.3.6 Vergleich Simulation 5.7-1 bis Simulation 5.7-5



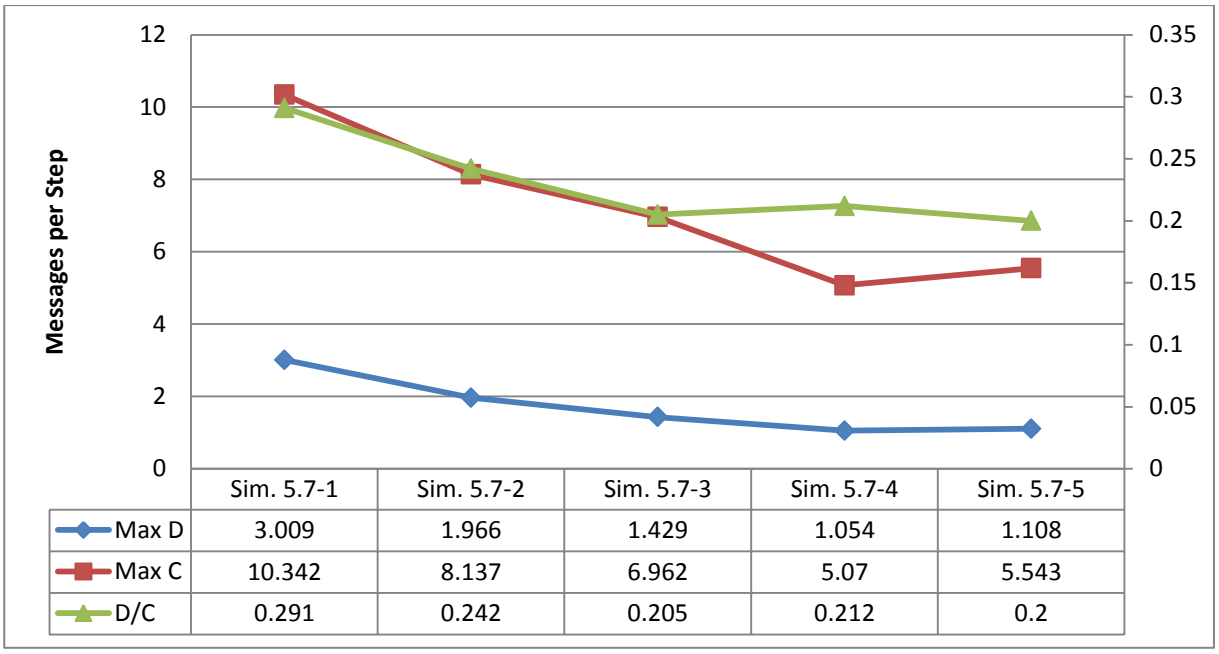
	Sim. 5.7-1	Sim. 5.7-2	Sim. 5.7-3	Sim. 5.7-4	Sim. 5.7-5
Mean D	0.075	0.054	0.043	0.037	0.044
Mean C	0.153	0.121	0.101	0.082	0.085
D/C	0.488	0.441	0.425	0.456	0.511

Diagramm 5.38: Durchschnittsverbrauch der Simulationen 5.7-1 bis 5.7-5 im Vergleich
Die Abkürzung D steht für den dezentralen Algorithmus, C für den zentralen; Auf der x-Achse ist die Simulation angegeben. Die blaue und rote Kurve sind in absoluten Werten angegeben (linke y-Achse), die grüne Kurve zeigt relative Werte (rechte y-Achse).



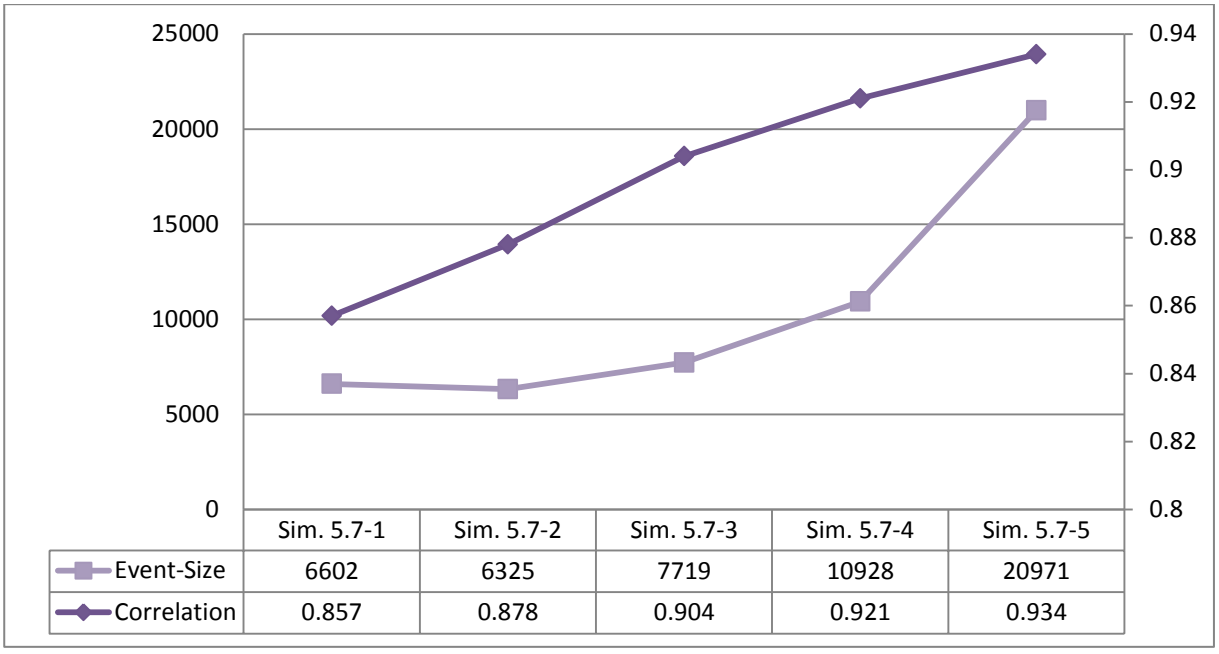
	Sim. 5.7-1	Sim. 5.7-2	Sim. 5.7-3	Sim. 5.7-4	Sim. 5.7-5
Std Dev D	0.26	0.172	0.101	0.098	0.088
Std Dev C	0.897	0.717	0.484	0.469	0.411
D/C	0.29	0.239	0.208	0.212	0.214

Diagramm 5.39: Standardabweichung der Simulationen 5.7-1 bis 5.7-5 im Vergleich
Die Abkürzung D steht für den dezentralen Algorithmus, C für den zentralen; Auf der x-Achse ist Simulation angegeben. Die blaue und rote Kurve sind in absoluten Werten angegeben (linke y-Achse), die grüne Kurve zeigt relative Werte (rechte y-Achse).



	Sim. 5.7-1	Sim. 5.7-2	Sim. 5.7-3	Sim. 5.7-4	Sim. 5.7-5
Max D	3.009	1.966	1.429	1.054	1.108
Max C	10.342	8.137	6.962	5.07	5.543
D/C	0.291	0.242	0.205	0.212	0.2

Diagramm 5.40: Maximal belastete Knoten der Simulationen 5.7-1 bis 5.7-5 im Vergleich.
Die Abkürzung D steht für den dezentralen Algorithmus, C für den zentralen; Auf der x-Achse ist die Simulation angegeben. Die blaue und rote Kurve sind in absoluten Werten angegeben (linke y-Achse), die grüne Kurve zeigt relative Werte (rechte y-Achse).



	Sim. 5.7-1	Sim. 5.7-2	Sim. 5.7-3	Sim. 5.7-4	Sim. 5.7-5
Event-Size	6602	6325	7719	10928	20971
Correlation	0.857	0.878	0.904	0.921	0.934

Diagramm 5.41: Durchschnittliche Grösse und Korrelation der Ereignisse
Dieses Diagramm zeigt die durchschnittlichen absoluten Grössen der Ereignisse (hellviolett) und die Korrelation (dunkelviolett) gemäss Beschreibung in Kapitel 5.7.1 für die Simulationen 5.7-1 (ganz links) bis 5.7-5 (ganz rechts). Die linke y-Achse gilt für die Ereignisgrösse, die rechte für die Korrelation.

5.7.4 Diskussion

5.7.4.1 Veränderung der Umweltsituation

Obwohl es in dieser Arbeit eigentlich darum geht, die Performance der Algorithmen und nicht die Veränderung der Umweltsituation zu beurteilen, soll hier dennoch kurz zusammengefasst werden, wie sich die Umwelt durch die Veränderung des Parameters „consumeFuel“ verändert, da dies auch einen grossen Einfluss auf die Interpretation der Daten des Geosensor Network hat.

Wie aus Diagramm 5.41 herausgelesen werden kann, hat sich die Umwelt in etwa so verhalten, wie es erwartet wurde. Durch das langsamere Abbrennen des Brennmaterials wuchs die durchschnittliche Gesamtereignisgrösse von 6602m^2 bei Simulation 5.7-1 auf 20971m^2 bei Simulation 5.7-5 an. Aus den in den Erwartungen beschriebenen Gründen wuchs die durchschnittliche Ereignisgrösse jedoch nicht gleich schnell an wie sich die Brandzeit verlängerte. Immerhin brennt ein Feuer in Simulation 5.7-5 rund 16 Mal länger als in Simulation 5.7-1. Dennoch ist ein markanter Sprung der Brandfläche von Simulation 5.7-4 zu Simulation 5.7-5 auszumachen. Inwiefern diese beinahe Verdoppelung der Ereignisfläche auf die Performance der beiden Algorithmen durchschlägt, gilt es in den beiden nächsten Kapiteln zu beurteilen. Die relative Gesamtereignisgrösse wurde hier nicht speziell betrachtet, weil sich diese genau gleich wie die absolute Ereignisgrösse verändert, da die Ausmasse des Untersuchungsgebiets konstant gehalten wurden.

Bezüglich der räumlichen Autokorrelation haben sich die Annahmen auch bewahrheitet. Dass auch schon bei Simulation 5.7-1 ein Wert von 0.857 zu beobachten ist, zeigt, dass es sich schon hier um hochgradig autokorrelierte Ereignisse handelt (unter Annahme der räumlichen Auflösung von 1m^2). Jedoch zeigt sich, dass sich diese Werte von Simulation 5.7-1 bis Simulation 5.7-5 permanent vergrössert haben. Wie schon im Experiment-Teil angetönt, sollten diese Werte aber nicht quantitativ verglichen werden, da das MAUP hier ein zu grosses Problem darstellt und genaue Zahlen somit nicht verglichen werden sollen.

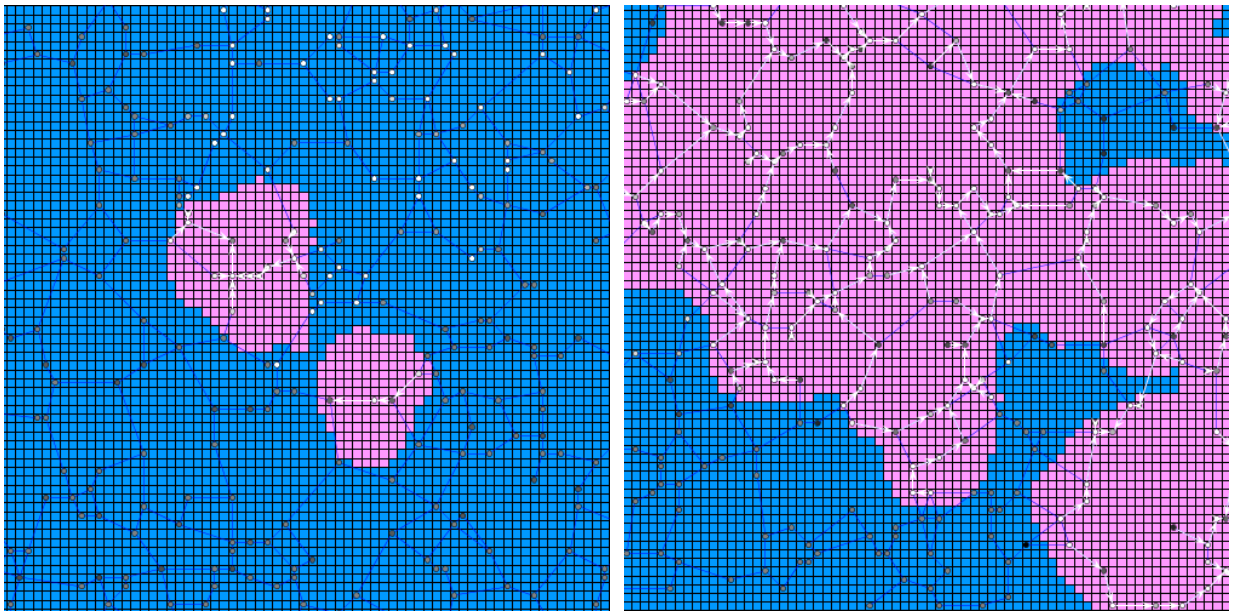


Abbildung 5.3: Vergleich unterschiedlicher räumlicher Autokorrelationen von Ereignissen
Auf der linken Abbildung sind zwei typische Ereignisse aus der Simulation 5.7-1 mit einem durchschnittlichen Korrelationsmass von 0.857 gezeigt. Auf der rechten Abbildung ist ein Ausschnitt eines Ereignisses aus Simulation 5.7-5 mit einem durchschnittlichen Korrelationsmass von 0.934 bei gleichem Massstab abgebildet. Die rosa Fläche ist von einem Ereignis/Feuer betroffen, die blaue Farbe steht für nicht betroffene Flächen.

