FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Department Informatik

Technical Reports / ISSN 2191-5008

Juergen Eckert, Abdalkarim Awad,
Kai-Steffen Hielscher, David Eckhoff (Hrsg.)

14. GI/ITG KuVS Fachgespräch Sensornetze

Technical Report CS-2015-06

23.-24. September 2015

Please cite as:
Juergen Eckert, Abdalkarim Awad, Kai-Steffen Hielscher, David Eckhoff (Eds.), “Proceedings of the 14. GI/ITG KuVS Fachgespräch Sensornetze,” Friedrich-Alexander-Universität Erlangen-Nürnberg, Dept. of Computer Science, Technical Reports, CS-2015-06, September 2015.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Department Informatik

Martensstr. 3 · 91058 Erlangen · Germany

www.cs.fau.de

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

14. GI/ITG KuVS Fachgespräch Sensornetze

der GI/ITG Fachgruppe
Kommunikation und Verteilte Systeme

23.-24. September 2015 in Erlangen

KuVS

GI ITG

Organisation

General Chair
Juergen Eckert

Technical Program
Abdalkarim Awad
Kai-Steffen Hielscher

Publication
David Eckhoff

Local Arrangements
Christian Götz

Social Event
Sebastian Schellenberg

Inhaltsverzeichnis

Session 1.1: Anwendungen

Session 1.2: Internet of Things

Session 1.3: Quo Vadis?

Session 2.1: Sicherheit

Session 2.2: Zuverlässigkeit

Octocopter Based Autonomous Cost over Coverage Deployment of Wireless Sensor Networks

Rafael Funke
University of Koblenz-Landau
Email: rfunke@uni-koblenz.de

Hannes Frey
University of Koblenz-Landau
Email: frey@uni-koblenz.de

***Abstract*—We present a prototype implementation for autonomous sensor network deployment. The utilized deployment algorithm considers two conflicting goals: (1) maximizing the coverage area while (2) maintaining good multi-hop connectivity from sensor nodes to a data sink. In an interdisciplinary research project we have built tiny sensor nodes fitting into bionic winged seeds that can be deployed with a flying drone. We had a couple of successful test flights with 15 nodes deployed on a soccer field. In this work we present these preliminary results as a proof of concept to demonstrate the principal applicability of the system.**

I. Introduction

Wireless sensor networks usually have two main functions. First, they shall monitor an area with their sensors. Second, they shall send sensor readings multi-hop to a sink. Both properties depend on the deployment of the sensor network. In our previous work [1] we proposed to use channel quality feedback within the deployment process to improve communication properties of the network. The idea is that whenever a node is deployed, the channel qualities are measured within the partially deployed network and used by the autonomous deployment device as feedback to determine the next best deployment position.

In this paper we consider a similar but improved deployment algorithm. We define a cost function that minimizes the ratio of expected multi-hop transmission costs to the sink over the sensor network coverage. For transmission costs we consider that each node must send one packet (containing its sensing values) multi-hop over erroneous channels to the sink. For computing sensor coverage we assume a uniform sensing disk around each sensor node that is larger than half the communication range. The covered area is the union over all disks.

We present the system implementation, simulation results that show the benefits of our algorithm and preliminary outdoor deployment experiments. We ran our algorithm on a flying copter which autonomously deployed sensor nodes on a soccer field. With these experiments we show that in principle our algorithm is applicable in the real world, provide a proof of concept that our implemented system is useful for testing autonomous deployment algorithms in a real world experiment, and, furthermore, share our experiences and encountered problems which should be valuable for other researchers who plan to do outdoor experiments with autonomous sensor network deployments.

II. Related Work

Deployment algorithms for wireless sensor networks can either be static or dynamic and as well either be controlled or random. The difference between static and dynamic deployments is whether nodes can be repositioned once they are deployed. Controlled and random deployments differ in whether the nodes are positioned individually or whether they are just dropped randomly.

In this work we consider static controlled deployments. Existing work uses regular patterns known in advance to deploy nodes. One approach is the regular triangle tesselation, which builds a lattice of equilateral triangles. It has been shown that it provides optimal coverage with least number of nodes [2], [3] when the sensing range is given as binary sensing disk. In the case that communication and coverage range are equal, Kar and Banerjee showed that the pattern r-strip [4] provides near-optimum solution. However, this assumes the network to be a unit disk graph (UDG) which a real-world topology will not be like.

An entirely different approach for static controlled deployment was given by Al-Turjman et al. [5]. They propose to do a so called leaser scan from an airplane to get a 3D model of the ground. With this model they predict channel qualities between positions offline. We see disadvantages in this approach compared to ours as it relies on a channel model, which is always an abstraction of the reality. Further, deviations from planned deployment distances to real distances may impact the predicted channel qualities. This especially holds if nodes are dropped from a flying drone rather than from a ground robot.

III. Cost-based Deployment Algorithm with Feedback

The considered deployment algorithm initially deploys two nodes. The first is put in the center of the deployment area and the second at static distance from it. After that, successively a mesh of triangles is set up such that the union over all sensing disks contains no holes (see Fig. 1 for an example deployment with 80 nodes; to avoid clutter only one sensing disk is shown). In each iteration the algorithm determines the position of the node on the border of the current triangle mesh that is closest to the center. That node, its right or left neighbor node (depending on which is closer to the center), and the next node v to be deployed form the next triangle to be added to the triangle mesh. The deployment position of v will be the position which minimizes the ratio between added costs over coverage gain. The costs are the expected number of transmissions for sending

the nodes' packet to the sink – including retransmissions due to lossy channels – when following the shortest path along the edges of the constructed triangle graph. The coverage gain is found geometrically. It is the area of the newly deployed node's sensing disk minus the overlapping parts of the other nodes' sensing disks.

The rationale behind transmission cost over coverage gain ratio is to consider two conflicting goals in one cost function. On the one hand reducing the number of nodes on a covered area reduces the number of sensor readings and their required transmissions. On the other hand reducing the number of nodes increases the distance between nodes and thus increases the number of required retransmissions to get the readings multi-hop from the sensor to the sink.

In our scenario, each deployed node introduces the costs of one more measurement sample that is sent via shortest path routing to the sink over erroneous channels. The shortest path is computed using the number of expected transmissions as link costs (per link one transmission plus the costs for retransmissions due to packet errors). The number of transmissions of each hop is derived from the channel quality in terms of signal-to-noise ratio (SNR) using outage probability. That gives us the bit error probability, which leads us to the packet error probability and the expected number of transmissions:

The outage probability in Equation 1[1] gives us the probability P_e of a bit error for given linear SNR value Γ:

$$P_e = \frac{1}{2(1+\Gamma)} \quad (1)$$

The probability P_c for the successful reception of a packet with length n bits is the probability of n correct bits, given by:

$$P_c = (1 - \frac{1}{2(1+\Gamma)})^n \quad (2)$$

We assume no error correction and thus one bit error results in a packet error. The expected number of packet retransmissions on one hop is then for n and Γ given by $1/P_c$.

With each newly deployed node the actual channel quality in terms of SNR of that node to all it's neighbors is measured and the transmission costs according to $1/P_c$ from Equation 2 are calculated. These values are then used to find the shortest path from a potential next deployment location to the data sink. This however requires an estimate of the SNR between the deployment position to be tested and its two neighbors in the triangle mesh. Since we need these values before a node is deployed, we estimate the SNR using a log-distance path-loss model and subsequently estimate the expected number of retransmissions based on Equation 2.

IV. Real-world prototype, experiences and challenges

So far we have implemented a preliminary prototype system consisting of a UAV for sensor node deployment and a set of lightweight sensor nodes which can be deployed from the air[2]. The system is planned to be used for prototype tests of sensor deployment algorithms. In this short paper we present

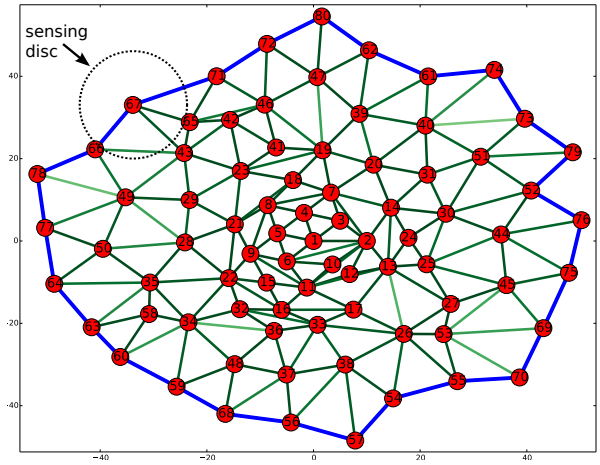


Fig. 1: Sample deployment graph for a simulation with 80 nodes. The border is shown as thick blue line.

measurements of the deployment algorithm briefly sketched in the previous section. The results shown are preliminary and of no statistical relevance. We present the results to sketch the current version of our system implementation, show the principal applicability of the system as a proof of concept, and to discuss aspects of our prototype system which still need improvement to be finally used as real system for autonomous sensor deployments.

We use a MikroKopter octocopter equipped with Global Positioning System (GPS), an additional microcontroller, a radio for interaction with the sensor network and a couple of wireless sensor nodes inside a magazine used to deploy the sensor network. The system autonomously[3] flies to certain GPS positions according to the deployment algorithm and drops a node at these positions. Whenever the magazine is empty, the copter returns to its starting position and has to be manually refilled. It then continues the deployment process where it previously stopped.

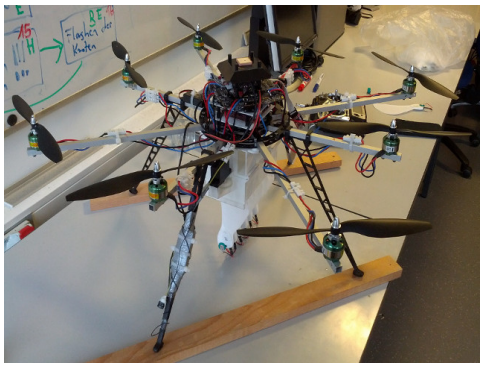
In an interdisciplinary research project together with the System and Circuit Technology group of the Electrical Engineering department and the Direct Manufacturing Research Center of the Mechanical Engineering department at the University of Paderborn, the required hardware parts were developed and built by researchers and students. The System and Circuit Technology group built tiny sensor nodes with IEEE 802.15.4 transceiver, see Figure 2d. The Direct Manufacturing Research Center group built winged bionic seeds where the sensor nodes with batteries could be sticked into (see Figure 2c). The purpose of the seeds is to soften the impact on hitting the ground and to have a protective casing. Furthermore, they built a deployment mechanism which is the magazine we use for dropping nodes from the flying drone (see Figure 2b).

The architecture of our experiments is shown in Figure 3. We have 48 sensor nodes which are flashed with a software that allows to read out channel qualities wireless. Our drone is an octocopter (MikroKopter platform) with a maximal payload

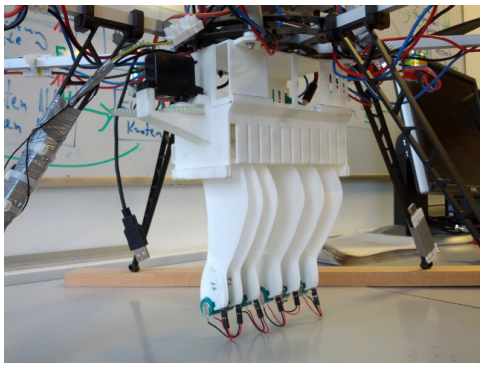
[1] assuming differential binary phase shift keying (BPSK) on slow flat-fading channels (Rayleigh distributed), which is used for example in sub-1GHZ IEEE 802.15.4 transceivers [6, p. 341]

[2] An advertising film can be found at http://URL

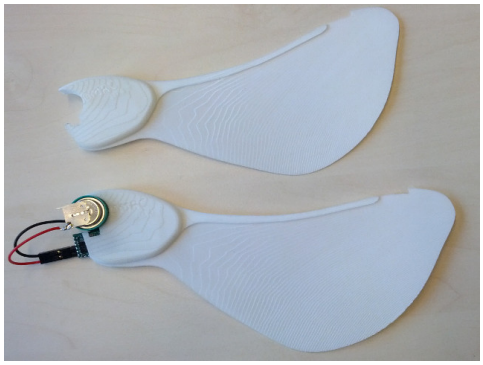
[3] Due to legal constraints, the mentioned autonomous flights still require a pilot to hold a remote control in his hand. In case of an algorithm failure, the pilot can immediately take over and safely land the octocopter by hand.



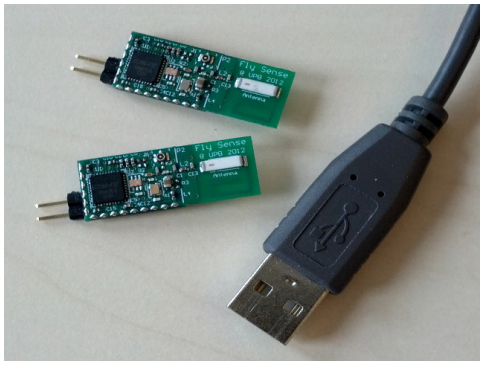
(a) Copter



(b) Filled deployment mechanism



(c) Bionic seeds



(d) Sensor nodes

Fig. 2: Hardware

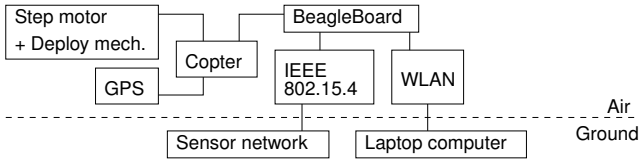


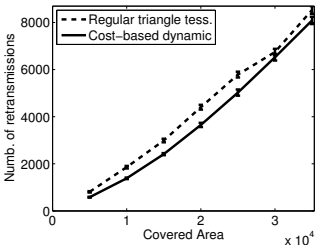
Fig. 3: Diagram of components used in experiments

of 2.5 kg. Attached to it is the deployment mechanism, a BeagleBoard running Linux and a GPS unit. The Linux board is connected to the octocopter via an interface which allows it to instruct the copter to fly to certain GPS positions and to drop sensor nodes. Further, it is connected to one sensor node via UART interface which allows it to send and receive data to and from other sensor nodes. The deployment algorithm is implemented on the Linux board, such that it can instruct the copter to fly to the deployment position, read out the current position of the copter, trigger the deployment of one seed and query channel quality feedback from already deployed nodes. We have small rechargeable batteries for each sensor node which powers them for about 2 hours if fully charged. We have five powerful rechargeable batteries for the octocopter where each gives it approximately 15 minutes of flight time, if fully charged. The testbed site we used was a soccer field at our university, which gave us an relatively large flat area without obstacles.

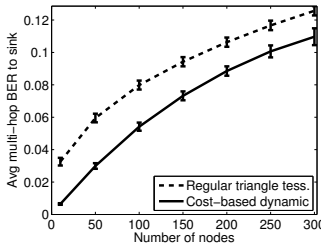
Due to the nodes communication range of approx 30 meters and due to the limited space on the soccer field, our experiments were limited to 15 nodes. Though the hardware allows to reduce communication range our current deployments require a distance of at least 15 m between the nodes to account for deviations in deployment positions. There are two major causes for such deviation. First, the impreciseness of GPS in combination with limited maneuverability of the copter. Due to that we had to set a tolerance range of 6 m around the target deployment position and drop the node when the copter got in that range. Second, the sensor nodes with the winged bionic seeds started rotating and floating slowly to the ground, as intended. However, affected by wind that further increased the variability of the deployment position.

A problem in producing statistical relevant data sets is that the experiments are very time consuming. Battery capacity is still very limited. In our case we have three chargers and with each charger we can charge either one copter battery or seven sensor node batteries at a time. The charging takes about 1 hour for copter batteries and 2 hours for sensor node batteries. In each flight, we can deploy five sensor nodes. Then the copter has to land and to be refilled. With one copter battery, at most 15 nodes can be deployed, at the moment. This number even reduces in windy conditions. The copter needs more power to stabilize. Typically, we can only deploy around 10 nodes then. One experiment with 15 nodes takes about $20 - 30$ minutes, including the time to change copter battery, refill the deployment mechanism in between and collecting all nodes

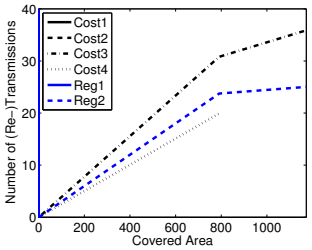
after finishing the experiment. So doing at most 5 experiments with 15 nodes is already half a day of work. Including all additional preparation this significantly limits the number of experiments one can do at one day.



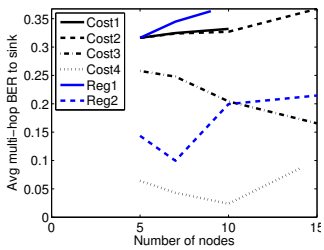
(a) Sum of number of (re)transmissions to center-based sink (Simulation)



(b) Avg. Multi-hop BER to sink for given number of nodes (Simulation)



(c) (Re)transmissions over covered area (outdoor experiment)



(d) Average multi-hop BER to sink (outdoor experiment)

Fig. 4: Results of simulations (4a and 4b) and six outdoor experiment runs ($4\times$ cost-based deployment, $2\times$ regular triangle tesselation in 4c and 4d).

In the end we had six successful experiment runs, four with our cost-based algorithm and two with the regular triangle tesselation. The measurements do not allow to compare both approaches to each other, but they give a range in which the metrics will be. The results for multi-hop BER to sink over number of nodes and (re-)transmissions over area are shown in Figure 4. We measured after deployment of 5, 7, 10 and 15 nodes, unless the experiment was aborted before. Each curve shows one experiment. We see that multi-hop BER ranges between 5 and 35%. In Figure 4c we see that the network was partitioned in three experiments, resulting in an infinite sum of transmission costs. In the other three experiment runs, the number of transmissions were in the range between 20 and 35 retransmissions.

For comparison and to show that our algorithm works well in principle, we show simulation results in Figure 4a and 4b. The simulation is implemented such that the deployment algorithm has perfect knowledge about the deterministic behaviour of the channel, i.e. the path loss exponent and the loss at reference distance. The non-deterministic log-normal shadowing is not known until nodes are deployed. The simulation is parametrized with path loss exponent $\alpha = 4.2$, path loss at reference distance 1 being 39.15 dB, a variance of 8.0 for the log normal distribution, the transmit power being 1 dBm, the noise floor being -100 dBm and the packet length being 64 bits. In Figure 4a we see that our cost-based deployment reduces the number of required transmissions by 5 to 28%. Further, we can see a reduction in multi-hop BER for our cost-based deployment in Figure 4b. Thus, simulations show that our cost-based algorithm is beneficial.

V. Conclusion

We introduce a novel approach for the deployment of sensor networks which achieves full coverage of an area and aims in providing cheap multi-hop transmissions from all nodes to sink in terms of number of transmissions on erroneous channels. We make use of the concept of iterative deployment with channel quality feedback, which we introduced in our previous work[1]. Our major new contribution in this work is the cost-based approach, where possible deployment positions are assessed according to the ratio of transmission costs to coverage gain. We evaluated our algorithm in outdoor experiments and in a graph-based simulation and compared it to the regular triangle tesselation which is optimal in terms of coverage.

We built a hardware prototype for outdoor real-world testing of deployment algorithms. It consists of a flying copter drone equipped with sensor nodes which it drops according to the deployment algorithm. Using this prototype, we did the proof of concept that our algorithm works in the real world. Due to limited space for deployment in combination with limited positioning accuracy of our practical deployment, we did not get statistically meaningful data, but we provide samples of a real-world measurement.

The simulation results show that our cost-based deployment algorithm reduces the number of (re-)transmissions for collecting sensing values from each node to a center-based sink by 5 to 28% compared to the regular triangle tesselation. Further, our algorithm reduces the average multi-hop BER considerably. Thus, our cost-based approach is beneficial for deploying sensor networks.

References

[1] R. Funke and H. Frey, "Iterative Sensor Node Deployment with Channel Quality Feedback," in *2013 IEEE International Conference on Distributed Computing in Sensor Systems*. IEEE, May 2013, pp. 402–408. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6569463

[2] S. Slijepcevic and M. Potkonjak, "Power efficient organization of wireless sensor networks," in *ICC 2001. IEEE International Conference on Communications.*, vol. 2. IEEE, 2011, pp. 472–476.

[3] R. Kershner, "The Number of Circles Covering a Set," *American Journal of Mathematics*, vol. 61, no. 3, pp. pp. 665–671, 1939. [Online]. Available: http://www.jstor.org/stable/2371320

[4] K. Kar and S. Banerjee, "Node Placement for Connected Coverage in Sensor Networks," in *WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, Mar. 2003.

[5] F. Al-Turjman, H. S. Hassanein, and M. A. Ibnkahla, "Connectivity Optimization for Wireless Sensor Networks Applied to Forest Monitoring," in *2009 IEEE International Conference on Communications*. IEEE, Jun. 2009, pp. 1–6.

[6] T. S. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed. Prentice Hall PTR, 2002.

We acknowledge all persons involved in the prototype implementation of the sensor deployment system. This includes in particular our collaborators from the Electrical Engineering and Mechanical Engineering departments at the University of Paderborn and all students participating in the semester projects Flynet I, II and III.

Most notably, we want to acknowledge Ivan Stojmenovic who contributed to this work as part of a Humboldt Research Award.

Verteilte Sensor-Signalverarbeitung für die Impedanzspektroskopie von vielzelligen Batterien

Nico Sassano, Valentin Roscher, Karl-Ragmar Riemschneider
Hochschule für Angewandte Wissenschaften Hamburg
nico.sassano@haw-hamburg.de valentin.roscher@haw-hamburg.de karl-ragmar.riemschneider@haw-hamburg.de

***Kurzfassung* —Große Batterien mit vielen Zellen werden in Fahrzeugen als Starter-, Puffer- und als Antriebsbatterien verwendet. Um diese Zellen einzeln zu überwachen, wurden bereits drahtlose Sensorsysteme im Detail vorgestellt [1]–[6]. Dieser Artikel soll sich auf die dezentrale Signalverarbeitung in diesem verteilten System fokussieren. Insbesondere soll am Beispiel der Berechnung der Impedanzspektroskopie für Batteriezustands- und Alterungsanalyse dargestellt werden, dass ein Teil der Signalverarbeitungsaufgabe vorteilhaft in jedem Zellensensor zeitlich parallel erfolgen kann. Weil jede Zelle über einen eigenen Controller im Sensor verfügt, ist die Bearbeitungszeit weitgehend unabhängig von der Zellenzahl der Batterie. Da durch die verteilte Signalverarbeitung bereits verdichtete Ergebniswerte vom Sensor anstelle von Rohdatenblöcken übertragen werden, wird der Funkkanal erheblich entlastet. Es werden praktische Ergebnisse einer Implementierung mit eigener Hardware und proprietärem Protokoll ebenso wie eine Funktionsdemonstration vorgestellt. Die Methode der Impedanzspektroskopie wird heute in aufwendigen Laboruntersuchungen für Batterien verwendet. Die vorgestellten Lösungen sollen ein Beitrag sein, dieses aussagekräftige Verfahren auch in der täglichen Fahrzeuganwendung zu ermöglichen.**

I. Einführung

Bisher war die robuste Blei-Säure-Technologie im Fahrzeug de facto Standard, sowohl als Starterbatterie für den Verbrennungsmotor als auch als Antriebsbatterie für Gabelstapler. Die Entwicklung der Lithium-Technologie in den 1990er Jahren erlaubte eine drei- bis fünffach höhere Energiedichte, allerdings zum Preis eines viel aufwendigeren Batteriemanagements. Zunächst waren kleine, meist einzellige Batterien für mobile Geräte im Fokus. Die Massenproduktion ermöglichte stark sinkende Batteriekosten für diese Technologie, so dass der Elektroantrieb von Straßenfahrzeugen aktuell eine Renaissance erlebt. In aktuellen Elektrofahrzeugen werden Hochvoltbatterien mit typisch 50-200 Zellen verwendet. Zunehmend werden Lithiumbatterien für alle Einsatzgebiete in Fahrzeugen geplant, auch für Starter-/Pufferbatterien und sogar für Gabelstapler. Die Leistungsfähigkeit, Sicherheit und Lebensdauer von Batterien wird essentiell vom Batteriemanagement bestimmt. Dabei ist es zwingend notwendig, jede einzelne Zelle zu überwachen.

Als Lösungskonzept wurden von der Arbeitsgruppe an der HAW Hamburg drahtlos kommunizierende Zellensensoren vorgestellt [1]–[6]. Auf diese Weise können viele Nachteile der verkabelten Sensorik vermieden werden. Eine Reihe von Arbeiten anderer Gruppen verfolgt ähnliche Ansätze [7]–[11]. Das Verbundprojekt IntLiIon setzt Zellensensoren mit Powerline Communication ein [12]–[14].

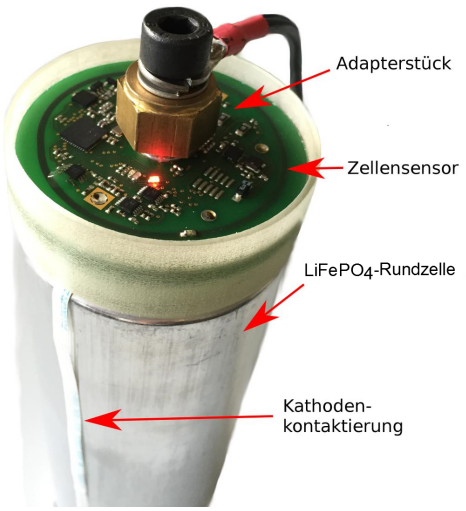


Abbildung 1. Drahtloser Zellensensor montiert auf eine große Lithium-Batteriezelle (45 Ah $LiFePO_4$-Zelle, ECC Batteries). In diesem Zellensensor ist bereits ein Teil der Signalverarbeitung zur Zellzustandsanalyse implementiert.

II. Modulare Funktionen des Zellensensors

Die Zellensensoren sind in mehreren Versionen entstanden, in denen schrittweise erweiterte Funktionen in Software und Module in Hardware umgesetzt wurden:

- Mehrere Sensorcontroller aus der MSP430-Familie
- Transmitter-IC ohne Quarz und entsprechendes Übertragungsverfahren
- Transceiver-IC mit Kommunikationsprotokollen
- Konverter für die Versorgungsspannung (Step Up/Down), dazu passend zeitlich gesteuerte Störunterdrückung sowie Referenzspannungsquellen
- Wake-Up-Funktion (passiver Empfänger)
- Temperaturerfassung mit Warnfunktion
- Schaltbarer Strompfad für die Ladungsbalancierung
- Hochdynamische Ereigniserfassung mit Zwischenspeicherung und Zeitstempeln (z.B. Motorstart)
- Mikrosekundengenaue funksynchronisierte Messung durch zentrale Triggerpulse an alle Sensoren
- Impedanzspektroskopie

Für das Batteriesteuergerät wurden zunächst verbrauchssparsame Controller des Typs MSP430F169 eingesetzt. Im Projektverlauf wurde sich für einen Controller ARM Cortex-M4 (Texas Instruments TM4C1294) mit 120 MHz Takt entschieden, um erweiterte Signalverarbeitung, komplexe Batteriemodelle und verschiedene Schnittstellen verfügbar zu machen. Es ist ein Kommunikationskonzept mit Kommandos, Nachrichten und Broadcastfunktionen

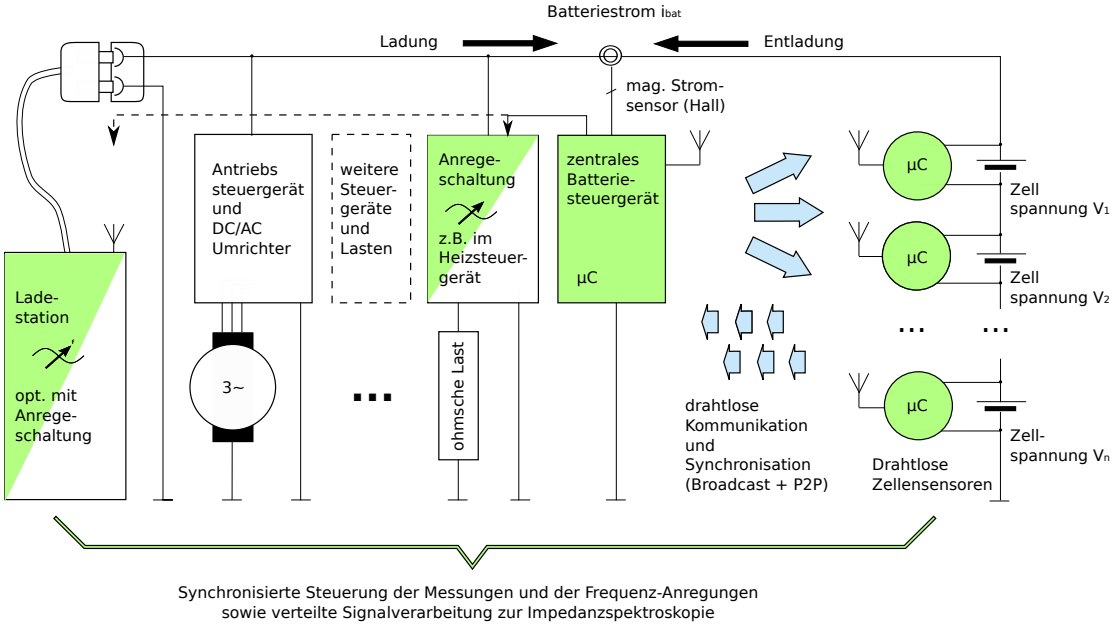


Abbildung 2. Schematischer Überblick einer Anordnung für die Impedanzspektroskopie im Elektroauto

entwickelt worden. Dabei wurden verschiedene Verfahren wie adressierte synchronisierte Kommunikation, sternförmige Broadcast- und Multicastfunktionen sowie ein verlustbehaftetes Aloha-Konzept erprobt. Wegen des Aufwandes und der Zeitanforderungen wurden die Sende- und Empfangs-ICs mit proprietären Übertragungsprotokollen gesteuert. Als Modulationsart wurde OOK mit 433 MHz Träger verwendet, die Bandbreite entspricht der Datenrate und wird auf die jeweils genutzte Empfängerbandbreite optimiert.

Mit in der Arbeitsgruppe praktisch umgesetzten Modulen können Sensoren verschiedener Klassen zusammengestellt werden, die unterschiedliche Anforderungen in der Funktion und an den Kostenrahmen abdecken [2], [4].

III. Anwendungsbeispiel für dezentrale Signalverarbeitung

Nachdem der Aufbau der Sensoren und die Kommunikation bereits dargestellt worden sind, soll ein Beispiel der Funktionsverteilung diskutiert werden. Hierzu wird die Signalverarbeitung eines Verfahrens zur Bestimmung des Lade- und insbesondere des Alterungszustandes ausgewählt. Dieses Verfahren ist als elektrochemische Impedanzspektroskopie (EIS) in der Batterieforschung bekannt. Normalerweise werden hierfür spezielle Labor-Großgeräte genutzt, die mehrere zehntausend Euro kosten.

Die Motivation, dieses Verfahren auf das Zellensensorsystem zu übertragen, war es, diese Mittel der 'Batterie-Befundung' in jedem Elektrofahrzeug verfügbar zu machen. Damit kann im täglichen Betrieb der Zellzustand mit guter Aussagekraft ermittelt werden.

