

# Clusterung von Sensorknoten zur Waldbranddektion

Robert Oehlmann, Jan Winkelmann  
Hamburg University of Technology  
Institute of Communication Networks

## I. EINLEITUNG

**D**IESER Bericht dokumentiert ein Protokoll zur Bildung von Clustern aus Sensorknoten um Strom zu sparen. Innerhalb der Cluster wird kein Routing benötigt, und jedes Mitglied des Clusters kann die Verwaltungsaufgaben übernehmen.

Wir beginnen mit einer weiterführenden Beschreibung des Anwendungsszenarios und den getroffenen Annahmen in Abschnitt II. Darauf folgt Abschnitt III mit der Spezifikation der funktionalen Anforderungen. Abschnitt IV beschäftigt behandelt das entwickelte Protokoll. Abschnitt ?? und VI **beschreibt robert**. Als letztes werden in Abschnitt VII Ansätze in für zukünftige Arbeit erwähnt, und in Abschnitt VIII ein Fazit gezogen.

## II. SETTING

Dieser Abschnitt beschreibt zuerst das Anwendungsgebiet, und danach die Annahmen die bei dem Design des Protokolles gemacht wurden. Während des Entwerfens des Protokolles, wurde nicht drauf geachtet das Anwendungsszenario realistisch zu gestalten. Weder das Anwendungsszenario noch Annahmen beruhen auf recherchierten Fakten.

### A. Anwendungsgebiet

Um Frühwarnsysteme für Waldbrände zu verbessern könnten z.B. kleine Sensoren eingesetzt werden, welche sich untereinander vernetzen um so große Flächen messen zu können. In unserem Szenario seien Sensoren gegeben, die neben dem Messen von Metriken die für die Waldbranderkennung nötig sind, auch über zwei Kommunikationsmethoden verfügen. Eine Kommunikationsmöglichkeit zur Verständigung mit anderen Sensoren in der unmittelbaren Umgebung, wie zum Beispiel WLAN, und eine zum Senden der Messergebnisse an eine Senke, wie zum Beispiel GPRS.

Ziel dieses Projektes es nun, Sensorknoten in *Cluster* zu organisieren. Cluster benutzen Kurzstrecken-Kommunikation um den Sensoren zu ermöglichen Batterien zu sparen, indem die Messdaten bei manchen Sensoren gesammelt werden, und so nicht jeder Sensor seine Daten einzeln zu der Senke senden muss. Der Sensor der die Messdaten zur Senke sendet bezeichnen wir als *Clusterhead*.

### B. Annahmen

Die technischen Voraussetzungen für unseren Ansatz sind die Folgenden:

- Alle Sensorknoten sind baugleich, d.h. jeder Sensorknoten kann mit anderen Sensorknoten und der Senke kommunizieren.
- Keine zwei Sensorknoten versuchen gleichzeitig einem Cluster beizutreten. Dies wird Momentan dadurch erreicht, dass alle sich Sensorknoten nacheinander aktivieren.
- Sensorknoten bewegen sich nicht.
- Keine Ausfälle von einzelnen Verbindungen. Ein Sensorknoten erreicht entweder alle Knoten seines Clusters, oder keine.
- Keine temporären Ausfälle. Ein Sensorknoten fällt entweder nicht oder komplett aus.
- Wenn ein Sensorknoten ausfällt, so merken es alle Sensorknoten im Cluster gleichzeitig und mit Sicherheit, aber nicht zwingend instantan.

Zusätzlich nehmen wir an, dass ein unterliegendes Protokoll existiert, das folgendes garantiert:

- Garantierte Zustellung von Nachrichten
- Zustellung der Nachrichten in der richtigen Reihenfolge

Ein Beispiel hierfür wäre ein TCP/IP Stack.

## III. FUNKTIONALE ANFORDERUNGEN

### 1) Implementation des beschriebenen Protokoll

- *Beschreibung:*  
Das Protokoll bietet die Hauptfunktionalität für das Anwendungsszenario. Eine vollständige und fehlerfreie Implementation ist daher *kritisch*.
- *Siehe:*  
Abschnitt IV für die Spezifikation des Protokolls.