5.7.4.2 Vergleich des durchschnittlichen Energieverbrauchs der beiden Algorithmen

Bei kleiner räumlicher Autokorrelation (kleine Ereignisse) scheinen beide Algorithmen einen relativ hohen durchschnittlichen Energieverbrauch aufzuweisen. Vergleicht man Simulation 5.7-1 mit Simulation 5.7-4 (Standardsetting), so weisen beide Algorithmen bei ersterer einen etwa doppelt so hohen Durchschnittsverbrauch auf. Da die durchschnittliche Gesamtereignisgrösse absolut wie auch relativ jedoch bei der Simulation 5.7-1 um fast 40% kleiner ist als bei der Simulation 5.7-4, müsste laut den Erkenntnissen aus Kapitel 5.4 der durchschnittliche Energieverbrauch eher ca. 40% geringer ausfallen als 100% höher. Da die Einflüsse der einzelnen Faktoren bei den Ergebnissen nicht voneinander getrennt werden können, kann über die Ursache dieses Verhaltens nur spekuliert werden. Durch eine Betrachtung der Dynamik der verschiedenen Simulationen lässt sich jedoch eine plausible Erklärung finden: Vergleicht man den Parameter „consumeFuel“ der beiden Simulationen 5.7-1 und 5.7-4, so sieht man, dass sich der Brennmaterialverbrauch zwischen den beiden Simulationen um den Faktor 8 unterscheidet. Dies bedeutet, dass ein Feuer bei Simulation 5.7-4 etwa 8 Mal länger brennt. Da die Sensorknoten nur eine Veränderung ihres Zustandes (in einem Ereignis/in keinem Ereignis) rapportieren, bedeutet dies einen potentiell achtmal höheren Energieverbrauch bei Simulation 5.7-1. Nimmt man ein theoretisches Extrem zu Hilfe, ist der Sachverhalt noch viel einfacher zu erklären: Würde ein Feuer kein Brennmaterial verbrauchen, würde es theoretisch unendlich lange brennen. Dadurch würde der Zustand eines Sensorknotens nur gerade einmal verändert. Aus diesem Grund könnte trotz einer grossen Ereignisfläche ein sehr niedriger Energieverbrauch resultieren. Genau dieses Verhalten der Feuer liess sich während den verschiedenen Simulationen so beobachten, kann jedoch hier nur über den Energieverbrauch der Knoten abgeleitet werden. So scheint der durchschnittliche Energieverbrauch der beiden Algorithmen in diesem Experiment nicht sehr eng mit der gesamten Ereignisgrösse oder der räumlichen Autokorrelation zu korrelieren, da die eben beschriebene Dynamik der Feuer einen grösseren Einfluss auf den Verbrauch ausübt.

Eine jedoch viel interessantere Schlussfolgerung lässt der Vergleich der beiden Algorithmen zu. Da der zentrale wie auch der dezentrale Algorithmus gleich stark vom dynamisch unterschiedlichen Verhalten der Ereignisse betroffen sind, lassen sich diese beiden Algorithmen relativ zueinander durchaus vergleichen. Denn wie in Kapitel 5.2 gezeigt wurde, ändert ein vermehrtes Auftreten von ähnlichen Ereignissen das Verhältnis zwischen den beiden Algorithmen nicht, denn die Belastung beider Algorithmen verändert sich um den gleichen Prozentsatz, was sich bei einer Division des einen durch den anderen wieder wegkürzt. Die Kurve des relativen Verbrauchs des dezentralen Algorithmus zum zentralen Algorithmus (grüne Kurve in Diagramm 5.38) zeigt ziemlich genau das auf, was erwartet wurde. Es scheint sich die Vermutung zu bewahrheiten, dass der dezentrale Algorithmus optimal bei einer mittleren räumlichen Autokorrelation bezüglich der Ereignisse, oder anders ausgedrückt, bei einer mittleren durchschnittlichen Ereignisgrösse zu arbeiten scheint. Jedoch soll hier dennoch erwähnt werden, dass es wohl als sehr glücklich zu bezeichnen ist, dass über nur fünf Simulationen genau dieses Optimum zufällig bei der mittleren Simulation 5.7-3 auftritt. Würde das Experiment bei Simulation 5.7-3 beginnen und die vier nachfolgenden Simulationen allesamt ein höheres oder tieferes räumliches Korrelationsmass aufweisen, hätte dieser Effekt hier so nicht eindeutig festgestellt werden können.

5.7.4.3 Vergleich des maximalen Energieverbrauchs der beiden Algorithmen

Wie erwartet verhält sich der Energieverbrauch des maximal belasteten Knotens beim zentralen Algorithmus ähnlich dem durchschnittlichen Energieverbrauch. Beim dezentralen Algorithmus ist dasselbe Phänomen auch zu erkennen, wobei wie erwartet bei der Simulation 5.7-5 wegen des enormen Anstieges der durchschnittlichen Gesamtereignisgrösse wohl der Verbrauch nochmals geringfügig angestiegen ist. Auch beim relativen Vergleich zueinander scheint der allgemeine Trend zu sein, dass bei höherer räumlicher Autokorrelation der Ereignisse der dezentrale Algorithmus immer besser abschneidet. Jedoch scheint bei einer relativen Belastung des am stärksten belasteten Knotens beim dezentralen Algorithmus bezogen auf den zentralen Algorithmus bei ca. 20% eine Grenze erreicht zu sein, wie dies von Simulation 5.7-3 bis Simulation 5.7-5 zu beobachten ist. Ein Grund dafür dürfte die Tatsache sein, dass der Informationsfluss an den Sink-Node bei einer weiteren Vergrösserung des Ereignisses prozentual nicht mehr abnimmt, da eine maximale Flächenänderung des Ereignisses definiert wurde, nach dem wieder eine Information an den Sink-Node gesendet wird. Da dieser Wert jedoch absolut und nicht relativ zur Ereignisfläche definiert wurde, bleibt die Informationsmenge bei weiterem Anwachsen des Ereignis konstant, wie es beim zentralen Algorithmus auch der Fall ist, weshalb die relative Differenz der Informationsdichte auch konstant bleiben dürfte.

5.7.4.4 Vergleich zum Verhalten auf Veränderung der Sensordichte

Vielleicht haben sich einige aufmerksame Leser schon die Frage gestellt, warum bei diesem Experiment nicht genau dieselben Ergebnisse herauskommen wie bei der Variation der Sensordichte, denn theoretisch hat eine Reduktion der durchschnittlichen Ereignisfläche auf die halbe Grösse eigentlich den gleichen Einfluss wie eine Halbierung der Sensordichte: Dasselbe Ereignis wird von nur noch von ca. halb so vielen Sensoren registriert. Diese Überlegung scheitert aber daran, dass bei beiden Varianten weiterhin alle anderen Parameter gleich bleiben. Als Beispiel kann man sich ein normales Auto und ein Modellauto vorstellen: Wenn das Modellauto zehn Mal kleiner ist, muss es, damit es ein realistisches Abbild des Originals ist, auch zehn Mal langsamer fahren, eine zehn Mal kleinere Reichweite besitzen etc., da es sonst zum Beispiel eine Distanz relativ zur Wagenlänge in nur einem Zehntel der Zeit absolvieren würde. So ist dies auch bei den Experimenten im Geosensor Network: Sollte eine Halbierung der durchschnittlichen Ereignisfläche die gleichen Ergebnisse liefern wie eine Halbierung der Sensordichte, müssten die Sendedistanz, die Ausbreitungsgeschwindigkeit des Feuers, die absolute Informationsmenge vom Head-Node zum Sink-Node, die Grösse des Untersuchungsgebietes, die räumliche Auflösung der simulierten Umwelt und alle anderen Parameter auch dementsprechend angepasst werden. Speziell bei der Grösse des Untersuchungsgebiets wird dies sehr offensichtlich, da bei der Variation der Sensordichte bei gleich bleibender Grösse die Anzahl der Sensorknoten halbiert werden musste, um eine geringere Dichte zu erhalten. Aus diesem Grund können diese beiden Experimente nicht einfach so verglichen werden, wie sich auch aus den Erwartungen sowie den Ergebnissen und den unterschiedlichen Argumentationen in der Diskussion ablesen lässt.

6 Schlussfolgerung

6.1 Erkenntnisse

Alle Ergebnisse dieser Arbeit beruhen auf dem Vergleich der beiden verwendeten Algorithmen. Diese Algorithmen standen in dieser Arbeit als Stellvertreter ihrer Gattung, der dezentralen sowie der zentralen Algorithmen. Da es jedoch eine Vielzahl an Algorithmen gibt und in der Zukunft noch geben wird, wäre es, wie in Kapitel 1.3 schon angemerkt, nicht korrekt, die Ergebnisse als absolut bezüglich dezentraler und zentraler Datenverarbeitung zu interpretieren, zumal die Dezentralität sowie die Zentralität eines Algorithmus wohl eher in einem Kontinuum verstanden werden müssten und nicht als fixe Einteilung. So müsste man ganz korrekt wohl eher von dezentraleren und zentraleren Algorithmen sprechen. Trotzdem lassen die Ergebnisse dieser Arbeit einige allgemeine Schlüsse qualitativer Art bezogen auf die in Kapitel 1.3 gestellten Forschungsfragen zu, wobei sich Kosten jeweils auf die Energiekosten beziehen und die Aussagen bezüglich dezentraler Methoden nicht als absolut sondern als relativ zu zentralen Methoden zu verstehen sind:

- Es gibt Anwendungen, in denen dezentrale Algorithmen die erforderlichen Informationen günstiger bereitstellen können.
- Dezentrale Algorithmen scheinen die Energiebelastung bei einem statischen Routing tendenziell besser auf die Knoten zu verteilen, was tendenziell einer längeren Lebensdauer des Geosensor Network gleichkommt.
- Durch dezentrale Verarbeitung von Daten lässt sich die gewünschte Informationsmenge steuern, womit ein für die Anwendung spezifisches optimales Gleichgewicht zwischen erforderlicher Informationsgenauigkeit und Energieverbrauch definiert werden kann (vgl. Kapitel 5.3).
- Mit zunehmender Distanz und somit zunehmenden Kosten für die Datenübermittlung zum Sink-Node sowie ansteigender Grösse des Geosensor Network steigt der Nutzen dezentraler Datenverarbeitung tendenziell an (vgl. Kapitel 5.4, Forschungsfrage 1).
- Mit zunehmender Sensordichte erhöhen sich bei dezentraler Verarbeitung zwar die Durchschnittskosten, jedoch werden diese Kosten auch viel besser auf die einzelnen Sensorknoten verteilt (vgl. Kapitel 5.5).
- Dezentrale Algorithmen verlieren ihre Vorteile bei extrem niedriger positiver oder sogar negativer Autokorrelation sowie bei einer sehr hohen positiven Autokorrelation der zu messenden Ereignisse. Sie scheinen am besten bei einer mittleren positiven Autokorrelation im Verhältnis zur gewählten Sensordichte zu funktionieren (vgl. Kapitel 5.7, Forschungsfrage 2).

Quantitative Schlussfolgerungen werden keine gezogen. Dies hat den Grund, da solch quantitative Aussagen von zu vielen Faktoren, nicht zuletzt von den verwendeten Algorithmen, abhängen würden. So liesse sich fast jede beliebige Aussage erzielen, welche dann jedoch auch nur für ein sehr spezifisches Problem repräsentativ wäre.

Eine weitere wichtige Erkenntnis dieser Arbeit ist, dass es oft wohl nicht ausreicht, den durchschnittlichen oder den gesamten Energieverbrauch der Sensorknoten als alleinigen Indikator für die Energieeffizienz von Algorithmen zu verwenden. So konnte man zum Beispiel in Kapitel 5.4 beobachten, dass der durchschnittliche Energieverbrauch sich entgegengesetzt zum Energieverbrauch stark belasteter Knoten entwickeln kann, weshalb direkte Rückschlüsse allein aus einem der beiden Indikatoren zu Fehlschlüssen bezüglich der Lebensdauer solcher Sensornetzwerke führen können.

6.2 Ausblick

Diese Arbeit untersuchte nur einen ganz kleinen Teil dezentraler und zentraler Algorithmen in sehr stark idealisierten Geosensor Networks: die Energieeffizienz in vordefinierten Szenarien von exakt zwei Algorithmen. Weder wurde die Informationsgenauigkeit der Algorithmen betrachtet, noch wurde zum Beispiel erwähnt, wie am Schluss herausgefunden werden kann, wo im Untersuchungsgebiet sich ein Ereignis wirklich befindet, wobei hierfür auch schon einige Lösungen in der Literatur zu finden sind (Wälchli, Skoczylas, Meer, & Braun, 2007; Römer, 2005; Will, Dziengel, & Schiller, 2009). Da das Forschungsgebiet jedoch noch relativ jung ist, und Experimente mit Netzwerkgrössen, wie sie in den Simulationen verwendet wurden, in der realen Umwelt finanziell mit dem aktuellen Stand der Technik kaum umsetzbar sind, sollte diese Arbeit trotzdem einen kleinen Beitrag dazu leisten, die Einsatzmöglichkeiten dezentraler Algorithmen in Geosensor Networks besser einschätzen zu können. Gerade im Hinblick darauf, dass in Zukunft von der Möglichkeit massiv grösserer Netzwerke ausgegangen wird (Smart Dust), werden Themen wie Energieeffizienz oder Netzwerklastenverteilung (Load Balance) weiter eine zentrale Rolle spielen, weshalb das Thema „dezentrale Algorithmen in Geosensor Networks“ an Wichtigkeit in Zukunft eher noch zunehmen dürfte.

Wie das oft der Fall ist, wenn geforscht wird, sind auch im Verlauf dieser Arbeit viele neue Fragen und Ideen aufgetaucht. Als zukünftige Forschungsarbeiten im engen Bezug zu dieser Arbeit wäre es interessant zu untersuchen, inwiefern der entwickelte Algorithmus noch weiter verbessert werden könnte. Hierfür wurden schon einige Ideen in Kapitel 4.1.4 angedacht. Weiter könnte der Algorithmus so erweitert werden, dass er auch in einer realen Umgebung bestehen könnte. Dazu gehört das Verhalten bei Ausfällen von Knoten, Messfehlern, umweltbedingten Positionsveränderungen einzelner Knoten, Übertragungsfehlern etc., wobei in all diesen Themen auch schon viel Grundlagenforschung verrichtet wurde. Auch wäre es prüfenswert, inwiefern Ideen dieses Algorithmus mit anderen Methoden wie jener von Hefeeda und Bagheri (2009) kombiniert werden können, um noch effizienter zu werden. Des Weiteren könnten weitere Simulationen in verschiedenen Szenarien und vor allem auch unter anderen Gesichtspunkten als der Energieeffizienz durchgeführt werden, um noch weitere mögliche Verbesserungen zu eruieren sowie auch eventuell nicht erkannte Probleme im dezentralen Algorithmus zu finden. Auch wäre es interessant zu untersuchen, inwiefern sich der Algorithmus auf andere Problemstellungen als die Detektion von Ereignissen im Raum adaptieren lässt.