A. Elektrochemische Impedanzspektroskopie

Mit Hilfe der elektrochemischen Impedanzspektroskopie (EIS) kann der Zellzustand unter verschiedenen Gesichtspunkten beurteilt werden. Dabei fließen Kenngrößen der Elektrodenoberflächen, des Elektrolyten, des aktiv speichernden Materials, der ohmschen Widerstände der Stromsammler/Leitfolien und der Anschlüsse sowie der Temperatur ein, die zu Aussagen über den Lade- und Alterungszustand verdichtet werden.

Die EIS basiert auf synchronen Spannungs- und Strommessungen der Zelle, wenn diese mit einer bestimmen Anregung aus Gleich- und Wechelstromanteilen beaufschlagt wird. Präzise Messungen der Reaktion der Zellspannung auf einen extern angelegten Batteriestrom liefern den komplexen Zellinnenwiderstand, bestehend aus ohmschen und kapazitiven Widerstandsanteilen. Trägt man den komplexen Zelleninnenwiderstand mit Real- und Imaginärteil über viele Anregefrequenzen auf, entsteht das Impedanzspektrum. Es kann als Nyquist-Plot (Abb. 4) dargestellt werden.

Entsprechend der Kinetik der Batterieprozesse können die Effekte bestimmten Anregungsfrequenzen zugeordnet werden. Diese Zuordnung und die chemisch/physikalische Interpretation ist ausführlich in der Fachliteratur dargestellt [15]–[18] (Abb. 4). Das Impedanzspektrum kann herangezogen werden, um auf Oberflächeneffekte, Alterungserscheinungen an den Elektroden, den Ladezustand u.a. schließen zu können.

B. AC-Anregung im Lade- und Entladebetrieb

Die Wechselstromanregung für die Impedanzspektroskopie kann auf verschiedene Arten erzeugt werden. Im einfachsten Fall wird nur bei wenigen Frequenzen gemessen, es kann aber auch ein breites Frequenzspektrum durchfahren werden. Oft wird aber nur eine begrenzte Anzahl an Frequenzwerte ausgewählt. Zudem kann die Anregung auf einen positiven oder negativen Offset gelegt werden, was einer Ladung bzw. Entladung der Zelle entspricht. Eingesetzt werden Sinusanregungen mit Frequenzen ab 10 mHz, typischerweise ab 100 mHz. Als obere Frequenz sind maximal 30 kHz bekannt, typischerweise beschränkt man sich auf wenige kHz. Bei höheren Frequenzen treten

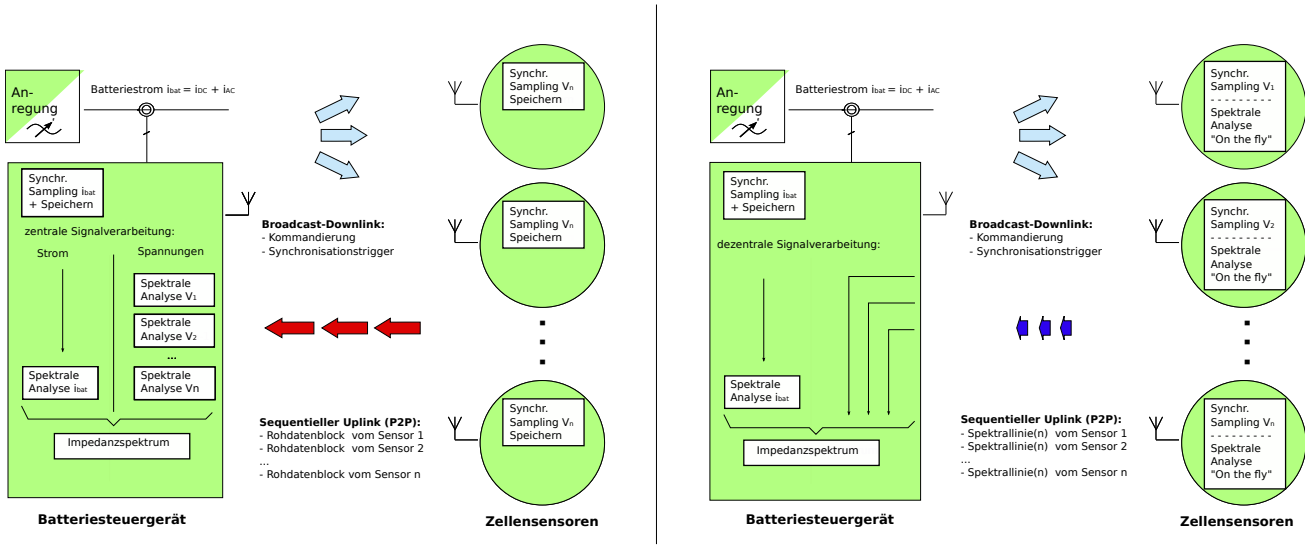


Abbildung 3. **Links:** Zellensensoren ohne Signalverarbeitung speichern die Spannungswerte und übertragen große Rohdatenblöcke. Die Blockgröße ist durch den Speicher des Sensorcontrollers begrenzt. Die gesamte Signalverarbeitung erfolgt auf dem Batteriesteuergerät, der Zeitaufwand steigt mit der Zellenzahl fast proportional. **Rechts**: Zellensensoren mit Signalverarbeitung verarbeiten die Spannungswerte fortlaufend (on-the-fly) und wenige Spektraldaten werden übertragen. Die Netto-Übertragungszeit ist mehr als zwei Größenordnungen kleiner als die der Rohdaten. Die Blockgröße wird durch den Speicher des Sensorcontrollers nicht mehr begrenzt.

vermehrt induktive Anteile auf, welche der Schaltung aber nicht der Batteriezelle zuzuordnen sind.

Es gibt auch Varianten, in denen die Anregung im Frequenzbereich pseudozufällig gespreizt wird (Multifrequenzanregung [17]). Diese erfordern typischerweise erhöhten Rechenaufwand und sind bisher nicht umgesetzt worden.

Da die Zellen sehr niedrige Innenwiderstände aufweisen, müssen erhebliche Anregeströme eingesetzt werden, um günstig messbare Spannungseffekte an den Zellen hervorzurufen. Der Wechselstromanteil bei EIS-Messungen ist so zu wählen, dass die Spannungsauslenkung im linearen Bereich verbleibt (bis zu 10 mV Auslenkung [16]), dies kann Strom-Amplituden von mehreren Ampere für große Batteriezellen erfordern, da ihre Innenwiderstände bei nur 0,1 bis 20 mΩ liegen. Im Versuchsaufbau wurden als Quellen für die AC-Anregung ein Leistungsoperationsverstärker (Kepco BOP36-5M) sowie alternativ der Ausgang eines Labor-EIS-Meters (Fuelcon TrueEIS) verwendet. Mit beiden Quellen waren mehrere Ampere AC- und DC-Anregungsanteile frei wählbar möglich. Im Fahrzeug und/oder an der Ladestation muss eine vergleichbare Anregeschaltung später realisiert werden.

C. Funksynchronisierte Messung

An zentraler Stelle im Batteriesteuergerät wird der Anregestrom erfasst. In jedem Zellensensor wird die Zellenspannung erfasst, dabei werden zwei Messungen durchgeführt. Zum einen wird die Spannung mit grober Auflösung (12 bit entspr. U_{lsb} = 1.25 mV) im vollen Messbereich von 0 bis 5 V abgetastet. Zum anderen wird mit einer feinen Auflösung gemessen, jedoch mit reduziertem Messbereich um einen vom Controller vorgebbaren DC-Offset-Spannungswert. Zu diesem Zweck wird bereits in der analogen Vorverarbeitung die DC-Offset-Spannung abgezogen und um den Faktor 10 bis 100 verstärkt. Damit sind auch mit dem 12 bit ADC des Controllers Spannungen bis $U_{lsb} = 10\,\mu$V auflösbar. Alternativ erprobt wurden auch Schaltungen mit einem hochauflösenden ADC (bis 24 bit) als externer IC. Diese litten jedoch unter der geringen Abtastrate und schwierig umsetzbaren Anforderungen an die Schaltungstechnik.

Für die Messung entscheidend ist die präzise zeitliche Übereinstimmung aller Abtastwerte im gesamten Sensorsystem. Der Zeitpunkt der zueinander gehörenden Messungen sollte nur maximal eine Mikrosekunde voneinander abweichen. Hierzu ist eine besondere Betriebsart, die funksynchronisierte Messung, in Hard- und Software realisiert worden. Die Spannungsabtastung auf allen einzelnen Zellensensoren wird durch den Empfang eines sehr kurz eingeschalteten Triggerpulses des 433-MHz-Trägersignals ausgelöst. Dazu wird der Träger ohne Daten-Protokoll unmittelbar demoduliert. Der Triggerpuls wird vom Steuergerät synchron zur Abtastung der Strommessung gesendet. Die Details dazu sind bereits publiziert worden [3]. Es wurde dort zunächst dargestellt, dass alle Rohdaten als Block vom Steuergerät abgerufen werden. Diese Lösung ist in der Anwendung der EIS zugunsten einer Signalverarbeitung auf dem Sensor abgelöst worden. Abb. 3 stellt dies gegenüber und verdeutlicht den Vorteil der verteilten Signalverarbeitung.

D. Verteilte Signalverarbeitung im Frequenzbereich

Das synchronisierte Abtasten der Zellspannungen und des Anregestroms erfolgt blockweise getrennt für jede gewählte Anregefrequenz. Jeder Block wird vom Zeitbereich in den Frequenzbereich transformiert. Generell stehen dafür als Verfahren die diskrete Fouriertransformation (DFT), die schnelle Fouriertransformation (FFT) oder der Goertzel-Algorithmus (GA) [19] zur Verfügung. Sie liefern komplexwertige Spektren bzw. Spektrallinien. Das erlaubt die genaue Berechnung der Phasenverschiebung zwischen

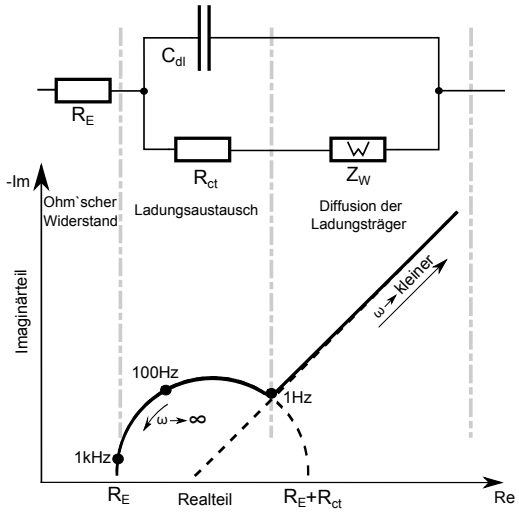


Abbildung 4. Ersatzschaltbild der elektrochemischen Impedanzspektroskopie und schematischer Nyquist-Plot der komplexem Impedanz einer Batteriezelle. Die Effekte werden entsprechend ihrer Geschwindigkeit drei verschiedenen Bereichen zugeordnet. Z_W: Warburg-Impedanz, C_{dl}: Doppelschicht-Kapazität R_{ct}: Ladungstransferwiderstand (Elektronen) R_E: Ohmsche Leitungswiderstände.

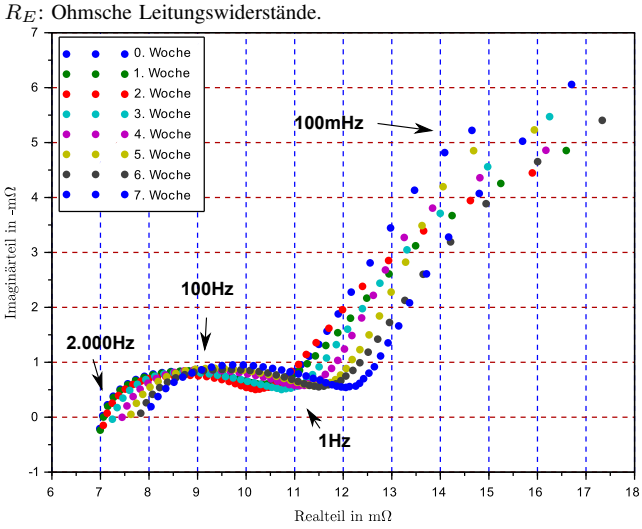


Abbildung 5. Fortschreitende Alterung $LiFePO_4$-Zelle (A123 ANR26650M1-B). Diese Zelle wurde bei einer Temperatur von 55°C über 7 Wochen hinweg künstlich gealtert. In dieser Zeit wurden 700 volle Zyklen geladen und entladen. Einmal pro Woche wurde das Impedanzspektrum vermessen.

Strom und Spannung, und damit des Real- und Imaginärteils der Impedanz. Das Impedanzspektrum ergibt sich dann für jeden Punkt durch den Vergleich der Phase des Stroms mit den verschiedenen Phasen der Zellspannungen im Frequenzbereich. Der Gleichspannungsanteil jeder Zelle ist für EIS nicht relevant, allerdings muss der Arbeitspunkt durch den Gleichstromanteil (bzw. Gleichspannungsanteil) während der Messung festgehalten werden.

Weil die Anregung im quasilinearen Bereich erfolgt, wurde entschieden, Harmonische (Oberwellen) zu vernachlässigen. Außerdem wurde für jeden Messabschnitt nur eine Sinus-Anregefrequenz angelegt und damit nur eine Spektrallinie bestimmt. Daher wurde der Goertzel-Algorithmus ausgewählt und implementiert [19]. Dieser ist in diesem Spezialfall am wenigsten aufwendig und ist auch von den Sensorcontrollern zu leisten.

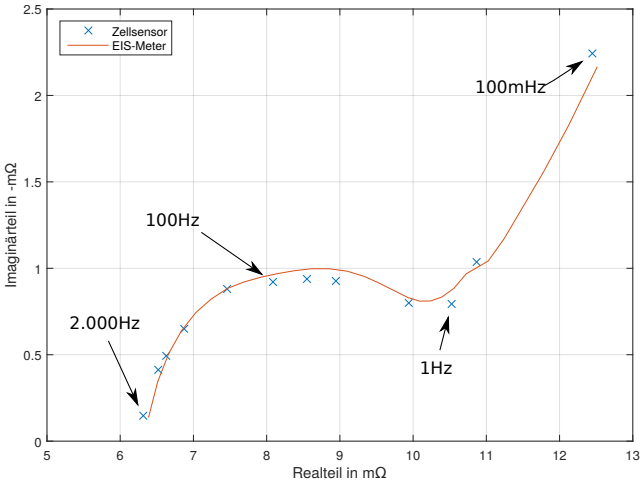


Abbildung 6. Vergleich einer Impedanzmessung zwischen einem Labormessgerät (Typ Fuelcon TrueEIS) und einer Messung mittels eines Zellensensors an einer A123 ANR26650M1-B $LiFePO_4$-Zelle im Bereich von 100 mHz bis 2 kHz.

IV. Anwendung und Erprobung

Um die Verwendung für die Alterungszustandsbestimmung zu demonstrieren, wurden 700 Zyklen an Zellen mit erhöhter Umgebungstemperatur angelegt. Durch diese künstliche Alterung sank die Kapazität der Zellen innerhalb von 7 Wochen auf 20% der Nennkapazität. Der Alterungsprozess war im Impedanzspektrum deutlich zu verfolgen, insbesondere bei einer Anregefrequenz um 1 Hz. Hier ist eine Verschiebung des Minimums gut zu sehen (Abb. 5). Um die korrekte Funktion der EIS auf dem Sensorsystem zu demonstrieren, wurden Messungen mit den Sensoren und dem kommerziellen EIS-Meter zeitgleich, d.h. bei jeweils gleichen Zellzuständen, vorgenommen. Dabei wurde eine gute Übereinstimmung erreicht (Abb. 6). Die Demonstration an Lithium-Starterbatterien ist geplant. Hierbei soll das Messsystem im Rahmen einer Fahrzeugerprobung insbesondere bei Startvorgängen und tiefen Temperaturen getestet werden.

V. Formale Beschreibung

A. Impedanzwerte und Impedanzspektrum

Die Impedanz ist der komplexe Widerstand aus einem realen ohmschen Anteil (Re) und einem imaginären Anteil (Im), der je nach Vorzeichen kapazitiv oder induktiv ist. Er kann als komplexer Quotient einer Wechselspannung und eines Wechselstroms definiert werden und ist allgemein auch von der Frequenz abhängig.

$$\underline{Z}(t,f) = Re(t,f) + \mathrm{Im}(t,f) \;= \frac{\underline{u}(t,f)}{\underline{i}(t,f)}$$

Weil das Batterieverhalten für die Erfassung eines Datenblocks der Länge n an den äquidistanten Abtastzeitpunkten $t_0, ..., t_n$ als stationär angesehen wird, kann vereinfacht werden:

$$\underline{Z}(f) \approx \underline{Z}(t,f)$$

Ebenso wird vereinfachend der Parameter t bei $\underline{u}(t,f)$ und $\underline{i}(t,f)$ weggelassen. Das Impedanzspektrum wird als eine Reihe von komplexen Widerstandswerten für eine diskrete

Anzahl m ausgewählter (nicht unbedingt äquidistanten) Anregungsfrequenzen bestimmt. Die Blocklänge n der Spannungs- und Stromabtastungen bleibt typischerweise konstant, entsprechend wird der Abstand der Abtastzeitpunkte $t_i - t_{i+1}$ der jeweiligen Frequenz f_x angepasst, sodass mehrere Perioden erfasst werden:

$$\underline{Z}(f) = Re(f_x) + \mathrm{I}m(f_x) \; = \frac{\underline{u}(f_x)}{\underline{i}(f_x)}$$

$$\text{mit } f_x = f_1, .., f_m$$

Um den ganzen Block der Spannungs- und Stromabtastungen zusammenfassend zu betrachten, wird dieser vom Zeitbereich in den Frequenzbereich überführt. Dabei ist insbesondere die jeweilige Anregungsfrequenz f_x interessant (Linearitätsannahme, Kleinsignalanregung). Wenn nur eine Frequenz (d.h. die Antwort auf die Anregungsfrequenz) benötigt wird, ist der Goertzel-Algorithmus für die Transformation dieser Spektrallinie i.d.R. vorteilhaft [20]. Das Symbol für die Transformation ist $\circ\!\!-\!\!\bullet$.

$$\underline{u}(f_x) \quad \circ\!\!-\!\!\bullet \quad \mathcal{F}(\underline{u}(f_x)) \tag{1}$$
$$\underline{i}(f_x) \quad \circ\!\!-\!\!\bullet \quad \mathcal{F}(\underline{i}(f_x)) \tag{2}$$

Aus diesen nur noch zwei komplexen Werten wird nun ein komplexer Impedanzwert $\underline{Z}(t, f)$ mit den Real- und Imaginärteilen errechnet.

$$\underline{Z}(f_x) = \frac{\mathcal{F}(\underline{u}(f_x))}{\mathcal{F}(\underline{i}(f_x))} \tag{3}$$

B. Impedanzspektren von vielen Zellen

Für eine p-zellige Batterie wird für jede Zelle mit dem Index $k = 0, ..., p$ ein Impedanzspektrum ermittelt. Dafür wird die Lokalisation der Operationen und die notwendige Übertragung mit betrachtet. Dazu werden zwei verschiedene Transportoperatoren eingeführt. Sie verändern die Werte nicht, verlagern diese aber durch eine räumliche Übertragung. Der Transportoperator $A \Longrightarrow B$ bezeichnet die Funkübertragung von n Spannungsabtastwerten eines Blocks vom Ort A zum Ort B. Der Transportoperator $A \Rightarrow B$ bezeichnet die Funkübertragung von lediglich zwei Werten einer komplexen Spektrallinie. Die Lokalisation kann mit diesen Transport- und den mathematischen Operatoren symbolisch dargestellt werden.

Die Gleichung 4 beschreibt die Variante mit zentraler Signalverarbeitung einer Batteriezelle. Für eine p-zellige Batterie muss dieser Vorgang p-mal wiederholt werden. Weiterhin muss f_x alle Anregefrequenzen durchlaufen. Es gilt $\forall x = 1, ..., m$.

$$\begin{array}{ccc} u_0(f_x) & \Longrightarrow & \underline{u}_0(f_x) \circ\!\!-\!\!\bullet \mathcal{F}(\underline{u}_0(f_x)) \\ \vdots & \vdots & \vdots \\ \underbrace{\underline{u}_p(f_x)}_{\text{in den Zellensensoren}} & \Longrightarrow & \underbrace{\underline{u}_p(f_x) \circ\!\!-\!\!\bullet \mathcal{F}(\underline{u}_p(f_x))}_{\text{seriell im Batteriesteuergerät}} \end{array} \tag{4}$$

$n \cdot p \cdot m$ Werte

Die verteilte Variante der Signalverarbeitung kann wie in Gleichung 5 dargestellt werden.

$$\begin{array}{ccc} \underline{u}_0(f_x) \circ\!\!-\!\!\bullet \mathcal{F}\underline{u}_0(f_x)) & \Rightarrow & \mathcal{F}(\underline{u}_0(f_x)) \\ \vdots & \vdots & \vdots \\ \underbrace{\underline{u}_p(f_x) \circ\!\!-\!\!\bullet \mathcal{F}(\underline{u}_p(f_x))}_{\text{parallel in den Zellensensoren}} & \Rightarrow & \underbrace{\mathcal{F}(\underline{u}_p(f_x))}_{\text{im Batteriesteuergerät}} \end{array} \tag{5}$$

$2 \cdot p \cdot m$ Werte

Für beide Fälle werden im zentralen Batteriesteuergerät die Stromwerte erfasst bzw. deren Fouriertransformierte (komplexe Spektrallinienwerte) berechnet. Auch im Fall der verteilten Signalverarbeitung verbleibt die Berechnung nach Gleichung 2 im Steuergerät. Da der Strom für alle Zellen identisch ist, gibt es keine Abhängigkeit von der Anzahl der Zellen in der Batterie. Diese Berechnung wird also nur m-fach durchgeführt.

$$\underbrace{\underline{i}(f_x) \circ\!\!-\!\!\bullet \mathcal{F}(\underline{i}(f_x))}_{\text{m-fach im Batteriesteuergerät}} \tag{6}$$

Die Berechnung der Impedanz nach Gleichung 3 erfolgt für alle Anregefrequenzen f_x mit $\forall x = 1, ..., m$ zentral im Batteriesteuergerät.

$$\underbrace{\underline{Z}(f_x) = \frac{\mathcal{F}(\underline{u}(f_x))}{\mathcal{F}(\underline{i}(f_x))}}_{\text{m-fach im Batteriesteuergerät}} \tag{7}$$

Die Berechnungen nach Gl. 6 und 7 haben in beiden Varianten den gleichen Zeitaufwand. Ebenso ist der Aufnahmezeitbedarf gleich[1]. Das gilt zumindest bei geringen Abtastraten.

Der entscheidende Unterschied ist also in Gleichung 4 und 5 zu finden. In Gleichung 5 sinkt der Bedarf der Übertragungszeit von n auf 2, wobei n typischerweise zwischen 100 und 2000 liegt. Der Zeitbedarf einer Transformation (ohne Übertragungszeit) steigt jedoch erheblich, da die Controller der Zellensensoren leistungsschwächer als der eingesetzte Controller des Batteriesteuergeräts sind. Jedoch werden alle Transformationsberechnungen parallel durchgeführt. Dies bringt erst bei einer sehr großen Anzahl von parallel rechnenden Zellensensoren (ab ca. 400) einen Vorteil, diese Anzahl wird praktisch nicht erreicht.

Der Nachteil der daher niedrigeren Transformationsgeschwindigkeit wird überkompensiert:

a) weil die Übertragungsgeschwindigkeit vergleichsweise niedrig ist;
b) weil die Aufnahmezeit (Sampling) und die Berechnungszeit (Transformation) sich bei der 'on-the-fly' Implementierung nicht vollständig addieren müssen[1].

[1]Bei niedrigen Frequenzen kann der Zeitbedarf für das Sampling einen Großteil des Gesamtaufwands beitragen, da die Periodendauer und der zeitliche Abstand zw. zwei Abtastwerten entsprechend lang ist. Daher kann die Transformation des Goertzel-Algorithmus mit dem Sampling schritthaltend implementiert werden, da ein abgetasteter Wert sofort verarbeitet wird. Diese 'on the fly'-Rechnung macht es möglich, den Aufwand der Transformation in den Lücken des unvermeidlichen Sampling-Zeitbedarfs 'zu verstecken'.

C. Beispiel

Die Nachfolgende Tabelle 1 zeigt einen Vergleich der benötigten Zeiten der beiden Varianten mit zentraler und verteilter Signalverarbeitung. Es wurden im Beispiel Impedanzwerte für 12 Frequenzwerte bestimmt. Diese liegen zw. 2 kHz und 100 mHz und sind in Abbildung 6 durch Kreuze gekennzeichnet. Für die Erfassung der Sample werden unterschiedliche Abtastraten benutzt, jedoch werden immer 2000 Messwerte pro Block erfasst. Die Summe aller Erfassungszeiten beträgt ca. 87 s. Dafür wurde die Berechnungszeit des Goertzel-Algorithmus einer Spektrallinie auf dem Zellensensor sowie auf dem Batteriesteuergerät gemessen. Die Berechnungszeit des Batteriesteuergeräts ergab sich zu 9,3 ms, der Zellensensor benötigt für die Berechnung 3,79 s. Der Sensorcontroller (Texas Instruments CC430F5137) wird dabei mit einer Geschwindigkeit von 16 MHz betrieben. Der Mikrocontroller des Batteriesteuergerätes (Texas Instruments TM4C1294) ist mit 120 MHz getaktet. Für die Übertragung von 2000 Werten wird eine Zeit von 6,98 s benötigt. Bei einer Übertragung von 2 Werten einer komplexen Zahl werden lediglich 40 ms benötigt. Für beide Varianten wird eine physikalische Übertragungsrate von 100 kBit/s gewählt. Diese kann prinzipiell nicht ausgenutzt werden, da Protokolloverhead nötig ist (Fehlerschutz, individuelle Adressierung u.a.). Dennoch kann hier noch erheblich optimiert werden, z.B. bei Blockgrößen, Wartezeiten, Broadcastabfragen und weiteren Softwareverbesserungen. Um diese Optimierungen umsetzen zu können, muss in weiteren Arbeiten festgestellt werden, welche Blockgrößen, Frequenzen, Duty Cycle u.a für die Anwendung passen.

	zentrale Signalverarbeitung	verteilte Signalverarbeitung
	4 Zellen	**4 Zellen**
Aufnahme	für 12 Freq. zus. ≈87 s [c])	für 12 Freq. zus. ≈87 s [c])
Übertragung	12 · 4 · 6,98 s ≈335 s	12 · 4 · 40 ms ≈1,9 s
Berechnung	12 · (4+1) · 9,3 ms ≈500 ms [a])	12 · 1 · 3,79 s ≈45 s [b])
Gesamtzeit	≈422 s	≈ 137 s
	100 Zellen	**100 Zellen**
Aufnahme	für 12 Freq. zus. ≈87 s [c])	für 12 Freq. zus. ≈87 s [c])
Übertragung	12 · 100 · 6,98 s ≈ 8376 s	12 · 100 · 40 ms ≈48 s
Berechnung	12 · (100+1) · 9,3 ms ≈11,2 s [a])	12 · 1 · 3,79 s ≈45 s [b])
Gesamtzeit	≈ 8474 s	≈ 180 s

[a]) +1 für die Berechnung des komplexen Stromwertes
[b]) Wegen Parallelverarbeitung wird die längste Rechenzeit gewertet
[c]) Untersch. Aufnahmezeiten für die Frequenzen von 2 kHz bis 100 mHz

Tabelle 1: Vergleich zwischen zentraler und verteilter Signalverarbeitung für Spektrum mit 12 Impedanzwerten

VI. Zusammenfassung und Ausblick

Für das in der Arbeitsgruppe entwickelte Batteriemanagementsystem wurde ein neuartiges Verfahren vorgestellt, das ohne aufwendige Laborgeräte die individuellen elektrochemischen Impedanzspektren von in Reihe geschalteten Batteriezellen bestimmen kann. Die verteilte Signalverarbeitung findet während der Messdatennahme auf den Zellensensoren statt (on-the-fly). Damit ist der Speicher des Sensorcontrollers nicht mehr limitierend. Die Übertragungskapazität des Funkkanals wird auch bei Einsatz sehr vieler Zellen geschont. Im Vordergrund der nächsten Arbeiten steht die Vorbereitung des Transfers in die Industrie. Das betrifft sowohl technische als auch anwendungsseitige Fragen und die Hochintegration als Chip.

Projektpartner und Förderung

Das Projekt 'Drahtlose Zellensensoren für Fahrzeugbatterien - BATSEN' wurde vom BMBF gefördert (FKZ 17001X10) und von den Unternehmen Volkswagen AG, Bertrandt AG, Still GmbH, ECC Batteries, Fey Electronic GmbH und Coilcraft Ltd. unterstützt. Weitere Arbeiten wurden im EU-Projekt 'E-Mobility NSR' und in der Graduiertenschule 'Key Technologies for Sustainable Energy Systems' der Universität und der HAW Hamburg durchgeführt.