- *Abnahmekriterium:*

Diese Anforderung gelte als erfüllt, wenn die Implementation folgendes leistet: das Bilden von Verbünden, das Behandeln von Ausfällen von Sensorknoten und die Rotation von Verbundsleitern anhand des genannten Protokolls.

## 2) Kapselung der Komponenten

- *Beschreibung:*

Einzelne Systemkomponenten haben auf realistische Weise voneinander gekapselt sein. Dies bezieht sich insbesondere, aber nicht ausschließlich auf den Motes zur Verfügung stehenden Informationen, wie Lage, Informationen über andere Sensorknoten, sowie Zustellung der Kommunikationen unter dem Motes durch einen von dem Sensorknoten unabhängigen Mechanismus.

- *Problembeschreibung:*

Um eine möglichst realistische Simulation zu sein, und um als echter Proof-Of-Concept zu gelten, müssen die simulierten Komponenten voneinander entkoppelt sein, also über Schnittstellen kommunizieren, welche auch in einem Anwendungsszenario bestehen würden.

- *Abnahmekriterium:*

Zur Erfüllung dieser Anforderung sind nötig: Kapselung der Mote-Objekte untereinander, Kapselung der Mote-Objekte von der Simulationsumgebung und ein Mechanismus zum Nachrichtenversenden.

## 3) Graphische Oberfläche

- *Beschreibung:*

Die Implementation hat eine graphische Benutzeroberfläche zur Verfügung zu stellen, welche die Simulation visualisiert. Die Bedienung der Oberfläche hat ein sinnvolles Subset der Gesamtfunktionalität zu implementieren.

- *Problembeschreibung:*

Um die Simulation zu visualisieren ist eine graphische Oberfläche unentbehrlich. Sie dient zur demonstration des Proof-Of-Concepts. Das Hinzufügen und Entfernen von Motes erlaubt das Demonstrieren des vollen Umfanges der Funktionalität.

- *Abnahmekriterium:*

Die Implementation muss eine graphische Benutzeroberfläche haben, welche die simulierte Umgebung, samt Sensorknoten darstellt. Gebildete Cluster müssen farblich markiert sein. Zusätzlich soll man Sensorknoten hinzufügen und entfernen können.

## IV. PROTOKOLL

Dieser Abschnitt beschäftigt sich mit den entwickelten Algorithmen für das bilden von Clustern, die Ausfallsicherheit, und das Rotieren von Clusterheads.

### A. Clusterbildung

Die Grundlage des Bildes von Clustern ist das Finden von vollständigen Sub-Graphen. Die Knoten des Graphen sind die Sensorknoten, und die Kanten die Verbindungen zu anderen Sensorknoten in Reichweite. Dadurch reduziert sich die Clusterbildung auf das Cliquesproblem, ist aber aufgrund der getroffenen Annahmen nicht in *NP*.

Sensorknoten ordnen sich nach der Aktivierung einem Cluster zu, und verlassen diesen nicht, bis der Cluster sich auflöst. Es werden keine Optimierungen an bestehenden Clustern vorgenommen. Siehe Bild IV-A für die Finite-State-Machine des Protokolls.

Cluster werden durch folgendes Protokoll gebildet:

- 1) Falls sich ein Sensor aktiviert, so sendet sie zuerst eine Nachricht die nach vorhandenen Clustern sucht. Findet der Sensor keine vorhandenen Cluster, so bildet er selber einen.
- 2) Alle schon vorhandenen Clusterheads antworten auf die Anfragen von neuen Knotens. Diese Antwort enthält einen Identifikator des Clusters und die Anzahl der Sensoren in dem Cluster.
- 3) Der neue Sensor speichert alle Antworten der vorhanden Cluster, ordnet sie nach Größe und versucht der Reihe nach einem der Cluster beizutreten, beginnend mit dem Kleinsten.
- 4) Der erste Schritt zum Beitreten eines Clusters, das Senden eine Nachricht, auf die alle Mitglieder des Clusters mit ihrer Id antworten.
- 5) Nach dem Ablauf eines Timeouts, sendet der neue Sensor die Ids aller empfangenen Sensoren an den Clusterhead. Dies stellt sicher, dass der neue Sensor alle schon vorhandenen Mitglieder erreichen kann.
- 6) Falls die Nachricht des neuen Sensors alle Ids des aktuellen Clusters enthält, so sendet der Clusterhead dem neuen Sensor eine Nachricht mit der Bestätigung, dass er neue Sensor dem Cluster beigetreten ist. Zusätzlich ordnet der Clusterhead dem neuen Sensor einen Slot zu. Dieser Slot wird nötig, falls der Clusterhead ausfällt.
- 7) Falls die Nachricht des neuen Sensors nicht alle Ids enthalten sollte, so sendet der Server eine Ablehnung und der Client versucht dem nächst größeren Cluster beizutreten.

Bild 2 ist ein UML-Sequenzdiagramm welches einen erfolgreichen Clusterbeitritt mit einem weiterem Sensorknoten illustriert.