Spätestens, wenn einzelne Sensorknoten so klein und günstig in der Produktion werden, dass diese zu Millionen produziert und eingesetzt werden können, wird es wohl unabdingbar sein, Algorithmen zu entwickeln, welche den Aufwand soweit reduzieren können, dass der Energiebedarf beinahe konstant gehalten werden kann. Ansonsten werden solche Netzwerke wohl kaum sinnvoll einsetzbar, es sei denn, die technische Entwicklung macht solch grosse Fortschritte, dass die Sensorknoten komplett unabhängig von endlichen Energiequellen werden (Thermosäulen, Solarenergie, etc.).

7 Literatur

Akyildiz, I., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). A Survey on Sensor Networks. *IEEE Communications Magazine*, 102-114.

Alippi, C., Anastasi, G., Di Francesco, M., & Roveri, M. (2009). Energy Management in Wireless Sensor Networks with Energy-Hungry Sensors. *IEEE Instrumentation & Measurement Magazine*, 16-23.

Alsalih, W., Islam, K., Núñez-Rodríguez, Y., & Xiao, H. (2008). Distributed Voronoi diagram computation in wireless sensor networks. *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures* (S. 364-364). New York: ACM.

Anselin, L. (1995). Local Indicators of Spatial Association-LISA. In *Geographical Analysis* (Bd. 27, S. 93-115).

Bapat, S. (1994). *Object-Oriented networks, Models for Architecture, Operations, and Management.* Prentice-Hall International.

Barnett, V., & Lewis, T. (1994). *Outliers in statistical Data* (3. Ausg.). New York: John Wiley & Sons.

Carle, J., & Simplot-Ryl, D. (2004). Energy-Efficient Area Monitoring for Sensor Networks. *Computer, 37*(2), 40-46.

Chandola, V., Banerjee, A., & Kumar, V. (2007). *Outlier Detection: A Survey.* University of Minnesota.

Chang, J.-H., & Tassiulas, L. (2000). Energy Conserving Routing in Wireless Ad-hoc Networks. *IFOCOM*.

Czárán, T., & Bartha, S. (1992). Spatiotemporal Dynamic Models of Plant Population and Communities. *Trends in Ecology & Evolution, 7*(2), 38-42.

De Smith, M., Goodchild, M., & Longley, P. (2007). *Geospatial Analysis - A Comprehensive Guide to Pronciples, Techniques and Software Tools* (2te Ausg.). Matador.

Deligiannakis, A., Kotidis, Y., Stoumpos, V., & Delis, A. (2009). Building Efficient Aggregation Trees for Sensor Network Event-Monitoring Queries. In N. Trigoni, A. Markham, & S. Nawaz, *GSN 2009* (S. 63-76). Springer-Verlag Berlin.

Dijkstra, E. (1959). A Note on Two Problems in Connexion with Graphs. In *Numerische Mathematik* (Bd. 1, S. 269-271). Amsterdam.

Duckham, M., & Reitsma, F. (2009). Decentralized environmental simulation and feedback in robust geosensor networks. In *Computers, Environment and Urban Systems 33* (S. 256-268). Elsevier.

Dziengel, N., Wittenburg, G., & Schiller, J. (2008). Towards Distributed Event Detection in Wirless Sensor Networks. *Adjunct Proceedings of the 4th IEEE/ACM International Conference on Distributed Computing in Sensor Systems.* Santorini Island.

Easwaran, C. V. (2008). An Efficient In-Network Event Detection Algorithm for Wireless Sensor Nodes. In T. Sobh, K. Elleithy, A. Mahmood, & M. A. Karim, *Novel Algorithms and Techniques In Telecommunications, Automation and Industrial Electronics* (S. 318-322). Springer Netherlands.

Hefeeda, M., & Bagheri, M. (2009). Forest Fire Modeling and Early Detection using Wireless Sensor Networks. *Ad Hoc & Sensor Wireless Networks, 7*, 169-224.

Hsu, V. S., Kahn, J. M., & Pister, K. S. (1998). *Wireless Communication for Smart Dust.* Berkeley: Department of Electrical Engineering and Computer Science, University of California.

Kim, H., Seok, Y., Choi, N., Choi, Y., & Kwon, T. (2005). Optimal Multi-sink Positioning and Energy-Efficient Routing in Wireless Sensor Networks. In C. Kim, *Lecture Notes in Computer Science* (S. 264-274). Springer-Verlag Berlin.

Krishnamachari, B., & Iyngar, S. (2004). Distributed Bayesian Algorithms for Fault-Tolerance Event Region Detection in Wireless Sensor Networks. *IEEE Transactions on Computers, 53*(3), 241-250.

Laube, P., & Duckham, M. (2009). Decentralized Spatial Data Mining for Geosensor Networks. In H. Miller, & J. Han, *Data Mining and Knowledge Discovery* (S. 411-433). London: Taylor and Francis.

Lloret, J., Garcia, M., Bri, D., & Sendra, S. (2009). A Wireless Sensor Network Deployment for Rural and Forest Fire Detection and Verification. *Sensors*, 8722-8747.

Lynch, N. (1996). *Distributed Algorithms.* San Francisco: Morgan Kaufmann Publishers.

Michaelides, M. P., & Panayiotou, C. G. (2007). Event Detection Using Sensor Networks: A Case for a Hybrid Detector. *46th IEEE Conference on Decision and Control*, (S. 1559-1564). New Orleans.

Nittel, S., Duckham, M., & Kulik, L. (2004). Information Dissemination in Mobile Ad-Hoc Geosensor Networks. In M. Egenhofer, C. Freksa, & H. Miller, *GIScience 2004* (S. 206-222). Springer-Verlag Berlin.

Nittel, S., Stefanidis, A., Cruz, I., M., E., D., G., Howard, A., et al. (2004). Report from the First Workshop on Geo Sensor Networks. *33*(1), 141-144.

Openshaw, S., & Taylor, P. (1979). A million or so correlation coefficients: three experiments on the modifiable areal unit problem. In *Statistical Applications in the Spatial Sciences* (S. 127-144). London: Pion.

Quartieri, J., Mastorakis, N., Iannone, G., & Guarnaccia, C. (2010). A Cellular Automata Model for Fire Spreading Prediction. *Latest Trends on Urban Planning and Transportation* (S. 173-179). WSEAS Press.

Römer, K. (2005). *Time Synchronization and Localization in Sensor Networks.* ETH Zürich.

Römer, K. (2008). Discovery of frequent distributed event patterns in sensor networks. *Proceedings of the 5th European conference on Wireless sensor networks* (S. 106-124). Bologna: Springer-Verlag Berlin.

Sadeq, M., & Duckham, M. (2008). Effect of Neighborhood on In-Network Pricessing in Sensor Networks. *5th international conference on Geographic Information Science* (S. 133-150). Park City: Springer-Verlag.

Shebli, F., Dayoub, I., & Rouvaen, J. (2007). Minimizing energy consumption within wireless sensors networks using optimal transmission range between nodes. *IEEE International Conference on Signal Processing and Communication*, (S. 105-108). Dubai.

Shekar, S., Lu, C. T., & Zhang, P. S. (2003). A Unified Approach to Detecting Spatial Outliers. *GeoInformatica, 7*(2), 139-166.

Shekhar, S., Zhang, P., Huang, Y., & Vatsavai, R. R. (2003). Trends in Spatial Data Mining. In H. Kargupta, A. Joshi, K. Sivakumar, & Y. Yesha, *Data Mining: Next Generation Challenges and Future Directions.* MIT/AAAI Press.

Singh, S., Woo, M., & Raghavendra, C. (1998). Power-aware routing in mobile ad hoc networks. *AXM/IEEE International Conference on Mobile Computing and Networking.*

Tobler, W. R. (1970). A Computer Movie Simulating Urban Growth in the Detroit Region. *Economic Geography, 46, Supplement: Proceedings. International Geographical Union. Commission on Quantitative Methods*, 234-240.

Toumpis, S., & Gupta, G. (2005). Optimal placement of nodes in large sensor networks under a general physical layer model. *Sensor and Ad Hoc Communications and Networks*, 275-283.

Trigoni, N., Yao, Y., Demers, A., Gehrke, J., & Rajaraman, R. (2005). Multi-query Optimization for Sensor Networks. In V. Prasanna, S. Iyengar, P. Spirakis, & M. Welsh, *Distributed Computing in Sensor Systems* (Bd. 3560, S. 307-321). Springer-Verlag Heidelberg.

Vuran, M. C., Akan, O. B., & Akyildiz, I. F. (2004). Spatio-Temporal Correlation: Theory and Applications for Wireless Sensor Networks. *Computer Networks Journal, 45*(3), 245-259.

Wälchli, M., Skoczylas, P., Meer, M., & Braun, T. (2007). Distributed Event Localization and Tracking with Wirless Sensors. In F. Boavida, E. Monteiro, S. Mascolo, & Y. Koucheryavy, *Wired/Wireless Internet Communications* - (S. 247-258). Springer-Verlag Berlin.

Will, H., Dziengel, N., & Schiller, J. (2009). Distance-Based Distributed Multihop Localization in Mobile Wireless Sensor Networks. *Proceedings of the 8th GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze" (FGSN'09).* Hamburg.

Wittenburg, G., Dziengel, N., Wartenburger, C., & Schiller, J. (2010). A system for distributed event detection in wireless sensor networks. *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks* (S. 94-104). New York: ACM.

Yu, L., Wang, N., & Meng, X. (2005). Real-time Forest Fire Detection with Wireless Sensor Networks. *Wireless Communications, Networking and Mobile Computing*, 1214-1217.

Zhang, J., Li, W., Han, N., & Kan, J. (2007). Forest fire detection system based on a ZigBee wireless sensor network. *Journal of Beijing Forestry University, 29*(4), 41-45.

Zhang, Y., Meratnia, N., & Havinga, P. J. (2008). *Outlier Detection Techniques For Wireless Sensor Networks: A Survey.* Enschede: Centre for Telematics and Information Technology, University of Twente.

8 Anhang

8.1 Quellcode

In den folgenden Kapiteln kann der gesamte Quelltext, welcher als Grundlage für die Simulationen in Kapitel 5 diente, nachgelesen werden. Dieser Quelltext ist geistiges Eigentum des Autors und darf nur für nichtkommerzielle Forschungszwecke weiterverwendet werden. Der Autor entschied sich für eine Veröffentlichung des Quelltextes, da er von folgenden Ideen überzeugt ist:

- Nur durch transparente Forschung können Fehler nachvollzogen und Ergebnisse reproduziert werden.
- Durch die Offenlegung des Quelltextes haben Forscher, welche sich auch mit diesem Thema beschäftigen, die Möglichkeit, von vorhandenem Wissen zu profitieren. Dies führt dazu, dass die gleichen Erkenntnisse nicht zuerst mehrmals gewonnen werden müssen sowie Fehler nicht mehrmals gemacht werden, was dem wissenschaftlichen Fortschritt dienen soll.

Folgende Unterkapitel beinhalten folgenden Code:

8.1.1 model.score	Hier steht der XML-Code, welcher nur für die Simulationsumgebung Repast Simphony relevant ist. In dieser Datei werden unter anderem auch die Standardparameter definiert, welche in Kapitel Tabelle 5.1 zu finden sind und in den verschiedenen Simulationen variiert wurden.
8.1.2 ContextCreator.java	Hier steht der Java-Code, welcher für die Initialisierung der Simulation in Repast Simphony zuständig ist.
8.1.3 Environment.java	Dieser Code ist für die Initialisierung sowie das Funktionieren der Umwelt zuständig. Bei der Initialisierung werden die Sensor-Knoten verteilt, der Wind berechnet, alle Umweltvariablen gespeichert, etc. Beim Funktionieren der Umwelt geht es um das Auftauchen neuer Ereignisse, die Regulierung der Temperatur- sowie der Ressourcen-Layer etc.
8.1.4 SimpleAgent.java	In dieser Java-Datei wird ein einfacher Agent erstellt, welcher die grundlegenden Methoden aller Agenten zur Verfügung stellt. Die Unterklassen Event.java, SensorNode.java, Monitor.java und indirekt auch die SinkNode.java erben von dieser Klasse.
8.1.5 Event.java	Hier steht der Code, der das Verhalten der Ereignisse bestimmt. Aus dieser Klasse entstehen Ereignis-Objekte (in dieser Arbeit sind dies Feuer), welche sich als zelluläre Automaten selber verbreiten beziehungsweise, sobald kein „Brennstoff“ mehr vorhanden ist, wieder verlöschen. Diese Ereignis-Objekte verändern in einem gewissen Radius den Ereignis-Wert (=Event-Value, hier die Temperatur), welcher dann von den Sensor-Knoten gemessen wird.
8.1.6 SensorNode.java	Dieser Code bildet das Kernstück der Arbeit. Hier sind die Methoden der Sensor-Knoten und speziell natürlich der dezentrale Algorithmus, welcher in Kapitel 4.1 entwickelt wurde, definiert.
8.1.7 SinkNode.java	In der SinkNode.java, welche von SensorNode.java und somit auch von SimpleAgent.java erbt, werden alle weiteren Methoden definiert, welche die Sink-Nodes gegenüber den „normalen“ Sensor-Knoten zusätzlich beherrschen.
8.1.8 Monitor.java	Monitor.java beinhaltet nur Code, um zusätzliche Daten auswerten zu können, wie zum Beispiel die Korrelation oder die Gesamtereignisgrösse während den Simulationen.