Literatur

[1] Krannich T., Plaschke S., Riemschneider K.-R., Vollmer J.: Drahtlose Sensoren für Batterie-Zellen - ein Diskussionsbeitrag aus Sicht einer Anwendung; 8. GI/ITG KuVS FGSN, Hamburg-Harburg, 2009

[2] Ilgin, S., Jegenhorst, N., Riemschneider, K.-R., Schneider, M. et al.: Zellenweiser Messbetrieb, Vorverarbeitung und drahtlose Kommunikation bei Fahrzeugbatterien; 10. GI/ITG KuVS FGSN, Paderborn, 2011

[3] Riemschneider, K.-R., Roscher, V., Sassano, N. et al.: Synchronisierte Messung durch Trigger-Broadcast und weitere Funktionen für drahtlose Batteriesensorik; 13. GI/ITG KuVS FGSN, Potsdam, 2014

[4] Schneider, M., Ilgin, S., Jegenhorst, N., Kube, R., Püttjer, S., Riemschneider, K.-R., Vollmer, J.: Automotive Battery Monitoring by Wireless Cell Sensors; IEEE I2MTC, Graz, 2012

[5] Riemschneider, K.-R., Roscher, V., Sassano, N. et al.: Synchronisation using Wireless Trigger-Broadcast for Impedance Spectroscopy of Battery Cells; IEEE SAS, Zadar, 2015

[6] Riemschneider, K.-R.: Wireless Battery Management System, Pat.appl. WO2004/047215A1, US020060152190A1, EP000001573851A1

[7] Wenger, M. M., Filimon, R., Lorentz, V. R. H., Marz, M.: A robust contactless capacitive communication link for high power battery systems; 23rd IEEE ISIE, 2014

[8] Takeuchi, T., Terada, T.: Evaluation of Wireless Communication Performance in a Li-ion Battery System, J. of Auto. Contr. Eng.; 2014

[9] Terada, T., et al.: Low power and high receiving sensitivity wireless wake-up receiver ... for lithium-ion battery systems; IEEE APEC, 2014

[10] Steinhorst, S., et al.: Smart cells for embedded battery management; IEEE CPSNA, 2014

[11] Bacquet, S., Maman, M.: Radio frequency communications for smart cells in battery pack for electric vehicle; IEEE IEVC, 2014

[12] Ouannes, I., Nickel, P., Dostert, K.: Cell-wise monitoring of Lithium-ion batteries for automotive traction applications by using power line communication; IEEE ISPLC, 2014

[13] Alonso, D., et al.:Towards a Wireless Battery Management System: Evaluation of Antennas and Radio Channel Measurements Inside a Battery Emulator; IEEE VTC Fall, 2014

[14] Opalko, O., Alonso, D., Dostert, K.: Measurements on Rogowski coils as coupling elements for power line communication in traction lithium-ion batteries; IEEE ISPLC, 2015

[15] Huet, F.: A review of impedance measurements for determination of the state-of-charge or state-of-health of ... batteries; JPS 70, 1998

[16] Macdonald, J. R., Barsoukov, E.: Impedance spectroscopy: theory, experiment, and applications; John Wiley & Sons 2005

[17] Kiel, M.: Impedanzspektroskopie an Batterien unter besonderer Berücksichtigung von Batteriesensoren...; Diss. Shaker 2013

[18] Bard, A. J., Faulkner, L. R., Electrochemical Methods: Fundamentals and Applications, 2nd Edition, Wiley, New York, 2001

[19] Goertzel, G.: An algorithm for the evaluation of finite trigonometric series; American mathematical monthly, 1958

[20] Sassano, N.: Entwicklung eines Messsystems zur funksynchronisierten elektrochemischen Impedanzspektroskopie von Batterie-Zellen; Masterthesis HAW Hamburg, 2015

[21] Bergveld, H. J., Danilov, D., Regtien, P. P., Notten, P. H.: Battery Management Systems: Accurate State-of-Charge Indication for Battery-Powered Applications. Springer, 2008

PotatoNet – Outdoor WSN Testbed for Smart Farming Applications

Ulf Kulau, Sebastian Schildt, Stephan Rottmann, Björn Gernert and Lars Wolf
Institute of Operating Systems and Computer Networks
Technische Universität Braunschweig
Email:[kulau|schildt|rottmann|gernert|wolf]@ibr.cs.tu-bs.de

***Abstract*—We present PotatoNet, an outdoor testbed for Wireless Sensor Networks (WSNs). Its primary focus is robustness, reliability and flexibility. PotatoNet is designed to operate without on-site maintenance for extended periods of time. It can withstand heat, dust and rain and has already been tested running outside for several months.**

I. Motivation

Many testbeds for WSNs have been proposed with the goal of enabling users to test new methods for such networks in a more realistic environment. However, many of these testbeds have only been developed for indoor use [1], [2], [3], [4]. The difference between an outdoor and an indoor testbed is not only hardware. Of course, nodes in an outdoor testbed need to be weather proof, and withstand the environmental conditions specific to their application. However, because outdoor testbeds might be installed in remote areas, software and architecture requirements are equally important: When it is not possible to perform regular maintenance to the testbed equipment, and any hard- or software failure leads to a prolonged downtime, robustness and reliability become the main focus of hard- and software design.

Nowadays more and more WSN applications operating outdoors are developed. Smart Farming is an upcoming application area for WSNs, which are used to analyze and monitor the conditions of the soil and crops in order to optimize yield and minimize the use of fertilizer and pesticides. While PotatoNet is currently used for Smart Farming research, it has been developed to be a generic testbed that can be used for any outdoor scenarios. Smart Farming provides an especially challenging environment, as nodes not deployed in protected areas, but sit out in the open field. They are not protected from direct sunlight and the environment can get very rough [5]. Additionally, they need to be permanently water-proof as they are exposed to rain and irrigation (see Figure 1).

Nevertheless, there are also some outdoor testbeds deployed previously. Indeed, [6] reports the experience of running a testbed on a potato field and highlighted many challenges encountered when operating a testbed outdoors. Due to the complexities involved, many outdoor testbeds are rather special purposes WSN deployments like X-Sense [7] or ASWP [8] and do not offer the same flexibility and generic usability like indoor testbeds. With the PotatoNet we combine the robustness needed for an outdoor testbed with maximum flexibility.

II. Implementation

The overall architecture of PotatoNet is shown in Figure 3. The basic architecture consists of a central box providing management functionalities and distributing power to all nodes. For cost-efficiency reasons the nodes are designed to be cheap, while complexity and redundancy features are moved to the central component. We are using Ethernet with passive PoE to distribute power and managing field nodes. The sensor node's IEEE 802.15.4 interface is not required for management or operation of the testbed and can be used exclusively or experiments.

Several features in the architecture lead to PotatoNet's high robustness and resilience against failures:

- Every WSN node is paired with a small embedded Linux board running OpenWRT Linux. The separate programming platform is able to flash the sensor node using ISP. PotatoNet does not rely on bootloaders. This makes it virtually impossible to brick sensor nodes by flashing broken firmware.
- PotatoNet offers two separate Internet uplinks. The main uplink is using Ethernet and can be connected to a standard SOHO router. The management server in the central box includes a cellular data card. As long as the management server has power, it is possible to log into PotatoNet remotely.
- Power to field nodes can be switched on and off via software individually. In case a short circuit triggers a reboot, all field nodes are powered off. This allows switching them one by one identifying the faulty one. Once the faulty node has been identified, PotatoNet can run with the node powered off until it can be replaced.

A. Hardware

We use the INGA [9] WSN node version 1.6.1 in PotatoNet's field nodes. This node is a Dynamic Voltage Scaling (DVS) and Undervolting capable sensor node which is based on a design presented in [10]. However, PotatoNet can be adapted to other WSN nodes easily.

The programming platform inside each field node is based on the WRTNode[1]. The WRTNode is a USD 25 MIPS-based (Mediatek MT7620N) embedded Linux board with 680 MHz 64 MiB RAM which runs the embedded Linux distribution

[1] http://wrtnode.com

(a) Dust during dry days

(b) Exposure to direct sunlight

(c) Water from irrigation and rain

Figure 1: PotatoNet Environment

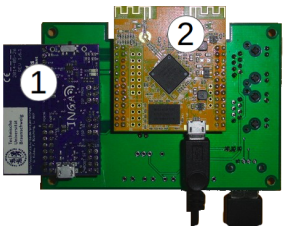
OpenWRT[2]. It supports many GPIO pins which are used for ISP programming of the INGA's two microcontrollers. By using Linux boards in field, in the testbed you can use the same software for programming or analyzing logs as you would do in your test setup on your desk. A large amount of standard Linux software is available for install directly from the OpenWRT repositories. Because each WRTNode includes a complete Wi-Fi implementation with PCB antenna, PotatoNet can also double as a Wi-Fi testbed.

We developed a special companion board that provides an efficient DC-DC converter to supply the WRTnode and INGA from the PoE supply voltage. The WRTNode and the INGA can be plugged into the companion board that includes two RJ45 jacks that can be used to cascade several field nodes.

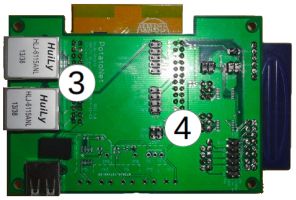
All components of the central box are mounted inside an 73 liter aluminum box (600 x 400 x 410 mm^3), We use rugged industrial connectors for Ethernet and Power to protect against dust and humidity (see Figure 2a). For the enclosure of the field nodes we use standard PVC-U tubing with 160 mm diameter. The enclosure consists of a drainage double socket closed with two socket plugs (see Figure 2b). As only one (or two for cascading) Ethernet cable needs to go into enclosure we used simple cable glands to connect the Ethernet cable and seal the enclosure. By doing so the actual RJ45 connector resides inside the enclosure and thus needs no weather-proofing.

(a) Central Box

(b) Field Node



(c) Companion board front side with INGA WSN node (1) and WRTNode (2)



(d) Companion board back side with Ethernet jacks (3) and extension headers (4)

Figure 2: PotatoNet Components

III. Lessons Learned

While developing PotatoNet we tried to keep an eye on costs, especially for the field nodes as the plan is to increase their amount steadily. However, a major cost factor you should keep in mind is the cost for cables. Overall, even in the initial deployment, PotatoNet includes more than 1 km of cables. One reason is, that in the deployment area cellular connectivity is very bad, so that using cellular networks for both uplinks was not an option. However, the nearest usable DSL landline was 450 meters away. For connecting the main uplink we covered that distance with a Wi-Fi bridge over a street and a dedicated VDSL link through a small forest.

As is common the research field is protected by an electric fence against boars and other animals. They are charged with short electric pulse up to 10 kV to shock any animal touching the fence. It turns out that the shielding of network cables lying near the fence is very good in capturing the jolts and distributing them throughout the testbed. Small shocks can be experienced when touching the shielding and ground at the same time somewhere in the testbed. This is not dangerous, but it shows that robust electrical design is a must.

IV. Conclusions

We presented the PotatoNet outdoor WSN testbed. PotatoNet is a generic testbed for outdoor applications of WSN nodes with a strong focus on robustness and availability. It is currently deployed in a Smart Farming scenario and will be successively extended with more field nodes in the future.

[2]http://openwrt.org/

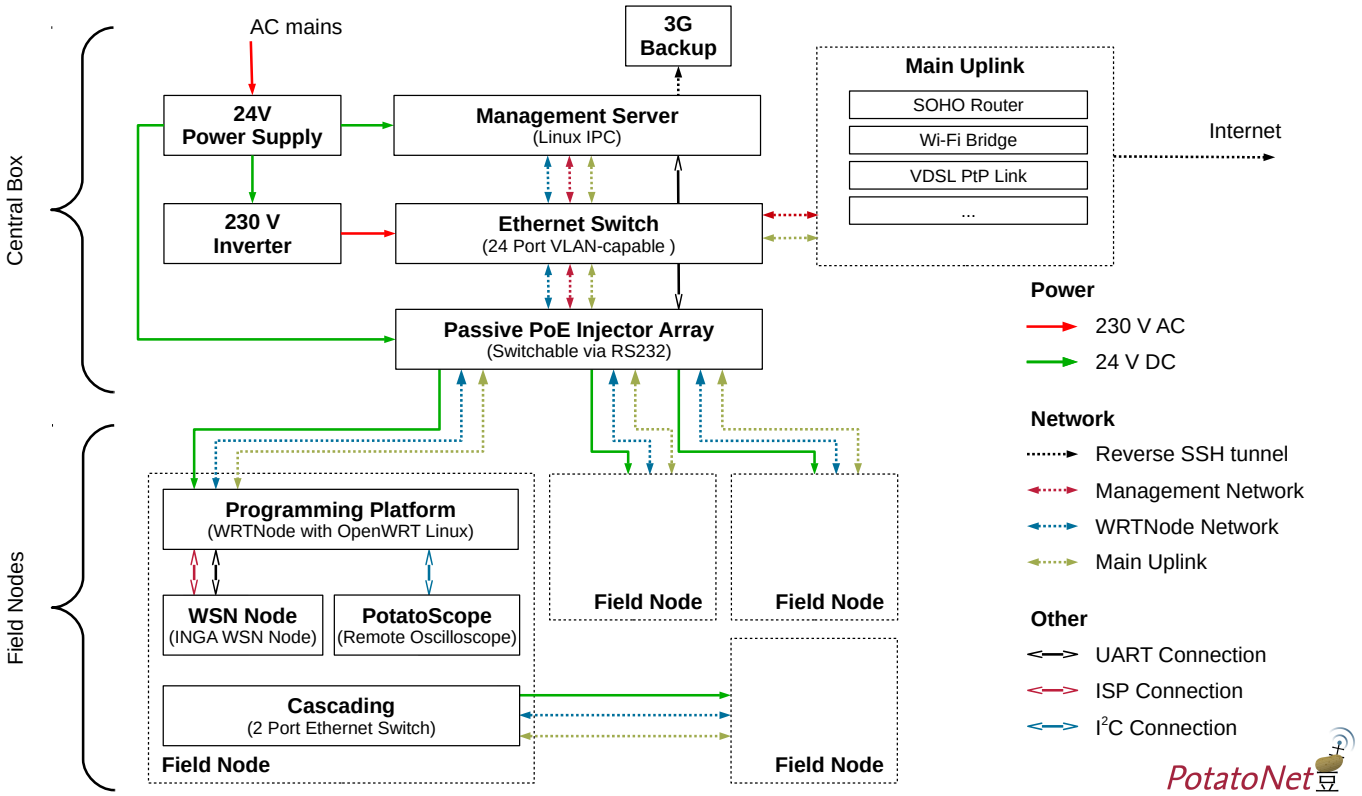


Figure 3: PotatoNet System Architecture

Software and custom-designed hardware schematics will be made available at the PotatoNet project page[3].

Acknowledgments

We would like to thank the VSD Dethlingen potato research station[4] for providing the field and technical support.

References

[1] G. Werner-Allen, P. Swieskowski, and M. Welsh, "Motelab: a wireless sensor network testbed," in *Fourth International Symposium on Information Processing in Sensor Networks (IPSN)*, April 2005.

[2] P. Hurni, M. Anwander, G. Wagenknecht, T. Staub, and T. Braun, "Tarwis - a testbed management architecture for wireless sensor network testbeds," in *7th International Conference on Network and Service Management (CNSM)*, Oct 2011.

[3] J.-P. Sheu, C.-J. Chang, C.-Y. Sun, and W.-K. Hu, "Wsntb: A testbed for heterogeneous wireless sensor networks," in *First IEEE International Conference on Ubi-Media Computing*, July 2008.

[4] H. Hellbruck, M. Pagel, A. Kroller, D. Bimschas, D. Pfisterer, and S. Fischer, "Using and operating wireless sensor network testbeds with wisebed," in *The 10th IFIP Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, June 2011.

[5] H. Wennerstrom, F. Hermans, O. Rensfelt, C. Rohner, and L.-A. Norden, "A long-term study of correlations between meteorological conditions and 802.15. 4 link performance," in *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2013 10th Annual IEEE Communications Society Conference on*. IEEE, 2013, pp. 221–229.

[6] K. Langendoen, A. Baggio, and O. Visser, "Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture," in *20th International Parallel and Distributed Processing Symposium*, April 2006.

[7] J. Beutel, B. Buchli, F. Ferrari, M. Keller, M. Zimmerling, and L. Thiele, "X-sense: Sensing in extreme environments," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011.

[8] M. Navarro, T. W. Davis, Y. Liang, and X. Liang, "A study of long-term wsn deployment for environmental monitoring," in *24th IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, Sept 2013.

[9] F. Büsching, U. Kulau, and L. Wolf, "Architecture and Evaluation of INGA - An Inexpensive Node for General Applications," in *IEEE Sensors 2012*, October 2012.

[10] U. Kulau, F. Büsching, and L. Wolf, "Undervolting in WSNs - Theory and Practice," *Internet of Things Journal, IEEE*, Dec 2014. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6991505

[3] https://www.ibr.cs.tu-bs.de/projects/potatonet/

[4] http://www.vsd-dethlingen.de

Ein Internet-gestütztes Sensornetz zur Erhebung von Mikroklimadaten in Weinbergen

Frank Bohdanowicz, Hannes Frey
Universität Koblenz
Email: bohdan@uni-koblenz.de, frey@uni-koblenz.de

***Zusammenfassung*—Im Projekt Weinberg-Sensornetz wird ein Netz aus drahtlosen Sensoren zur verteilten Messung von Mikroklimadaten in Weinbergen entwickelt. Ziel des Projektes ist die Realisierung eines Sensornetzes bei geringem Kosten- und Arbeitsaufwand über die Programmierumgebung und die Bibliotheken des Arduino-Projektes. Es wurde der Prototyp eines Sensorknotens entwickelt der sich zu einem Ad-Hoc Mesh-Sensornetz vernetzen und drahtlos untereinander kommunizieren kann. Zur Bestimmung des Mikroklimas werden von den Sensorknoten Temperatur- und Luftfeuchtigkeitswerte sowie Helligkeitswerte an verschiedenen Stellen innerhalb eines Weinberges in variierbaren Intervallen gemessen und an einen in der Nähe befindlichen Gatewayknoten gesendet. Der Gatewayknoten sammelt die Sensordaten und übermittelt diese über Mobilfunk an einen Internetserver zur weiteren Verarbeitung. Vom Internetserver können in Gegenrichtung Konfigurationsdaten an den Gatewayknoten und weiter an die Sensorknoten übertragen werden, um beispielsweise die Messintervalle zu regulieren.**

I. Einführung

Unter Präzisionsweinbau wird Weinbaubewirtschaftung unter Berücksichtigung kleinräumiger Boden-, Klima und Bestandsunterschiede verstanden. Mittels digitaler Sensortechnologie erhält ein Winzer zusätzliche Informationen, die auf der detaillierten Analyse seines Weinbergs beruhen und ihn bei der Entscheidungsfindung für die nächsten Arbeitsvorgänge unterstützen sollen. Im Projekt Weinberg-Sensornetz wird auf Basis der Arduino Programmierumgebung und deren Bibliotheken ein batterie- und solarbetriebenes Sensornetz entwickelt, welches verteilt in einem Weinberg Mikroklimawerte als Datengrundlage für Präzisionsweinbau sammeln soll. Der entwickelte Sensorknoten misst in variierbaren Intervallen verschiedene Parameter wie Luftfeuchtigkeit, Temperatur und Sonneneinstrahlung und charakterisiert so eine Mikroklimazone innerhalb einer Weinbergsparzelle oder zwischen verschiedenen Weinbergen. Die Sensorknoten eines Sensornetzes verbinden sich automatisch mit einem Gatewayknoten, der die erfassten Messdaten sammelt und zur weiteren Auswertung über Mobilfunk an einen Datenserver im Internet sendet. Über den Datenserver kann ein Weinbaubetrieb so die tagesaktuellen, mikroklimatischen Sensordaten von zentraler Stelle erhalten. Die Auswertung der Daten soll zu einer besseren Koordinierung von Beobachtungsfahrten und Arbeitsvorgängen führen. Beispielsweise müssen für die Eisweinlese vorgeschriebene Minustemperaturen erreicht worden sein oder in einem Teil der Weinbergsparzelle soll das vorhandene Pilzrisiko über ein stärkeres Ausdünnen des Blattwerks zur Erhöhung der Luftzirkulation und Verringerung der Luftfeuchtigkeit erreicht werden. Das Sensornetz soll von zentraler Stelle über das Internet verwalt- und steuerbar sein. So sollen die Messintervalle reguliert werden können um eine maximale Energieffizienz mit ausreichend detaillierten Messungen für jede Jahreszeit zu erreichen. Da die Kosten für die Herstellung des Sensornetzes für die Wirtschaftlichkeit eine entscheidende Rolle spielen, wurde ein eigener Sensorknotentyp mit möglichst geringem Kostenaufwand entwickelt.

Die weitere Beschreibung des Projektes ist wie folgt aufgebaut. Im nächsten Kapitel folgt eine kurze Beschreibung verwandter Arbeiten aus dem Bereich Agrarsensortechnologie. Danach wird das Projekt mit seinen Komponenten beschrieben. Es folgt eine Evaluation des bisher entwickelten Prototypen. Diese Ausarbeitung schließt mit einer Zusammenfassung und einem Ausblick zur weiteren Vorgehensweise.

II. Verwandte Arbeiten und Stand der Forschung

Die Anzahl der Veröffentlichungen, die sich mit drahtlosen Sensornetzen in der Präzisionslandwirtschaft beschäftigen, steigt seit einigen Jahren kontinuierlich. Viele dieser Arbeiten greifen auf den IEEE 802.15.4 Standard zurück, wie beispielsweise in [1], [2]. In [2] wird ein drahtloses Sensornetz in einem Weinberg auf Sizilien mit 30 TelosB Sensorknoten der Firma Moteiv aufgebaut. Das Sensornetz nutzt IEEE 802.15.4 im 2,4 GHz Band. Die spanische Firma Libellium baut kleine Sensorstationen zum Erfassen jeglicher Daten in einem wie auch immer gearteten Umfeld. Libellium stellt einige Einsatzbeispiele für den Einsatz der Sensorknoten im Präzisionsweinbau auf seiner Webseite vor [3], [4]. Diese Projekte nutzen generische Hardware oder investieren hohen Entwicklungsaufwand und -kosten in das Design eines neuen Sensorknotens. Ziel des hier beschriebenen Weinberg-Sensornetz Projektes ist die Verwendung möglichst kostengünstiger Hardware. Darüber hinaus soll ein Sub-GHz Band für die Kommunikation im Sensornetz verwendet werden. Form und Größe eines Sensorknotens soll so gestaltet sein, dass er den Betriebsablauf im Weinberg nicht stört und minimal-invasiv ausgebracht werden kann.

III. Projektbeschreibung

Das Ziel des Projektes ist die Entwicklung komplexer Sensornetze bestehend aus verschiedenen Sensorknontentypen auf der Basis erweiterbarer Sensorknoten. Beim Bau des Sensorknoten wurde auf Kriterien wie die Verfügbarkeit der Einzelteile in beliebiger Menge und ein möglichst günstiger Preis geachtet. Im folgenden werden die Kosten für die Sensorknoten nach oben gerundet angegeben (Stand 01.07.2015). Alle Einzelteile wurden über im Internet zugängliche Auktions- und Handelsplattformen von meist asiatischen Versendern bezogen. Das Sensornetz ist hierarchisch aufgebaut. Es besteht aus mindestens einem Gateway und mehreren Sensorknoten. In Abb. 1 ist ein Sensorknoten (1a) sowie das Datenmodul

und ein Gateway (1b) abgebildet. Die Sensorknoten sammeln Messdaten und senden diese in regelmäßigen Abständen an den Gateway, der die Daten zuerst lokal auf eine SD-Karte speichert bevor er sie über Mobilfunk an einen Datenserver im Internet zur Auswertung weitersendet. Der entwickelte Sensorknotentyp lässt sich in die folgenden Funktionseinheiten aufteilen

1 *Basismodul* - Mikrocontroller und 433MHz-Funkmodul
2 *Sensormodul* - Temperatur-, Luftfeuchtigkeits- und Helligkeitssensor
3 *Datenmodul* - SD-Karten-Modul und GSM-Modul

Ein Sensorknoten besteht dabei aus den ersten beiden Einheiten, während ein Gatewayknoten zusätzlich noch aus dem Datenmodul besteht.
Zur Erreichung einer möglichst langen Betriebszeit verbringen Sensorknoten und Gateway den Großteil der Zeit im Schlafmodus, in dem sie nur wenig Energie verbrauchen. Während dieser Schlafzyklen ist das Funk- und das Datenmodul ausgeschaltet, es können aber Sensormessungen stattfinden. Die Wachzyklen, in denen die Sensorknoten die Sensorwerte an den Gateway senden, werden von diesem über Hello-Nachrichten bekannt gegeben. Die Sensorknoten synchronisieren sich dann mit den Aufwachzeitpunkten des Gateways. Wird ein Sensorknoten initial aktiviert, so sendet er in kurzen regelmäßigen Abständen eine Hello-Nachricht aus und wartet auf die Beantwortung durch den Gateway oder einen anderen Sensorknoten, der den Aufwachzeitpunkt des Gateways bereits kennt. Erst wenn der Sensorknoten einen Aufwachzeitpunkt erhalten hat, geht er bis zu diesem Zeitpunkt in den Schlafzyklus über. Der Gateway errechnet den nächsten Aufwachzeitpunkt immer am Anfang des aktuellen Wachzyklus. Neben dem Aufwachzeitpunkt teilt der Gateway auch die länge des Schlaf- und des Wachzyklus mit. Sollte ein Sensorknoten einmal keine Verbindung zum Gateway herstellen können, so kann er sich den Zeitraum, in dem der nächste Wachzyklus stattfinden sollte, errechnen. Wenn auch das mehrere Male gescheitert ist, geht der Sensorknoten wieder in den Initialzustand über und sendet in kurzen regelmäßigen Abständen Hello-Nachrichten aus. Die Synchronisation zwischen Sensorknoten und Gateway wird schwieriger je länger sie sich im Schlafzyklus befinden, da die Uhren auseinanderlaufen können.

(a) Sensorknoten (b) Datenmodul und Gateway

Abbildung 1: Sensorknoten und Gatewayknoten

A. Sensorknoten

In Abb. 1a (links) ist der Sensorknoten des Projektes abgebildet. Er besteht aus einem Atmel Atmega 328P-PU und einem HOPE RF 69HW Sendemodul (433 MHz). Auf der Unterseite befindet sich ein DHT22 Temperatur- und Luftfeuchtigkeitssensor sowie ein Photowiderstand zur Messung der Helligkeit. Die Sensoren sind auf der Unterseite angebracht, da sie aus dem Gehäuse herausschauen und so vor direkten Wettereinflüssen geschützt sind. Eine Solarzelle kann zusätzlich angeschlossen werden, welche die Batterie entlastet aber nicht lädt. Als Gehäuse wurde ein handelsübliches Kleingehäuse (72x50x40mm) mit Laschen verwendet. Die Kosten für den prototypischen Sensorknoten belaufen sich auf ca. 25 € ohne Batterien wobei Gehäuse und Solarzelle davon ca. die Hälfte betragen und die eigentlichen Sensoren ca. ein Viertel des Preises ausmachen.

B. Gateway

Der Gateway ist ein Sensorknoten, der mit einem zusätzlichen Datenmodul ausgestattet ist. Das Datenmodul verfügt über eine eigene Batterieleitung und eigene Solarzelle aufgrund des modularen Aufbaus und des höheren Leistungsbedarfs. Das Datenmodul ist mit einem SIM800L GSM-Modul von SIMCOM und einem Micro-SD Kartenmodul von LC Technology ausgestattet. Sowohl dem Micro-SD Kartenmodul als auch dem GSM-Modul sind Spannungswandler vorgeschaltet, da der höhere Leistungsbedarf über höhere Spannungen zur Verfügung gestellt wird. So benötigt das GSM-Modul bis zu 2 Ampere für eine stabile GPRS-Verbindung ins Internet. Auch die Solarzelle des Datenmoduls darf eine höhere Spannung als die des Sensormoduls liefern. Die Kosten für den prototypischen Gatewayknoten belaufen sich auf ca. 60 €. Sie entsprechen den Kosten für einen Sensorknoten mit größerem Gehäuse (128x59x47mm), den Kosten für das Datenmodul von ca. 25 € sowie den Kosten für eine Solarzelle mit höherer Spannungsausbeute und einem größeren Batteriehalter.

C. Internetserver

Der Gatewayknoten baut in variablen Intervallen eine GPRS-Verbindung auf und sendet die Daten via UDP an den Internetserver. Das Transportprotokoll UDP erlaubt ein schnelles Versenden der Daten, so kann die GPRS-Verbindung so kurz wie möglich gehalten werden. Die Übertragung beginnt mit einer Statusnachricht die vom Internetserver bestätigt werden muss. Im Anschluss wird der aktuelle Inhalt der SD-Karte ausgelesen und gesendet. Danach wird erneut eine Statusnachricht vom Gateway gesendet die beantwortet werden muss. Bei Ausbleiben einer Bestätigung wird die Übertragung der Daten durch den Gateway wiederholt. Bei korrekter Übertragung werden die Daten auf der SD-Karte des Gateways gelöscht. Beim Internetserver handelt es sich um einen einfachen UDP-Server geschrieben in Python, der die Daten entgegennimmt und derzeit an die freie Hausautomationssoftware FHEM [1] zur Anzeige in Diagrammen weiterleitet. Der Internetserver antwortet auf die Statusnachrichten des Gateways zu Beginn und am Ende einer jeden Verbindung mit der Serversystemzeit. Auch andere Konfigurationsdaten und eine Bestätigung des korrekten Empfangs der Daten können so übermittelt werden.

D. Nachrichtenprotokoll im Sensornetz

Das Protokoll zwischen den Sensorknoten und dem Gateway besteht aus zwei verschiedenen Nachrichtenformaten, Hello-Nachrichten für die Synchronisation (HelloValues) und Daten-Nachrichten für die Übermittlung der eigentlichen Sensorwerte (SensorValues). Zum Zwecke der Eniergieeffizienz synchronisiert der Gateway seine Wach- und Schlafzyklen mit den Sensorknoten über Hello-Nachrichten. Nach dem Aufwachen aus dem Schlafzyklus versucht jeder Sensorknoten den

[1] http://www.fhem.de

Gateway mit einer Hello-Nachricht zu erreichen (SYN). Der Gateway antwortet auf jede Hello-Nachricht (SYN) mit einer eigenen Hello-Nachricht (SYN_ACK). Anschließend sendet der angesprochene Sensorknoten seine gemessenen Mikroklimawerte über mehrere Daten-Nachrichten an den Gateway. Die Übertragung der Werte wird mit einer Hello-Nachricht (FIN) vom Sensorknoten abgeschlossen, die der Gateway mit einer Hello-Nachricht beantwortet (FIN_ACK).
Ein Sensorknoten setzt während des Schlafmodus alle Netzwerkeinstellungen zurück. Nach dem Aufwachen versucht er sich über eine Default Netzwerk-ID mit dem Gateway zu synchronisieren. Dabei nutzt er die Default Netzwerk-ID als Sender-ID und als Empfänger-ID, bis er vom Gateway eine Netzwerk-ID zugeteilt bekommt, die für diesen Wachzyklus gilt. Die eigentliche Identifikation eines Sensorknotens findet über eine feste Knoten-ID und eine Gruppen-ID statt, die beide im EEPROM des Knotens fest eingespeichert sind. Eine Verschlüsselung der Übertragung wäre eine weitere Möglichkeit der Gruppierung oder Identifizierung von Sensorknoten.
Der Gateway besitzt die feste Netzwerk-ID 1 und die Knoten-ID 000 und empfängt alle eingehenden Nachrichten (promiscouos-mode). Bei der Beantwortung von Hello-Nachrichten belässt er die in der Nachricht enthaltene Knoten-ID und Gruppen-ID für die Identifikation des Empfängers.

Alle Sensorknoten, die die Hello-Nachrichten (SYN_ACK) des Gateways empfangen, übernehmen die Uhrzeit und den Zeitpunkt des nächsten Aufwachens sowie den Wach- und Schlafzyklus. Über die in der Hello-Nachricht angegebene Knoten-ID und Gruppen-ID bestimmt der Gateway an welchen Sensorknoten die Nachricht eigentlich gerichtet ist. Nur dieser Sensorknoten übernimmt dann noch weitere Konfigurationen wie z.B. die übermittelte Netzwerk-ID. Der Gatewayknoten wartet dann auf das Senden der Sensorwerte.
Wenn der anfragende Sensorknoten die Hello-Nachricht (SYN_ACK) des Gateways erhalten hat, übernimmt er die Netzwerk-ID als Sender-ID und sendet die Sensorwerte an die Netzwerk-ID des Gateway (1). Im Anschluss sendet er wieder eine Hello-Nachricht (FIN) an den Gateway, die von diesem auch wieder beantwortet wird (FIN_ACK). Ein Sensorknoten wird somit initiativ über den Gateway konfiguriert und nur die Knoten- und Gruppen-ID muss einmalig vorkonfiguriert werden.

Die Anzahl der zu übermittelnden Sensorwerte wird dem Gateway über die Hello-Nachricht (SYN) mitgeteilt. Die Sensorwerte werden dann für die Übertragung zum Gateway nummeriert. So kann der Gateway den korrekten Empfang aller Sensorwerte überprüfen und fehlende Werte erkennen. In der abschließende Hello-Nachricht (FIN_ACK) teilt der Gateway dem Sensorknoten über eine Bitmaske mit, welche Sensorwerte empfangen (1) oder nicht empfangen (0) wurden. Aufgrund der Bitmaske mit fester Länge ist die Anzahl auf einmal empfangbarer Sensorwerte zwar begrenzt. Allerdings kann der Gateway aufgrund seines begrenzten Arbeitsspeichers (SRAM) auch nur eine begrenzte Anzahl an Datennachrichten mit Sensorwerten auf einmal empfangen. Er speichert die empfangenen Sensorwerte zunächst in seinem Arbeitsspeicher. Erst nach Abschluss der Kommunikation werden sie auf die SD-Karte geschrieben. Anschließend ist der Arbeitsspeicher wieder frei für den Empfang weiterer Sensorwerte.