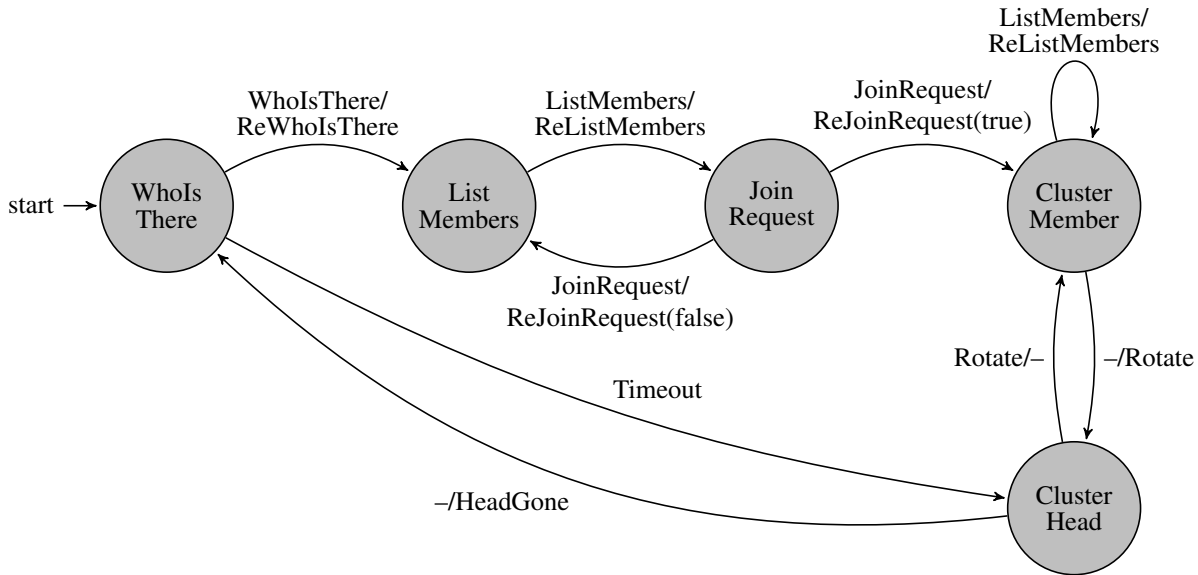


Fig. 1. State Machine des Cluster Protokolls

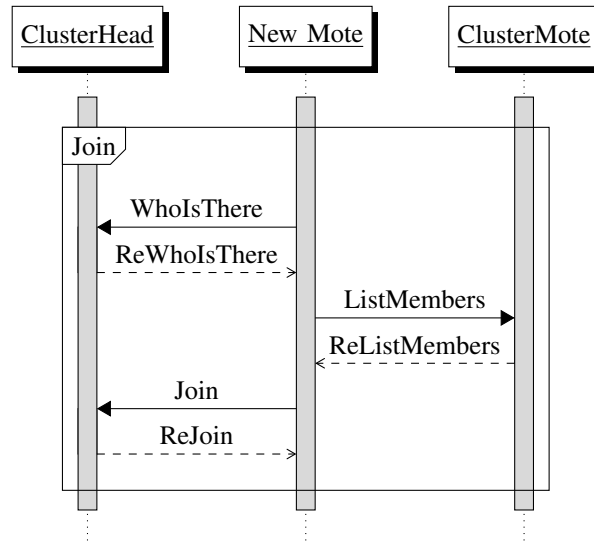


Fig. 2. Sequenzdiagramm für einen erfolgreichen Clusterbeitritt

### B. Ausfallsicherheit

Aufgrund der erwähnten Annahmen (TCP/IP, keine Bewegung der Sensorknoten, etc) gibt es nur zwei Ausfallszenarien die betrachtet werden müssen, der Ausfall des Clusterheads, sowie der Ausfall eines Sensorknotens der normales Mitglied in einem Cluster ist. Falls ein normaler Sensorknoten ausfallen sollte, so muss der zugehörige Clusterhead dies merken und die Liste der zum Cluster zugehörigen Sensorknoten aktualisieren. Da Sensorknoten die dem Cluster betreten wollen Kontakt zu allen Sensorknoten nachweisen müssen, die der Cluster als zum Cluster zugehörig gespeichert hat, würde der unbemerkte Ausfall eines Sensorknotens dazu führen, dass keine neuen Knoten beitreten können. In einem realen Szenario erkennt der Clusterhead das Ausfallen eines normalen Knotens an dem Fehler von gesendeten Sensordaten. In der nie vorgestellten Simulation wird durch das Löschen direkt beim Server ein Löschevent.

Der Ausfall von einem Clusterhead wird durch das Fehlen von Bestätigungsnachrichten erkannt. Aufgrund unserer Annahme geschickt dies bei allen Knoten im Cluster gleichzeitig. Sobald ein Knoten den Ausfall des Clusterheads wahrnimmt, so wird der Cluster als tot angenommen. Der Knoten wartet nun eine bestimmte Zeit  $b$  und versucht dann einem neuen Cluster beizutreten.  $b$  berechnet sich durch die upper bound für das Beitreten eines Sensorknotens zu einem Cluster multipliziert mit dem "slot" welcher dem Knoten beim Beitreten des jetzt toten Clusters zugewiesen wurde. Der Slot ist ein streng monoton ansteigender, pro Cluster eindeutiger Integer. Somit wird sichergestellt, dass die Knoten alle nacheinander in der ursprünglichen Beitrittsreihenfolge versuchen einem neuen Cluster beizutreten.

### C. Rotation der Clusterheads

Die Rotation des Clusterhead ist der Prozess welcher von dem aktuellen Clusterhead ausgeführt wird, und einem anderen Knoten die Führung des Clusters zuweist. Dieser Rotation kann ausgelöst werden von z.B. einem timeout oder dem Batteriestand. Bei der Durchführung der Rotation wählt der Clusterhead den neuen Cluster und sickt alle relevanten Daten an ihn. Die gesendeten Daten sind: Der Slot der als nächstes vergeben wird, eine Liste der Ids aller Knoten in dem Cluster, sowie die Id des Clusters.

## V. IMPLEMENTIERUNG

Um die dargelegten Algorithmen zu testen und zu demonstrieren, haben wir eine Simulation entwickelt. Im folgenden wird zunächst die gewählte Entwicklungsplattform vorgestellt und begründet, anschließend werden die Implementierung des Äthers, der Motes und die Spezifika der Simulations-Umgebung dargestellt.

### A. JavaScript

Die Simulation ist in JavaScript implementiert; als Entwicklungsplattform haben wir Webbrowser mit aktueller JavaScript-Engine (z.B. Googles V8 und Mozillas JägerMonkey) und Unterstützung für 2D-Grafikdarstellung mithilfe des canvas-Tags gewählt.

Mit JavaScript als dynamisch-typisierte Skriptsprache lassen sich ohne große Deklarations-Overhead schnell sichtbare Ergebnisse produzieren. Mit dem neuen HTML5 Canvas Element steht eine einfach zu nutzende Zeichenfläche zur Verfügung und für alle gängigen Betriebssysteme gibt es Browser die die Simulation aufführen können.