Tabelle 8.1: Inhaltsangabe zum Quellcode

8.1.1 model.score

```
<?xml version="1.0" encoding="UTF-8"?>
<score:SContext xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:score="http://scoreabm.org/score"
label="GeosensorNetwork" ID="geosensorNetwork " pluralLabel="GeosensorNetwork">
  <attributes label="xDim" ID="xDim" pluralLabel="xDims" sType="INTEGER" defaultValue="2000"/>
  <attributes label="yDim" ID="yDim" pluralLabel="yDims" sType="INTEGER" defaultValue="100"/>
  <attributes label="initialNumberOfSensorNodes" ID="initialNumberOfSensorNodes" pluralLabel="initialNumberOfSensorNodess"
sType="INTEGER" defaultValue="10000"/>
  <attributes label="initialNumberOfSinkNodes" ID="initialNumberOfSinkNodes" pluralLabel="initialNumberOfSinkNodess"
sType="INTEGER" defaultValue="1"/>
  <attributes label="sensorNodeSendingDistance" ID="sensorNodeSendingDistance" pluralLabel="sensorNodeSendingDistances"
sType="FLOAT" defaultValue="10"/>
  <attributes label="eventSparkProbability" ID="eventSparkProbability" pluralLabel="eventSparkProbabilities" sType="FLOAT"
defaultValue="0.05"/>
  <attributes label="eventSpreadingProbability" ID="eventSpreadingProbability" pluralLabel="eventSpreadingProbabilities"
sType="FLOAT" defaultValue="0.45"/>
  <attributes label="windStrength" ID="windStrength" pluralLabel="windStrengths" sType="FLOAT" defaultValue="0.2"/>
  <attributes label="initialEventValue" ID="initialEventValue" pluralLabel="initialEventValues" sType="FLOAT"
defaultValue="25"/>
  <attributes label="maxEventValue" ID="maxEventValue" pluralLabel="maxEventValues" sType="FLOAT" defaultValue="900"/>
  <attributes label="eventValueThreshold" ID="eventValueThreshold" pluralLabel="eventValueThresholds" sType="FLOAT"
defaultValue="750"/>
  <attributes label="reduceEventValue" ID="reduceEventValue" pluralLabel="reduceEventValues" sType="FLOAT"
defaultValue="10"/>
  <attributes label="initialFuel" ID="initialFuel" pluralLabel="initialFuels" sType="FLOAT" defaultValue="1.5"/>
  <attributes label="consumeFuel" ID="consumeFuel" pluralLabel="consumeFuels" sType="FLOAT" defaultValue="0.01"/>
  <attributes label="fuelGrow" ID="fuelGrow" pluralLabel="fuelGrows" sType="FLOAT" defaultValue="0.002"/>
  <attributes label="maxFuel" ID="maxFuel" pluralLabel="maxFuels" sType="FLOAT" defaultValue="1.5"/>
  <attributes label="minAbsoluteAreaDifferenceToReport" ID="minAbsoluteAreaDifferenceToReport"
pluralLabel="minAbsoluteAreaDifferenceToReports" sType="FLOAT" defaultValue="50"/>
  <attributes label="relativeAreaDifferenceToReport" ID="relativeAreaDifferenceToReport"
pluralLabel="relativeAreaDifferenceToReports" sType="FLOAT" defaultValue="2"/>
  <attributes label="maxAbsoluteAreaDifferenceToReport" ID="maxAbsoluteAreaDifferenceToReport"
pluralLabel="maxAbsoluteAreaDifferenceToReports" sType="FLOAT" defaultValue="1000"/>
  <implementation package="testproject3" className="" basePath="../TestProject3" mode="AUTO"/>
  <agents xsi:type="score:SContext" label="Environment" ID="environment" pluralLabel="Environments">
    <implementation className="Environment"/>
    <agents label="SensorNode" ID="sensorNode" pluralLabel="SensorNodes">
      <implementation className="SensorNode"/>
    </agents>
    <agents label="SinkNode" ID="sinkNode" pluralLabel="SinkNodes">
      <implementation className="SinkNode"/>
    </agents>
```

```
    <agents label="Monitor" ID="monitor" pluralLabel="Monitors">
      <implementation className="Monitor"/>
    </agents>
    <agents label="Event" ID="event" pluralLabel="Events">
      <implementation className="Event"/>
    </agents>
    <projections xsi:type="score:SGrid" label="Grid" ID="grid" pluralLabel="Grids"/>
    <projections xsi:type="score:SNetwork" label="Network" ID="network" pluralLabel="Networks"/>
    <projections xsi:type="score:SNetwork" label="EventNetwork" ID="eventNetwork" pluralLabel="EventNetworks"/>
    <valueLayers label="EventValue" ID="eventValue" pluralLabel="EventValues"/>
    <valueLayers label="Fuel" ID="fuel" pluralLabel="Fuels"/>
    <valueLayers label="BooleanEventValue" ID="booleanEventValue" pluralLabel="BooleanEventValues"/>
  </agents>
</score:SContext>
```

8.1.2 ContextCreator.java

```
package geosensornetwork;
import repast.simphony.context.Context;
import repast.simphony.dataLoader.ContextBuilder;

/**
 *
 * @author Raphael Renaud
 *
 */

public class ContextCreator implements ContextBuilder<Object> {

	public Context<Object> build(Context<Object> context) {

		Environment environment = new Environment();
		context.addSubContext(environment);
		context.add(environment);

		return context;
	}
}
```