Alle Sensorwerte die nicht korrekt empfangen oder nicht gesendet werden konnten sendet der Sensorknoten über einen weiteren Kommunikationsaufbau innerhalb des jetzigen oder nächsten Wachzyklus erneut.

Ein Sensorknoten der den Gateway erreicht hat, steht für einige Sekunden als alternativer Gateway dem Sensornetz zur Verfügung. Abgeleitet von seiner temporären Netzwerk-ID kann er selbst anderen Sensorknoten temporäre Netzwerk-IDs vergeben und wie ein Gateway für diese Sensorknoten funktionieren. Eine Distanz-Variable (hop-count metric) sorgt in den Hello-Nachrichten für einen baumartigen Aufbau des Sensornetzes während eines Wachzyklus und verhindert in den SensorValue-Nachrichten, dass Sensorwerte im Sensornetz kreisen. Das ermöglicht es Sensorwerte via Multi-hop über mehrere Sensorknoten oder zu verschiedenen Gatewayknoten innerhalb eines Sensornetzes zu senden.

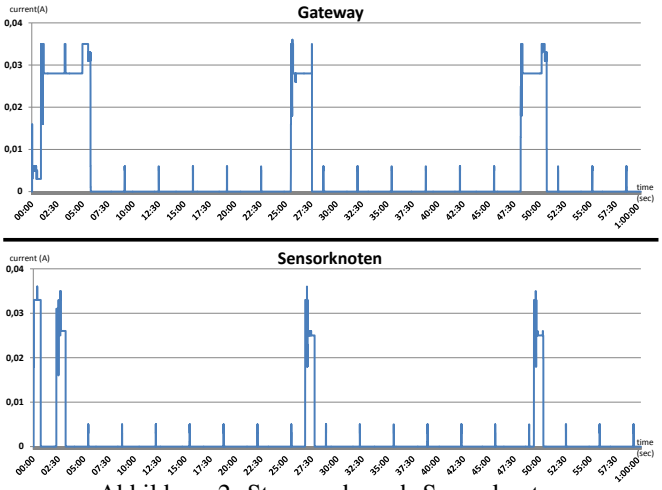


Abbildung 2: Stromverbrauch Sensorknoten

IV. Evaluation

Im folgenden ist eine kurze Analyse der Sensorknoten aufgeführt. Zum einen eine Analyse des Stromverbrauchs des Gateways und eines Sensorknoten und zum anderen der Einsatz der beiden Knoten innerhalb eines Sensornetzes in einem Weinberg.

A. Energieverbrauch

In Abb. 2 ist der Energieverbrauch des Gateways und eines Sensorknoten dargestellt. Beim Gateway ist lediglich der Verbrauch des Sensormoduls gezeigt. Das Datenmodul des Gatways mit dem GSM- und dem SD-Karten-Modul verfügt über eine eigene Stromversorgung. Das Datenmodul ist nur sehr kurz während des Wachzyklus aktiv, wenn Sensordaten über das Sensornetz empfangen wurden und auf die SD-Karte geschrieben werden oder wenn das GSM-Modul am Ende eines Wachzyklus aktiviert wird. Für das folgende Verbrauchsdiagramm wurde ein Sensormessintervall von 5 Minuten, ein Sendeintervall von Sensor- zu Gatewayknoten von 25 Minuten (sleepCycle) und ein GSM-Intervall von 50 Minuten (2 * sleepCycle) gesetzt, d.h. alle 5 Minuten misst der Knoten die Sensorwerte, alle 25 Minuten sendet er die gemessenen Sensorwerte an den Gateway und alle 50 Minuten sendet der Gateway die Sensorwerte an den Internetserver. Die Messungen im Diagramm in Abb. 2 wurden mit einem Labornetzgerät bei 4,2V (ohne Solarzelle) durchgeführt. Das Diagramm zeigt, dass der Verbrauch des Sensor- und des Gateways für den Schlafzyklus

auf unter 1 mA (ca. 15 μA) gesenkt werden kann. Das Auslesen der Temperatur-, Feuchtigkeits- und Helligkeitswerte benötigt ca. 5mA und dauert ca. eine Sekunde. Bis zu 35mA wird für das Senden (Sensorknoten) und Empfangen (Gatway) der Nachrichten benötigt. Die Wachphase des Gateway ist dabei voreingestellt und beträgt hier 2 Minuten. Für die Initialisierung des Sensornetzes wird die Zeit verdoppelt, wobei zu Beginn und zum Ende der Initialisierungsphase eine GPRS-Verbindung zum Internetserver aufgebaut wird. Zu Beginn meldet sich der Gateway beim Internetserver an und erhält so Konfiguratinosdaten wie die aktuelle Uhrzeit. Zum Ende werden alle während der Initialisierungsphase erkannten Sensorknoten dem Internetserver mitgeteilt. Über eine Messzeit von 60 Minuten hatte der Sensorknoten einen durchschnittlichen Verbrauch von 2mA und der Gatewayknoten einen durchschnittlichen Verbrauch von 5mA gezeigt.

B. Erhebung im Weinberg

Ein prototypisches Sensornetz wurde am 04.07.2015 in einem Weinberg[2] ausgebracht. Die Sensormesswerte der Prototypen wurden zuvor unter identischen klimatischen Bedingungen auf Gleichheit getestet. Der Sensorknoten wird mit drei handelsüblichen AA Batterien mit jeweils 1,5V und ca. 1800mAh in Reihe sowie Solarzellen mit einer Leistung von 3,5V und 80mA betrieben. Das Sensormodul des Gateways wurde mit drei AA Batterien und einer Solarzelle mit 4V und 120mA ausgestattet. Das Datenmodul des Gateway hat eine eigene Stromversorgung und wird mit sieben AA Batterien in Reihe und einer Solarzelle mit 9V und 40mA betrieben. Der Prototyp benötigt eine Mindestbatteriespannung von 3,6V. Es wurde ein Sensormessintervall von 10 Minuten, ein Sendeintervall zwischen Sensor- und Gatewayknoten von 60 Minuten (sleepCycle) und ein GSM-Intervall von 6h (6 * sleepCycle) eingestellt. Wobei eine GPRS-Verbindung ins Internet auch immer dann am Ende eins Wachzyklus aufgebaut wird, wenn die Solarzelle den Gatewayknoten ausreichend betreiben kann. Werden die vorherigen Energiemessungen als Grundlage genommen, so ist bei ausreichend Sonne von einer Laufzeit von 3-4 Wochen auszugehen. Abb. 3 zeigt das Ergebnis der Messungen bis zum 17.07.2015 für den Gatewayknoten und den Sensorknoten 001. Die Knoten hängen in diesem Feldversuch nah beiander (ca. 10 Meter) und es lassen sich daher auch sehr ähnliche Verläufe in den Temperatur- und Luftfeuchtigkeitskurven ablesen. Sehr deutlich ist der starke Fall der Betriebsspannung des Gatewayknotens zu erkennen, der durch die Solarzelle bedingt aufgehalten wird. Beim Sensorknoten sind die Messungen insgesamt ungenauer, was an den größeren Spannungsschwankungen liegt wie im Diagramm zu sehen ist. Hier finden auch Messungen während des Wachzyklus statt, was beim Gatewayknoten nicht der Fall ist und sich offenbar negativ auf die Gültigkeit der Messwerte auswirkt.

V. Zusammenfassung

Das Projekt Weinberg-Sensornetze zeigt die konstengünstige Entwicklung eines komplexen Sensornetzes. Die grundlegende Funktion des drahtlosen Sensornetzes konnte erreicht werden. Verbessert werden muss, neben der Ansteuerung der eigentlichen Sensoren und der Erfassung der Sensorwerte,

[2]http://unikorn.uni-koblenz.de/vineyardwsn/

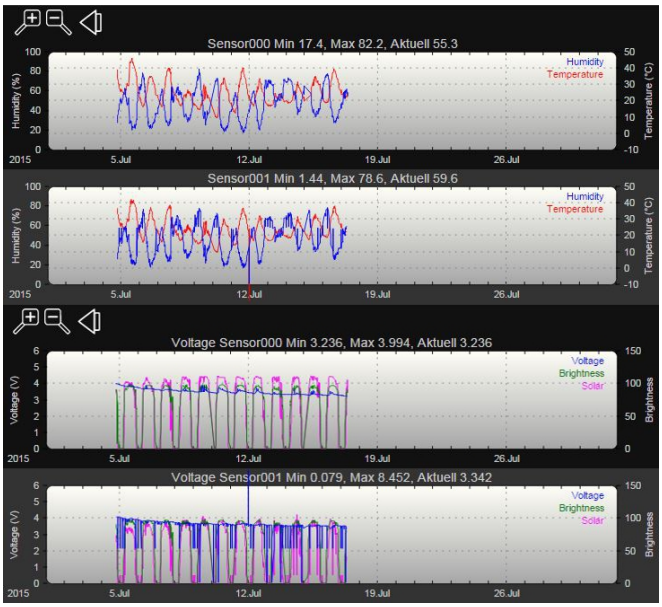


Abbildung 3: Ausgabe des Diagramms auf dem Internetserver

noch die Betriebszeit der einzelnen Knoten und die Sendereichweite der Funkmodule. Durch den Austausch des 433MHz-Funkmoduls konnten die letzten beiden Punkte bereits verbessert werden. Zur Verbesserung der Sendereichweite wird ein anderer Aufbau und ein anderes Gehäuse überprüft, da die gegenwärtige Form zwar wetterfest ist aber das Funksignal stark abdämpft. Zur Verbesserung der Betriebszeit wird eine Solarladeschaltung und der Betrieb über Akkus geprüft. Für eine bessere Synchronisation, gerade bei initialen Sensorknoten, wird ein ständig aktives Funkmodul bei ausreichendem Solarstrom für den Betrieb angestrebt. Grundsätzlich ist ein solches Sensornetz auch für den Einsatz in anderen Dauerkulturen wie Obst- und Hopfenanbau oder Baumschulen möglich. Insgesamt lässt sich Feststellen, dass die Realisierung eines drahtlosen Sensornetzes mit überschaubarem Aufwand und geringem Mitteleinsatz durchführbar ist.

Literatur

[1] I. Cuinas, S. Cervera, and J. A. Gay-Fernandez, “Benefits of using wireless sensor networks to predict plagues in vineyards,” in *Progress In Electromagnetics Research Symposium Proceedings* (P. I. E. R. S. Proceedings, ed.), August 2013.

[2] P. Catania, M. Vallone, G. L. Re, and M. Ortolani, “A wireless sensor network for vineyard management in sicily (italy),” in *Agricultural Engineering International: CIGR Journal, Vol 15, No 4 (2013)*, 2013.

[3] A. Bielsa, “Smart agriculture project in galicia to monitor vineyards with waspmote,” June 2012.

[4] J. Martinez, “Smart viticulture project in spain uses sensor devices to harvest healthier,” February 2014.

[5] R. Morais, M. A. Fernandes, S. G. Matos, C. Serôdio, P. J. S. G. Ferreira, and M. J. C. S. Reis, “A zigbee multi-powered wireless acquisition device for remote sensing applications in precision viticulture,” *Comput. Electron. Agric.*, vol. 62, pp. 94–106, July 2008.

[6] F. J. Pierce and T. V. Elliott, “Regional and on-farm wireless sensor networks for agricultural systems in eastern washington,” *Comput. Electron. Agric.*, vol. 61, pp. 32–43, Apr. 2008.

[7] S. Shanmuganthan, A. Ghobakhlou, and P. Sallis, “Sensors for modeling the effects of climate change on grapevine growth and wine quality,” in *In Proceeding of the 12th WSEAS International Conference on CIRCUITS Heraklion*, 2008.

[8] Aqeel-Ur-Rehman, A. Z. Abbasi, N. Islam, and Z. A. Shaikh, “A review of wireless sensors and networks’ applications in agriculture,” *Comput. Stand. Interfaces*, vol. 36, pp. 263–270, Feb. 2014.

Leistungsmessung eines modularen Netzwerk-Stacks für das IoT-Betriebssystem RIOT

Peter Kietzmann, Martin Landsmann, Thomas C. Schmidt
Internet Technologies Group, Dept. Informatik, HAW Hamburg, Germany
{peter.kietzmann, martin.landsmann, t.schmidt}@haw-hamburg.de

Hauke Petersen, Martine Lenders, Matthias Wählisch
Institut für Informatik, Freie Universität Berlin, Germany
{hauke.petersen, m.lenders, m.waehlisch}@fu-berlin.de

***Zusammenfassung*—Das „Internet der Dinge“ (IoT) beschreibt die Entwicklung, wie maschinengebundene, eingebettete Systeme schrittweise Standardprotokolle der Internet-Welt adaptieren und damit selbst Teil des globalen Inter-Netzwerks werden. Das IoT entwickelt sich gegenwärtig sehr schnell, und eine zunehmende Professionalisierung erfordert den Einsatz einer Systemarchitektur, die Hardware und Kommunikationskomponenten in der Abstraktionsschicht eines Betriebssystems zusammenführt. RIOT, das *freundliche Betriebssystem für das Internet der Dinge*, bildet eine solche Abstraktionsschicht; seit seiner Einführung auf der IEEE INFOCOM 2013 erfreut sich dieses vollständig offene System einer schnell wachsenden Beliebtheit [1]. Bei hoher Effizienz zielt RIOT auf eine modulare Architektur mit programmierfreundlicher Abstraktionsschicht und Unterstützung von C und C++.**

In dieser Arbeit vermessen wir den aktualisierten Netzwerk-Stack von RIOT, welcher heterogene Interface-Treiber mit den gängigen IoT-Protokollen in einer modular geschichteten Architektur vereint. Es ist unser Ziel, die Leistungsfähigkeit der Neuimplementierung des Netzwerk-Stacks in RIOT zu untersuchen. Neben dem Parameter des Datendurchsatzes wird der Energieverbrauch und der Speicherbedarf gemessen.

I. Einleitung

IoT-Geräte haben eng begrenzte Ressourcen. Nicht nur fehlende Hardwarekomponenten wie Speicherverwaltungseinheiten (MMUs), sondern insbesondere der eng begrenzte Arbeitsspeicher macht es auf diesen Geräten unmöglich, etablierte Standard-Betriebssysteme wie Linux zu betreiben. Klasse 1 Geräte [2] verfügen über Speicher von nicht mehr als 10 kB RAM und 100 kB ROM. In diesem Segment ist RIOT [3] das wohl jüngste, am schnellsten an Verbreitung gewinnende Betriebssystem. Es liefert trotz des geringen Speicheraufwands von wenigen Kilobytes RAM und ROM, eine ähnliche Programmierumgebung wie Linux. Eine Minimalkonfiguration des Betriebssystems für ARM Cortex-M-Plattformen benötigt ca. 2,5 kB RAM und 10 kB ROM. RIOT bietet außerdem echtes Multi-Threading, sehr leichtgewichtige Interprozesskommunikation (IPC), Echtzeit-Scheduling, ein uniformes Treibermodell sowie partielle POSIX-Kompatibilität.

Ein zentraler Bestandteil eines IoT-Betriebssystems ist der Netzwerk-Stack. Um im Internet zu kommunizieren, müssen Geräte die Standard-Internetprotokolle IPv6, UDP oder TCP beherrschen. Darüber hinaus werden für den gezielten Einsatz auf eingebetteten Systemen optimierte Protokolle wie 6LoWPAN [4] und COAP [5] erforderlich, welche die effiziente Übertragung von IPv6-Paketen bzw. REST-Zuständen [6] ermöglichen. Aufgrund der steigenden Anzahl verfügbarer Protokolle, sehen wir einen hohen Grad an Modularität und Erweiterbarkeit als zwingende Voraussetzung für einen modernen IoT-Netzwerk-Stack.

Um diese Anforderungen umzusetzen, wurde in RIOT kürzlich ein neuer Netzwerk-Stack, welcher sich durch eine klar definierte Architektur im Hinblick auf Modularisierung, interne Schnittstellen und Erweiterbarkeit auszeichnet [7], entworfen und implementiert. In Abschnitt II geben wir einen Überblick über die Architektur und Implementierung des neuen Netzwerk-Stacks. In Abschnitt III vermessen wir den neuen Netzwerk-Stack hinsichtlich der Systemparameter Datendurchsatz, Energie– und Speicherverbrauch. In Abschnitt IV erfolgt die Zusammenfassung der Ergebnisse.

II. Die RIOT Netzwerk-Stack Architektur

Der neue RIOT Netzwerk-Stack implementiert eine strikt modulare Architektur. Die drei wesentlichen Konzepte sind hierbei:

- eine Aufteilung auf mehrere Threads
- eine Vereinheitlichung der Schnittstellen
- eine deduplizierende Datenhaltung

Diese Architektur erlaubt eine einfache Erweiterbarkeit und Konfigurierbarkeit. Abbildung 1 zeigt eine vereinfachte, beispielhafte Konfiguration des neuen Netzwerk-Stacks mit drei Netzwerkschnittstellen. Hierbei symbolisiert jedes Rechteck einen eigenen Thread. Diese Threads kommunizieren über eine einheitliche Schnittstelle miteinander.

A. Multi-Threading Architektur

Ein hoher Grad an Modularität wird dadurch erreicht, dass jedes Modul des Netzwerk-Stacks in einem eigenen Thread läuft. Module sind in diesem Kontext typischerweise Protokollimplementierungen wie beispielsweise IPv6, UDP oder auch 6LoWPAN. Durch die Implementierung in separaten Threads können verschiedene Module weitestgehend unabhängig voneinander entwickelt werden.

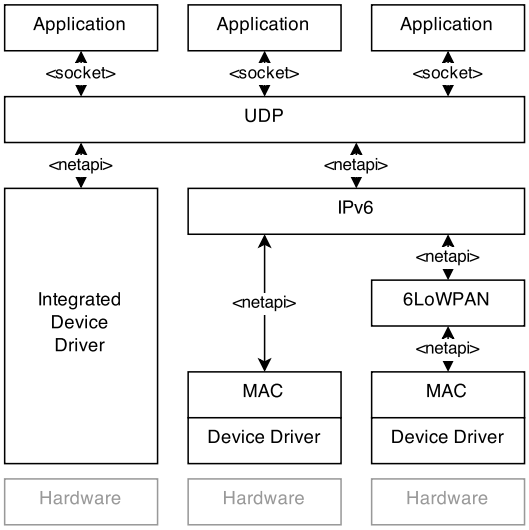


Abbildung 1. Darstellung einer beispielhaften Konfiguration des neuen RIOT Netzwerk-Stacks mit drei Netzwerkschnittstellen

Ein Nachteil dieser Designentscheidung liegt in einem potentiell erhöhten RAM-Verbrauch. Dieser ist dadurch bedingt, dass jeder einzelne Thread einen Stack-Speicher sowie einen „Thread Control Block“ (TCB) benötigt, welche beide im RAM abgelegt werden müssen. In modernen IoT-Betriebssystemen wie RIOT hält sich der Mehraufwand an benötigtem Speicher jedoch in Grenzen, da diese Systeme sehr leichtgewichtiges Threading implementieren. In RIOT ist die Standard-Stackgröße für ARM Cortex-M Plattformen 1024 Bytes, wobei der TCB von 40 Bytes bereits in diesem Speicher enthalten ist. Dieser Stack-Speicher muss von jedem Modul im RAM reserviert werden.

B. Vereinheitlichte Inter-Modul Schnittstelle

Das zweite grundlegende Konzept für die Umsetzung einer modularen Architektur ist die Einführung einer einheitlichen Schnittstelle zwischen den einzelnen Modulen des Netzwerk-Stacks. Die Schnittstelle *netapi* basiert auf der IPC-Infrastruktur des unterliegenden Betriebssystems. Die Kommunikation zwischen den Netzwerk-Stack Modulen findet somit auf Basis von Nachrichten statt, die zwischen den Threads des Netzwerk-Stacks ausgetauscht werden. Diese lose Kopplung erlaubt eine einfache Konfigurierbarkeit von Modulen sowie Erweiterbarkeit des Netzwerk-Stacks mit neuen Modulen.

Die *netapi* Schnittstelle basiert auf vier verschiedenen Nachrichtentypen. Die Nachrichtentypen *send* und *receive* werden asynchron übertragen. Die Nachrichtentypen *get* und *set* zum lesen und setzten von Optionen werden synchron übertragen.

C. Zentraler Paketspeicher

In eingebetteten Umgebungen ist der Arbeitsspeicher eine strikt limitierte Ressource. Im Kontext eines Netzwerk-Stacks sind dabei die Nutzdaten die im Stack verarbeitet werden, eine große Herausforderung. Als Beispiel sei die Verarbeitung von IPv6-Paketen mit einer „Maximum Transmission Unit“ (MTU) Größe von 1280 Bytes auf Geräten mit 16 kB Arbeitsspeicher genannt. Es wird deutlich, dass eine effiziente Datenhaltung unumgänglich ist und Nutzdaten nicht mehrfach im Speicher gehalten werden oder dupliziert werden sollten. In dem neuen Netzwerk-Stack wird dieses Problem mit dem zentralen Paketspeicher *pktbuf* gelöst.

Der Paketspeicher arbeitet auf einem zur Compile-Zeit fest eingestellt Speicherbereich. Mit dieser Designentscheidung kann der Speicher zur Laufzeit nicht wachsen. Die gewählte Paketspeichergröße für IPv6-basierte Netzwerke ist 4 kB in RIOT, was der Größe von drei vollen IPv6-Paketen inklusive eines gewissen Puffers entspricht.

Innerhalb des Paketspeichers werden die Daten in Form von verlinkten Listen von Fragmenten, namentlich *snips*, verschiedener Längen abgelegt. Der Aufbau einer Liste entspricht der Struktur von Netzwerkpaketen, welche aus Nutzdaten (Payload) und den Headern der verschiedenen Schichten im Netzwerk-Stack bestehen. Durch Architektur ist es möglich, einzelne Header effizient zu bearbeiten und in ihrer Größe zu verändern.

III. Evaluierung der Netzwerk-Stack Performance

Die Betrachtung der Systemperformance umfasst die Metriken des Datendurchsatzes, des Energieaufwands und des Speicherbedarfs im RAM. Die Messprogramme verwenden den Netzwerk-Stack mit den Protokollen UDP, IPv6 und 6LoWPAN. Es wurden link-lokale Unicast-IPv6-Adressen [8] für die Adressierung verwendet. Die 6LoWPAN-Adaptionsschicht verwendet IP-Header-Kompression (IPHC) [9].

A. Aufbau und Durchführung

Die Messungen wurden auf einer IoT-typischen Plattform durchgeführt. Der „IoT-lab M3“ Sensorknoten [10] ist Bestandteil einer Testbed Umgebung [11], welche über mehr als 2500 dieser Sensorknoten verfügt. Jeder dieser Knoten ist mit einer externen Messeinheit zur Überwachung verbunden. Die Hardware der Sensorknoten hat die folgenden Spezifikationen:

- ARM Cortex-M3, 32Bit Prozessor
- 72 MHz CPU Frequenz
- 64 kB RAM Arbeitsspeicher
- 512 kB ROM Flash Speicher
- IEEE802.15.4 Funkmodul
- Vier diskrete Sensoren, 3 LEDs

Die Durchsatzmessungen wurden für Nutzdaten der Größen 0 Byte - 1232 Bytes vorgenommen. Das entspricht der MTU von IPv6 (1280 Bytes) abzüglich UDP- und IPv6 Header. Für jeden Messpunkt wurde die Software mit 1000 zu übertragenden Paketen unter Volllast gesetzt. Die Funkdatenübertragung und die Steuerung der Funk-Hardware über den SPI-Bus sind kritische Aspekte hinsichtlich der Geschwindigkeit der Datenverarbeitung. Um die reine Software-Performance des Netzwerk-Stacks zu messen, haben wir die Funkmodule und die zugehörige Treiber-Schicht ausgegrenzt.

Die Energiemessungen wurden mit Nutzdaten der Größen 20 Bytes und 1000 Bytes vorgenommen, um einen Vergleich zwischen „kleinen“ und „großen“ Paketen anzustellen. Dabei wurden wie bei der Duschsatzmessung jeweils 1000 Pakete prozessiert. In dem Messintervall wurde der Energiebedarf gemittelt. Der Verbrauch des ungenutzten Funkmoduls im Wartezustand [12] wurde von den Messungen abgezogen. Des weiteren erfolgte die Messung mit eingeschaltetem UART-Modul, um den Programmablauf zu kontrollieren. Textausgaben fanden nur vor und nach der Datenverarbeitung statt, um das Zeitverhalten der Messung nicht zu verfälschen. Dieser Anteil wurde nicht von den Messungen entfernt.

Der Speicherverbrauch wurde zur Compile-Zeit aus den gelinkten Objekt-Dateien ermittelt. Einen wesentlichen Bestandteil des RAMs allozieren die Stack-Speicher der einzelnen Threads. Der tatsächlich verwendete Stack-Speicher durch die Threads wurde zur Laufzeit ermittelt. Der nicht verwendete, aber allozierte Stack-Speicher, wurde zur Darstellung der Systemperformance von dem RAM-Verbrauch abgezogen.

B. Datendurchsatz

Ein wesentliches Kriterium für die Performance-Evaluierung von Netzwerk-Software, ist die Messung des Datendurchsatzes im Sender und Empfänger. Die Ergebnisse sind in Abbildung 2 dargestellt.

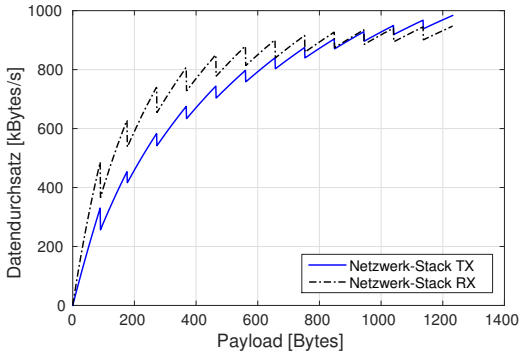


Abbildung 2. Datendurchsatz des RIOT Netzwerk-Stacks unter Ausgrenzung der Funk-Hardware

Erwartungsgemäß steigt der Datendurchsatz mit zunehmender Paketgröße, wobei sich Sender und Empfänger angleichen. Wir vermuten das Volllaufen von Nachrichten-Warteschlangen als obere Performancegrenze. Weiterhin lässt sich eine Zackenform in beiden Graphen erkennen, welche durch die 6LoWPAN Paketfragmentierung von IPv6-Paketen zustande kommt. Es ist zu erkennen, dass der Datendurchsatz des Senders bis zu Payloads von ca. 1000 Bytes unter dem des Empfängers liegt. Als Ursache für die Performanceeinbußen sehen wir im Wesentlichen das zweimalige Allozieren von Daten im Paket-Speicher beim Senden. Das Senden erfordert die Allokation der Payload und der Datenstruktur für die *snips* (Vgl. Abschnitt II-C). Hingegen werden empfangene Daten lediglich einmal im Paketspeicher abgelegt und anschließen markiert. Dieser Einfluss sinkt mit zunehmender Datengröße.

C. Energieverbrauch

Typische IoT-Geräte sind für eine geringe Leistungsaufnahme im Batteriebetrieb ausgelegt. Aufgrund der begrenzten Energieversorgung ist die Betrachtung des Energieverbrauchs ein wesentliches Kriterium in Hinblick auf die Evaluierung der Systemperformance. Eingebettete IoT-Plattformen bestehen aus einem einzelnen Mikrocontroller sowie angeschlossener Peripherie wie Sensoren und Funkmodulen. Des weiteren arbeitet der Kernel von RIOT ohne ein fixes Zeitscheibenverfahren [13]. Anhand dieser Eigenschaften lässt sich ableiten, dass die Systemauslastung während der Bearbeitung von Paketen immer 100% beträgt. Messungen haben diese Annahme bestätigt. Für alle durchgeführten Messungen lag der Verbrauch im relevanten Zeitintervall bei 0,227 W $\pm$ 0,3 $\%$. Der mittlere Verbrauch im Wartezustand betrug ca. 0,21 W. Der relevante Parameter für den gesamten Energieverbrauch ist die Rechenzeit.

In Abbildung 3 ist der tatsächliche Energieverbrauch in μJ pro Payload-Byte im Sender und Empfänger, abzüglich des Leerlaufverbrauchs des Funkmoduls, für die Übertragung von 20 Bytes und 1000 Bytes großen UDP-Paketen dargestellt.

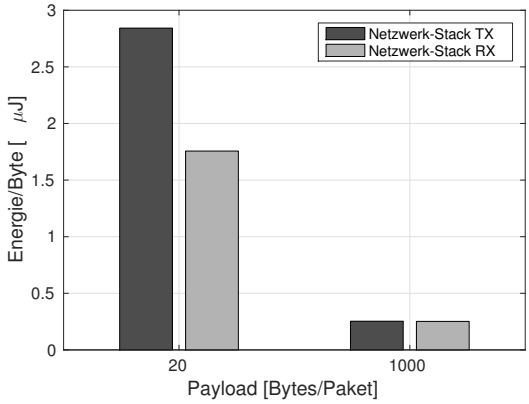


Abbildung 3. Vergleich der Energieaufnahme pro Payload-Byte im RIOT Netzwerk-Stack unter Ausgrenzung der Funk-Hardware; UART-Modul und Funk-Hardware im Wartezustand

Die Ergebnisse korrelieren mit denen aus Sektion III-B. Der Vergleich der Messungen zeigt, dass der Energiebedarf im Sender und Empfänger, normiert auf ein Payload-Byte, mit steigender Payload pro Paket insgesamt deutlich sinkt. Dieses Verhalten lässt sich durch den höheren Datendurchsatz für größere Payloads begründen. Des weiteren wird deutlich, dass sich der Energieaufwand beider Software-Komponenten mit zunehmender Payload angleicht.

D. Speicherbedarf

Die Messung des RAM-Speichers erfolgte mit einer Konfiguration des Netzwerk-Stacks, bestehend aus den Modulen UDP, IPv6 und 6LoWPAN. Das Code-Image beinhaltet außerdem den RIOT-Betriebssystemkern, die nötigen Hardware-Treiber sowie ein minimales Anwendungsprogramm. Für die Messung des Senders beinhaltet dieses die Initialisierung des

Netzwerk-Stacks und das Senden eines vordefinierten UDP-Pakets. Für den Empfänger wird ein weiterer Thread als UDP-Server gestartet. Messungen haben belegt, dass der genutzte Speicher unabhängig von der Payload-Größe des versendeten bzw. empfangenen Pakets ist.