### B. Motes

Die simulierten Motes sind Instanzen der Klasse Mote, deren Implementierung sich über die Dateien `mote.js`, `mote_member.js` und `mote_head.js` erstreckt.

Das in Abschnitt ?? dargestellte Protokoll ist vollständig, aber zustandslos, in der Klasse Mote implementiert.

Wird eine neue Mote instanziiert, initialisiert der Konstruktor einige interne Variablen, registriert die neue Mote dann beim Äther und kann dann gestartet ("eingeschaltet") werden.

### C. Äther

Die Motes kommunizieren untereinander nur über Nachrichten. Den (virtuellen) Versand der Nachrichten übernimmt der simulierte Äther, implementiert als Singleton "MoteList" in der Datei `motelist.js`.

Ähnlich dem Observer-Entwurfsmuster stellt MoteList eine Methode "register" bereit, die neuer Motes aufnimmt. Sie wird vom Mote-Konstruktor aufgerufen. Zudem gibt es eine Methode "send", die den Versand von Nachrichten anstößt.

Beim Versand wird zunächst die Position des Absenders abgerufen, anschließend wird die Distanz zu allen anderen registrierten Motes berechnet. Ist die Distanz nicht größer als ein festgelegter Senderadius, wird die Nachricht zugestellt indem sie der "onRecv" Methode der jeweiligen Empfänger-Mote übergeben wird.

### D. Grafische Oberfläche

Die grafische Oberfläche beschränkt sich im wesentlichen auf eine "Karte" in der alle Motes, möglichen Verbindungen, sowie Cluster verzeichnet sind.

Die Zeichenfunktion ist dabei in der MoteList implementiert, da nur diese Zugriff auf die Positionen der Motes hat.

Die Motes sind als kleine Quadrate auf der Karte dargestellt, oberhalb eines Cluster-Heads wird jeweils die ID des Clusters angezeigt. Zwischen Motes die nicht mehr als einen Senderadius voneinander entfernt sind, ist eine blasser Linie eingezeichnet. Zwischen zwei Motes die zu einem Cluster gehören, wird diese Linie farblich hervorgehoben.

Weitere Motes werden hinzugefügt, wenn man auf eine freie Fläche der Karte klickt. Klickt man eine vorhandene Mote an, wird diese entfernt.

Die Rotation der Cluster-Heads wird durch Drücken der Taste 'r' angestoßen. Die Taste 'a' schaltet eine automatische Rotation der Cluster-Heads in einem fixen Intervall ein oder aus.

## VI. ANALYSE

Robert... (This is also a placeholder)

## VII. FUTURE WORK

Die zur Verfügung gestellte Implementierung des beschriebenen Protokolls erfüllt die funktionalen Anforderungen im vollen Maße. Um aber den Einsatz des Protokolls realistischer zu machen, ist noch Arbeit an den Annahmen aus Abschnitt II nötig. So wäre es möglich das Protokoll um die Möglichkeit zu erweitern mehreren Sensorknoten zu erlauben gleichzeitig einem Cluster beizutreten. Dies wäre zum Beispiel realisierbar, in dem der Clusterhead bei einem schon laufenden join-Vorgang blockiert, und der Sensorknoten wartet und danach erneut versucht dem Cluster beizutreten.

Zusätzlich müsste das Protokoll auf in der Realität genutzte drahtlose Kommunikationprotokolle, die unter anderem kein Multicast unterstützen, angepasst werden.

## VIII. FAZIT

Dieser Bericht stellt ein Protokoll vor, welches das Bilden von Sensorknoten-Clustern erlaubt. Es unterstützt neben dem Bilden von Clustern auch den Ausfall von Knoten, sowie die Rotation des Clusterheads. Bereitgestellt wird zusätzlich eine Implementation des Protokolls in Form einer Simulation. Umfang der Simulation ist neben einer graphischen Darstellung auch ein gekapseltes Simulationsumfeld, und die Möglichkeit Motes hinzuzufügen, oder zu entfernen.