8.1.3 Environment.java

```
1  package geosensornetwork;
2  import repast.simphony.context.DefaultContext;
3  import repast.simphony.context.space.graph.NetworkFactoryFinder;
4  import repast.simphony.context.space.grid.GridFactoryFinder;
5  import repast.simphony.engine.environment.RunEnvironment;
6  import repast.simphony.engine.schedule.ISchedule;
7  import repast.simphony.engine.schedule.ScheduledMethod;
8  import repast.simphony.parameter.Parameters;
9  import repast.simphony.space.graph.Network;
10 import repast.simphony.space.grid.Grid;
11 import repast.simphony.space.grid.GridBuilderParameters;
12 import repast.simphony.space.grid.GridPoint;
13 import repast.simphony.space.grid.RandomGridAdder;
14 import repast.simphony.space.grid.StrictBorders;
15 import repast.simphony.valueLayer.GridValueLayer;
16
17 /**
18  *
19  * @author Raphael Renaud
20  *
21  */
22
23 public class Environment extends DefaultContext<SimpleAgent> {
24       private final boolean log = true;              //Console-log on / off
25
26       private int xDim;                              //Environment-size in x-direction
27       private int yDim;                              //Environment-size in y-direction
28       private int numSensorNodes;                    //Number of placed SensorNodes
29       private int numSinkNodes;                      //Number of placed SinkNodes
30       private double initialEventValue;              //InitialEventValue
31       private double maxEventValue;                  //Maximum possible eventValuee
32       private double eventValueThreshold;            //Threshold which defines whether a value represents an event or not
33       private double reduceEventValue;               //Reduction-rate of eventValue per step
34       private double initialFuel;                    //InitialFuel
35       private double maxFuel;                        //Maximum possible fuel
36       private double fuelGrow;                       //Fuel-grow per step
37       private double windDirection;                  //Wind-direction
38       private double windStrength;                   //Wind-strength (1 = 100%, 0 = 0%)
39       private double eventSparkProbability;          //Probability for a new event (for each step)
40       private double consumeFuel;                    //How much fuel an event will consume per step
41       private double eventSpreadingProbability;      //Average event-spreading probability
42       private double eventSpreadingProbabilityNorth;//Probability for event-spreading: northwards
```

```
43          private double eventSpreadingProbabilityEast; //Probability for event-spreading: eastwards
44          private double eventSpreadingProbabilitySouth;//Probability for event-spreading: southwards
45          private double eventSpreadingProbabilityWest; //Probability for event-spreading: westwards
46          private ISchedule schedule;
47          private Monitor monitor = new Monitor();
48
49          public Environment() {
50                  super("Environment");
51
52                  Parameters p = RunEnvironment.getInstance().getParameters();
53                  this.setXDim((Integer)p.getValue("xDim"));
54                  this.setYDim((Integer)p.getValue("yDim"));
55                  this.setNumSensorNodes((Integer)p.getValue("initialNumberOfSensorNodes"));
56                  this.setNumSinkNodes((Integer)p.getValue("initialNumberOfSinkNodes"));
57                  this.setInitialEventValue((Double)p.getValue("initialEventValue"));
58                  this.setMaxEventValue((Double)p.getValue("maxEventValue"));
59                  this.setEventValueThreshold((Double)p.getValue("eventValueThreshold"));
60                  this.setReduceEventValue((Double)p.getValue("reduceEventValue"));
61                  this.setFuelGrow((Double)p.getValue("fuelGrow"));
62                  this.setMaxFuel((Double)p.getValue("maxFuel"));
63                  this.setInitialFuel((Double)p.getValue("initialFuel"));
64                  this.setWindDirection(Math.random() * 2 * Math.PI);
65                  this.setWindStrength((Double)p.getValue("windStrength"));
66                  this.setEventSparkProbability((Double)p.getValue("eventSparkProbability"));
67                  this.setConsumeFuel((Double)p.getValue("consumeFuel"));
68                  this.setEventSpreadingProbability((Double)p.getValue("eventSpreadingProbability"));
69                  double sp = this.getEventSpreadingProbability();
70                  double wd = this.getWindDirection();
71                  double ws = this.getWindStrength();
72                  this.setEventSpreadingProbabilityNorth(sp - Math.sin(wd) * ws);
73                  this.setEventSpreadingProbabilityEast(sp - Math.cos(wd) * ws);
74                  this.setEventSpreadingProbabilitySouth(sp + Math.sin(wd) * ws);
75                  this.setEventSpreadingProbabilityWest(sp + Math.cos(wd) * ws);
76                  this.setSchedule(RunEnvironment.getInstance().getCurrentSchedule());
77
78                  //Outputs some wind-parameters
79                  this.out("WindDirection: " + this.getWindDirection());
80                  this.out("North: " + this.getEventSpreadingProbabilityNorth());
81                  this.out("East: " + this.getEventSpreadingProbabilityEast());
82                  this.out("South: " + this.getEventSpreadingProbabilitySouth());
83                  this.out("West: " + this.getEventSpreadingProbabilityWest());
84
85                  //Create EnvironmentGrid
86                  Grid<SimpleAgent> grid = GridFactoryFinder.createGridFactory(null).createGrid(
```

```
87                                  "Grid",
88                                  this,
89                                  new GridBuilderParameters<SimpleAgent>(
90                                        new StrictBorders(),
91                                        new RandomGridAdder<SimpleAgent>(), //This will add all agents to a random position
92                                        true,
93                                        this.getXDim(),
94                                        this.getYDim()
95                                  )
96                        );
97
98                        //Create EventValue-layer
99                        GridValueLayer eventValueLayer = new GridValueLayer(
100                                 "EventValue",
101                                 true,
102                                 new StrictBorders(),
103                                 this.getXDim(),
104                                 this.getYDim()
105                       );
106                       this.addValueLayer(eventValueLayer);
107
108                       //Create BooleanEventValue-layer
109                       GridValueLayer booleanEventValueLayer = new GridValueLayer(
110                                 "BooleanEventValue",
111                                 true,
112                                 new StrictBorders(),
113                                 this.getXDim(),
114                                 this.getYDim()
115                       );
116                       this.addValueLayer(booleanEventValueLayer);
117
118                       //Create Fuel-layer
119                       GridValueLayer fuelLayer = new GridValueLayer(
120                                 "Fuel",
121                                 true,
122                                 new StrictBorders(),
123                                 this.getXDim(),
124                                 this.getYDim()
125                       );
126                       this.addValueLayer(fuelLayer);
127
128                       //Create Network
129                       Network<SimpleAgent> network = NetworkFactoryFinder.createNetworkFactory(null).createNetwork(
130                                 "Network",
```

```
131                         this,
132                         false
133                 );
134 
135                 //Create EventNetwork
136                 Network<SimpleAgent> eventNetwork = NetworkFactoryFinder.createNetworkFactory(null).createNetwork(
137                         "EventNetwork",
138                         this,
139                         true
140                 );
141 
142                 //Set the eventValue globally to initialEventValue
143                 for(int i = 0; i < this.getXDim(); i++) {
144                         for(int j = 0; j < this.getYDim(); j++) {
145                                 eventValueLayer.set(this.getInitialEventValue(), i, j);
146                                 fuelLayer.set(this.getInitialFuel(), i, j);
147                         }
148                 }
149 
150                 //Tests if there are too many nodes in the environment and
151                 if((this.getXDim() * this.getYDim()) < (this.getNumSensorNodes() + this.getNumSinkNodes())) {
152                         System.out.println("Too many nodes in the environment!");
153                 } else {
154                         //Adds the SinkNodes
155                         for(int i = 0; i < this.getNumSinkNodes(); i++) {
156                                 SinkNode sinkNode = new SinkNode();
157                                 this.add(sinkNode);
158                                 GridPoint gp = grid.getLocation(sinkNode);
159                                 //The first sink node is in the center of the right border
160                                 if(i == 0) {
161                                         grid.moveTo(sinkNode, (int)this.getXDim() - 1, (int)this.getYDim() / 2);
162                                 }
163                                 for(Object o : grid.getObjectsAt(gp.getX(), gp.getY())) {
164                                         if(!sinkNode.equals(o)) {
165                                                 this.remove(sinkNode);
166                                                 i--;
167                                         }
168                                 }
169                         }
170                         //Adds the SensorNodes
171                         for(int i = 0; i < this.getNumSensorNodes(); i++) {
172                                 SensorNode sensorNode = new SensorNode();
173                                 this.add(sensorNode);
174                                 GridPoint gp = grid.getLocation(sensorNode);
```

```
175                             for(Object o : grid.getObjectsAt(gp.getX(), gp.getY())) {
176                                     if(!sensorNode.equals(o)) {
177                                             this.remove(sensorNode);
178                                             i--;
179                                     }
180                             }
181                     }
182             }
183
184             //Adds the first event in the center of the environment and the monitor which stores simulation-data
185             Event event = new Event();
186             this.add(event);
187             grid.moveTo(event, this.getXDim()/2, this.getYDim()/2);
188             this.add(monitor);
189     }
190
191     @ScheduledMethod(start = 1, interval = 0.5, shuffle=true)
192     public void step() {
193             if(this.getSchedule().getTickCount()%1 == 0) {
194                     this.step1();
195             } else {
196                     this.step2();
197             }
198     }
199
200     public void step1() {
201             //Sets a new event with probability eventProbability
202             if(Math.random() < this.getEventSparkProbability()) {
203                     Event event = new Event();
204                     this.add(event);
205                     Grid<SimpleAgent> grid = (Grid<SimpleAgent>)this.getProjection("Grid");
206                     GridPoint gp = grid.getLocation(event);
207                     grid.moveTo(event, gp.getX(), gp.getY());
208                     gp = grid.getLocation(event);
209
210                     //Removes the event if there is already an event
211                     for(Object o : grid.getObjectsAt(gp.getX(), gp.getY())) {
212                             if(o instanceof Event && !o.equals(event)) {
213                                     this.remove(event);
214                             }
215                     }
216             }
217
218             //Updates the eventLayer as well as the fuelLayer
```

```
219             //The eventValue decreases every step until it reaches initialEventValue. If there is an event on a cell, the
    event takes control of the eventValue.
220             //The fuelLayer increases until initialFuel is reached. An event will consume fuel.
221             for(int i = 0; i < this.getXDim(); i++) {
222                     for(int j = 0; j < this.getYDim(); j++) {
223                             boolean onEvent = false;
224                             Grid<SimpleAgent> grid = (Grid<SimpleAgent>)this.getProjection("Grid");
225
226                             //Check if there is an event on the cell; if yes, the event takes control of eventValue and fuel
227                             for(Object o : grid.getObjectsAt(i, j)) {
228                                     if(o instanceof Event) {
229                                             onEvent = true;
230                                     }
231                             }
232                             if(!onEvent) {
233                                     this.reduceEventValue(i, j);
234                                     this.grow(i, j);
235                             }
236
237                             //Updates the booleanEventValueLayer
238                             if(this.getEventValue(i, j) > this.getEventValueThreshold()) {
239                                     this.setValue(1, i, j, "BooleanEventValue");
240                             } else {
241                                     this.setValue(0, i, j, "BooleanEventValue");
242                             }
243                     }
244             }
245     }
246
247     public void step2() {
248             //No action
249     }
250
251     /**
252      * The fuel will grow at point x/y
253      * @param x
254      * @param y
255      */
256     public void grow(int x, int y) {
257             this.setFuel(this.getFuel(x, y) + this.getFuelGrow(), x, y);
258     }
259
260     /**
261      * Reduces the eventValue by reduceEventValue
```

```
262         * @param x
263         * @param y
264         */
265        public void reduceEventValue(int x, int y) {
266               this.setEventValue(this.getEventValue(x, y) - this.getReduceEventValue(), x, y);
267        }
268
269        /**
270         * Sets the new fuel at position x/y to the layer "Fuel"
271         * @param fuel
272         * @param x
273         * @param y
274         */
275        public void setFuel(double fuel, int x, int y) {
276               if(fuel > this.getMaxFuel()) {
277                      this.setValue(this.getMaxFuel(), x, y, "Fuel");
278               } else if(fuel < 0) {
279                      this.setValue(0, x, y, "Fuel");
280               } else {
281                      this.setValue(fuel, x, y, "Fuel");
282               }
283        }
284
285        /**
286         * Returns the fuel at position x/y.
287         * @param x
288         * @param y
289         * @return the fuel
290         */
291        public double getFuel(int x, int y) {
292               return this.getValue(x, y, "Fuel");
293        }
294
295        /**
296         * Sets the new eventValue at position x/y to the layer "EventValue"
297         * @param eventValue
298         * @param x
299         * @param y
300         */
301        public void setEventValue(double eventValue, int x, int y) {
302               if(eventValue > this.getMaxEventValue()) {
303                      this.setValue(this.getMaxEventValue(), x, y, "EventValue");
304               } else if(eventValue < this.getInitialEventValue()) {
305                      this.setValue(this.getInitialEventValue(), x, y, "EventValue");
```

```
306                 } else {
307                         this.setValue(eventValue, x, y, "EventValue");
308                 }
309         }
310
311         /**
312          * Returns the eventValue at position x/y.
313          * @param x
314          * @param y
315          * @return the eventValue
316          */
317         public double getEventValue(int x, int y) {
318                 return this.getValue(x, y, "EventValue");
319         }
320
321         /**
322          * Sets a new value at position x/y to the layer layerName
323          * @param value
324          * @param x
325          * @param y
326          * @param layerName
327          */
328         public void setValue(double value, int x, int y, String layerName) {
329                 GridValueLayer layer = (GridValueLayer)this.getValueLayer(layerName);
330                 if(inEnvironment(x, y)) {
331                         layer.set(value, x, y);
332                 }
333         }
334
335         /**
336          * Returns a value at position x/y of the layer "layerName". If x/y is outside the environment, -1 will be returned
337          * @param x
338          * @param y
339          * @param layerName
340          * @return the value
341          */
342         public double getValue(int x, int y, String layerName) {
343                 double value = -1;
344                 if(inEnvironment(x, y)) {
345                         GridValueLayer layer = (GridValueLayer)this.getValueLayer(layerName);
346                         value = layer.get(x, y);
347                 }
348                 return value;
349         }
```

```
350
351         /**
352          * Returns true if the point is inside the environment and false if the point is outside
353          * @param x
354          * @param y
355          * @return if the point is in area
356          */
357         public boolean inEnvironment(int x, int y) {
358                 return (x < this.getXDim() && x >= 0 && y < this.getYDim() && y >= 0);
359         }
360
361         /**
362          * Calculates the euclidean distance between two points (x1/y1, x2/y2)
363          * @param x1
364          * @param y1
365          * @param x2
366          * @param y2
367          * @return the distance
368          */
369         public double distance(int x1, int y1, int x2, int y2) {
370                 int dx = x1 - x2;
371                 int dy = y1 - y2;
372
373                 return (double)Math.sqrt((dx * dx) + (dy * dy));
374         }
375
376         //Helper methods
377
378         public void out(String message) {
379                 if(this.isLog()) {
380                         System.out.println(message);
381                 }
382         }
383
384         public boolean isLog() {
385                 return log;
386         }
387
388         //Getter and setter methods
389
390         public int getXDim() {
391                 return xDim;
392         }
393
```

```
394         public void setXDim(int dim) {
395                 xDim = dim;
396         }
397
398         public int getYDim() {
399                 return yDim;
400         }
401
402         public void setYDim(int dim) {
403                 yDim = dim;
404         }
405
406         public int getNumSensorNodes() {
407                 return numSensorNodes;
408         }
409
410         public void setNumSensorNodes(int numSensorNodes) {
411                 this.numSensorNodes = numSensorNodes;
412         }
413
414         public int getNumSinkNodes() {
415                 return numSinkNodes;
416         }
417
418         public void setNumSinkNodes(int numSinkNodes) {
419                 this.numSinkNodes = numSinkNodes;
420         }
421
422         public double getInitialEventValue() {
423                 return initialEventValue;
424         }
425
426         public void setInitialEventValue(double initialEventValue) {
427                 this.initialEventValue = initialEventValue;
428         }
429
430         public double getMaxEventValue() {
431                 return maxEventValue;
432         }
433
434         public void setMaxEventValue(double maxEventValue) {
435                 this.maxEventValue = maxEventValue;
436         }
437
```

```
438         public double getEventValueThreshold() {
439                 return eventValueThreshold;
440         }
441
442         public void setEventValueThreshold(double eventValueThreshold) {
443                 this.eventValueThreshold = eventValueThreshold;
444         }
445
446         public double getReduceEventValue() {
447                 return reduceEventValue;
448         }
449
450         public void setReduceEventValue(double reduceEventValue) {
451                 this.reduceEventValue = reduceEventValue;
452         }
453
454         public double getInitialFuel() {
455                 return initialFuel;
456         }
457
458         public void setInitialFuel(double initialFuel) {
459                 this.initialFuel = initialFuel;
460         }
461
462         public double getMaxFuel() {
463                 return maxFuel;
464         }
465
466         public void setMaxFuel(double maxFuel) {
467                 this.maxFuel = maxFuel;
468         }
469
470         public double getFuelGrow() {
471                 return fuelGrow;
472         }
473
474         public void setFuelGrow(double fuelGrow) {
475                 this.fuelGrow = fuelGrow;
476         }
477
478         public double getWindDirection() {
479                 return windDirection;
480         }
481
```

```
482         public void setWindDirection(double windDirection) {
483                 this.windDirection = windDirection;
484         }
485
486         public double getWindStrength() {
487                 return windStrength;
488         }
489
490         public void setWindStrength(double windStrength) {
491                 this.windStrength = windStrength;
492         }
493
494         public double getEventSparkProbability() {
495                 return eventSparkProbability;
496         }
497
498         public void setEventSparkProbability(double eventSparkProbability) {
499                 this.eventSparkProbability = eventSparkProbability;
500         }
501
502         public double getConsumeFuel() {
503                 return consumeFuel;
504         }
505
506         public void setConsumeFuel(double consumeFuel) {
507                 this.consumeFuel = consumeFuel;
508         }
509
510         public double getEventSpreadingProbability() {
511                 return eventSpreadingProbability;
512         }
513
514         public void setEventSpreadingProbability(double eventSpreadingProbability) {
515                 this.eventSpreadingProbability = eventSpreadingProbability;
516         }
517
518         public double getEventSpreadingProbabilityNorth() {
519                 return eventSpreadingProbabilityNorth;
520         }
521
522         public void setEventSpreadingProbabilityNorth(
523                         double eventSpreadingProbabilityNorth) {
524                 this.eventSpreadingProbabilityNorth = eventSpreadingProbabilityNorth;
525         }
```

```
526
527         public double getEventSpreadingProbabilityEast() {
528                 return eventSpreadingProbabilityEast;
529         }
530
531         public void setEventSpreadingProbabilityEast(
532                         double eventSpreadingProbabilityEast) {
533                 this.eventSpreadingProbabilityEast = eventSpreadingProbabilityEast;
534         }
535
536         public double getEventSpreadingProbabilitySouth() {
537                 return eventSpreadingProbabilitySouth;
538         }
539
540         public void setEventSpreadingProbabilitySouth(
541                         double eventSpreadingProbabilitySouth) {
542                 this.eventSpreadingProbabilitySouth = eventSpreadingProbabilitySouth;
543         }
544
545         public double getEventSpreadingProbabilityWest() {
546                 return eventSpreadingProbabilityWest;
547         }
548
549         public void setEventSpreadingProbabilityWest(
550                         double eventSpreadingProbabilityWest) {
551                 this.eventSpreadingProbabilityWest = eventSpreadingProbabilityWest;
552         }
553
554         public ISchedule getSchedule() {
555                 return schedule;
556         }
557
558         public void setSchedule(ISchedule schedule) {
559                 this.schedule = schedule;
560         }
561
562         public Monitor getMonitor() {
563                 return monitor;
564         }
565
566         public void setMonitor(Monitor monitor) {
567                 this.monitor = monitor;
568         }
569 }
```

8.1.4 SimpleAgent.java

```
1  package geosensornetwork;
2  import repast.simphony.engine.schedule.ScheduledMethod;
3  import repast.simphony.util.ContextUtils;
4
5  /**
6   *
7   * @author Raphael Renaud
8   *
9   */
10
11 public class SimpleAgent {
12
13        public SimpleAgent() {
14
15        }
16
17        @ScheduledMethod(start = 1, interval = 0.5, shuffle=true)
18        public void step() {
19              Environment environment = (Environment)ContextUtils.getContext(this);
20              //One step is divided into two half-steps (step1() and step2())
21              if(environment.getSchedule().getTickCount()%1 == 0) {
22                    this.step1();
23              } else {
24                    this.step2();
25              }
26        }
27
28        public void step1() {
29              //Overridden by subclasses
30        }
31
32        public void step2() {
33              //Overridden by subclasses
34        }
35
36        public double getTickCount() {
37              Environment environment = (Environment)ContextUtils.getContext(this);
38              return environment.getSchedule().getTickCount();
39        }
40 }
```

8.1.5 Event.java

```
1  package geosensornetwork;
2  import repast.simphony.space.grid.Grid;
3  import repast.simphony.space.grid.GridPoint;
4  import repast.simphony.util.ContextUtils;
5
6  /**
7   *
8   * @author Raphael Renaud
9   *
10  */
11
12 public class Event extends SimpleAgent {
13        private int age;          //The age of the event
14
15        public Event() {
16              super();
17              this.setAge(0);
18        }
19
20        public void step1() {
21              Environment environment = (Environment)ContextUtils.getContext(this);
22              Grid<SimpleAgent> grid = (Grid<SimpleAgent>)environment.getProjection("Grid");
23              GridPoint gp = grid.getLocation(this);
24
25              //Removes event if there is no fuel anymore
26              if(environment.getFuel(gp.getX(), gp.getY()) < environment.getConsumeFuel()) {
27                    environment.remove(this);
28              } else {
29                    //If age == 2, then the event can spread
30                    if(this.getAge() == 2) {
31                          if(Math.random() <= environment.getEventSpreadingProbabilityNorth()) {
32                                this.setEvent(gp.getX()+1, gp.getY());
33                          }
34                          if(Math.random() <= environment.getEventSpreadingProbabilityEast()) {
35                                this.setEvent(gp.getX(), gp.getY()+1);
36                          }
37                          if(Math.random() <= environment.getEventSpreadingProbabilitySouth()) {
38                                this.setEvent(gp.getX()-1, gp.getY());
39                          }
40                          if(Math.random() <= environment.getEventSpreadingProbabilityWest()) {
41                                this.setEvent(gp.getX(), gp.getY()-1);
42                          }
```

```
43                      }
44                      this.setAge(this.getAge()+1);
45                      this.updateTemperatureLayer();
46                      this.updateFuelLayer();
47              }
48      }
49
50      /**
51       * Sets a new event to x/y if there is not already another event
52       * @param x
53       * @param y
54       * @return whether or not the event has been set
55       */
56      private boolean setEvent(int x, int y) {
57              boolean setEvent = true;
58              Environment environment = (Environment)ContextUtils.getContext(this);
59              Grid<SimpleAgent> grid = (Grid<SimpleAgent>)environment.getProjection("Grid");
60              if(environment.inEnvironment(x, y)) {
61                      for(Object o : grid.getObjectsAt(x, y)) {
62                              if(o instanceof Event) {
63                                      setEvent = false;
64                              }
65                      }
66                      if(setEvent) {
67                              Event event = new Event();
68                              environment.add(event);
69                              grid.moveTo(event, x, y);
70                      }
71              }
72              return setEvent;
73      }
74
75      /**
76       * Updates the temperatureLayer
77       */
78      private void updateTemperatureLayer() {
79              Environment environment = (Environment)ContextUtils.getContext(this);
80              Grid<SimpleAgent> grid = (Grid<SimpleAgent>)environment.getProjection("Grid");
81              GridPoint gp = grid.getLocation(this);
82              int x = gp.getX();
83              int y = gp.getY();
84              int radius = 5;
85              int xStart = x - radius;
86              int xStop = x + radius;
```

```
 87                int yStart = y - radius;
 88                int yStop = y + radius;
 89
 90                for(int i = xStart; i <= xStop; i++) {
 91                      for(int j = yStart; j <= yStop; j++) {
 92                            double distance = environment.distance(i, j, x, y);
 93                            if(distance <= radius) {
 94                                  double value = environment.getMaxEventValue() / 10 / (1 + Math.pow(distance, 2));
 95                                  environment.setEventValue(environment.getEventValue(i, j) + value, i, j);
 96                            }
 97                      }
 98                }
 99          }
100
101         /**
102          * Updates the fuelLayer
103          */
104         private void updateFuelLayer() {
105               Environment environment = (Environment)ContextUtils.getContext(this);
106               Grid<SimpleAgent> grid = (Grid<SimpleAgent>)environment.getProjection("Grid");
107               GridPoint gp = grid.getLocation(this);
108               int x = gp.getX();
109               int y = gp.getY();
110
111               environment.setFuel(environment.getFuel(x, y) - environment.getConsumeFuel(), x, y);
112         }
113
114         //Getter and setter methods
115
116         public int getAge() {
117               return age;
118         }
119
120         private void setAge(int age) {
121               this.age = age;
122         }
123   }
```