Zur Darstellung sind die Module in logisch geordnete Gruppen gegliedert. Die Gruppe *base* beinhaltet alle Basisfunktionen von RIOT ohne zusätzlich geladene Module sowie das Anwendungsprogramm. Für das Empfängerbeispiel wird der UDP-Server ebenfalls in diese Gruppe gerechnet. *base_net* umfasst die protokollunabhängigen Module die der Netzwerk-Stack verwendet, u.A. den Paketspeicher *pktbuf* und die Inter-Modul Schnittstelle *netapi*. Die Gruppen *UDP*, *IPv6* und *6LoWPAN* fassen die Funktionen und Bestandteile zur Verarbeitung der jeweiligen Protokollimplementierung zusammen. Die Gruppe *mac+driver* stellt alle wesentlichen Funktionen in Sicherungsschicht und darunter dar. Dies beinhaltet den Treiber für das Funkmodul sowie die MAC-Schicht mit ihrer Schnittstelle zu darüber liegenden Netzwerkschichten. Der Speicherverbrauch der einzelnen Module im RAM ist in Abbildung 4 dargestellt. Zur Betrachtung der Performance wird der tatsächlich verwendete RAM-Speicher angezeigt. Der insgesamt allozierte RAM liegt darüber.

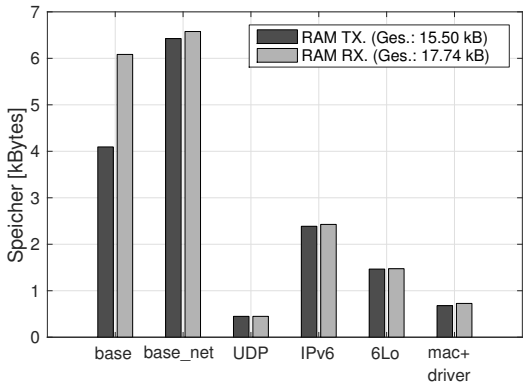


Abbildung 4. Speicherverbrauch des RIOT Netzwerk-Stacks im RAM

Es ist zu erkennen, dass sich der RAM-Bedarf im Sender und Empfänger insgesamt ähnlich verhält. Die Abweichung in der Gruppe *base* lässt sich durch den zusätzlichen Thread des UDP-Servers im Empfänger begründen. Die Gruppen *IPv6* und *6LoWPAN* haben gegenüber *UDP* und *mac+driver* einen erhöhten Speicherbedarf, was sich durch die höhere Komplexität dieser Protokolle begründen lässt. Es ist zu erkennen, dass die Zielhardware aus Sektion III die Anforderungen an den RAM-Speicher problemlos erfüllt. Bei der Betrachtung des Gesamtverbrauchs von ca. 15,5 kB bzw. 17,4 kB im RAM wird deutlich, dass sich der Bedarf an Arbeitsspeicher im RIOT Netzwerk-Stack derzeit an der oberen Grenze für Klasse 1 Geräte [2] befindet.

IV. Zusammenfassung und Ausblick

In diesem Beitrag haben wir den neuen Netzwerk-Stack im IoT-Betriebssystem RIOT vorgestellt. Dabei haben wir das Architekturkonzept und die Implementierung des Netzwerk-Stacks erörtert und die Software-Komponente hinsichtlich der Systemperformance untersucht. Der neue Netzwerk-Stack zeichnet sich durch eine sehr modulare, Thread-basiert Architektur aus, wobei die Kommunikation der Module durch eine uniforme IPC-basierte Schnittstelle erfolgt. Dies führt zu einem hohen Grad an Konfigurierbarkeit. Des weiteren zeichnet sich die Neuimplementierung durch einen zentralen Paket-Speicher aus, auf welchen alle Module des Netzwerk-Stacks zugreifen können. Durch diesen Ansatz werden Datenduplikate vermieden und der Anspruch an den Arbeitsspeicher gering gehalten. Zudem wird so die Anzahl zeitintensiver Kopiervorgänge minimiert. Weiterhin wurden die Systemparameter Datendurchsatz sowie der Energie- und Speicherbedarf gemessen und der Einfluss der Payload-Größe auf die Systemperformance dargelegt. Bei einer Payload von 1000 Bytes pro Paket leistet die Software-Komponente sowohl in Sende- als auch in Empfangsrichtung einen Datendurchsatz von ca. 900 kB/s. Der Energiebedarf der Paket-Prozessierung durch die Software liegt bei ca. 0,2 μJ pro Payload Byte. Beide Komponenten haben noch Optimierungspotenzial. Der Mehraufwand durch die modulare, Thread-basierte Implementierung kann jedoch schon mit diesen Ergebnissen legitimiert werden.

Literatur

[1] (2015, Jul.) Contributions to riot-os. [Online]. Available: https://github.com/RIOT-OS/RIOT/graphs/contributors

[2] C. Bormann, M. Ersue, and A. Keranen, "Terminology for Constrained-Node Networks," IETF, RFC 7228, May 2014.

[3] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *Proc. of the 32nd IEEE INFOCOM. Poster*. Piscataway, NJ, USA: IEEE Press, 2013.

[4] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," IETF, RFC 4944, September 2007.

[5] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," IETF, RFC 7252, June 2014.

[6] Z. Shelby, "Constrained RESTful Environments (CoRE) Link Format," IETF, RFC 6690, August 2012.

[7] H. Petersen, M. Lenders, M. Wählisch, O. Hahm, and E. Baccelli, "Old Wine in New Skins? Revisiting the Software Architecture for IP Network Stacks on Constrained IoT Devices," in *1st Int. Workshop on IoT Challenges in Mobile and Industrial Systems (IoT-Sys15)*. Florence, Italy: ACM, May 2015.

[8] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," IETF, RFC 4291, February 2006.

[9] J. Hui and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," IETF, RFC 6282, September 2011.

[10] O. Fambon, É. Fleury, G. Harter, R. Pissard-Gibollet, and F. Saint-Marcel, "FIT IoT-LAB tutorial: hands-on practice with a very large scale testbed tool for the Internet of Things," INRIA, Tech. Rep., Jun 2014.

[11] "IoT-LAB: a very large scale open testbed," https//www.iot-lab.info/, 2015.

[12] *Low Power 2.4 GHz Transceiver for IEEE 802.15.4, ZigBee, 6LoWPAN, RF4CE, SP100, WirelessHART and ISM Applications*, Atmel Corporation, 9 2009, rev.811C - 09/09.

[13] H. Will, K. Schleiser, and J. Schiller, "A real-time kernel for wireless sensor networks employed in rescue scenarios," in *34th Annual IEEE Conf. on Local Computer Networks Workshops (LCN Workshops)*. IEEE Press, Oct 2009, pp. 834–841.

cowbusconfig – Ansatz zur dezentralen Konfiguration von Gebäudeautomation

Patrick Kanzler
patrick.kanzler@fau.de

Michael Zapf
michael.zapf@fau.de

***Zusammenfassung*—Um die Vorzüge intelligenter Haussteuerung einer breiten Masse zugänglich zu machen, müssen die Systeme einfach handhabbar werden. Besonders die Konfiguration, also die Zuordnung von Sensorereignis zu Schaltaktion, muss einfach durchführbar und ebenso einfach anpassbar bzw. revidierbar sein. Diese Arbeit schlägt einen Ansatz vor, der in einem dezentralen Sensor-Aktor-Netzwerk eine einfache Konfigurationsmöglichkeit bieten soll. Einfache Wenn-Dann-Regeln werden in den Aktorknoten gespeichert und sind jederzeit auslesbar und änderbar. Im Betrieb wird dabei auf zentrale Komponenten verzichtet. Die eigentliche Konfiguration erfolgt per Webbrowser und über ein Gateway, das den Zugang zum Sensor-Aktor-Netz ermöglicht.**

I. Einleitung

„Smart Home“ ist ein Schlagwort, um das man im 21. Jahrhundert kaum noch herum kommt. Der moderne Mensch möchte sein Zuhause vernetzen. Vor allem in gewerblich genutzten Gebäuden ist es inzwischen verbreitet, auf die in der Vergangenheit übliche harte Verdrahtung von Sensoren und Aktoren – zum Beispiel Lichtschalter und Leuchte – zu verzichten und auf intelligente Einzelkomponenten zu setzen, die meist in einer Bustopologie verbunden sind. Der Vorteil liegt auf der Hand: Durch das Fehlen der festen und starren Zuordnung zwischen auslösendem Ereignis des Sensors (*Lichtschalter gedrückt*) und Wirkung des Aktors (*Leuchte an*) können softwarebasiert Ursache-Wirkung-Beziehungen modelliert, evaluiert und korrigiert werden.

Etablierte Systeme setzen dabei in der Regel auf eine Konfigurationssoftware, in der die teils komplexen Kommunikations- und Schaltvorgänge modelliert werden. Die einzelnen an den Bus angeschlossenen Komponenten werden häufig nur mit dem logischen Endergebnis konfiguriert, aus dem sich die ursprünglichen Absichten nicht mehr vollständig ableiten lassen.

In dieser Arbeit beschäftigen wir uns mit dem Gedanken, Knoten in einem Gebäudeautomationsnetzwerk so konfigurierbar zu gestalten, dass die Konfiguration jederzeit mit minimalem Aufwand auslesbar und änderbar ist. Die Idee ist im Kontext einer Vorlesung zum Thema Do-It-Yourself (DIY) von Dr.-Ing. Jürgen Eckert als studentisches Projekt entstanden und zielt deshalb auf einfachste Anforderungen ab.

II. Verwandte Arbeiten

Gebäudeautomation ist inzwischen sowohl in der Praxis als auch in der Forschung verbreitet. Hierbei konkurrieren verschiedene Systeme und Technologien, denen unterschiedlichste Paradigmen zugrunde liegen.

Der unter der Abkürzung EIB bekannte **Europäische Installationsbus** ist ein offener Standard und nach subjektivem Empfinden das in der Praxis wohl meistgenutzte System in der Gebäudeautomation. Inzwischen wird er unter dem Namen **KNX** entwickelt und vertrieben. Es sind weltweit fast 7000 Produkte auf dem Markt[1], die mit einer Twisted-Pair-Leitung verbunden werden (weitere Medien – auch drahtlose – sind verfügbar), über die Nachrichten direkt untereinander ausgetauscht werden[2]. Die Konfiguration der Schaltregeln etc. erfolgt nach der Installation der Komponenten durch einen Fachmann mithilfe von Software aus der ETS-Familie. Dabei handelt es sich um proprietäre Software, die von der KNX Association entwickelt und vertrieben wird[3]. KNX eignet sich besonders im Bereich großer Gebäude, in denen die elektrischen Systeme von einem Fachmann geplant, installiert und auch über Jahre hinweg gewartet werden. Ein entscheidender Nachteil von KNX ist, dass die Konfiguration, sobald sie einmal in die Steuergeräte geschrieben ist, nicht mehr ohne Weiteres ausgelesen werden kann. Um die Konfiguration zu ändern (um zum Beispiel neue Komponenten hinzuzufügen oder auch nur eine einfache Schaltregel anzupassen), wird jedes Mal die ursprünglich erstellte Projektdatei aus der ETS-Software benötigt.

T. Haenselmann et al. [1] beschreiben hingegen ein **zentrales System**. Um die Konfiguration bequem wiederauslesbar und jederzeit änderbar zu gestalten, schlagen sie einen zentralen Server vor, bei dem alle Nachrichten des Systems zusammenlaufen. Sensoren melden ihre Messwerte an diesen, der dann Schaltnachrichten an die Aktoren sendet. Die Entscheidung, welche Aktionen ausgelöst werden, trifft der Server mithilfe von Skripten, die der Benutzer hinterlegt. Diese Skripte können auch über eine Weboberfläche erstellt und verwaltet werden.

Der Nachteil des Ansatzes liegt auf der Hand: Sobald alle Schaltaufgaben von einem zentralen Punkt aus bearbeitet werden, entsteht eine Abhängigkeit des gesamten Systems von einer einzelnen Komponente. Fällt diese aus, funktioniert kein einziger Lichtschalter mehr.

[1] KNX Association, „KNX – Technologie – Einführung“, http://www.knx.org/knx-de/knx/technologie/einfuehrung/index.php

[2] KNX Association, „KNX – Technologie – Kommunikationsmedien“, http://www.knx.org/knx-de/knx/technologie/kommunikationsmedien/index.php

[3] KNX Association, „KNX System Specifications – Architecture“, 2001, http://www.knx.org/media/docs/downloads/KNX-Standard/Architecture.pdf

III. Ansatz

Inspiriert von Haenselmann et al. [1] möchten wir ein System vorschlagen, das ebenso über eine einfache Weboberfläche konfiguriert werden kann. Der Benutzer kann dort einfache Wenn-Dann-Regeln anlegen, bearbeiten und löschen, die das Verhalten der Aktoren in Abhängigkeit von Sensorereignissen festlegen.

Der grundlegende Unterschied im Ansatz besteht jedoch darin, dass wir auf eine zentrale Kommunikationseinheit verzichten.

A. Voraussetzungen, Annahmen

Wir gehen davon aus, dass uns ein paketvermitteltes Netz vorliegt, in dem alle Teilnehmer eindeutig adressierbar sind.

Als Test- und Referenzumgebung dient uns der *cowbus*. Dabei handelt es sich um ein studentisches Projekt, das sich zum Ziel gesetzt hat, eine kostengünstige und einfach nachzubauende Plattform zu schaffen, die einen Einstieg in den Arbeits- und Forschungsbereich des „Smart Home“ bzw. „Connected Home“ ermöglichen soll. Der Nachrichtenaustausch erfolgt dabei drahtlos über eine $2{,}4\,\mathrm{GHz}$-Funkverbindung als geflutete Broadcasts in einer Mesh-Topologie. Eine umfangreichere Beschreibung des Projektes steht auf der Projektseite des Lehrstuhles zur Verfügung, die Daten des Projekts sind bei GitHub verfügbar[4].

B. Idee

Da **Sensoren** häufig höhere Anforderungen an einen sparsamen Energieverbrauch stellen, sollen sie in diesem System „dumm“ und zustandslos sein. Sie erarbeiten sich also kein Wissen über die anderen Netzwerkkomponenten und reagieren nicht, sondern agieren ausschließlich. Das ermöglicht uns, sie die meiste Zeit in Energiesparmodi zu betreiben und nur dann aufzuwecken, wenn sie selbst ein Ereignis auslösen, zum Beispiel wenn der Benutzer einen Taster betätigt.

Aktoren hingegen sind häufig mit stabilen Energiequellen ausgestattet, da sie meist gewisse Lasten schalten. Als Beispiele wären Leuchten, Jalousiemotoren oder auch Ventilatoren zu nennen. Daher bietet es sich an, logische Verknüpfungen hier vorzunehmen. Aktoren empfangen also die Ereignisse, die von Sensoren generiert werden. Diese sind im einfachsten und üblichsten Fall die Zustände der Sensoren, zum Beispiel eine Temperatur oder eine Schalterposition. Auf Basis dieser Ereignisse treffen Aktoren dann mithilfe von Regeln (siehe III-C) Entscheidungen und führen Schaltaktionen aus.

Ein **Gateway** verbindet das Sensor-Aktor-Netz mit einem vorhandenen herkömmlichen Local Area Network (LAN) oder Wireless Local Area Network (WLAN). Es muss mit einem Funkmodul ausgestattet sein, um mit den Sensoren und Aktoren zu kommunizieren, und mit einer LAN- bzw. WLAN-Schnittstelle. Es stellt einen Webserver zur Verfügung, der die Konfigurationswebsite im LAN- bzw. WLAN anbietet.

[4]Dokumentation siehe: http://www7.cs.fau.de/de/wp-content/uploads/sites/2/2014/08/projektdoku_cowbus.pdf; Projekt siehe: https://github.com/michz/diy14bus.

Die eigentliche **Konfiguration** der Aktoren wird dann über ein webfähiges Gerät vorgenommen, das über das LAN bzw. WLAN die Konfigurationswebsite des Gateways anzeigt. Dabei werden Konfigurationsnachrichten „in-band“ gesendet, also neben den Nutzdaten auf demselben Funkkanal. Erkannt werden diese Pakete von den Knoten an speziellen Nachrichtentypen.

Die **Website** kommuniziert über ein WebSocket mit dem Gateway, welches die Anfragen in Pakete für das Mesh-Netzwerk übersetzt. Dies wird zum einen dazu genutzt, um eine Liste der verfügbaren Knoten zu erzeugen und für diese Liste die bestehende Konfiguration auszulesen. Zum anderen erlaubt die Konfigurationswebsite das Erstellen neuer Regeln, indem sie den Knoten entsprechende Pakete sendet. Daraufhin aktualisieren die betroffenen Knoten ihre internen Regeltabellen und bilden damit die neue Sensor-Aktor-Verknüpfung ab. Darüber hinaus ist es über die Website möglich alle Nachrichten, die das Gateway auf dem Funkkanal „sieht“, zu protokollieren und so zu Zwecken der Fehlersuche zu nutzen.

Wie bereits angeklungen basiert die Konfiguration auf **Regeln**, auf die im Abschnitt III-C genauer eingegangen wird. Jeder Aktorknoten muss eine oder mehrere Regeln speichern, auf Anfrage auslesen und rückmelden, sowie löschen können.

C. Regeln

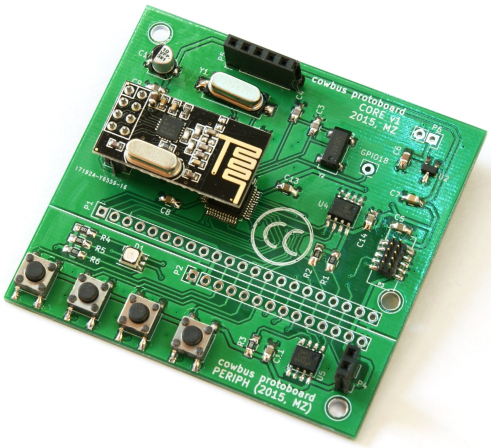
Basis der Logik sind einfache Wenn-Dann-Regeln. Ein (Aktor)-Knoten kann theoretisch beliebig viele dieser Regeln befolgen.

Eine Regel besteht aus mehreren Komponenten. Als Beispiel dient ein Temperatursensor, der einen Ventilator nur dann anschalten soll, wenn die Temperatur in einem bestimmten Intervall liegt.

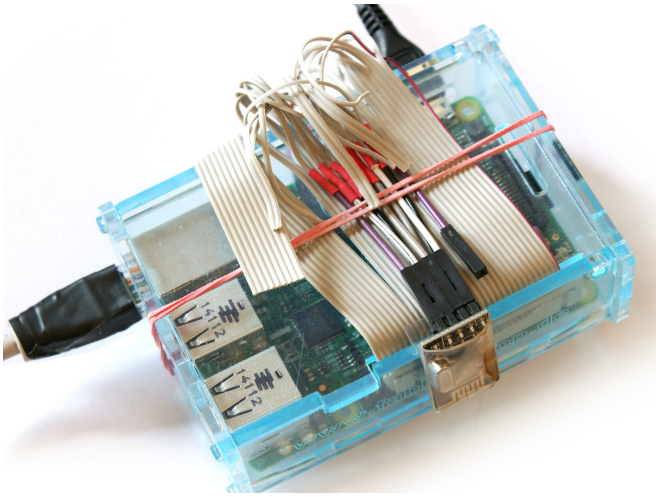
1) Festgelegt wird, auf Pakete welcher **Adresse** die Regel zutrifft. Im Beispiel wäre dies die Adresse des Temperatursensors.
2) Der **Operator** legt fest, wie der angegebene Schwellwert der Regel mit dem im Paket übertragenen Wert verglichen wird. Im Beispiel nutzen wir einen Intervalloperator.
3) Verglichen wird jeweils mit dem bereits erwähnten **Schwellwert**. Wenn als Operator eine Intervallprüfung gewählt wird, müssen zwei Schwellwerte (ein oberer und ein unterer) angegeben werden. Im Beispiel wird das Intervall $[18; 25]$ gewählt.
4) Die **Aktion** gibt an, was der Aktor macht, wenn ein Paket eingetroffen ist, auf das die Regel zutrifft. Hier wird der Ventilator an- bzw. abgeschaltet.

Soll ein Ventilator aktiviert werden, wenn der Wert eines Temperatursensors im Intervall $[18; 25]$ liegt, könnten die zugehörigen Regeln wie folgt aussehen:

	Regel 1	**Regel 2**
Adresse:	Sensoradresse	
Operator:	„$\leq\geq$“	„nicht $\leq\geq$“
Schwellwert A:	18	
Schwellwert B:	25	
Aktion:	„Ventilator an“	„Ventilator aus“



(a) Sensor- sowie Aktorknoten werden durch cowbus-Prototyping-Boards realisiert.



(b) Die Funktion des Gateways wird von einem Raspberry Pi übernommen.

Abbildung 1: Komponenten der Testumgebung

D. Konfigurationsnachrichten

Um die Konfiguration jederzeit auslesbar und änderbar zu gestalten, sind folgende Nachrichtentypen vorgesehen:

- **LIST** liest die Konfiguration eines Knotens.
- **ADD** fügt einem Knoten eine Regel hinzu.
- **DELETE_ONE** löscht eine Regel auf einem Knoten.
- **DELETE_ALL** löscht alle Regeln auf einem Knoten.
- **DELETE_ADDRESS** löscht alle Regeln auf einem Knoten, die sich auf eine bestimmte Sensoradresse beziehen.

IV. Evaluation

Das Testnetzwerk besteht aus drei Komponenten:

- Als **Sensor** dient ein Taster auf einem cowbus-Prototyping-Board (Abb. 1a).
- Als **Aktor** dient eine Leuchtdiode (LED), die ebenfalls auf einem cowbus-Prototyping-Board sitzt.
- Die Rolle des **Gateways** übernimmt ein Raspberry Pi, der um ein Funkmodul erweitert wurde (Abb. 1b). Alternativ wäre auch denkbar ein USB-Dongle zu entwickeln, das ebenfalls um ein Funkmodul verfügt und an einen beliebigen Rechner angeschlossen werden kann.

Der vorgestellte Ansatz funktioniert in diesem kleinen Maßstab sehr gut. Einfache Schaltregeln lassen sich damit ohne großen Aufwand realisieren. Werden die Regeln in den Knoten persistent gespeichert, stellen diese eine ähnlich stabile und dauerhafte Konfiguration dar, wie sie die eingangs erwähnten Systeme bieten.

V. Zusammenfassung und Ausblick

Mit cowbusconfig haben wir einen Ansatz zur einfachen Konfigurierbarkeit von Sensor-Aktor-Netzwerken im Hausautomationsbereich vorgestellt. Die Besonderheit gegenüber bisher üblichen oder erforschten Ansätzen ist dabei die Möglichkeit auf zentrale „Logik“-Komponenten zu verzichten und trotzdem die Konfiguration wiederauslesbar und änderbar zu gestalten. Damit ist es möglich, auf einfache Art und Weise neue Komponenten in das System zu integrieren und entsprechende Schaltregeln hinzuzufügen bzw. anzupassen, ohne das komplette System neu einstellen zu müssen.

Offen bleibt jedoch die Frage, wie komplexere logische Abhängigkeiten dargestellt werden können, wie zum Beispiel die Aggregation von Ereignissen bzw. Werten. So könnte es im Beispiel des Ventilators sinnvoll sein, diesen nur zu aktivieren, wenn der Temperaturwert über einige Minuten lang konstant über dem Schwellwert liegt. Mögliche Ansätze reichen von der Pufferung innerhalb eines Aktors bis hin zu autonomen Aggregatorknoten, die Ereignisse beobachten und daraus neue Ereignisse generieren.

Implementierungen

Beispielhardware und -code sind unter freien Lizenzen im cowbus-GitHub-Projekt verfügbar:
https://github.com/michz/diy14bus

Danksagung

Wir möchten uns bei Dr.-Ing. Jürgen Eckert bedanken, der uns mit der Lehrveranstaltung „DIY: Personal Fabrication“[5] und seiner persönlichen Unterstützung ermöglicht hat einen Einblick in die Forschung im Gebiet der modernen Sensor-Aktor-Netze zu gewinnen und dabei selbst sehr frei und selbstständig Erfahrungen zu sammeln.

Literatur

[1] T. Haenselmann, T. King, W. Effelsberg, M. Busse, and M. Fuchs, “Skriptbasierte drahtlose Gebäudeautomation mit Sensornetzen,” *PIK-Praxis der Informationsverarbeitung und Kommunikation*, vol. 30, no. 3, pp. 163–169, 2007.

[5] „DIY: Personal Fabrication“, Wintersemester 2014/2015, FAU, http://www7.cs.fau.de/de/teaching/diy-2014w/

RESTful Services für 6LoWPAN Networks

Thomas Scheffler
Beuth Hochschule für Technik Berlin
University of Applied Sciences
Luxemburger Str. 10, 13353 Berlin, Germany
Email: scheffler@beuth-hochschule.de

***Zusammenfassung*—Die Verknüpfung netzwerkfähiger Sensoren und Aktoren zum Internet der Dinge ist momentan ein wichtiger Treiber für die Forschung und Entwicklung im Netzwerkbereich. Die fortschreitende Vernetzung elektrisch steuerbarer Geräten ermöglicht neuartige Nutzungs- und Interaktionsszenarien. Die Entwicklung von Softwareanwendungen für diese Netze sollte auf etablierte Entwurfsmuster verteilter Anwendungen zurückgreifen. Dabei muss beachtet werden, dass aufgrund der stark eingeschränkten Ressourcen der Geräte eine verstärkte Integration und Anpassung der Softwarekomponenten nötig ist.**

Im Rahmen dieser Arbeit wird gezeigt wie sich durch den Einsatz geeigneter Netzprotokolle und Betriebssysteme die Eigenheiten eines Mikrocontrollersystems verbergen lassen, so dass verteilte Anwendungen mittels bekannter Paradigmen und Applikationsframeworks entworfen und umgesetzt werden können.

I. Einführung

Die Vernetzung intelligenter Geräte, so genannter Smart Objects, bringt einige neue Herausforderungen mit sich. Anders als bei Computern und Smartphones stellt die Netzwerkfähigkeit keine Hauptanforderung dar, sondern wird als Zusatznutzen zur eigentlichen Gerätefunktion implementiert. Die Produktlebenszyklen, Support- und Installationsanforderungen an solche Geräte sind teilweise sehr verschieden von bekannten Anwendungsszenarien für Netzwerktechnik. So sollen sich zum Beispiel Sensornetze mit tausenden von Sensoren automatisch konfigurieren, Netzwerkverkehr nach Kriterien der Energieeffizienz routen und im Batteriebetrieb mehrere Jahre einsatzfähig sein. Darüberhinaus sollen die einzelnen Geräte nur wenige Euro kosten.

Diese Anforderungen haben in der Vergangenheit dazu geführt, dass solche Netzwerke anwendungsspezifische Protokolle einsetzen, die einen reduzierten Funktionsumfang bereitstellen und eine sehr starke vertikale Integration aufweisen (Zigbee, Bluetooth). Die typischen Erfolgsmerkmale eines vollständigen Internet-Protokoll Stacks, wie globale Adressierbarkeit und Erreichbarkeit, Bereitstellung einer abstrakten Socket-Schnittstelle für die Netzwerkintegration von Anwendungen und der Einsatz unterschiedlicher Transportprotokolle lassen sich in solchen Netzen nur sehr eingeschränkt realisieren und setzen in der Regel den Einsatz von Gateways voraus.

Fortschritte auf dem Gebiet der Mikrocontroller, der Übertragungstechnik und des Netzprotokolldesigns haben es mittlerweile aber möglich gemacht, die oben genannten Anforderungen auf Basis kostengünstiger und energieeffizienter Mikrocontroller zu realisieren, welche über eine standardkonforme Implementierung des Internet-Protokolls verfügen. Als Softwarebasis für die Entwicklung eines netzwerkfähigen Geräts bietet sich zum Beispiel Contiki an [1]. Contiki ist ein lizenzfreies, minimales Betriebssystem, welches auf verschiedenen 8-, 16- und 32-bit-Mikrocontrollern, als auch nativ als Linux-Prozess lauffähig ist.

In dieser Arbeit wird die prototypische Entwicklung eines Webservices auf einem Mikrocontroller-Betriebssystems vorgestellt, welches auf dem REST-Paradigma beruht. Primäres Ziel ist dabei nicht der Einsatz eines effizienten Übertragungsprotokolls sondern die möglichst vollständige Abstraktion der Eigenheiten des Mikrocontroller-Systems. Das Ziel der Entwicklung ist die Unterstützung der praktischen Lehre und Ausbildung im Schwerpunkt 'Verteilte Kommunikationsplattformen und -dienste' der Beuth-Hochschule. Die Studierenden sollen in die Lage versetzt werden verteilte Anwendungsszenarien im Rahmen des Übungsbetriebs zu entwerfen und umzusetzen sowie durch Funktions- und Leistungstests die kritische Randparameter ihrer Lösung selbst testen zu können.

II. Vernetzung auf Basis von IEEE 802.15.4

Internetfähige Geräte müssen über eine geeignete Sicherungsschicht verfügen um IP-Pakete transportieren zu können. An dieser Stelle führen die erwähnten Anforderungen dazu, dass etablierte Protokolle und Netzzugriffsverfahren aus dem PC-Bereich ungeeignet sind, weil ihr Energiebedarf zu hoch ist oder bestimmte Netztopologien voraussetzt.

IEEE 802.15.4 ist ein Funkstandard für Home Networks, Industrie- und Gebäudeautomatisierung [2]. Der Standard wurde Anfang der 2000er Jahre entworfen und hat das Ziel, kostengünstige, niedrig-bitratige Vernetzung für Geräte zu ermöglichen, die bisher nicht netzwerkfähig waren. Dafür musste insbesondere der Energiebedarf optimiert werden. Die Anforderungen an Datenrate und Latenz sind dagegen gering und sollten für einfache Kontroll- und Steuerungsfunktionen ausreichen. Gegenüber alternativen Technologien wie Bluetooth-LE bieten Netze auf Basis von IEEE 802.15.4 weiterhin den Vorteil der größeren Zahl adressierbarer Knoten (16-bit MAC-Adressen) und der möglichen Vollvermaschung (Mesh-Network). Knoten mit Routingfunktionalität können Nachrichten weiterleiten und mit wenigen Edge-Routern ein weitflächiges Netzwerk aufspannen.

Der Standard definiert mehrere Übertragungsverfahren in verschiedenen lizenzfreien Übertragungsbändern:

- 20 kbit/s bei 868 MHz
- 40 kbit/s bei 915 MHz

- 250 kbit/s bei 2.4 GHz (DSSS)

III. IPv6

Implementierungen für das Internet der Dinge sollten sinnvollerweise direkt auf dem Internetprotokoll in Version 6 aufsetzen. Aufgrund des beschränkten Adressraums von IPv4 ist es nicht sinnvoll, ein Netzwerk mit mehreren Milliarden erwarteten Geräten mit 32-bit-Adressen zu konfigurieren.

Das IPv6-Kernprotokoll wurde 1998 von Steve Deering und Rob Hinden im RFC 2460 beschrieben [3] und stellt durch 128-bit Adressen eine nahezu unerschöpfliche Ressource dar. Gegenüber IPv4 weist es die Besonderheit auf, dass spezielle Protokollfunktionen in Erweiterungsheader ausgelagert wurden und eine stärker strukturierte Adressierung effizientes Routing auch innerhalb einer Site erlaubt.

IPv6-Endsysteme können sich im Netz automatisch konfigurieren, was für den vorliegenden Anwendungsfall von besonderem Interesse ist. Die bis vor wenigen Jahren noch mangelhafte Verbreitung von IPv6 stellt heute kein Hindernis mehr dar. Laut aktueller Statistik wurden Mitte 2015 in Deutschland bereits über 15% aller Anfragen an Google über IPv6 abgewickelt[1].

IV. 6LoWPAN

IEEE 802.15.4 definiert eine Framegröße von maximal 127 Byte. Je nach Adressierungsart und aktiven Sicherheitsfeatures können davon 93-118 Bytes als Payload genutzt werden. Die geringe Framegröße ergibt sich in direkter Konsequenz aus der niedrigen Datenrate. Größere Frames würden zu einer erhöhten Wahrscheinlichkeit führen, dass der Frame wegen eines Bitfehlers verworfen wird. Weiterhin minimieren kleinere Frames die Belegungszeit des Funkkanals. Hier ergibt sich ein Problem: Der IPv6-Standard fordert eine Path MTU von mindestens 1280 Bytes und benötigt bereits 40 Bytes für die Übertragung des einfachen IPv6-Headers. Für die Übertragung von IPv6-Paketen in IEEE 802.15.4 Frames wird daher eine Anpassungsschicht benötigt.

Der 6LoWPAN-Adaptation-Layer ist unter anderem in RFC 4919 [4] und RFC 4944 [5] standardisiert. 6LoWPAN definiert einen Mechanismus zu Link-Layer Fragmentierung/Defragmentierung, ein Verfahren zur zustandslosen (stateless) Header Compression für den IP- und UDP-Header und unterstützt verschiedene Topologieoptionen.

V. Contiki

Contiki ist ein quelloffenes Betriebssystem für Embedded Devices. Es wurde ursprünglich am Swedish Institute of Computer Science (SICS) entwickelt[2] und verfügt über eine aktive internationale Entwicklergemeinde. Es werden verschiedene Hardwareplattformen unterstützt (z. B. Linux nativ, AVR, ARM, MSP430). Weiterhin ist es vergleichsweise einfach, eine Portierung selbst vorzunehmen, da Contiki komplett in C geschrieben ist.

[1] https://www.google.com/intl/de/ipv6/statistics.html#tab=per-country-ipv6-adoption

[2] https://www.sics.se/projects/contiki-connecting-the-next-billion-devices

Contiki bietet kooperatives Multitasking in Form sogenannter Protothreads. Es verfügt über einen IPv4/IPv6-Netzwerkstack und benötigt für minimale Betriebssystemfunktionen ca. 40 kB Flash und etwa 2 kB RAM. Je nach implementierter Anwendung fallen diese Anforderungen natürlich höher aus, sind aber immer noch auf einfachen Mikrocontrollern wie zum Beispiel einem ATmega 1281 realisierbar. Für den Einstieg in die Entwicklung mit Contiki biete es sich an, ein komplett konfiguriertes VM Image auf Basis von Ubuntu von der Webseite des Contiki-Projekts herunterzuladen.

VI. REST (Representational State Transfer)

REST bezeichnet einen Architekturstil für verteilte Anwendungen und wird als leichtgewichtige Alternative zu Mechanismen wie RPC (Remote Procedure Calls) und Web Services (SOAP, WSDL, etc.) genutzt. Anwendungen die nach dem REST-Prinzip aufgebaut sind, werden auch als RESTful-Webservices bezeichnet. Sie benutzen URIs um Ressourcen zu adressieren, HTTP-Request-Methoden als einheitliches Interface und XML oder JSON (JavaScript Object Notation) für die Repräsentation der Daten.

Ressourcen sind das Schlüsselelement eines RESTful-Designs. Die Gesamtheit der Ressourcen in Kombination mit den Zugriffsmethoden definiert die REST API.

A. Ressourcen:

- werden durch URIs identifiziert,
- repräsentieren Zustände und Funktionalität einer verteilten Anwendung und
- lassen sich aus allen Teilen des Systems unmittelbar ansprechen.

B. Zugriffsmethoden:

REST nutzt für alle Operationen der Anwendung (Create, Read, Update, Delete) ausschließlich die standardisierten HTTP-Methoden (GET, POST, PUT, DELETE). Es existieren keine benannten *Befehle* oder *Services* wie in RPC oder SOAP. Es ist in REST nicht notwendig, einen bestimmten Methodennamen zu kennen und aufzurufen (wie z. B. *getProductPrice(x)*). Stattdessen wird ein HTTP-Request auf eine bestimmte URI ausgeführt.

Bei der Anwendung ist zu beachten, dass insbesondere die HTTP PUT- und POST-Methoden semantische Unterschiede hinsichtlich ihrer Ausführung aufweisen. So ist PUT eine idempotente Methode. Durch die Anwendung von PUT wird eine Ressource erzeugt oder vollständig aktualisiert. Das bewirkt, dass die mehrfache Ausführung von PUT immer zum gleichen Ergebnis führt. Die POST-Methode hingegen ist nicht idempotent und erzeugt bei jedem Aufruf eine Child-Ressource oder fügt Daten zu einer existierenden Repräsentation hinzu.

Die folgenden Beispiele verdeutlichen die Anwendung der HTTP-Methoden:

GET wird verwendet, um Daten oder Statusinformationen des Geräts zu erfragen.

PUT kann die Gerätekonfiguration auf einen definierten Wert ändern.

POST kann verwendet werden, um zwischen verschiedenen Optionen zyklisch zu wechseln.

DELETE wird verwendet, um Zustandsinformationen, die in Form von Ressourcen auf Servern und Caches gespeichert sind, zu löschen.

Für die Implementierung eines RESTful-Services ist es sinnvoll, sich an die folgenden Designrichtlinien zu halten:

Client-Server-Architektur: RESTful-Services funktionieren nach dem Muster der Client-Server-Kommunikation. Client-Komponenten können Dienste für weitere Komponenten verfügbar machen und verhalten sich gegenüber anderen Clients damit selbst als Server.

Ressourcen-Web: Eine einzelne Ressource sollte nur relevante Daten bereitstellen, kann aber Links auf weitere Ressourcen enthalten (so wie im Web-Kontext allgemein üblich).

Zustandsverwaltung: Die Interaktion ist zustandslos, allerdings können Server und Ressourcen zustandsbehaftet sein. Um ein gute Skalierbarkeit zu gewährleisten, sollte jeder neue Request alle Daten für seine Ausführung beinhalten und nicht von früheren Interaktionen abhängig sein.

Caching: Ressourcen sollten cacheable sein (Angabe eines Ablaufdatums/-zeit). Dafür kann der HTTP-Cache-Control-Header benutzt werden.

Proxy Server: können Teil der Architektur sein, um Performance und Skalierbarkeit zu erhöhen.

Beispiel: Ein Service macht über folgende URL den Zugriff auf Berichte verschiedener Nutzer möglich:

```
http://example.com/reports/
```

Über einen HTTP GET-Aufruf lassen sich die Berichte der einzelnen Nutzer aufrufen:

```
http://example.com/reports/alice
```

oder alle vorhandenen Berichte abfragen:

```
http://example.com/reports/all
```

C. *Repräsentation der Daten:*

Anfragen werden als HTTP GET-Request an die spezifizierte Ressource (URI) gesendet:

```
GET /name/meier/alter/43 HTTP/1.1
Host: echo.jsontest.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8)
```

RESTful-Services liefern als Antwort üblicherweise maschinenlesbare Daten und kein HTML. Dabei kommen XML-, JSON- und CSV-Formate zum Einsatz:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=ISO-8859-1
Date: Sun, 01 Dec 2013 19:24:42 GMT

{
   "name": "meier",
   "alter": "43"
}
```

Wie erwähnt, werden RESTful-Services als Alternative zu Webservices und RPC eingesetzt. Vergleichbar mit Webservices, haben REST-Services folgende Eigenschaften:

- Platform-independent (Server läuft unter Unix, Client unter Windows, ...),
- Language-independent (C#-Server kommuniziert mit Java-Client, etc.),
- basierend auf etablierten Standards (HTTP, XML, JSON),
- Einsatz über Firewalls hinweg möglich.

REST ist kein Standard, sondern verwendet die Designprinzipien des WWW für den Datenaustausch zwischen Anwendungen. Verschiedene Frameworks[3] unterstützen und implementieren RESTful-Services und da es sich um ein sehr einfaches Konzept handelt, sind komplett eigene Implementierungen möglich. Für den praktischen Einsatz ist weiterhin zu bedenken, dass REST keine inhärenten Sicherheitsfeatures wie Verschlüsselung oder Session-Management bietet. Nutzer-Authentifizierung durch Benutzername-/Passwort-Token sowie die Nutzdatenverschlüsselung per Transport Layer Security (TLS) sind möglich, müssen aber separat implementiert werden.

VII. Anwendungsentwicklung unter Contiki OS

Um eine Anwendung für einen Mikrocontroller zu entwickeln, müssen wir uns kurz mit den Eigenarten der Software-Entwicklung für diese Plattform vertraut machen. Aufgrund der sehr eingeschränkten Ressourcen eines Mikrocontrollers findet die Entwicklung nicht direkt auf dem Zielsystem statt. Der Quellcode wird mit Hilfe eines Cross-Compilers auf dem Entwicklungssystem übersetzt und anschließend über ein Programmiergerät auf das Zielsystem geflasht. Einfache Mikrocontroller verfügen weder über eine Speicherverwaltungseinheit noch über ein Dateisystem. Daher werden Anwendung, Betriebssysteme und Treiber in der Regel zu einem einzigen Binärobjekt übersetzt. All diese Komponenten stehen üblicherweise als Quellcode-Dateien zur Verfügung.

Der Contiki Quellcode hat eine feste Ordnerstruktur:

/ Im Contiki-Wurzelverzeichnis befindet sich das globale Makefile `Makefile.include`.

apps Hier befinden sich alle von Contiki bereitgestellten Applikationen. Diese sind nicht standardmäßig aktiviert. Der Quellcode benötigter Apps kann über das Projekt-Makefile mit der Definition `APPS=[Name des App-Ordners]` hinzugefügt werden.

core In diesem Ordner befindet sich das eigentliche Betriebssystem, u. a. auch der Ordner `net`, der den kompletten Quellcode für den Netzwerkstack enthält. Im Ordner `sys` befindet sich der Quellcode für das Prozessmanagement und die Timer.

cpu Hier findet sich der Quellcode für die unterstützenden Prozessoren, also z. B. der Ordner `avr` für ATmega-Prozessoren oder der Ordner `arm` für ARM-Prozessoren. In den jeweiligen Unterordnern sind die Treiber für die unterstützte Peripherie zu finden.

[3] http:\\restlet.com, http:\\pylonsproject.org, u.a.

doc Enthält die Dokumentation sowie Beispielcode für und über Contiki.

examples Dieser Ordner enthält fertige Contiki-Projekte. Er wird typischerweise zum Erstellen neuer Projekte genutzt. Existierende Projekte lassen sich nachnutzen.

platform Enthält alle spezifischen Einstellungen für die verwendete Hardware-Plattform und das verwendete Entwicklungsboard.

tools Sonstige Werkzeuge, die bei der Arbeit mit Contiki hilfreich sind, z. B. der Simulator *cooja* oder Quellcode für den SLIP-Border Router auf der PC-Seite.

Für die Anwendungsentwicklung unter Contiki steht ein ausgeklügeltes System von Makefiles bereit, welches die Details der plattformspezifischen Eigenheiten verbirgt. Es muss daher nur ein einfaches, projektspezifisches Makefile erzeugt werden. Das Makefile des *Hello-world* Beispiels in `~/contiki/examples/hello-world/` sieht folgendermaßen aus:

```
CONTIKI_PROJECT = hello-world
all: $(CONTIKI_PROJECT)
CONTIKI = ../..
include $(CONTIKI)/Makefile.include
```

Zuerst wird der Name der Anwendung spezifiziert. Die CONTIKI-Variable verweist auf das Wurzelverzeichnis des Contiki-Quellcodes. Zum Schluss wird das systemweite Makefile `Makefile.include` eingebunden.

Durch die Angabe eines Targets für `make` kann die Anwendung für verschiedene Hardware-Plattformen übersetzt und bereitgestellt werden. Das Target `minimal-net` erlaubt dabei die Ausführung der Anwendung direkt als Prozess der Entwicklungsumgebung. Dadurch kann die Entwicklung beschleunigt und auf den Einsatz von Software-Debuggern zurückgegriffen werden. Wenn die Anwendung fehlerfrei läuft, kann diese einfach für die Zielplattform (z.B. Atmel Raven) neu übersetzt werden:

```
$ make TARGET=avr-raven
```

VIII. RESTful-Webserver

Das Contiki-Projekt stellt bereits einen REST HTTP-Server zur Verfügung, mit dessen Hilfe wir auf einfachem Weg einen eigenen REST-Service erstellen können. Dazu reicht es am Anfang, die Datei `rest-server-example.c` im Verzeichnis `~/contiki/examples/rest-example` entsprechend anzupassen.

Über das RESOURCE-Makro wird der Name der Ressource, die HTTP-Zugriffsmethode und die URL definiert. Wollen wir zum Beispiel vom REST-Server eine Zufallszahl über den Aufruf der URL `/rand` abfragen, so wird die Ressource wie folgt definiert:

```
RESOURCE(rand, METHOD_GET, "rand");
```

Jede Ressource benötigt weiterhin eine Handler-Funktion, welche die entsprechenden Daten erzeugt oder verarbeitet. Der Name der Handler-Funktion wird nach dem Muster *[resource name]_handler* gebildet.

Die nachfolgend dargestellte Funktion erzeugt beispielhaft ein einfaches JSON-Objekt, welches eine Zufallszahl zwischen 0 und 10 enthält und diese im char-Array `temp` abspeichert. Anschließend wird der Content-Typ des HTTP-Response-Headers gesetzt und die Payload generiert:

```
void
rand_handler(REQUEST* request, RESPONSE* response)
{
  /*Construct a JSON Object containing random numbers*/
  sprintf(temp,"{\n\"test\" : \"%d\"\n}", rand() % 11);

  /*Set the content response header to JSON*/
  rest_set_header_content_type(response, APPLICATION_JSON);
  rest_set_response_payload(response, (uint8_t*)temp,
  strlen(temp));
}
```

Als letzte Aufgabe muss die neu erzeugte Ressource noch im Server-Thread bekanntgegeben werden. Der entsprechende Funktionsaufruf ist im Beispiel dargestellt:

```
PROCESS_THREAD(rest_server_example, ev, data)
{
  ...
  rest_activate_resource(&resource_rand);
}
```

Das passende Makefile für unser Beispiel sieht folgendermaßen aus:

```
all: rest-server-example
CONTIKI=../..
CFLAGS += -DPROJECT_CONF_H=\"project-conf.h\"
CFLAGS += -DWITH_HTTP
APPS += rest-http
CONTIKI_WITH_IPV6 = 1

include $(CONTIKI)/Makefile.include
```

Anschließend kann das Projekt mit

```
$ make TARGET=minimal-net
```

übersetzt und mit

```
$ ./rest-server-example.minimal-net
```

ausgeführt werden.

Der Contiki-Prozess unter Linux erzeugt ein TAP-Interface, um mit der Anwendung zu kommunizieren. Server und Interface verfügen erst einmal nur über link-lokale IPv6-Adressen und der Server erwartet Anfragen auf Port 8080. Um die Anwendung einfacher testen zu können, sollten noch einige Änderungen am Quellcode vorgenommen werden. Zuerst werden in der Datei `~/contiki/apps/rest-http/http-common.h` die Werte der folgenden Makros angepasst:

```
#define HTTP_PORT 80
#define HTTP_DATA_BUFF_SIZE 800
```

Weiterhin verwendet Contiki standardmäßig das Routing-Protokoll RPL [6], um das Netzwerkpräfix an die Netzknoten zu verteilen. Für den Test der Anwendung auf dem Linux-System ist das allerdings eher hinderlich. Daher entfernen wir in der Datei `~/contiki/platform/minimal-net/contiki-conf.h` den Kommentar vor dem Makro:

```
#define HARD_CODED_ADDRESS      "fdfd::"
```

und fügen unmittelbar danach die folgenden beiden Zeilen ein:

```
#undef UIP_CONF_IPV6_RPL
#define UIP_CONF_IPV6_RPL       0
```

Anschließend kann das Projekt erneut übersetzen und ausgeführen werden. Das TAP-Interface wird mit folgendem Befehl konfiguriert:

```
$ sudo ifconfig tap0 add fdfd::1/64
```

Damit ist der REST-Service unter folgender URL erreichbar:

```
http:// [fdfd::206:98ff:fe00:232]/rand
```

Die zuletzt beschriebenen Änderungen sind nur für die Ausführung als nativer Linux-Prozess relevant. Wenn die Anwendung anschließend für eine passende Mikrocontroller-Plattform übersetzt wird, kommt standardmäßig wieder RPL zum Einsatz.

IX. Fazit

Auf Basis des Betriebssystems Contiki lassen sich verteilte Anwendungen nach überschaubarer Einarbeitungsszeit schnell und standardkonform implementieren. Es wurde daher die Entscheidung getroffen eine Mikrocontrollerplattform in die praktischen Übungen im Masterstudiengang 'Kommunikations- und Informationstechnik' der Beuth Hochschule zu integrieren.

Der sichtbare Vorteil gegenüber dem Einsatz von dedizierten, horizontal und vertikal integrierten Lösungen wie z.B. Zigbee besteht in der schnellen prototypischen Entwicklung und Integration eigener Lösungen. Die Toolunterstützung für die Anwendungsentwicklung sowie das Testen und Debuggen von Netzwerkanwendungen ist hervorragend. Hierbei bieten die eingesetzten ASCII Protokolle eine gute Lesbarkeit und Nachverfolgbarkeit auf der Netzwerkebene welches der Vermittlung und der Übertragbarkeit des erworbenen Wissens zuträglich ist.

Durch die Verwendung von Application Frameworks mit REST-Unterstützung, ist eine schnelle prototypische Umsetzung eigener Projektideen möglich, ohne bereits zu Projektbeginn an Hürden binärer Protokolle wie byte-ordering und der Unterstützung durch Programmiersprachen und Protokollparsern zu scheitern.

Literatur

[1] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, ser. LCN '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 455–462.

[2] J. Gutierrez, M. Naeve, E. Callaway, M. Bourgeois, V. Mitter, and B. Heile, "IEEE 802.15.4: A developing standard for low-power low-cost wireless personal area networks," *IEEE Network Magazine*, vol. 15, no. 5, pp. 12–19, September/October 2001.

[3] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," Internet Engineering Task Force, RFC 2460, Dec. 1998. [Online]. Available: http://tools.ietf.org/html/rfc2460

[4] N. Kushalnagar, G. Montenegro, and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals," Internet Engineering Task Force, RFC 4919, Aug. 2007. [Online]. Available: http://tools.ietf.org/html/rfc4919

[5] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," Internet Engineering Task Force, RFC 4944, Sep. 2007. [Online]. Available: http://tools.ietf.org/html/rfc4944

[6] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," Internet Engineering Task Force, RFC 6550, Mar. 2012. [Online]. Available: http://tools.ietf.org/html/rfc6550

20 Jahre Sensornetze - Praxisorientierte Forschung?

Alexander von Bodisco
Hochschule für angewandte Wissenschaften Augsburg
Fakultät für Informatik
Augsburg
Email: alexander.vonbodisco@hs-augsburg.de

***Zusammenfassung*—Sensornetze sind ohne Zweifel eines der am meisten untersuchten Forschungsgebiete der letzten 20 Jahre. Die nahezu unbegrenzten Einsatzgebiete dieser vielversprechenden Technologie, welche sich aus einer großen Anzahl kollaborativ, selbst-organisiert und energieautark arbeitenden Knoten ergeben würden, sind für Wissenschaftler ein sehr attraktives Forschungsumfeld. Während der Hochphase wurde Sensornetzen eine Vielzahl von Eigenschaften zugesprochen, die in der Praxis unter realistischen Bedingungen, leider noch nicht oder nur begrenzt umsetzbar sind. So ist es trotz intensiver Forschung noch ein weiter Weg bis willkürlich positionierte Sensorknoten in großer Anzahl autonom und selbst-organisierend agieren.**

In diesem Artikel wird die Forschung der letzten Jahre im Bereich der Sensornetze einer kritische Betrachtung hinsichtlich ihrer Praxistauglichkeit unterzogen. Ebenfalls wird eine kurze Bilanz der erreichten und nicht-erreichten Ziele gezogen.

I. Einleitung

Sensornetze wurden bereits im letzten Jahrtausend als Schlüsseltechnologie erkannt [1]. Seitdem haben sie Forscher und Entwickler herausgefordert und inspiriert. Die stark limitierten Ressourcen in Form von Rechenleistung, Arbeitsspeicher, Datenrate und Energie einerseits und die große Anzahl an Sensorknoten andererseits. Gerade in den Anfängen der Sensornetze wurden Anwendungen beschrieben, welche den selbst-organisierten Zusammenschluss von mehreren Hunderten oder gar Tausenden von Sensorknoten voraussetzen [2], [3]. Zusätzlich sollen die Knoten energieautark arbeiten oder zumindest eine Laufzeit von mehreren Jahren ohne externe Energieversorgung aufweisen.

Eingeschränkte Ressourcen machten insbesondere in den ersten Jahren eine Optimierung unausweichlich. Zunächst wurden die Schwerpunkte auf die limitierte Hardware sowie auf Kanalzugriffs- und Routingprotokolle gelegt. Schnell wurde die Kommunikation als Hauptenergieverbraucher identifiziert, so dass der erweiterte Fokus auf energieeffiziente Transceiver, Algorithmen und Kommunikationsprotokolle gelegt wurde. Nachdem eine ausreichende Wissensbasis vorhanden war, wurde verstärkt an der Integration von Technologien zur Gewinnung von Energie gearbeitet, um die Lebenszeit von Sensoren zu verlängern oder gar dauerhaften Betrieb zu ermöglichen.

Neue Herausforderungen wurden in den Bereichen mobile Sensornetze, echtzeitfähige Sensornetze, Sensornetze zur Unterwasserkommunikation [4] und für Multimediaanwendungen [5] gesucht und gefunden. Darüber hinaus wurden viele Protokolle und Algorithmen für spezielle Anwendungsszenarien mit unterschiedlichsten Anforderungen im Laufe der Jahre entwickelt. In den letzten Jahren rücken Sensornetze immer mehr als Schlüsseltechnologie für das Internet der Dinge in den Vordergrund. In diesem Zusammenhang findet das wichtige Thema IT-Sicherheit zunehmend an Beachtung.

Während die üblicherweise für Sensorknoten verwendete energiesparende Hardware weitestgehend auf dem gleichen Leistungsniveau geblieben ist, hat sich das konkurrierende Umfeld angeführt durch preiswerte Bastelrechner in Form von Raspberry PI und Arduino rasant weiterentwickelt. Letztere drängen sich verstärkt für rechenintensive Aufgaben auf und können bei entsprechenden Duty Cycle mit einem durchschnittlichen Energieverbrauch im unteren einstelligen Milliwattbereich betrieben werden [6]. Dadurch lassen sich bereits Laufzeiten von ca. einem Jahr erreichen. Auch wenn sich andere Technologien zunehmend der Energieeffizienz von Sensorknoten annähern, bleiben diese auf Grund der speziellen Optimierung weiterhin unerreicht [7], [8].

Für nahezu alle genannten Herausforderungen und Anwendungen wurden Lösungen vorgestellt [2], welche zumindest theoretisch eine akzeptable Performanz erreichen. Gerade im Hinblick auf Skalierbarkeit und Energieeffizienz von Sensornetzen basieren viele Forschungsergebnisse allerdings auf die Realität vereinfachenden Annahmen für Analysen und Simulationen, da ein entsprechend dimensioniertes Testnetz i.d.R. mit zu hohen Kosten verbunden ist. Sensornetze und Testbeds mit mehr als 50 bis 100 Knoten stellen zur Zeit noch eine absolute Ausnahme dar [9]. Selbst wenn ein ausreichend dimensioniertes Testnetz unter Laborbedingungen eine hinreichend gute Performanz verspricht, zeigen sich unter realen Bedingungen oft gravierende unerwartete Schwachstellen und Probleme [10]. Hinzu können Fehler und Ungenauigkeiten bei der statistischen Auswertung kommen, welche die Belastbarkeit der vorgestellten Leistungsdaten schmälern [11].

Welche Ergebnisse können als belastbar angesehen werden und welche Verfahren, Algorithmen, Protokolle und Technologien haben sich in der Praxis bewährt? Welche Anwendungen lassen sich damit realisieren, wo liegen die aktuellen Grenzen und welche Anwendungen können erst in naher oder ferner Zukunft verwirklicht werden? Wie viel Optimierung ist tatsächlich notwendig und wo sollte zu Gunsten der Flexibilität auf Optimierung verzichtet werden, um die leider oft praktizierte Überanpassung für nicht-praxisrelevante Sonderfälle zu vermeiden? In dieser Arbeit wird versucht, einige Antworten zu geben und gleichzeitig aufgerufen, die eigene Forschung kritisch in Bezug auf Praxisrelevanz zu hinterfragen.

Zunächst werden in Abschnitt II Anwendungen und deren Anforderungen an drahtlose Sensornetze diskutiert. Der aktuelle Stand der Technik in den Bereichen der Kanalzugriffs- und Routingprotokolle wird in den Abschnitten III und IV behandelt. Eine kritische Betrachtung des aktuell Möglichen

ist in Abschnitt V zu finden. Die Arbeit schließt mit einem Ausblick in Abschnitt VI.

II. Anwendungen und Anforderungen

Die Anforderungen an Sensornetze hängen überwiegend von der Zielanwendung, der eingesetzten Hardware und den Gegebenheiten des Einsatzortes ab, weshalb bestimmte Eigenschaften unabdingbar sind(siehe Tabelle I).

Tabelle I: Eigenschaften und Anforderungen in Sensornetzen

Eigenschaften	Anforderungen
Limitierte Ressourcen (Rechenleistung, Speicher, Bandbreite, Energie)	Effizienter Umgang mit Ressourcen
Dynamische und extreme Bedingungen am Einsatzort	Adaptive Kommunikations-protokolle
Redundanz der Daten	Datenfusion und lokale Verarbeitung
Unzuverlässige Kommunikation	Robuste, zuverlässige Kommunikationsprotokolle
Keine einzigartige ID der Sensorknoten	Kommunikation mit zentraler Datensenke
Fehleranfällige Hardware	Fehlertolerante Kommunikation
Sensornetze mit hoher Knotenanzahl	Skalierbarkeit der Protokolle, Selbst-konfiguration, Selbstorganisation

Die Kollaboration von Sensorknoten stellt hierbei einen großen Vorteil zu konventionellen Ad Hoc- und Meshnetzen dar. Zunächst beschränkte sich die praxisorientierte Forschung auf verzögerungstolerante Sensornetze zur Überwachung von Wildtieren [12], [13]. Diese aus heutiger Sicht relativ einfach umzusetzenden Aufgaben waren von immenser Bedeutung, um erste Erfahrungen zu sammeln und stellten zum damaligen Zeitpunkt ein technisches Novum dar.

Nach und nach traten andere komplexere Anwendungen zur Überwachung großer Gebiete, z.B. in Katastrophenszenarien, in kritischen Regionen oder auch in der Landwirtschaft [10], in den Vordergrund, welche sich nur mit einer großen Anzahl von kollaborativ arbeitenden Knoten realisieren ließen. Zum einen sollten sich die Knoten selbst organisieren, um ein schnellen und flexiblen Aufbau zu ermöglichen. Zum anderen sollten die Knoten in der Lage sein Daten lokal zu verarbeiten, um wertvolle Bandbreite zu sparen.

Typischerweise ist die Positionierung der Sensorknoten innerhalb eines Einsatzgebietes nur bedingt vorab planbar, da sich die lokalen Gegebenheiten und äußeren Einflüsse, wie z.B. Interferenz, Wetterverhältnisse, Signalausbreitung stets unterscheiden. Dadurch ergeben sich dynamische Bedingungen hinsichtlich der Kommunikation, weshalb adaptive und robuste Kommunikationsprotokolle die Grundvoraussetzung für ein gut funktionierendes Sensornetz sind.

Ebenfalls können sich die Charakteristika eines Sensornetzes stark in Bezug auf Knotendichte und Datenverkehrsaufkommen unterscheiden. Sensornetze zur Strukturüberwachung stark beanspruchter kritischer Bauteile weisen i.d.R. eine extrem hohe Knotendichte und event-basierten Datenverkehr auf, wodurch hohe Anforderungen an die Kollisionsauflösung gestellt werden [14], [15]. Entsprechende Mechanismen zur Kollisionsauflösung können auf verschiedene Arten, wie z.B. Synchronisation oder dem Einsatz von Belegungssignalen, umgesetzt werden. In jedem Fall ist deren Einsatz stets mit einer Reduzierung der Bandbreite verbunden. Ein genauerer Einblick diesen Bereich wird in Abschnitt III gegeben.

Heterogene Hardware (Funkchips und Antennen) führen zu stark asymmetrischen Verbindungen zwischen Sensorknoten, so dass in vielen Fällen nur eine eingeschränkte bidirektionale Kommunikation möglich ist. Dies erschwert Routingprotokollen den Aufbau einer stabilen und zuverlässigen Topologie, welche für die meisten Anwendungen notwendig ist. In Abschnitt IV werden wichtige Aspekte im Bereich Routing zusammengefasst und der aktuelle Stand der Technik aus praxisorientierter Sicht diskutiert.

III. Kanalzugriffsprotokolle

Die limitierten Fähigkeiten heutiger energiesparender Funkchips in Bezug auf Kanalabtastung, Umschaltzeiten und Datenratesowie der hohe Anteil der Funkchips am gesamten Energieverbrauch der Sensorknoten führte zu einer wahren Flut an Kanalzugriffsprotokollen [16]. Die Protokolle verfolgen dabei unterschiedliche Ziele, wie z.B. Senkung des Energieverbrauchs, Erhöhung der Zuverlässigkeit, Erhöhung der Datendurchsatzrate, Minimierung der Kanalzugriffsverzögerung oder prioritäts-basierten Zugriff [17].

A. Carrier Sense Multiple Access (CSMA)

Funkchips für Sensornetze unterstützen typischerweise Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA). Ein Kanal wird dabei als belegt erkannt, wenn die durchschnittliche Signalstärke der letzten acht empfangenen bzw. detektierten Symbole über einem bestimmten Schwellenwert liegt. Problematisch ist diese Vorgehensweise, falls von einem schwachen Signal weniger als 8 Symbole empfangen wurden. In diesem Fall wird ein belegter Kanal fälschlicherweise als frei angenommen. Umgekehrt führt ein zu niedrig angesetzter Schwellenwert dazu, dass für eine Übertragung unkritische Interferenzen als belegter Kanal erkannt werden und somit eine Übertragung verhindern. Alle spezialisierten Protokolle sind auf diese Funktionalität angewiesen, um eine zuverlässige Übertragung zu ermöglichen.

Für den Betreiber eines Sensornetzes stellt sich die Frage, ob der Einsatz eines spezialisierten Protokolls tatsächlich notwendig ist, zumal diese die Komplexität stark erhöht. Insbesondere Protokolle, welche eine Synchronisation der Knoten benötigen, sind hier an erster Stelle zu nennen. Die in Sensorknoten eingesetzten Oszillatoren sind nicht exakt und stark temperaturabhängig, wodurch eine Re-Synchronisation häufig angestoßen werden muss. In der Praxis ist das CSMA-CA Protokoll für die meisten Anwendungen vollkommen ausreichend, auch wenn bei hoher Auslastung deutliche Einschränkungen hinzunehmen sind [18]. Klassische nicht-sicherheitskritische Anwendungen zur Überwachung von Temperatur, Feuchtigkeit, Helligkeit und Bewegung, welche mit einer Rate von weniger als 10 Hz abtasten, benötigen bei einer Knotendichte von weniger als 10 Knoten innerhalb eines Kommunikationsradius unter sonst normalen Bedingungen kein spezialisiertes Protokoll.

B. Spezialisierte Kanalzugriffsprotokolle

Eine Ausnahme stellen Sensornetze für sicherheitskritische Anwendungen [15] dar. In diesem Fall sollten optimierte

Protokolle herangezogen werden, um eine zuverlässige Kommunikation zu gewährleisten. Jedoch gilt auch hier, dass stets Protokolle mit niedriger Komplexität bevorzugt werden sollten. Hochkomplexe, überoptimierte Protokolle erreichen oft nur unter speziellen Laborbedingungen ihr maximales Potential. In der Praxis sind viele Netzeigenschaften unvorhersehbar und/oder nur begrenzt planbar, weshalb weniger spezialisierte Protokolle häufig besser geeignet sind.

Eine weitere Ausnahme bilden Sensornetze, bei denen ein niedriger Energieverbrauch höchste Priorität hat. Um Energie zu sparen, müssen die Funkchips der Sensorknoten so häufig wie möglich abgeschaltet werden. Da zur erfolgreichen Kommunikation beide Knoten ihre Funkchips aktiviert haben müssen, ist eine entsprechende Abstimmung notwendig. Hier ist es ebenfalls möglich, auf Synchronisation zurückzugreifen. Allerdings ist diese Vorgehensweise mit hohem Aufwand und zusätzlicher Komplexität verbunden. Für viele Sensornetze ist bereits eine geringe Datenrate ausreichend. In der Praxis erzielen präambel-basierte Protokolle [19], wie z.B. X-MAC [20] oder BPS-MAC [14], sehr gute Ergebnisse.

IV. Routingprotokolle

Sensornetze schöpfen ihr Potential überwiegend aus der großen Anzahl von kollaborierenden Sensorknoten, wodurch sich große Gebiete überwachen lassen und auch komplexere Aufgaben [21] mit hoher Präzision bewältigt werden können. Neben einem erfolgreichen Kanalzugriff hängt die Leistungsfähigkeit von Sensornetzen vor allem von einer effizienten und stabilen Topologie ab. Oft können Sensorknoten nur an suboptimalen Orten [22], wie z.B. auf dem Boden oder an Wänden auf niedriger Höhe platziert werden, so dass die Signalausbreitung und damit die Signalstärke bei Nachbarknoten sehr eingeschränkt ist. Zusätzlich arbeiten viele Sensorknoten mit Chipantennen, welche keine optimale Abstrahlcharakteristik aufweisen. Interferenzen limitieren die Möglichkeiten der Kommunikation ebenfalls, weshalb die Erstellung und Aufrechterhaltung eine der anspruchsvollsten Aufgaben im Forschungsumfeld der Sensornetze ist.

A. Routingtechniken

Die Anzahl der entwickelten Routingprotokolle und Routingmechanismen [23] übersteigt vermutlich sogar die Anzahl der Kanalzugriffsprotokolle. Ein Großteil der Forschungsgemeinde hat sich diesem Thema angenommen, zumal sich ähnliche Problematiken in Ad Hoc- und Meshnetzen wiederfinden. Routingprotokolle in drahtlosen Sensornetzen verwenden überwiegend reaktive Ansätze, das heißt sie bauen Routen erst bei Bedarf auf. Häufig wird die Kommunikation auf ein Ziel bzw. wenige Ziele beschränkt, wodurch die Komplexität und der Routingoverhead deutlich verringert werden.

Routingprotokolle müssen stets grundlegende Funktionen erfüllen, welche unabhängig von den eingesetzten Techniken sind. Um eine optimale Routingentscheidung treffen zu können, ist eine permanente Überwachung aller relevanten Verbindungen zwischen den Teilnehmern erforderlich. Dabei gilt es in erster Linie, die Signalstärke und Zuverlässigkeit auf einer Verbindung zu überprüfen und zu protokollieren. Neben dem Senden und Empfangen von Routingnachrichten kann zusätzlich Datenverkehr der Nachbarknoten ausgewertet werden, um weitere Informationen bzgl. der Linkqualitäten zu erhalten. Eine Erhöhung der Senderate von Routingnachrichten führt - sofern die vorhandenen Links nicht überlastet werden - zu aktuellerer Information und erlaubt dadurch bessere Routingentscheidungen.

Die notwendige Verteilung von Routingnachrichten zur Messung der Linkgüte stellt in größeren Sensornetzen ein nicht zu unterschätzendes Problem dar. Einerseits sollen möglichst alle Verbindungen zuverlässig überprüft werden. Andererseits möchte man eine Mehrfachüberprüfung pro Durchgang vermeiden, um den Routingoverhead so gering wie möglich zu halten.

B. Hierarchie

Das Problem der Skalierbarkeit tritt vor allem in Sensornetzen mit flachen Hierarchien auf, da in diesem Fall bei einem Broadcast alle Knoten adressiert werden. Alternativ bietet sich eine hierarchische Topologie an, um das Fluten von Nachrichten zu limitieren und Knoten entsprechend ihren Aufgaben zusammenzufassen. Allerdings setzt dieser Ansatz entsprechende Planung voraus. Eine weitere Möglichkeit stellt das sogenannte Clustering dar. Mehrere Teilnehmer bilden sich zu einer Gruppe (Cluster) zusammen und bestimmen den Clusterhead, welcher die Kommunikation mit anderen Clustern übernimmt. Der Clusterhead muss dadurch mehr Aufgaben übernehmen und verstärkt Daten übertragen, wodurch diese Aufgabe mit einem hohen Energieverbrauch einhergeht. Üblicherweise übernimmt ein Knoten die Aufgabe des Clusterheads nur für eine kurze Zeit, um den Energieverbrauch auf alle Knoten innerhalb des Clusters gleichmäßig zu verteilen.

C. Routingmetrik

Im Laufe der Jahre wurde eine Vielzahl an Routingmetriken vorgestellt [24], um die Zuverlässigkeit der Protokolle weiter zu steigern. Die Metriken beziehen überwiegend die Signalstärke, Zuverlässigkeit, Verzögerung und ggf. deren Schwankungen mit ein. Häufig wird auch die verbleibende Energie des empfangenden Knotens in der Metrik mit berücksichtigt. Tatsächlich kommt in der Praxis überwiegend die Hopcount-Metrik zum Einsatz, da sie in vielen Fällen ähnlich gute Routen liefert wie komplexe und kombinierte Metriken [25]. Als Ausnahmen sind an dieser Stelle mobile Sensornetze und Sensornetze mit Fokus auf Energieverbrauch zu nennen. In diesen Fällen kann mit entsprechenden Metriken eine Leistungssteigerung erreicht werden.

V. Stand der Technik

In den letzten Jahren wurden viele interessante Protokolle und Verfahren vorgestellt, um die Leistungsfähigkeit von Sensornetzen weiter zu steigern. Welche Protokolle und Verfahren sind wirklich praxistauglich und was lässt sich mit dem heutigen Wissensstand mit vertretbarem Aufwand erreichen?

A. Hardware

Die eingesetzte Hardware in Testnetzen hat sich in den letzten Jahren nur sehr langsam entwickelt [9]. Der Trend zu hocheffizienten und sehr performanten Einchip-Lösungen wie dem CC3200 [8] ermöglichen hohe Datenraten bei geringem

Stromverbrauch, wodurch der Einsatz von leistungsstarken Routingprotokollen aus dem Ad Hoc-Bereich für Sensornetze möglich wird. Laufzeiten von mehr als einem Jahr sind für viele Anwendungen realistisch geworden.

B. Kanalzugriffsprotokolle

Die Forschungsgemeinde hat in den letzten Jahren einen großen Fundus an Protokollen für nahezu alle erdenklichen Situationen entwickelt. Der limitierende Faktor ist hier insbesondere in der Dauer für das Abtasten des Kanals zu sehen. Das sogenannte Clear Channel Assessment (CCA) Delay beträgt bei den eingesetzten Funkchips typischerweise 96-192μs. Hinzukommt die Turnaround Time, die Umschaltzeit zwischen Senden und Empfangen, während der der Kanal nicht abgetastet werden kann. Beide Faktoren sind maßgeblich für Kollisionen verantwortlich, welche in Sensornetzen durch den ereignis-basierten Datenverkehr verstärkt auftreten. Trotz der großen Anzahl an Protokollen wird in der Regel auf CSMA und präambel-basierte Protokolle zurückgegriffen.

C. Routingprotokolle

Im Bereich Routing wurden unterschiedlichste Verfahren vorgestellt, welche in Simulationen und Analysen Sensornetze mit einer Größe von mehreren Hundert zufällig verteilten (mobilen) Knoten und einem Durchmesser von mehr als 30 Hops hervorragende Ergebnisse versprechen. Die Unzuverlässigkeit der drahtlosen Verbindungen sowie der hohe Bandbreitenverlust durch die wiederholte Weiterleitung von Daten, limitieren in der Praxis den (sinnvollen) Netzdurchmesser.

In vielen Feldstudien werden Sensorknoten im freien Feld mit Sichtverbindung auf ca. ein Meter hohen Pfosten montiert. Dieser Aufbau ist nachvollziehbar, da er systematisch ist und sich gut reproduzieren lässt. Inwieweit ein derartiger Aufbau allerdings realistischen Bedingungen entspricht soll an dieser Stelle unkommentiert bleiben.

Idealerweise ist die nächste Datensenke weniger als vier Hops entfernt. Bei größeren Entfernungen, hinsichtlich der Anzahl der Hops, erzielen auch Protokolle wie das populäre RPL Protokoll [26] nur geringen Durchsatz. RPL kann durchaus als eines der am ausgereiftesten Protokolle im Kontext von energiesparenden Netzen mit unzuverlässigen Verbindungen angesehen werden. Dennoch hat der Standard mit aktuell mehr als 150 Seiten eine bedenkliche Komplexität erreicht.

VI. Ausblick

In den vergangenen Jahren wurde eine umfangreiche Wissensbasis zu nahezu allen wichtigen Bereichen der Sensornetze geschaffen. Mit Hilfe spezialisierter Hardware und Software können für viele Anwendungen geeignete drahtlose Sensornetze angeboten werden. Komplexe, hochspezialisierte Insellösungen sind jedoch nur zu Beginn einer neuen Technologie akzeptabel. Die nächsten Schritte müssen zu einfacheren und flexibleren Sensornetzen führen, um schnell kostengünstige Lösungen für das industrielle und private Umfeld entwickeln zu können. Um dieses Ziel zu erreichen, sollten wir uns wieder an folgendes Zitat erinnern. *Man muss die Dinge so einfach wie möglich machen. Aber nicht einfacher.* [1]

[1] Albert Einstein (1879-1955)

Literatur

[1] Business Week, *21 Ideas for the 21st Century*, pp.78–167, August 1999.

[2] P. Rawat and K. Singh and H. Chaouchi and J. Bonnin, *Wireless Sensor Networks: A Survey on recent Developments and Potential Synergies*, The Journal of Supercomputing, Vol.68, pp.1-48, Springer US, 2014.

[3] I. F. Akyildiz, et al., *A Survey on Sensor Networks*, IEEE Communications Magazine, Vol.40, pp.102-114, 2002.

[4] K. Chen, et al., *A Survey on MAC Protocols for Underwater Wireless Sensor Networks*, IEEE Communications Surveys & Tutorials, Vol.16, pp.1433-1447, 2014.

[5] I. F. Akyildiz and T. Melodia and K. R. Chowdhury, *A Survey on Wireless Multimedia Sensor Networks*, IEEE Wireless Communications Magazine, Vol.14, pp.32-39, 2007.

[6] I. Ramos, Gadgetmakersblog, *http://gadgetmakersblog.com/arduino-power-consumption*, 02.08.2015.

[7] Texas Instruments, *Chipcon CC1310 Ultra-Low Power Sub-1Ghz Wireless MCU*, Datasheet rev. 1.0, http://www.ti.com/,

[8] Texas Instruments, *Chipcon CC3200 SimpleLink Wi-Fi and Internet-of-Things Solution, a Single-Chip Wireless MCU*, Datasheet rev. F, http://www.ti.com/, 2015.

[9] J. Horneber and A. Hergenroder, *A Survey on Testbeds and Experimentation Environments for Wireless Sensor Networks*, IEEE Communications Surveys & Tutorials, Vol.16, pp.1820-1838, 2014.

[10] K. Langendoen and A. Baggio and O. Visser, *Murphy Loves Potatoes: Experiences from a Pilot Sensor Network Deployment in Precision Agriculture*, 20th International Parallel and Distributed Processing Symposium (IPDPS), 2006.

[11] K. Pawlikowski, *Do Not Trust All Simulation Studies of Telecommunication Networks*, ICOIN '03, Vol. 2662, pp. 899-908, 2003.

[12] P. Juang, et al., *Energy-efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet*, SIGARCH Comput. Archit. News, Vol.30, pp.96-107, ACM, 2002.

[13] A. Mainwaring, et al., *Wireless Sensor Networks for Habitat Monitoring*, Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications, pp.88-97, ACM, 2002.

[14] A. Klein, *BPS-MAC: Backoff Preamble based MAC Protocol with Sequential Contention Resolution*, Lecture Notes in Computer Science (LNCS), Vol.6886, pp.39-50, Springer-Verlag Berlin Heidelberg, 2011.

[15] P. Suriyachai and U. Rödig and A. Scott, *A Survey of MAC Protocols for Mission-Critical Applications in Wireless Sensor Networks*, IEEE Communications Surveys Tutorials, Vol.14, No.2, pp.240-264, 2012.

[16] K. Langedoen, *The MAC Alphabet Soup served in Wireless Sensor Networks*, http://www.st.ewi.tudelft.nl/ koen/MACsoup/, 30.07.2015.

[17] I. Demirkol and C. Ersoy and F. Alagoz, *MAC Protocols for Wireless Sensor Networks: A Survey*, IEEE Communications Magazine, Vol.44, pp.115-121, 2006.

[18] C. Antonopoulos, et al., *CSMA-MAC Performance Evaluation for WSN Applications*, SENSORCOMM '09, pp.13-18, 2009.

[19] C. Cano, et al., *Low Energy Operation in WSNs: A Survey of Preamble Sampling MAC protocols*, Computer Networks, Vol.55, pp.3351-3363, 2011.

[20] M. Büttner, et al., *X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks*, SenSys '06, pp.307-320, ACM, 2006.

[21] G. Simon, et al., *Sensor Network-based Countersniper System*, Proceedings of the 2nd Int. Conference on Embedded Networked Sensor Systems, pp. 1–12., ACM, 2004.

[22] G. Werner-Allen, et al., *Deploying a Wireless Sensor Network on an Active Volcano*, IEEE Internet Computing, No.2, Vol.10, pp.18–25, 2006.

[23] J. N. Al-Karaki and A. E. Kamal, *Routing Techniques in Wireless Sensor Networks: A Survey*, IEEE Wireless Communications, Vol.11, pp.6-28, 2004.

[24] R. Baumann, et al., *A Survey on Routing Metrics*, Computer Engineering and Networks Laboratory, ETH Zürich, 2007.

[25] R. Draves and J. Padhye and B. Zill, *Comparison of Routing Metrics for Static Multi-Hop Wireless Networks*, SIGCOMM '04, pp.133-144, 2004.

[26] T. Winter, et al., *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*, IETF RFC6550, 2012.

Evaluation der Kryptografiebibliothek NaCl im Kontext drahtloser Sensornetze

Markus Jung
Institut für Telematik
Karlsruher Institut für Technologie
E-Mail: markus.jung@kit.edu

Anton Hergenröder*
E-Mail: anton.hergenroeder@kit.edu

Katharina Männle
Karlsruher Institut für Technologie
E-Mail: katharina.maennle@student.kit.edu

***Zusammenfassung*—Als Teil des Internet of Things erfassen, verarbeiten und kommunizieren drahtlose Sensorknoten potentiell sensible und schutzbedürftige Daten. Klassische Schutzziele wie Vertraulichkeit, Integrität und Authentizität können durch entsprechende kryptographische Primitive gewährleistet werden. In dieser Arbeit untersuchen wir die nach aktuellem Stand der Forschung als modern und sicher geltenden Primitive der Kryptographiebibliothek NaCl auf ihre Eignung für den Einsatz in drahtlosen Sensornetzen. Unsere Ergebnisse zeigen, dass insbesondere die authentifizierte Verschlüsselung mit NaCl deutlich effizienter als eine vergleichbare Kombination von AES-128-ECB und SHA-256 ist. Es war aber auch festzustellen, dass die Bibliothek verhältnismäßig viel Programmspeicher benötigt.**

I. Einleitung

Im *Internet of Things* spielen drahtlose Sensorknoten eine wichtige Rolle. Als Datenquelle sind sie ausgestattet mit Sensoren zur Erfassung ihrer Umgebung, bilden drahtlos kommunizierend Netzwerke und werden durch Batterien mit Energie versorgt. Dabei unterliegen sie starken Ressourcenbeschränkungen und stellen so eine besondere Herausforderung für die Entwicklung von Anwendungen und Protokollen für drahtlose Sensornetze dar. Insbesondere der energieeffizienten Kommunikation wird in diesem Kontext eine wichtige Bedeutung beigemessen.

Der Austausch von oftmals sensiblen Daten erfordert die Gewährleistung von Kommunikationssicherheit, gleiches gilt auch für Aspekte der persönlichen Sicherheit (Safety). Da klassische Sicherheitsmechanismen bisher den Aspekt der Energieeffizienz ungenügend berücksichtigen, ist es essentiell Wege zur energieeffizienten Realisierung von Kommunikationssicherheit unter den gegebenen Ressourceneinschränkungen zu erforschen.

Es existieren bereits einige für Mikrocontroller optimierte Kryptographiebibliotheken, welche eine Sammlung von kryptographischen Primitiven bereitstellen. Die im Bereich von drahtlosen Sensornetzen noch nicht untersuchte Bibliothek *NaCl* (ausgesprochen *Salt*) [1] verfolgt dagegen einen anderen Ansatz. Um zu vermeiden, dass einzelne sichere Primitive durch eine ungünstige Kombination zu einem unsicheren Gesamtsystem kombiniert werden, bietet NaCl unter anderem eine anwenderfreundliche Schnittstelle, welche eine authentifizierte Verschlüsselung auf sichere Weise realisiert.

*Bei Entstehung dieser Arbeit tätig am Institut für Telematik des Karlsruher Instituts für Technologie.

In dieser Arbeit wird der Einsatz der Kryptographiebibliothek NaCl für die sichere Kommunikation in Sensornetzen evaluiert. Hierzu wurde eine für Atmel AVR Mikrocontroller portierte Version von NaCl namens *AVRNaCl* [2], in *TinyOS*, einem Betriebssystem für drahtlose Sensorknoten, eingebunden und für die Verschlüsselung und Authentifizierung von Nachrichten verwendet. Als Sensornetzplattform kamen die weit verbreiteten *MICAz*-Sensorknoten zum Einsatz. Die Performanz und Energieeffizienz von NaCl wurde mit dem realitätsnahen Simulator *Avrora+* [3] evaluiert. Weiterhin wurden Experimente mit den Kryptographiebibliotheken *TinyECC* und *AVR-Crypto-Lib* durchgeführt und die jeweiligen Ergebnisse zum Vergleich mit NaCl herangezogen.

II. Grundlagen

Sichere Kommunikation in drahtlosen Sensornetzen stellt, speziell vor dem Hintergrund der beschränkten Ressourcen, eine große Herausforderung dar. Insbesondere asymmetrische Kryptoverfahren sind sehr rechen- und damit auch energieintensiv. Eine mögliche Maßnahme zur Steigerung der Energieeffizienz ist die Auswahl geeigneter Algorithmen und optimierter Implementierungen. Auch Verwendung dedizierter Kryptoprozessoren [4] kommt dabei in Betracht.

Die AVR-Crypto-Lib [5] stellt kompakte Implementierungen von kryptographischen Funktionen für Atmel AVR Mikrocontroller zur Verfügung. Der Schwerpunkt der Bibliothek liegt dabei auf symmetrischen Kryptoverfahren wie DES oder AES, weiterhin umfasst sie *Keyed-Hash Message Authentication Codes* (*HMAC*) wie beispielsweise HMAC-SHA-256.

Für die asymmetrische Kryptographie auf Basis elliptischer Kurven existiert mit TinyECC [6] eine für Sensorknoten geeignete Bibliothek. Sie implementiert unter anderem das *Elliptic Curve Integrated Encryption Scheme* (*ECIES*), ein hybrides Verfahren. Dieses nutzt zum Austausch eines gemeinsamen Geheimnisses elliptische Kurven, die eigentliche Verschlüsselung erfolgt symmetrisch.

Die Bibliothek NaCl [1] bietet sowohl auf symmetrischer als auch auf asymmetrischer Kryptographie basierende Verschlüsselungs- und Signaturverfahren zur sicheren Netzwerkkomunikation. Bei ihrer Implementierung wurde großer Wert auf Sicherheit, Effizienz und Performanz gelegt. Insbesondere wurde darauf geachtet, Kontroll- und Datenflussabhängigkeiten zu geheimen Daten zu vermeiden um so einen Schutz vor Timing-Angriffen zu gewährleisten.

Tabelle I. ZUR EVALUATION VERWENDETE BIBLIOTHEKS- UND ANWENDUNGSVERSIONEN

µNaCl / AVRNaCl	20140813
TinyECC	2.0
AVR-Crypto-Lib	SVN rev. 5989
avr-gcc	4.8.1
avr-libc	1.8.0
TinyOS	2.1.2
Avrora+	1.7.117

A. Die NaCl Programmierschnittstelle

Mit `crypto_box` und `crypto_secretbox` bietet NaCl zwei Schnittstellen zur Absicherung der am häufigsten in der Netzwerkkommunikation geforderten Schutzziele: Vertraulichkeit, Integrität und Authentizität [7]. Beide kapseln eine vorgegebene, sichere Kombination an Primitiven, die eine authentifizierte Verschlüsselung (*Authenticated Encryption with Associated Data, AEAD*) bereitstellt. Implementierungsfehler wie eine unsichere Komposition kryptographischer Primitive von Seiten des Anwenders sind damit ausgeschlossen.

Die `crypto_secretbox` basiert auf symmetrischer Kryptographie und setzt daher voraus, dass beide kommunizierende Parteien einen gemeinsamen geheimen Schlüssel teilen. Sie kombiniert die Stromchiffre *Salsa20* [8] mit dem MAC *Poly1305* [9].

Ein asymmetrisches Kryptosystem wird durch die `crypto_box` realisiert. Sie beruht auf einem hybriden Verfahren bei dem die eigentliche Verschlüsselung von Daten ebenfalls durch die `crypto_secretbox` erfolgt. Der dazu erforderliche symmetrische Schlüssel wird mit dem *Elliptic-Curve Diffie-Hellmann Schlüsselaustauschverfahren* (*ECDH*) erzeugt. Dieses operiert bei NaCl auf der elliptischen Kurve *Curve25519* [10], welche als schnell und sicher gilt.

B. µNaCl / AVRNaCl

Das Projekt *µNaCl* [2] stellt eine Portierung von NaCl für Mikrocontroller bereit. Bisher unterstützt werden Atmel AVR Mikrocontroller der ATmega-Familie, Implementierungen für weitere Architekturen sind in Arbeit. µNaCl gibt es in zwei verschiedenen Varianten, die auf Geschwindigkeit respektive geringen Speicherbedarf hin optimiert sind.

III. EVALUATION

Im Rahmen der Evaluation wird die Implementierung von NaCl hinsichtlich der Energieeffizienz, der erforderlichen Rechenzeit und dem Speicherbedarf untersucht. Ein wesentlicher Augenmerk liegt dabei auf einem Vergleich mit den Bibliotheken TinyECC und AVR-Crypto-Lib. Die verwendeten Versionen sind in Tabelle I aufgeführt

A. Methodik

Zur Evaluation der Bibliotheken wurden TinyOS-Anwendungen implementiert, welche die zu untersuchenden kryptographischen Operationen verwenden. Bei AVRNaCl handelte es sich dabei um die auf Geschwindigkeit optimierten Funktionen. Als Hardwareplattform wurden MICAz-Sensorknoten gewählt. Um äußere Störeinflüsse auszuschließen und identische Vergleichsbedingungen zu schaffen, erfolgte die Evaluation im Simulator Avrora+ [3]. Dieser erzielt durch die Emulation der einzelnen Bestandteile drahtloser Sensorknoten realitätsnahe Ergebnisse. Insbesondere entspricht das Laufzeitverhalten des emulierten Prozessorkerns dem echter Hardware.

Tabelle II. AVRORA+ MODELLPARAMETER FÜR MICAZ

Parameter	Wert
Taktfrequenz	7,3728 MHz
Versorgungsspannung	3,309 V
Stromaufnahme	9,357 mA

Tabelle III. RECHENAUFWAND FÜR DIE VER- UND ENTSCHLÜSSELUNG MIT `CRYPTO_SECRETBOX` (SYMMETRISCH) UND `CRYPTO_BOX` (ASYMMETRISCH)

Verfahren	Anzahl Zyklen	Dauer	Energiebedarf
Sym. Verschlüsseln	92'563	0,013 s	0,5 mJ
Sym. Entschlüsseln	126'972	0,017 s	0,7 mJ
Asym. Verschlüsseln	22'904'034	3,105 s	125,6 mJ
Asym. Entschlüsseln	22'938'445	3,110 s	125,8 mJ

Für jede der Anwendungen wurde die zur Ausführung der einzelnen Operationen erforderlichen Zyklenzahl erfasst. Aus dieser können, aufgrund der Architektur des bei MICAz eingesetzten Atmel AVR Mikrocontrollers, mit den in Tabelle II angegebenen Modellparametern die entsprechenden Werte für Rechenzeit und Energiebedarf direkt ermittelt werden.

B. crypto_secretbox und crypto_box im Vergleich

Für die Gegenüberstellung von `crypto_secretbox` und `crypto_box` wurde eine Nachricht von 128 Byte verbeziehungsweise entschlüsselt. Zum Vergleich: Die maximale Größe einer IEEE 802.15.4-Dateneinheit beträgt 127 Byte.

Wie die Ergebnisse in Tabelle III zeigen, führt die mit asymmetrischer Kryptographie realisierte `crypto_box` deutlich aufwändigere Berechnungen durch als `crypto_secretbox`. Da beide Nutzungsmodi die Daten nach dem gleichen Verfahren verschlüsseln, ergibt sich der Unterschied von über Faktor 200 aus dem bei `crypto_box` genutzten Schlüsselaustauschverfahren auf Basis elliptischer Kurven.

C. Vergleich der symmetrischen kryptographischen Primitive aus NaCl

Neben der authentifizierten Verschlüsselung mit der `crypto_secretbox` ermöglicht NaCl auch den direkten Zugriff auf die einzelnen Primitive. Zur Verschlüsselung kommt Salsa20 zum Einsatz, als Hashfunktion wird SHA-512 genutzt. Zum Vergleich der einzelnen Funktionen untereinander und zur Untersuchung des Skalierungsverhaltens wurden diese auf verschiedene Datenblöcke von 8 bis 256 Byte angewandt. Grafik 1 zeigt die Taktzyklen, die zur Bildung eines SHA-512 Hashwerts, eines HMAC-SHA-512-256 Authentifikators sowie zu dessen Prüfung erforderlich waren. Weiterhin dargestellt ist der Berechnungsaufwand für die Ver- und Entschlüsselung mit Salsa20 sowie mit der `crypto_secretbox`.