8.1.6 SensorNode.java

Für die beiden Algorithmen, insbesondere den dezentralen Algorithmus, sind die Zeilen 31 bis 39 (Intialvariablen) sowie die Zeilen 74 bis 399 (Methoden) von Bedeutung. In diesen Zeilen wurde Quelltext, welcher nur als Hilfe für das Monitoring des Energieverbrauches dient und für den Algorithmus nicht relevant ist, blau markiert. Weiter wurde der Code, welcher nur für den zentralen Algorithmus relevant ist, grün markiert. Orange markiert ist in diesen Zeilen aller Code, welcher sich auf die Simulationsumgebung Repast bezieht (Darstellung etc.), und nichts direkt mit dem Algorithmus zu tun hat:

```
 1  package geosensornetwork;
 2  import java.util.Vector;
 3  import repast.simphony.engine.environment.RunEnvironment;
 4  import repast.simphony.parameter.Parameters;
 5  import repast.simphony.space.graph.Network;
 6  import repast.simphony.space.grid.Grid;
 7  import repast.simphony.space.grid.GridPoint;
 8  import repast.simphony.util.ContextUtils;
 9  import repast.simphony.valueLayer.GridValueLayer;
10
11  /**
12   *
13   * @author Raphael Renaud
14   *
15   */
16
17  public class SensorNode extends SimpleAgent {
18        protected double sendingDistance;
19        protected double eventValue;
20        protected boolean initialized;
21        protected boolean networkInitialized;
22        protected double maxConnectionDistance;
23        protected double detectionArea;
24        protected int numOfMessagesDecentralized;
25        protected int numOfMessagesCentralized;
26        protected Vector<SensorNode> nodesInRange;
27        protected Vector<SensorNode> connectedNodes;
28        protected SensorNode shortestPathNode;
29        protected double shortestPathDistance;
30        //Start event-detection variables
31        protected int state = 0;
32        protected SensorNode reportTo = this;
33        protected Vector<SensorNode> reportFrom = new Vector<SensorNode>();
34        protected SensorNode head = this;
```

```
35          protected double area = 0;
36          protected double lastArea = 0;
37          protected double minAbsoluteAreaDifferenceToReport;  //Minimum absolute diff. after which a new report will be sent
38          protected double relativeAreaDifferenceToReport;     //Minimum relative diff. after which a new report will be sent
39          protected double maxAbsoluteAreaDifferenceToReport;  //Maximum absolute diff. after which a new report will be sent
40          //End event-detection variables
41
42          public SensorNode() {
43                  super();
44                  Parameters p = RunEnvironment.getInstance().getParameters();
45                  this.setSendingDistance((Double)p.getValue("sensorNodeSendingDistance"));
46                  this.setInitialized(false);
47                  this.setNodesInRange(new Vector<SensorNode>());
48                  this.setConnectedNodes(new Vector<SensorNode>());
49                  this.setDetectionArea(0);
50                  this.setNumOfMessagesDecentralized(0);
51                  this.setNumOfMessagesCentralized(0);
52                  this.setShortestPathNode(null);
53                  this.setShortestPathDistance(Double.MAX_VALUE);
54                  this.setMinAbsoluteAreaDifferenceToReport((Double)p.getValue("minAbsoluteAreaDifferenceToReport"));
55                  this.setRelativeAreaDifferenceToReport((Double)p.getValue("relativeAreaDifferenceToReport"));
56                  this.setMaxAbsoluteAreaDifferenceToReport((Double)p.getValue("maxAbsoluteAreaDifferenceToReport"));
57          }
58
59          public void step1() {
60                  //Initialize SensorNodes if not already done
61                  if(!this.isInitialized()) {
62                          this.initialize();
63                  } else {
64                          this.detect();      //Calls the method to detect events
65                  }
66          }
67
68          public void step2() {
69                  this.setEventValue(this.measureEventValue());
70          }
71
72          //Start event-detection methods
73
74          /**
75           * The main method for the event detection which will be called every step
76           */
77          public void detect() {
78                  if(this.getState() == 0) {//Not in an event
```

```
79                      //Only do something if "this" detects an event
80                      if(this.detectEvent()) {
81                              //Centralized report (onChange)
82                              this.report(this.getShortestPathNode(), this.getDetectionArea(), false);
83
84                              //Decentralized
85                              //Update node to which "this" should report
86                              this.searchReportTo();
87                      }
88              } else {
89                      //If "this" detects no event anymore, inform nodes which reported to "this"
90                      //Else if getState == 2, requestArea and send information to sink node
91                      if(!this.detectEvent()) {
92                              //Centralized report (onChange)
93                              this.report(this.getShortestPathNode(), this.getDetectionArea(), false);
94
95                              //Decentralized
96                              this.removeArea();
97                              this.setState(0);
98                              //Update reportTo for all nodes which reported to "this"
99                              while(this.getReportFrom().size() > 0) {
100                                     this.getReportFrom().get(0).updateReportTo();
101                             }
102                             this.setHead(this);
103                             this.setReportTo(this);
104                     }
105             }
106     }
107
108     /**
109      * Returns whether the sensor detects an event or not
110      * @return whether the sensor detects an event or not
111      */
112     public boolean detectEvent() {
113             Environment environment = (Environment)ContextUtils.getContext(this);
114             return (this.getEventValue() > environment.getEventValueThreshold()) ? true : false;
115     }
116
117     /**
118      * Report added area to headNode (area = this.getDetectionArea())
119      */
120     public void addArea() {
121             this.addArea(this.getDetectionArea());
122     }
```

```
123
124         /**
125          * Report added area to headNode
126          * @param area
127          */
128         public void addArea(double area) {
129                 this.setArea(this.getArea() + area);
130                 //If "this" is a headNode and the difference between actual area is big enough, "this" will report it to the
    sink-node
131                 //Else, send the information forward if !this.getHead().equals(this). This may happen if "this" was an isolated
    node
132                 if(this.getState() == 2) {
133                         //Minimum absolute difference
134                         if(Math.abs(this.getArea() - this.getLastArea()) > this.getMinAbsoluteAreaDifferenceToReport() ||
135 this.getLastArea() == 0) {
136                                 //Minimum relative difference && Maximum difference
                                  if(Math.abs(this.getArea() - this.getLastArea()) > this.getMaxAbsoluteAreaDifferenceToReport() ||
137 this.getLastArea() * this.getRelativeAreaDifferenceToReport() < this.getArea() || this.getArea() *
138 this.getRelativeAreaDifferenceToReport() < this.getLastArea()) {
                                          this.setLastArea(this.getArea());
                                          this.report(this.getShortestPathNode(), this.getArea(), true);
139                                 }
140                         }
141                 } else if(!this.getHead().equals(this)) {
142                         this.increaseNumOfMessages(true); //To monitor energy consumption
143                         this.getReportTo().addArea(area);
144                 }
145         }
146
147         /**
148          * Report removed area to headNode (area = this.getArea())
149          */
150         public void removeArea() {
151                 this.removeArea(this.getArea());
152         }
153
154         /**
155          * Report removed area to headNode
156          * @param area
157          */
158         public void removeArea(double area) {
159                 this.addArea(0 - area);
160         }
161
```

```
162         /**
163          * Sets this to head if reportTo has changed its state to 0 and try to find another head of another eventArea nearby
164          */
165         public void updateReportTo() {
166                 this.setState(2);
167                 //Check if another head of another eventArea can be found near this eventArea
168                 this.checkForOtherHead();
169         }
170
171         /**
172          * Checks whether or not there is another eventArea with another head nearby
173          * @return whether or not another eventArea with a new head had been found
174          */
175         public boolean checkForOtherHead() {
176                 boolean foundOtherHead = false;
177                 for(int i = 0; i < this.getConnectedNodes().size(); i++) {
178                         //Only check connected node if it is not already in reportFrom or reportTo
179                         if(!this.getReportFrom().contains(this.getConnectedNodes().get(i)) &&
180 !this.getReportTo().equals(this.getConnectedNodes().get(i))) {
181                                 //Connect only to node which already detected an event and has another head
182                                 this.increaseNumOfMessages(true); //Monitors only energy consumption
183                                 this.getConnectedNodes().get(i).increaseNumOfMessages(2, true); //Monitors only energy consumption
184 (two informations must be returned (head, state))
185                                 SensorNode headOfConnectedNode = this.getConnectedNodes().get(i).getHead();
186                                 if(this.getConnectedNodes().get(i).getState() != 0 && !this.getHead().equals(headOfConnectedNode))
    {
187                                         foundOtherHead = true;
188                                         double area = this.merge(this.getConnectedNodes().get(i), headOfConnectedNode);
189                                         this.increaseNumOfMessages(true); //Monitors only energy consumption
190                                         this.getReportTo().addArea(area);
191                                 }
192                         }
193                 }
194                 //If no other head has been found, check if another node which reports to this finds another head
195                 if(!foundOtherHead) {
196                         for(int i = 0; i < this.getReportFrom().size(); i++) {
197                                 //Check nodes which report to this until a new head has been found
198                                 this.increaseNumOfMessages(true); //Monitors only energy consumption
199                                 this.getReportFrom().get(i).increaseNumOfMessages(true); //Monitors only energy consumption
200                                 if(this.getReportFrom().get(i).checkForOtherHead()) {
201                                         break;
202                                 }
203                         }
204                 }
```

```
205                 this.increaseNumOfMessages(true); //Monitors only energy consumption
206                 return foundOtherHead;
207           }
208
209           /**
210            * Search node to which "this" will report to
211            */
212           public void searchReportTo() {
213                 boolean foundReportTo = false;
214                 boolean merge = false;
215                 //Search connected nodes for another node which already detected an event
216                 for(int i = 0; i < this.getConnectedNodes().size(); i++) {
217                       //Connect only to node which already detected an event
218                       this.increaseNumOfMessages(true); //Monitors only energy consumption
219                       this.getConnectedNodes().get(i).increaseNumOfMessages(2, true); //Monitors only energy consumption
220                       SensorNode headOfConnectedNode = this.getConnectedNodes().get(i).getHead();
221                       if(this.getConnectedNodes().get(i).getState() != 0) {
222                             //If no reportTo node has been found until now, set it to reportTo node
223                             //Else if the connected node is also in an event but has another head, merge these two areas
224                             if(!foundReportTo) {
225                                   foundReportTo = true;
226                                   this.setState(1);
227                                   this.setReportTo(this.getConnectedNodes().get(i));
228                                   this.setHead(headOfConnectedNode);
229                             } else {
230                                   if(!this.getHead().equals(headOfConnectedNode)) {
231                                         merge = true;
232                                         double area = this.merge(this.getConnectedNodes().get(i), headOfConnectedNode);
233                                         this.increaseNumOfMessages(true);
234                                         this.getReportTo().addArea(area);
235                                   }
236                             }
237                       }
238                 }
239                 //If no node to report to has been found, this is a headNode
240                 if(!foundReportTo) {
241                       this.setState(2);
242                       this.addArea();
243                 } else {
244                       if(!merge) {
245                             this.addArea();
246                       }
247                 }
248           }
```

```
249
250         /**
251          * Merges two different events to one new event
252          * @param reportTo
253          * @param head
254          * @return merged area
255          */
256         public double merge(SensorNode reportTo, SensorNode head) {
257                 double area = this.getDetectionArea();
258                 //Only nodes which have another head nodes must be informed
259                 if(!this.getHead().equals(head)) {
260                         //If this was a head, set state to 1 (because there is a new head) and set area to getArea();
261                         //Else, change reportTo of reportTo node to "this" (change direction of connection)
262                         if(this.getState() == 2) {
263                                 this.setState(1);
264                         } else {
265                                 this.increaseNumOfMessages(2, true); //Monitors only energy consumption
266                                 area += this.getReportTo().merge(this, head);
267                         }
268                         //Adds all areas from nodes which report to this
269                         for(int i = 0; i < this.getReportFrom().size(); i++) {
270                                 //Skip new reportTo which reported to this until now and the new node which reports to this,
    because it already returns the value
271                                 if(!reportTo.equals(this.getReportFrom().get(i)) &&
    !this.getReportTo().equals(this.getReportFrom().get(i))) {
272                                         this.increaseNumOfMessages(true); //Monitors only energy consumption
273                                         this.getReportFrom().get(i).increaseNumOfMessages(true); //Monitors only energy consumption
274                                         area += this.getReportFrom().get(i).getArea();
275                                 }
276                         }
277                         this.setReportTo(reportTo);
278                         this.setHead(head);
279                         this.setArea(area);
280                 }
281                 this.increaseNumOfMessages(true);
282                 return area;
283         }
284
285         /**
286          * Reports an event along shortestPath to next sinkNode
287          * @param to
288          * @param area
289          * @param decentralized
290          */
```

```
291         public void report(SensorNode to, double area, boolean decentralized) {
292                 this.increaseNumOfMessages(decentralized); //Monitors only energy consumption
293                 to.report(to.getShortestPathNode(), area, decentralized);
294         }
295 
296         public int getState() {
297                 return state;
298         }
299 
300         public void setState(int state) {
301                 //If "this" is a head node, set head and reportTo to this.
302                 //Else, if state == 0, set area to 0 (= reset area)
303                 if(state == 2) {
304                         this.setHead(this);
305                         this.setReportTo(this);
306                         this.setLastArea(0);
307                 } else if(state == 0) {
308                         this.setArea(0);
309                 }
310 
311                 this.state = state;
312         }
313 
314         public SensorNode getReportTo() {
315                 return reportTo;
316         }
317 
318         public void setReportTo(SensorNode reportTo) {
319                 Environment environment = (Environment)ContextUtils.getContext(this);
320                 Network<SensorNode> network = (Network<SensorNode>)environment.getProjection("EventNetwork");
321                 //Inform node to which this reported that it doesn't report to the node anymore
322                 network.removeEdge(network.getEdge(this, this.getReportTo()));
323                 this.increaseNumOfMessages(true); //Monitors only energy consumption
324                 this.getReportTo().getReportFrom().remove(this);
325                 this.reportTo = reportTo;
326                 //Inform new node to which this reports (if this != node to report to)
327                 if(!this.equals(this.getReportTo())) {
328                         network.addEdge(this, reportTo);
329                         this.increaseNumOfMessages(true); //Monitors only energy consumption
330                         this.getReportTo().getReportFrom().add(this);
331                 }
332         }
333 
334         public SensorNode getHead() {
```

```
335                 return head;
336         }
337
338         public void setHead(SensorNode head) {
339                 //Only do something if new head != actual head
340                 if(!this.getHead().equals(head)) {
341                         this.head = head;
342                         //Every node which reported to this must also know the new head
343                         for(int i = 0; i < this.getReportFrom().size(); i++) {
344                                 this.increaseNumOfMessages(true); //Monitors only energy consumption
345                                 this.getReportFrom().get(i).setHead(head);
346                         }
347                 }
348         }
349
350         public Vector<SensorNode> getReportFrom() {
351                 return reportFrom;
352         }
353
354         public void setReportFrom(Vector<SensorNode> reportFrom) {
355                 this.reportFrom = reportFrom;
356         }
357
358         public double getArea() {
359                 return area;
360         }
361
362         public void setArea(double area) {
363                 this.area = area;
364         }
365
366         public double getLastArea() {
367                 return lastArea;
368         }
369
370         public void setLastArea(double lastArea) {
371                 this.lastArea = lastArea;
372         }
373
374         public double getMinAbsoluteAreaDifferenceToReport() {
375                 return minAbsoluteAreaDifferenceToReport;
376         }
377
378         public void setMinAbsoluteAreaDifferenceToReport(
```

```
379                     double minAbsoluteAreaDifferenceToReport) {
380             this.minAbsoluteAreaDifferenceToReport = minAbsoluteAreaDifferenceToReport;
381         }
382 
383         public double getMaxAbsoluteAreaDifferenceToReport() {
384             return maxAbsoluteAreaDifferenceToReport;
385         }
386 
387         public void setMaxAbsoluteAreaDifferenceToReport(
388                     double maxAbsoluteAreaDifferenceToReport) {
389             this.maxAbsoluteAreaDifferenceToReport = maxAbsoluteAreaDifferenceToReport;
390         }
391 
392         public double getRelativeAreaDifferenceToReport() {
393             return relativeAreaDifferenceToReport;
394         }
395 
396         public void setRelativeAreaDifferenceToReport(
397                     double relativeAreaDifferenceToReport) {
398             this.relativeAreaDifferenceToReport = relativeAreaDifferenceToReport;
399         }
400 
401         //End event-detection methods
402 
403         /**
404          * Initializes the network-structure after placing SensorNodes
405          * @return whether or not initializing has finished
406          */
407         public void initialize() {
408             //Initialize nodesInRange
409             this.initializeNodesInRange();
410 
411             //Initialize network
412             this.initializeNetwork();
413 
414             //Calculate important static values
415             this.setMaxConnectionDistance(this.maxConnectionDistance());
416             this.setDetectionArea(this.detectionArea());
417 
418             this.setInitialized(true);
419         }
420 
421         /**
422          * Initialize network
```

```
423          */
424         public void initializeNetwork() {
425                 if(this.getNodesInRange().size() == 0) {
426                         this.initializeNodesInRange();
427                 }
428                 this.rng();
429 
430                 this.setNetworkInitialized(true);
431         }
432 
433         /**
434          * Returns approximately the covered area including connected nodes
435          * @return approximately the covered area including connected nodes
436          */
437         public double detectionArea() {
438                 return Math.pow(this.meanConnectionDistance() * 2, 2) / this.getConnectedNodes().size();
439         }
440 
441         /**
442          * Returns the mean connection distance of connected nodes
443          * @return the mean connection distance of connected nodes
444          */
445         public double meanConnectionDistance() {
446                 if(this.getConnectedNodes().size() == 0) {
447                         this.initializeNetwork();
448                 }
449                 double sum = 0;
450                 int n = this.getConnectedNodes().size();
451 
452                 for(int i = 0; i < n; i++) {
453                         sum += this.distance(this.getConnectedNodes().get(i));
454                 }
455 
456                 return sum / n;
457         }
458 
459         /**
460          * Returns the longest distance to a connected node
461          * @return the longest distance to a connected node
462          */
463         public double maxConnectionDistance() {
464                 if(this.getConnectedNodes().size() == 0) {
465                         this.initializeNetwork();
466                 }
```

```
467                 double maxConnectionDistance = 0;
468                 double distance;
469
470                 for(int i = 0; i < this.getConnectedNodes().size(); i++) {
471                       distance = this.distance(this.getConnectedNodes().get(i));
472                       if(distance > maxConnectionDistance) {
473                             maxConnectionDistance = distance;
474                       }
475                 }
476
477                 return maxConnectionDistance;
478           }
479
480           /**
481            * Returns the measured eventValue
482            * @return the measured eventValue
483            */
484           private double measureEventValue() {
485                 double measuredEventValue;
486
487                 Environment environment = (Environment)ContextUtils.getContext(this);
488                 Grid<SimpleAgent> grid = (Grid<SimpleAgent>)environment.getProjection("Grid");
489                 GridPoint gp = grid.getLocation(this);
490                 GridValueLayer eventValueLayer = (GridValueLayer)environment.getValueLayer("EventValue");
491                 measuredEventValue = eventValueLayer.get(gp.getX(), gp.getY());
492
493                 return measuredEventValue;
494           }
495
496           /**
497            * Search all nodes in range and save them in this.nodesInRange
498            */
499           private void initializeNodesInRange() {
500                 Environment environment = (Environment)ContextUtils.getContext(this);
501                 //This if-else-statement is just to optimize performance in a simulation
502                 if(environment.getNumSensorNodes() < Math.pow(this.getMaxConnectionDistance(), 2)) {
503                       SensorNode tempNode = null;
504                       for(int i = 0; i < environment.getObjects(SensorNode.class).size(); i++) {
505                             tempNode = (SensorNode)environment.getObjects(SensorNode.class).get(i);
506                             //Skip if tempNode = this
507                             if(this.equals(tempNode)) {
508                                   continue;
509                             } else if(this.distance(tempNode) < this.getSendingDistance()) {
510                                   this.getNodesInRange().add(tempNode);
```

```
511                                 }
512                           }
513                     } else {
514                           Grid<SimpleAgent> grid = (Grid<SimpleAgent>)environment.getProjection("Grid");
515                           GridPoint gp = grid.getLocation(this);
516                           for(int i = gp.getX() - (int)Math.ceil(this.getSendingDistance()); i < gp.getX() +
(int)Math.ceil(this.getSendingDistance()); i++) {
517                                 for(int j = gp.getY() - (int)Math.ceil(this.getSendingDistance()); j < gp.getY() +
(int)Math.ceil(this.getSendingDistance()); j++) {
518                                       if(environment.inEnvironment(i, j)) {
519                                             for(Object tempNode : grid.getObjectsAt(i, j)) {
520                                                   if(tempNode instanceof SensorNode && !tempNode.equals(this) &&
this.distance((SensorNode)tempNode) < this.getSendingDistance()) {
521                                                         this.getNodesInRange().add((SensorNode)tempNode);
522                                                   }
523                                             }
524                                       }
525                                 }
526                           }
527                     }
528               }
529
530               /**
531                * Initialize shortest path to the sinkNode
532                */
533               public void shortestPath(SinkNode sinkNode) {
534                     if(!this.isNetworkInitialized()) {
535                           this.initializeNetwork();
536                     }
537                     SensorNode nextNode;
538                     double distance;
539
540                     for(int i = 0; i < this.getNodesInRange().size(); i++) {
541                           nextNode = this.getNodesInRange().get(i);
542                           distance = this.getShortestPathDistance() + this.distance(nextNode);
543                           //Default value of shortestPathDistance is Double.MAX_VALUE
544                           if(nextNode.getShortestPathDistance() > distance) {
545                                 nextNode.setShortestPathDistance(distance);
546                                 nextNode.setShortestPathNode(this);
547                                 sinkNode.getShortestPathQueue().add(nextNode);
548                           }
549                     }
550               }
551
```

```
552         /**
553          * Builds the Gabriel-graph around this node
554          */
555         private void gg() {
556                 this.gg((double)Double.MAX_VALUE);
557         }
558 
559         /**
560          * Builds the Gabriel-graph around this node within a maximum distance
561          * @param maxDistance
562          */
563         private void gg(double maximumDistance) {
564                 SensorNode nodeToConnect = null;
565                 SensorNode otherNode = null;
566                 double r = 0;//The radius around the virtual point (x/y) between two points in which no other SensorNode must be
567                 double x = 0;//x-coordinate of the virtual point
568                 double y = 0;//y-coordinate of the virtual point
569                 boolean connect = true;
570                 Environment environment = (Environment)ContextUtils.getContext(this);
571                 Grid<SimpleAgent> grid = (Grid<SimpleAgent>)environment.getProjection("Grid");
572 
573                 for(int i = 0; i < this.getNodesInRange().size(); i++) {
574                         connect = true;
575                         nodeToConnect = this.getNodesInRange().get(i);
576                         GridPoint p1 = grid.getLocation(this);
577                         GridPoint p2 = grid.getLocation(nodeToConnect);
578                         x = (double)(p1.getX() + p2.getX()) / 2;        //Calculate the x-coordinate of the virtual point
579                         y = (double)(p1.getY() + p2.getY()) / 2;        //Calculate the y-coordinate of the virtual point
580                         r = this.distance(nodeToConnect) / 2;           //Calculate the radius around the virtual point
581 
582                         //Checks if no other point is in the radius r around the virtual point x/y
583                         for(int j = 0; j < this.getNodesInRange().size(); j++) {
584                                 otherNode = this.getNodesInRange().get(j);
585 
586                                 //The definition of the gg
587                                 if(otherNode.distance(x, y) <= r && !nodeToConnect.equals(otherNode)) {
588                                         connect = false;
589                                         break;
590                                 }
591                         }
592                         //Connect to nodeToConnect if connect = true
593                         if(connect) {
594                                 if(this.distance(nodeToConnect) <= maximumDistance) {
595                                         this.addEdge(nodeToConnect);
```

```
596                             }
597                         }
598                     }
599             }
600
601             /**
602              * Builds the relative neighborhood graph around this node
603              */
604             private void rng() {
605                     this.rng((double)Double.MAX_VALUE);
606             }
607
608             /**
609              * Builds the relative neighborhood graph around this node within a maximum distance
610              */
611             private void rng(double maximumDistance) {
612                     SensorNode nodeToConnect = null;
613                     SensorNode otherNode = null;
614                     double r = 0;//The distance between two nodes = the radius around both nodes
615                     boolean connect = true;
616                     Environment environment = (Environment)ContextUtils.getContext(this);
617                     Grid<SimpleAgent> grid = (Grid<SimpleAgent>)environment.getProjection("Grid");
618
619                     for(int i = 0; i < this.getNodesInRange().size(); i++) {
620                             connect = true;
621                             nodeToConnect = this.getNodesInRange().get(i);
622                             r = this.distance(nodeToConnect);               //Calculate the radius around the two nodes (=distance
    between the two nodes)
623
624                             //Checks whether no other point is in the intersection of the two circles around this and nodeToConnect
625                             for(int j = 0; j < this.getNodesInRange().size(); j++) {
626                                     otherNode = this.getNodesInRange().get(j);
627
628                                     //The definition of the rng
629                                     if(otherNode.distance(this) < r && otherNode.distance(nodeToConnect) < r) {
630                                             connect = false;
631                                             break;
632                                     }
633                             }
634                             //Connect to nodeToConnect if connect = true
635                             if(connect) {
636                                     if(this.distance(nodeToConnect) <= maximumDistance) {
637                                             this.addEdge(nodeToConnect);
638                                     }
```

```
639                         }
640                   }
641             }
642
643             /**
644              * Connect two nodes
645              * @param node2
646              * @return whether or not the connection could be made
647              */
648             public boolean addEdge(SensorNode node2) {
649                   boolean connected = false;
650                   Environment environment = (Environment)ContextUtils.getContext(this);
651                   Grid<SimpleAgent> grid = (Grid<SimpleAgent>)environment.getProjection("Grid");
652                   Network<SensorNode> network = (Network<SensorNode>)environment.getProjection("Network");
653                   double distance = this.distance(node2);
654
655                   //Don't connect a second time to the same node or if the node is too far away
656                   if(!this.getConnectedNodes().contains(node2) && !node2.getConnectedNodes().contains(this) && distance <=
this.getSendingDistance()) {
657                         this.getConnectedNodes().add(node2);
658                         node2.getConnectedNodes().add(this);
659                         network.addEdge(this, node2, distance);
660                         connected = true;
661                   }
662                   return connected;
663             }
664
665             /**
666              * Calculates the euclidean distance between two objects (this, o2)
667              * @param o2
668              * @return the distance
669              */
670             public double distance(SimpleAgent a2) {
671                   Environment environment = (Environment)ContextUtils.getContext(this);
672                   Grid<SimpleAgent> grid = (Grid<SimpleAgent>)environment.getProjection("Grid");
673                   GridPoint p1 = grid.getLocation(this);
674                   GridPoint p2 = grid.getLocation(a2);
675
676                   return this.distance((double)p1.getX(), (double)p1.getY(), (double)p2.getX(), (double)p2.getY());
677             }
678
679             /**
680              * Calculates the euclidean distance between an object (this) and a coordinate-pair (x2/y2)
681              * @param x2
```

```
682          * @param y2
683          * @return the distance
684          */
685         public double distance(double x2, double y2) {
686                Environment environment = (Environment)ContextUtils.getContext(this);
687                Grid<SimpleAgent> grid = (Grid<SimpleAgent>)environment.getProjection("Grid");
688                GridPoint p1 = grid.getLocation(this);
689
690                return this.distance((double)p1.getX(), (double)p1.getY(), x2, y2);
691         }
692
693         /**
694          * Calculates the euclidean distance between two coordinate-pairs (x1/y1, x2/y2)
695          * @param x1
696          * @param x2
697          * @param y1
698          * @param y2
699          * @return the distance
700          */
701         public double distance(double x1, double y1, double x2, double y2) {
702                double dx = x1 - x2;
703                double dy = y1 - y2;
704
705                return Math.sqrt((dx * dx) + (dy * dy));
706         }
707
708         /**
709          * Increases numOfMessages by 1
710          * @param decentralized
711          */
712         public void increaseNumOfMessages(boolean decentralized) {
713                this.increaseNumOfMessages(1, decentralized);
714         }
715
716         /**
717          * Increases numOfMessages by n
718          * @param n
719          * @param decentralized
720          */
721         public void increaseNumOfMessages(int n, boolean decentralized) {
722                if(decentralized) {
723                       this.setNumOfMessagesDecentralized(this.getNumOfMessagesDecentralized() + n);
724                } else {
725                       this.setNumOfMessagesCentralized(this.getNumOfMessagesCentralized() + n);
```

```
726                 }
727         }
728
729         /**
730          * Returns the average number of messages per step
731          * @return the average number of messages per step
732          */
733         public double averageNumOfMessagesPerStepDecentralized() {
734                 return this.getNumOfMessagesDecentralized() / this.getTickCount();
735         }
736
737         /**
738          * Returns the average number of messages per step (centralized)
739          * @return the average number of messages per step (centralized)
740          */
741         public double averageNumOfMessagesPerStepCentralized() {
742                 return this.getNumOfMessagesCentralized() / this.getTickCount();
743         }
744
745         /**
746          * Returns the ratio of sent messages between centralized and decentralized.
747          * 0 = centralized and decentralized sent the same amount of messages or one of them is 0
748          * x > 1 = decentralized algorithm sent x times more messages
749          * x < -1 = centralized algorithm sent -x times more messages
750          * @return ratio
751          */
752         public double ratioDecentralizedCentralized() {
753                 if(this.getNumOfMessagesCentralized() == 0 || this.getNumOfMessagesDecentralized() == 0 ||
    this.getNumOfMessagesCentralized() == this.getNumOfMessagesDecentralized()) {
754                         return 0;
755                 } else if(this.getNumOfMessagesDecentralized() > this.getNumOfMessagesCentralized()) {
756                         return this.getNumOfMessagesDecentralized() / this.getNumOfMessagesCentralized();
757                 } else {
758                         return -(this.getNumOfMessagesCentralized() / this.getNumOfMessagesDecentralized());
759                 }
760         }
761
762         //Getter and setter methods
763
764         public double getSendingDistance() {
765                 return sendingDistance;
766         }
767
768         public double getEventValue() {
```

```
769                 return eventValue;
770         }
771
772         public void setEventValue(double eventValue) {
773                 this.eventValue = eventValue;
774         }
775
776         public void setSendingDistance(double sendingDistance) {
777                 this.sendingDistance = sendingDistance;
778         }
779
780         public int getNumOfMessagesDecentralized() {
781                 return numOfMessagesDecentralized;
782         }
783
784         public void setNumOfMessagesDecentralized(int numOfMessagesDecentralized) {
785                 this.numOfMessagesDecentralized = numOfMessagesDecentralized;
786         }
787
788         public int getNumOfMessagesCentralized() {
789                 return numOfMessagesCentralized;
790         }
791
792         public void setNumOfMessagesCentralized(int numOfMessagesCentralized) {
793                 this.numOfMessagesCentralized = numOfMessagesCentralized;
794         }
795
796         public boolean isInitialized() {
797                 return initialized;
798         }
799
800         public void setInitialized(boolean initialized) {
801                 this.initialized = initialized;
802         }
803
804         public boolean isNetworkInitialized() {
805                 return networkInitialized;
806         }
807
808         public void setNetworkInitialized(boolean networkInitialized) {
809                 this.networkInitialized = networkInitialized;
810         }
811
812         public double getMaxConnectionDistance() {
```

```
813                 return maxConnectionDistance;
814           }
815
816           public void setMaxConnectionDistance(double maxConnectionDistance) {
817                 this.maxConnectionDistance = maxConnectionDistance;
818           }
819
820           public Vector<SensorNode> getNodesInRange() {
821                 return nodesInRange;
822           }
823
824           public void setNodesInRange(Vector<SensorNode> nodesInRange) {
825                 this.nodesInRange = nodesInRange;
826           }
827
828           public Vector<SensorNode> getConnectedNodes() {
829                 return connectedNodes;
830           }
831
832           public void setConnectedNodes(Vector<SensorNode> connectedNodes) {
833                 this.connectedNodes = connectedNodes;
834           }
835
836           public double getDetectionArea() {
837                 return detectionArea;
838           }
839
840           public void setDetectionArea(double detectionArea) {
841                 this.detectionArea = detectionArea;
842           }
843
844           public SensorNode getShortestPathNode() {
845                 return shortestPathNode;
846           }
847
848           public void setShortestPathNode(SensorNode shortestPathNode) {
849                 this.shortestPathNode = shortestPathNode;
850           }
851
852           public double getShortestPathDistance() {
853                 return shortestPathDistance;
854           }
855
856           public void setShortestPathDistance(double shortestPathDistance) {
```

```
857                 this.shortestPathDistance = shortestPathDistance;
858         }
859 }
```

8.1.7 SinkNode.java

```
1  package geosensornetwork;
2  import java.util.Vector;
3
4  /**
5   *
6   * @author Raphael Renaud
7   *
8   */
9
10 public class SinkNode extends SensorNode {
11
12        protected Vector<SensorNode> shortestPathQueue = new Vector<SensorNode>();
13
14        public SinkNode() {
15              super();
16        }
17
18        public void initialize() {
19              super.initialize();
20              this.setInitialized(false);
21
22              //Prepare calculation of shortestPath and build it
23              this.getShortestPathQueue().add(this);
24              this.setShortestPathDistance(0);
25              this.setShortestPathNode(this);
26              this.shortestPathQueue();
27
28              this.setInitialized(true);
29        }
30
31        /**
32         * Calls shortestPath of each sensorNode (similar to the breadthFirstSearch)
33         */
34        public void shortestPathQueue() {
35              while(this.getShortestPathQueue().size() > 0) {
36                    this.getShortestPathQueue().remove(0).shortestPath(this);
37              }
38        }
```

```
39
40          /**
41           * Reports a fire along shortestPath to next sinkNode
42           * @param to
43           * @param area
44           * @param decentralized
45           */
46          public void report(SensorNode to, double area, boolean decentralized) {
47                  //Do nothing (override the method in SensorNode.java; In reality, here would be a procedure which sends the
    information to a central server over the internet/satellite
48          }
49
50          //Getter and setter methods
51
52          public Vector<SensorNode> getShortestPathQueue() {
53                  return shortestPathQueue;
54          }
55
56          public void setShortestPathQueue(Vector<SensorNode> shortestPathQueue) {
57                  this.shortestPathQueue = shortestPathQueue;
58          }
59
60          public void setNumOfMessagesCentralized(int numOfMessagesCentralized) {
61                  this.numOfMessagesCentralized = 0;
62          }
63
64          public void setNumOfMessagesDecentralized(int numOfMessagesDecentralized) {
65                  this.numOfMessagesDecentralized = 0;
66          }
67  }
```

8.1.8 Monitor.java

```
1  package geosensornetwork;
2  import repast.simphony.util.ContextUtils;
3
4  public class Monitor extends SimpleAgent {
5        private boolean initialized;
6
7        private int correlationRadius = 1;
8        private double[][] correlationWeights = new double[2*this.getCorrelationRadius()+1][2*this.getCorrelationRadius()+1];
9        private double correlationSumWeights = 0;
10       private double correlation = 0;
11       private double correlationSum = 0;
12       private int eventSize = 0;
13
14       public Monitor() {
15             this.setInitialized(false);
16       }
17
18       public void step1() {
19             if(!this.isInitialized()) {
20                   this.setInitialized(this.initialize());
21             }
22       }
23
24       public void step2() {
25             this.setCorrelation(this.calculateCorrelation());
26             this.setCorrelationSum(this.getCorrelationSum() + this.getCorrelation());
27
28       }
29
30
31       /**
32        * Initializes the monitor
33        * @return true
34        */
35       public boolean initialize() {
36             //Calculates correlation's static parameters
37             Environment environment = (Environment)ContextUtils.getContext(this);
38             double[][] weights = new double[2*this.getCorrelationRadius()+1][2*this.getCorrelationRadius()+1];
39             double distance;
40             for(int i = 0; i < weights.length; i++) {
41                   for(int j = 0; j < weights[0].length; j++) {
42                         //The middlepoint is weighted 0
```

```
43                              distance = environment.distance(this.getCorrelationRadius(), this.getCorrelationRadius(), i, j);
44                              if(i == this.getCorrelationRadius() && j == this.getCorrelationRadius() || distance >
this.getCorrelationRadius()) {
45                                      weights[this.getCorrelationRadius()][this.getCorrelationRadius()] = 0;
46                              } else {
47                                      weights[i][j] = 1 / distance;
48                                      this.setCorrelationSumWeights(this.getCorrelationSumWeights() + weights[i][j]);
49
50                              }
51                      }
52              }
53              this.setCorrelationWeights(weights);
54
55              return true;
56      }
57
58      /**
59       * Calculates the correlation of the BooleanTemperature layer and counts the total event-size
60       * @return correlation
61       */
62      public double calculateCorrelation() {
63              if(!this.isInitialized()) {
64                      this.setInitialized(this.initialize());
65              }
66              Environment environment = (Environment)ContextUtils.getContext(this);
67              double sum = 0;
68              double sumLocal = 0;
69              int countLocalCorrelations = 0;
70              double sumWeights;
71              double correlation = 0;
72
73              for(int x = 0; x < environment.getXDim(); x++) {
74                      for(int y = 0; y < environment.getYDim(); y++) {
75                              if(environment.getValue(x, y, "BooleanEventValue") == 1) {
76                                      countLocalCorrelations++;
77                                      sumLocal = 0;
78                                      sumWeights = this.getCorrelationSumWeights();
79                                      for(int i = x-this.getCorrelationRadius(); i <= x+this.getCorrelationRadius(); i++) {
80                                              for(int j = y-this.getCorrelationRadius(); j <= y+this.getCorrelationRadius(); j++) {
81                                                      if(environment.inEnvironment(i, j)) {
82                                                              sumLocal += environment.getValue(i, j, "BooleanEventValue") *
this.getCorrelationWeights()[i-x+this.getCorrelationRadius()][j-y+this.getCorrelationRadius()];
83                                                      } else {
84                                                              sumWeights -= this.getCorrelationWeights()[i-
```

```
    x+this.getCorrelationRadius()][j-y+this.getCorrelationRadius()];
85                                                  }
86                                              }
87                                          }
88                                          sum += sumLocal / sumWeights;
89                                      }
90                                  }
91                          }
92                          //Probability can't be below 0
93                          if(countLocalCorrelations > 0 && sum > 0) {
94                                  correlation = sum / countLocalCorrelations;
95                                  this.setEventSize(countLocalCorrelations);
96                          }
97                          return correlation;
98                  }
99
100                 /**
101                  * Returns the average calculated Correlation
102                  * @return the average calculated Correlation
103                  */
104                 public double averageCorrelation() {
105                         return this.getCorrelationSum() / this.getTickCount();
106                 }
107
108                 public boolean isInitialized() {
109                         return initialized;
110                 }
111
112                 public void setInitialized(boolean initialized) {
113                         this.initialized = initialized;
114                 }
115
116                 public int getCorrelationRadius() {
117                         return correlationRadius;
118                 }
119
120                 public void setCorrelationRadius(int correlationRadius) {
121                         this.correlationRadius = correlationRadius;
122                 }
123
124                 public double[][] getCorrelationWeights() {
125                         return correlationWeights;
126                 }
127
```

```
128         public void setCorrelationWeights(double[][] correlationWeights) {
129                 this.correlationWeights = correlationWeights;
130         }
131
132         public double getCorrelationSumWeights() {
133                 return correlationSumWeights;
134         }
135
136         public void setCorrelationSumWeights(double correlationSumWeights) {
137                 this.correlationSumWeights = correlationSumWeights;
138         }
139
140         public double getCorrelation() {
141                 return correlation;
142         }
143
144         public void setCorrelation(double correlation) {
145                 this.correlation = correlation;
146         }
147
148         public double getCorrelationSum() {
149                 return correlationSum;
150         }
151
152         public void setCorrelationSum(double correlationSum) {
153                 this.correlationSum = correlationSum;
154         }
155
156         public int getEventSize() {
157                 return eventSize;
158         }
159
160         public void setEventSize(int eventSize) {
161                 this.eventSize = eventSize;
162         }
163 }
```

8.2 Persönliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und die den verwendeten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Villmergen, 10.01.2011

Raphael Renaud