Auffällig ist zum einen der große Geschwindigkeitsunterschied zwischen den SHA-512-basierten Funktionen und der mit Salsa20 und Poly1305 arbeitenden `crypto_secretbox`. Obwohl letztere Authentifizierung und Verschlüsselung bietet benötigt sie dafür eine deutlich

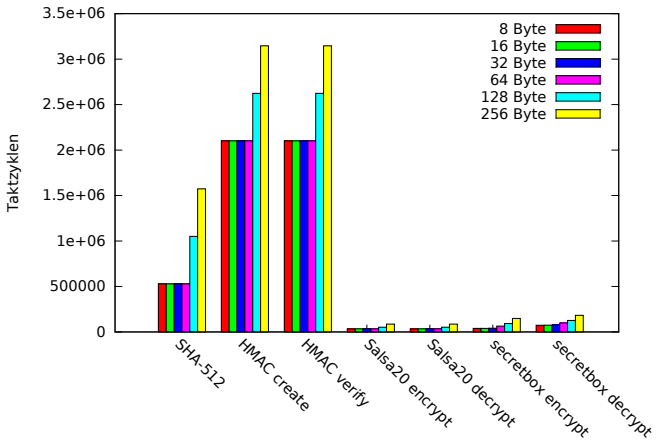


Abbildung 1. Skalierungsverhalten verschiedener symmetrischer kryptographischer Primitive aus NaCl

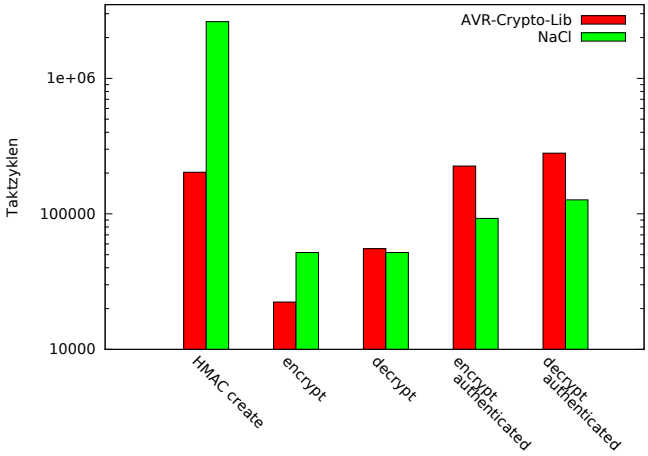


Abbildung 2. Rechenaufwand für die symmetrische Verschlüsselung von 128 Byte mit NaCl und der AVR-Crypto-Lib

geringere Anzahl an Taktzyklen und ist damit für den Einsatz in drahtlosen Sensornetzen besser geeignet. Weiterhin zeigt sich, dass die `crypto_secretbox` gut mit der Nachrichtenlänge skaliert. Der Berechnungsaufwand für die auf SHA-512 aufbauenden Funktionen steigt, durch die Arbeitsweise der Hashfunktion, in diskreten Stufen. SHA-512 verarbeitet Eingaben in Blöcken von je 1024 Bit. Aufgrund von Padding-Mechanismen erfolgt der Sprung von einem zu zwei Blöcken, mit einer entsprechenden Zunahme der Berechnungszeit, nicht erst bei einer Nachrichtengröße von 256 Byte sondern bereits bei 128 Byte. Diese Eigenschaft ist auch anhand der Graphen nachzuvollziehen.

D. Vergleich mit der AVR-Crypto-Lib

Im Folgenden werden SHA-256 als HMAC und AES-128-ECB als Verschlüsselungsverfahren aus der AVR-Crypto-Lib mit dem HMAC-SHA-512-256 und Salsa20 aus NaCl verglichen. Die `crypto_secretbox` aus NaCl realisiert eine authentifizierte Verschlüsselung. Um die gleichen Schutzziele mit der AVR-Crypto-Lib zu erreichen wurde AES mit dem HMAC-SHA256 kombiniert (*Encrypt-then-MAC*). Verarbeitet wurden jeweils Nachrichten mit einer Länge von 128 Byte.

Die für die verschiedenen Operationen benötigte Anzahl an Taktzyklen ist in Abbildung 2 dargestellt. Auffällig ist ein großer Geschwindigkeitsunterschied im Zusammenhang mit der HMAC-Implementierung in NaCl. Ursächlich dafür ist die Nutzung von SHA-512 als zugrundeliegende kryptographische Hashfunktion. HMAC-SHA512-256 verwirft eine Hälfte des Berechnungsergebnisses und entspricht damit, bei deutlich höherem Rechenaufwand, dem Sicherheitsniveau von HMAC-SHA-256. Auch Salsa20 schneidet bei der Verschlüsselung gegenüber AES-128-EBC schlechter ab und benötigt ungefähr die doppelte Berechnungszeit. Die Entschlüsselung mit AES erfolgt aber deutlich langsamer als die Verschlüsselung [11], Salsa20 zeigt dieses Verhalten nicht und liegt hier gleichauf. Bei der authentifizierten Ver- und Entschlüsselung ist die `crypto_secretbox` von NaCl im Vorteil, der zusätzliche Authentifikator ist mit Poly1305 bei nur geringem Mehraufwand zu berechnen.

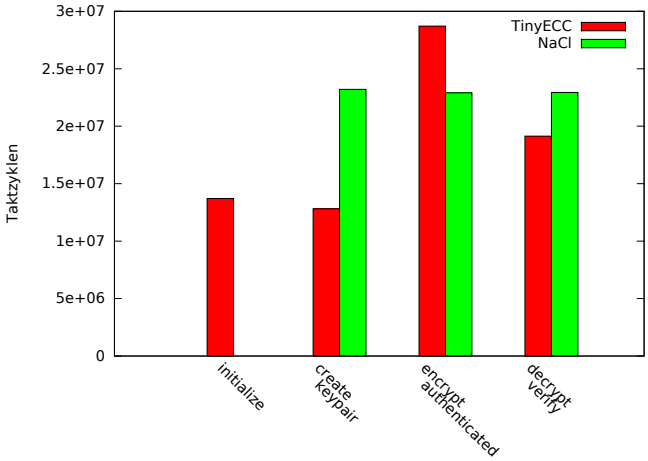


Abbildung 3. Rechenaufwand für die asymmetrischer Verschlüsselung von 128 Byte mit NaCl und TinyECC

E. Vergleich mit TinyECC

Für den Vergleich mit der `crypto_box` wurde die ECIES-Implementierung von TinyECC herangezogen. Diese arbeitet ähnlich wie die `crypto_box` in NaCl mit einem hybriden Schema und entspricht durch die Nutzung des HMAC-SHA1 dem Schutzniveau der durch NaCl realisierten authentifizierten Verschlüsselung. Für den Vergleich wurden Nachrichten mit einer Länge von 128 Byte verarbeitet.

In Abbildung 3 dargestellt ist der Berechnungsaufwand zur Initialisierung der Bibliothek, ein Schritt der nur bei TinyECC einmalig erfolgt. Weiterhin zeigt die Grafik die Anzahl der Taktzyklen, welche zum Erzeugen des Schlüsselpaares beziehungsweise zum Ver- und Entschlüsseln einer Nachricht erforderlich sind. Im direkten Vergleich sind die Unterschiede zwischen TinyECC und NaCl geringer ausgeprägt als bei NaCl und der AVR-Crypto-Lib. Der Aufwand für Initialisierung der Bibliothek und Erzeugung des Schlüsselpaares ist bei TinyECC rund 15% höher als bei NaCl. Auch die Verschlüsselung benötigt ungefähr 25% mehr Rechenzeit, im Gegensatz dazu ist der Aufwand zur Entschlüsselung und Authentifizierung bei NaCl aber knapp 20% größer.

Tabelle IV. SPEICHERBEDARF VON AVR-CRYPTO-LIB, NACL UND TINYECC

Bibliothek	Programmspeicher	Arbeitsspeicher
NaCl small	22 948 Byte	1613 Byte
Nacl fast	31 142 Byte	1485 Byte
AVR-Crypto-Lib	5482 Byte	1831 Byte
TinyECC	18 850 Byte	2249 Byte

F. Speicherbedarf der drei Bibliotheken

Eine Testanwendung zur Ver- und anschließenden Entschlüsselung einer Nachricht von 128 Byte wurde mit allen drei Bibliotheken implementiert. Dabei ergab sich der in Tabelle IV aufgeführte Bedarf an Programm- und Arbeitsspeicher. Letztere Angabe bezieht sich auf den statisch allozierten Speicher, zur Laufzeit werden weitere Variablen dynamisch auf dem Stack abgelegt. Bei NaCl wurden sowohl die `crypto_box` als auch die `crypto_secretbox` in der Testanwendung aufgerufen, der Unterschied zur getrennten Betrachtung war vernachlässigbar gering.

Im Vergleich mit den anderen Bibliotheken fällt auf, dass NaCl einen etwas geringeren Arbeitsspeicherbedarf hat, aber mehr Programmspeicher benötigt. Gegenüber der AVR-Crypto-Lib belegt NaCl annähernd die sechsfache Menge an Programmspeicher, der Unterschied zu TinyECC ist vergleichsweise gering.

IV. ZUSAMMENFASSUNG UND AUSBLICK

Die große Stärke von NaCl ist, von Implementierungsaspekten wie der gezielten Vermeidung von Seitenkanälen abgesehen, vor allem die hohen Geschwindigkeit bei der authentifizierten symmetrischen Verschlüsselung. Insbesondere benötigt diese weniger Zeit als die bloße Berechnung einer HMAC-SHA-256 mit der AVR-Crypto-Lib. Gegenüber TinyECC gestalteten sich die Ergebnisse ausgeglichen. Gleichzeitig ist aber auch festzustellen, dass NaCl deutlich mehr Programmspeicher erfordert als die beiden anderen betrachteten Bibliotheken.

Gegenwärtig befindet sich mit *ChaCha20* [12] eine weiterentwickelte Form von Salsa20 im Standardisierungsprozess als Verschlüsselungsverfahren für das Protokoll *Transport Layer Security* (*TLS*). Auch für Curve25519 gibt es entsprechende Bemühungen. Ferner ist für TLS 1.3 vorgesehen, nur noch Chiffren mit authentifizierter Verschlüsselung zuzulassen. Im Hinblick auf die Evaluationsergebnisse erscheinen Salsa20 (beziehungsweise ChaCha20 als Nachfolger) und Poly1305 für TLS 1.3 als gut geeignet.

TLS ist von einem zuverlässigen Transportprotokoll abhängig. Für den Einsatz in drahtlosen Sensornetzen ist *Datagram Transport Layer Security* (*DTLS*) daher besser geeignet. Beide Protokolle sehen eine Vielzahl von Verschlüsselungs- und Signaturverfahren vor, welche aber teilweise veraltet sind und nicht gezielt für Sensorknoten ausgewählt wurden. Die Arbeitsgruppe *DTLS in Constrained Environments* (*DICE*) der *Internet Engineering Taskforce* (*IETF*) beschäftigt sich daher unter anderem mit der Spezifikation eines auf ressourcenbeschränkte Systeme zugeschnittenen DTLS-Profils.

Über DICE hinaus geht das Konzept der Delegation aufwändiger Operationen, insbesondere des Schlüsselaustauschs, an leistungsstärkere Systeme im Netz, welches aktuell im Rahmen der IETF Arbeitsgruppe *Authentication and Authorization for Constrained Environments* (*ACE*) verfolgt wird. Inwiefern solche Ansätze tatsächlich den Energiebedarf verringern können ist unklar. Frühere Arbeiten zeigten, dass etwa beim Schlüsselaustauschprotokoll *Rich Uncle* die Wahl effizienter kryptographischer Verfahren wirkungsvoller als die Delegation von Berechnungen war [13].

Ausgehend von den Evaluationsergebnissen erscheint NaCl als gut geeignet für drahtlose Sensornetze. Für weitergehende Untersuchungen im Kontext von DTLS und DCAF sollte NaCl daher in eine DTLS-Implementierung integriert werden. Dabei stellt sich insbesondere die Frage, inwiefern DCAF den Energiebedarf von Sensorknoten zu reduzieren vermag und wie die Abwägung zwischen Kommunikations- und Rechenaufwand durch energieeffiziente Kryptoverfahren beeinflusst wird.

Danksagungen

Teile dieser Arbeit entstanden im Rahmen des, durch das Bundesministerium für Bildung und Forschung geförderten, Kompetenzzentrums für angewandte Sicherheitstechnologie (KASTEL, BMBF Förderkennzeichen 01BY1172)

LITERATUR

[1] D. J. Bernstein, T. Lange, und P. Schwabe, "The security impact of a new cryptographic library," in *Progress in Cryptology–LATINCRYPT 2012*. Springer, 2012, pp. 159–176.

[2] M. Hutter und P. Schwabe, "NaCl on 8-Bit AVR Microcontrollers," in *AFRICACRYPT*. Springer, 2013, pp. 156–172.

[3] C. Haas, J. Wilke, und V. Stöhr, "Realistic simulation of energy consumption in wireless sensor networks," in *Wireless Sensor Networks*. Springer, 2012, pp. 82–97.

[4] C. Haas, S. Munz, J. Wilke, und A. Hergenröder, "Evaluating Energy-Efficiency of Hardware-based Security Mechanisms," in *Proceedings of the 9th IEEE International Conference on Pervasive Computing and Communications Workshops (PerSeNS)*. IEEE, Mär. 2013, pp. 560–565.

[5] das labor, "Avr-crypto-lib." [Online]. Abzurufen unter: http://avrcryptolib.das-labor.org

[6] A. Liu und P. Ning, "TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks," in *Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, ser. IPSN '08. IEEE Computer Society, 2008, pp. 245–256.

[7] D. J. Bernstein, "Cryptography in NaCl," Mar. 2009.

[8] D. J. Bernstein, "The Salsa20 Family of Stream Ciphers," in *New Stream Cipher Designs*, ser. Lecture Notes in Computer Science, M. Robshaw und O. Billet, Eds. Springer, 2008, vol. 4986, pp. 84–97.

[9] D. J. Bernstein, "The Poly1305-AES Message-authentication Code," in *Proceedings of the 12th International Conference on Fast Software Encryption*, ser. FSE'05. Springer-Verlag, 2005, pp. 32–49.

[10] D. J. Bernstein, "Curve25519: New Diffie-Hellman Speed Records," in *Public Key Cryptography - PKC 2006*, ser. Lecture Notes in Computer Science, M. Yung, Y. Dodis, A. Kiayias, und T. Malkin, Eds. Springer, 2006, vol. 3958, pp. 207–228.

[11] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, und N. Ferguson, "Performance comparison of the AES submissions," in *Second AES Candidate Conference*. National Institute of Standards and Technology, 1999, pp. 15–34.

[12] D. J. Bernstein, "ChaCha, a variant of Salsa20," January 2008.

[13] C. Haas, J. Wilke, und F. Knittel, "Evaluating the Energy-Efficiency of the Rich Uncle Key Exchange Protocol in WSNs," in *Proceedings of the 38th IEEE Conference on Local Computer Networks (LCN)*. IEEE, Okt. 2013.

Design Space of Smart Home Networks from a Security Perspective

Philipp Morgner and Zinaida Benenson
University of Erlangen-Nuremberg, Germany
{philipp.morgner, zinaida.benenson}@fau.de

Christian Müller and Frederik Armknecht
University of Mannheim, Germany
{christian.mueller, armknecht}@uni-mannheim.de

***Abstract*—Smart homes are an important concept within the Internet of Things and describe households where a network connects multiple domestic devices and sensors. These networks can be managed via control panels, remote controls, or smartphone applications. Security mechanisms ensure that only authorized entities control and manage these networks. A rigorous analysis of security features in smart home networks is especially important in order to provide high security assurance for the residents. A necessary precondition for this kind of analysis is a formalized security model, consisting of a system model, an adversary model, and security goals. In this paper, we investigate the design space of smart home networks from a security perspective, identifying twelve dimensions to classify existing smart home systems. This design space constitutes an important part of the foundation for a comprehensive security analysis.**

I. Introduction

The basic idea of smart homes is to support and enhance the living in domestic environments, to allow remote control of household gadgets, as well as to save energy costs. The term 'smart home network' describes the network of interconnected household devices (smart objects) that communicate with each other, and can be controlled remotely, for example, via smartphone applications. A smart object is defined as an embedded system consisting of a physical entity and an embedded processor [1].

Wireless sensor networks (WSNs) are considered as a major technology within smart home networks, because they provide sensory interfaces to measure physical states in smart homes. The environmental information gathered by these sensory interfaces is essential for the functionality of a wide range of smart home applications, such as regulating the heating, or turning on the lights after sunset. In the next decades, a spreading deployment of smart home networks is predicted, with applications such as alarm systems, smart metering, or support systems for elderly and disabled people [2].

Security mechanisms of these applications must ensure that only authorized entities control and manage smart objects, as well as that only authorized smart objects communicate with each other and have access to the gathered information. Despite the enormous relevance of this topic, till today a comprehensive framework for a sound analysis of security features in smart home networks is missing. Instead, there exists only a small number of security frameworks for WSNs that provide rudimentary and informal security analysis, as outlined in Section II. In particular, these security frameworks for WSNs neither allow to compare nor to securely combine several schemes.

The aim of our research is to close this gap between deployed security mechanisms and their formal verification. In this paper, we introduce a design space as a first step towards a formal security model for smart home networks, thus building the foundation for a comprehensive security framework. A formal security framework would allow us to rigorously investigate security properties of real-world smart home networks, e.g., ZigBee [3] or Z-Wave [4], and to develop provably secure solutions for the most important security goals.

This paper is organized as follows: In Section II, we outline the related work. In Section III we discuss first thoughts on a security model that comprises an attacker model and security goals. Then, we propose a categorization of the design space for smart home networks from a security perspective in Section IV, divided into network-related dimensions (Section IV-A) and node-related dimensions (Section IV-B). We conclude this paper in Section V.

II. Related Work

Römer and Mattern [5] present a design space of WSNs according to twelve dimensions. Our design space for smart homes follows their methodology.

To the best of our knowledge, neither a formal concept of a security model nor a sound security framework has been published in the domain of smart home networks yet. However, there exists a number of security frameworks for WSNs. Most of these security frameworks focus on key management and key distribution (cf. [6], [7]), and provide no concept for a comprehensive security analysis.

III. Security Model

Security is a factor of high importance in smart home networks. Without security mechanisms, a system is vulnerable against a wide range of critical attacks, which compromise the security goals of the smart home network partially or even entirely. In this context, general security goals are:

1) Confidentiality: Shared information in smart home networks are only disclosed to authorized entities.

2) Integrity: Data shared between multiple entities of the network has not been modified during transport or in storage.

3) Availability: All smart objects of the smart home network are available to offer their services, even under critical environmental conditions.

4) Authenticity: The originator of data is authentic and can be verified by other authorized entities in the smart home network.

More concretely, numerous security-critical operations are performed in smart home networks. For instance, the distribution and management of cryptographic keys for encrypted communication or device authentication. Further security-critical operations are secure multicast and broadcast of control messages and secure data aggregation. These operations can be considered as advanced security goals.

Besides security goals, we categorize potential attackers based on [8] according to their capabilities: First, the external adversary is an entity which may have physical access to the communication medium or smart objects of the smart home network, although it is not authorized to participate in the network. Second, an internal adversary, i.e., the user, owns the smart home network and has complete physical and authorized access to each smart object. Essentially, a user exceeds an external adversary's capabilities as a *legitimate* participant of the system. Third, the manufacturer who produces the smart objects is a powerful entity with deep knowledge of the system in general, especially of possible device secrets such as keys. Even worse, a malicious manufacturer could build and backdoors into the system and exploit these later, e.g., to collect data about the user.

To achieve comprehensive security, one needs to protect against all these types of adversaries. However, this may not be sufficient. Besides each attacker entity acting alone, different parties may collaborate. Note that we omit the case in which an internal adversary collaborates with an external adversary due to its exceeding capabilities. A collaboration between the manufacturer and an external adversary enables the manufacturer to extend its capabilities according to the external adversary, e.g., the manufacturer is able to gain physical access to the communication medium. Otherwise, a collaboration between the internal adversary and the manufacturer extends the user's knowledge of the system and capabilities, e.g., the user may pose as a representative of the manufacturer but has additionally physical access to the devices.

IV. Design Space from a Security Perspective

This section explores several dimensions in the design space of the smart home networks from a security perspective. In Section IV-A, we present network properties, while we describe node properties in Section IV-B. The design space provides a solid foundation to classify smart home networks but does not claim completeness. We assume that further dimensions of smart home networks can be discussed from a security perspective.

A. Network-related Dimensions

This section describes dimensions related to properties that apply to a whole smart home network.

1) Deployment: Deployment can be carried out as a one-time activity and deployment as an extensible process. In the first case, all smart objects are deployed before first use. An example might be a hotel room that is equipped with a closed network of smart home devices. In the second category, smart objects can be added or removed over time after an initial setup. An example for this category are households that occasionally integrate new smart objects into their smart home network. In fact, the first category of deployment should be considered as very unusual because it does not allow for replacement of broken smart objects.

The category of deployment is an important factor that affects security because each new smart object needs to be authenticated and equipped with cryptographic keys assuming that appropriate security schemes exist.

2) Communication: Currently, the most common types of communication in deployed smart home networks are power line communication (PLC) as well as radio frequency (RF) communication. Less common types of communication include communication via infrared, coaxial cable, and optical fiber. The first home automation networks in the 1980s and 1990s used mainly PLC in which the home automation objects are attached to power lines, and send and receive commands via this medium. Notable examples for such technologies are X10 [9], C-BUS [10], EIB/KNX [17], and UPB [11]. In recent years, RF communication became the most common communication technology applied in standards such as ZigBee [3], Z-Wave [4], or Bluetooth Low Energy (BLE) [12]. The decision on the communication medium has to take into account building structure and wiring as well as mobility requirements of the smart objects.

The choice of the communication medium is an important decision that - depending on the attacker model - highly influences the level of security. For many attacks, the attacker needs access to the communication medium. For example, eavesdropping on PLC communication is more difficult than eavesdropping on wireless communication.

3) Network Infrastructure: Smart home networks can be divided into infrastructure-based networks and ad-hoc networks. In infrastructure-based networks, there exists one or multiple control entities that know all smart objects in the network. The smart objects communicate with one of the control entities, which relays the data to the other smart objects. In ad-hoc networks, neither a control entity nor a hierarchy of smart objects exists. The smart objects communicate directly with each other and are able to act as router that forwards data traffic to distant smart objects. Also, hybrid versions of these two types of network infrastructures are imaginable.

The network infrastructure affects security. In an infrastructure-based network, the attacker gains control of the entire system by compromising a control entity. In an ad-hoc network, on the one hand, it is easier for an attacker to add a malicious smart object to the network than in infrastructure-based networks. On the other hand, by adding a malicious device, an attacker usually only gains control of a part of the network.

4) Network Topology: Four main network topologies are suitable for smart home networks: star, tree, bus, and mesh networks. An illustration of these network topologies is shown in Figure 1.

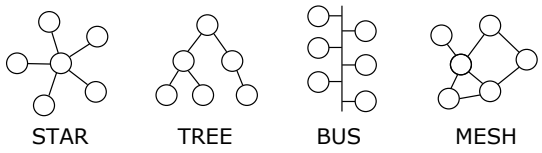


Fig. 1. Network topologies in smart home networks.

Star, tree, and bus network topologies are often infrastructure-based networks, while the mesh network topology is mainly used in ad-hoc networks.

The network topology influences the security, especially in terms of key management. Using a bus, star, or tree topology is suitable for a centralized key management scheme, while mesh networks are better suited for decentralized key management schemes.

5) Network Size: The size of a smart home network is determined by the number of smart objects. This number can vary from two smart objects to more than a hundred smart objects in a single smart home network. In the future, smart home network sizes with thousands of participating smart objects are imaginable.

The size of the network has an impact on security regarding key management. For a network with a small number of smart objects, a simple key management scheme such as pairwise shared keys is practical. However, in a network with hundreds of devices, this is infeasible and requires a more sophisticated approach, i.e., an advanced key management and key distribution schemes.

6) Privacy: Privacy is a sensitive topic in smart home networks. High amounts of data generated by smart objects allow to profile and track residents in a smart home. For example, smart meters can be used to reveal personal habits by analyzing the power consumption [13].

In general, smart home networks apply different approaches to protect a user's privacy. In fact, there are types of smart home networks that provide no explicit privacy-protecting mechanisms. In networks that consider privacy, the approaches can be classified in privacy-by-architecture and privacy-by-policy [14]. Privacy-by-architecture relies on technical means to protect data and identity of the user, whereas privacy-by-policy describes the approach in which users are informed through 'notice and choice' about how their data is used. Combining the technical privacy-enhancing mechanisms with notice and choice gives the users a certain degree of control over their personal data. Which privacy approach is enforced by a smart home network depends on the communication standard of the network specifying the content of the messages that are exchanged between smart objects.

Privacy affects other dimensions of smart home networks and is closely related to security, since privacy and security complement each other and partly rely on the same technical mechanisms. However, privacy and security can also be contradictory, for example, when security requirements call for a strong non-anonymous authentication while the privacy specifications demand the protection of the users' identity.

B. Node-related Dimensions

The following dimensions are specific for smart objects and might be different for every smart object of a smart home network.

1) Physical Properties: The physical properties of a smart object comprises primarily size and level of protection. In terms of security, physical properties are an important factor for physical security measures. In general, tiny smart objects can be stolen easier than huge ones, which makes them more prone to theft. Finally, if no or only little physical protection is provided, an attacker may compromise the node. A smart object under the control of an attacker can be used for different kinds of attacks, e.g., eavesdropping, denial of service, or man-in-the-middle attacks.

2) Functional Properties: Similar to physical properties, smart objects are also defined through functional properties that describe the capabilities of the smart objects. These capabilities provide a service at the application level and can be described as actuators. For instance, a 'smart light bulb' has the capability to light up a room, while a 'smart TV' enables the streaming of movies via an Internet connection.

Security plays an important role in this dimension because the compromise of these capabilities might have a real-life impact on the residents of the smart home. Therefore, the functional properties of smart objects are highly attractive targets for attackers. Assuming that an attacker controls the functional properties of a smart object, the attacker can scare and endanger residents, e.g., through randomly turning lights on and off.

3) Mobility: Mobility describes change of location of smart objects. From our point of view, there exist two categories of mobility: smart objects that are installed at a fixed location, e.g., sensors or large home appliances, and smart objects that change their location (in some cases frequently), e.g., robotic vacuum cleaners or remote controls. Furthermore, we distinguish two types of change of location: active and passive. An active change of location occurs if a smart object actively and intentionally changes its position, e.g., a robotic vacuum cleaner that is moving through the house. Passive location changes comprise all location changes resulting from human actions or environmental influences.

Mobility has a huge effect on communication because during a change of location, a smart object might lose contact to the network. This affects security in regards to availability and key managements. In hierarchical networks, a mobile smart object may connect to a different superior node, which demands flexible and dynamic key management schemes.

4) Power Supply: The power supply is a crucial design aspect of smart objects. Electrical power is not only necessary to make the communication with other objects possible but also

required to enable the functionality of the attached object. In smart home networks, the main power sources are either power lines or batteries. Batteries however may be rechargeable or non-rechargeable.

In deployment, the power supply is an important constraint because smart objects that are supplied via the power line need to be located near to power outlets. Otherwise, batteries have a limited lifetime, i.e., they need to be manually replaced or recharged.

The power supply also affects security by limiting the computational power of smart objects. Usually, power line supplied devices have a high computational power, and therefore, they are not restricted regarding the complexity of cryptographic algorithms and communication protocols. In contrast, battery supplied devices have only limited computational power and require low-power consuming communication protocols and cryptographic algorithms, e.g., lightweight cryptography [15].

5) Interoperability: Network communication follows predefined rules that are standardized in specifications. Some networks use proprietary communication methods, thus restricting the expansion of the network to smart objects using the same method. To avoid this restriction and enhance interoperability, alliances of industrial partners agreed on common standards. Today, several industrial standards for smart homes exist. ZigBee [3], Z-Wave [4], Bluetooth Low Energy [12], and EnOcean [16] being the most popular standards that use RF communication in home automation. Notable smart home standards using wired PLC communication are X10 [9] and EIB/KNX [17]. Nevertheless, many smart objects support multiple standards to achieve a higher compatibility with other objects.

The compliance of smart objects to standards also has a high impact on security because most standards support a set of different security mechanisms and cryptographic primitives. For instance, there is often a lack of interoperability between key management schemes, i.e., keys cannot be used on the same smart object for different standards.

6) Cryptographic Primitives: An important dimension of smart home networks are cryptographic primitives. In general, there exists a wide range of cryptographic primitives which serve different purposes. In fact, some specifications allow the possibility to apply no cryptographic mechanisms at all. Many specifications of smart home networks provide cryptographic suites that combine several cryptographic primitives to achieve integrity, confidentiality, and authenticity. An example is the *AES-128 bit encryption in Counter mode with CBC-MAC* (AES-CCM) as used in IEEE 802.15.4 [18].

The dimension of cryptographic primitives lays the technical foundation for all technical security mechanisms in smart home networks. Therefore, without cryptographic primitives a security reaching beyond organizational measures would not be feasible.

V. Conclusion

In this paper, we took a first step towards a security model for smart home networks by investigating the design space of smart home networks from a security perspective. Further work needs to be done to establish a comprehensive security framework for smart home networks. In the next steps, a security model, an attacker model as well as security goals need to be formulated based on the presented design space. Then, the system model in combination with the attacker model and the security goals describe the security model, which builds the foundation for the later security framework.

References

[1] H. Kopetz, "Internet of things," in *Real-time systems*. Springer, 2011, pp. 307–323.

[2] S. Dengler, A. Awad, and F. Dressler, "Sensor/actuator networks in smart homes for supporting elderly and handicapped people," in *21st International Conference on Advanced Information Networking and Applications (AINA 2007), Workshops Proceedings, Volume 2, May 21-23, 2007, Niagara Falls, Canada*. IEEE Computer Society, 2007, pp. 863–868. [Online]. Available: http://dx.doi.org/10.1109/AINAW.2007.325

[3] ZigBee Standards Organization, "ZigBee Specification – Document 053474r20," 2012.

[4] Zensys, "Software Design Specification – Z-Wave Protocol Overview," 2006.

[5] K. Römer and F. Mattern, "The design space of wireless sensor networks," *IEEE Wireless Commun.*, vol. 11, no. 6, pp. 54–61, 2004. [Online]. Available: http://dx.doi.org/10.1109/MWC.2004.1368897

[6] X. Chen, K. Makki, K. Yen, and N. Pissinou, "Sensor network security: a survey," *IEEE Communications Surveys and Tutorials*, vol. 11, no. 2, pp. 52–73, 2009. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/SURV.2009.090205

[7] Y. Zhou, Y. Fang, and Y. Zhang, "Securing wireless sensor networks: A survey," *IEEE Communications Surveys and Tutorials*, vol. 10, no. 1-4, pp. 6–28, 2008. [Online]. Available: http://dx.doi.org/10.1109/COMST.2008.4625802

[8] A. Atamli and A. Martin, "Threat-Based Security Analysis for the Internet of Things," in *2014 International Workshop on Secure Internet of Things (SIoT)*, Sep. 2014, pp. 35–43.

[9] X10, "X10 - Home gadgets since 1978," https://www.x10.com/.

[10] Clipsal - Integrated Systems, "Open C-Bus Serial Protocol Documents," http://training.clipsal.com/downloads/OpenCBus/ OpenCBus-ProtocolDownloads.html.

[11] Powerline Control Systems, "UPB Technology Description," http://www.pcslighting.com/resources/PulseWorx/Installation/ Documentation/UPBDescriptionv1.4.pdf.

[12] Bluetooth SIG, "Specification of the Bluetooth System – Master Table of Content & Compliance Requirements – Covered Core Package version 4.2," 2014.

[13] P. D. McDaniel and S. E. McLaughlin, "Security and privacy challenges in the smart grid," *IEEE Security & Privacy*, vol. 7, no. 3, pp. 75–77, 2009. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/MSP.2009.76

[14] S. Spiekermann and L. F. Cranor, "Engineering privacy," *IEEE Trans. Software Eng.*, vol. 35, no. 1, pp. 67–82, 2009. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TSE.2008.88

[15] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A survey of lightweight-cryptography implementations," *IEEE Design & Test of Computers*, no. 6, pp. 522–533, 2007.

[16] EnOcean Alliance, "EnOcean - No Wires. No Batteries. No Limits." https://www.enocean-alliance.org.

[17] H. Merz, T. Hansemann, and C. Hübner, *Building Automation: Communication Systems with EIB/KNX, LON and BACnet*. Springer Science & Business Media, 2009.

[18] "IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)," Tech. Rep., 2006. [Online]. Available: http://dx.doi.org/10.1109/ieeestd.2006.232110

Protecting Smart Objects from Denial-of-Service Attacks against SNMP

Samir Sharma
Beuth Hochschule für Technik Berlin
University of Applied Sciences
Luxemburger Str. 10, 13353 Berlin, Germany
Email: s53554@beuth-hochschule.de

Thomas Scheffler
Beuth Hochschule für Technik Berlin
University of Applied Sciences
Luxemburger Str. 10, 13353 Berlin, Germany
Email: scheffler@beuth-hochschule.de

***Abstract*—Smart objects are electrical devices that become part of a network architecture. Application security for such devices is a very important prerequisite, since the network connection now allows remote control and management of device functionality. SNMPv3 offers secure control and management functionality through application layer encryption and authentication provided by the integrated User Based Security Model (USM). However, an adversary can still attack the device by simply sending more messages that the device can handle or by exploiting weaknesses in the protocol design. The paper introduces a protection mechanism against Denial-of-Service attacks targeted at a SNMPv3 agent running on a smart object. We present implementation details and preliminary test results.**

I. Introduction

Many people see the *Internet of Things*, an interconnection of electric and electronic devices equipped with sensors or actuators, a microcontroller, a communication device and a power source to form a *smart object* [1], as the logical next step in the evolution of the Internet.

Most research has so far been targeted at the engineering challenges to build and operate such a network. Only recently have security issues come more into focus [2], [3]. Smart objects have a very high damage potential and attack surface through a number of reasons. They are numerous, they control physical device properties, are implemented on resource constrained computing platforms and they may use low-bandwidth network links. They may not be software-upgradeable and are operated in an open environment.

The primary computer security goals: confidentiality, integrity and authenticity, can also be enforced on smart objects through the correct application of cryptographic mechanisms such as encryption, digital signatures and hash functions. In a previous paper we have shown, that it is possible to implement reasonable strong application layer security mechanisms into smart objects based on the *Contiki OS* and an existing SNMP implementation [4]. However, network security assumes an attacker model, where an adversary has complete control over the communication channel and can capture, alter, reorder and inject arbitrary messages. It is therefore also necessary to focus on an additional security goal: availability.

An adversary can attack a smart object with the goal to make this device (or the whole network) unavailable to legitimate users – to conduct a Denial-of-Service attack. The problem with this kind of attack is, that there exist no simple defence mechanisms and smart objects simply have not enough resources to deal with such attacks.

Denial-of-service attacks can be broadly categorized in attacks that exploit protocol weaknesses and brute-force flooding attacks [5]. Network protocols may have properties that are exploitable by an attacker or the attacker may simply send more request that the device can handle. The latter attack may be mitigated through the implementation of strict *rate limiting* [6], while the former attack may require plausibility checks on the received protocol messages. Both mitigation techniques require considerable capabilities and are best implemented on a perimeter device, such as a network firewall.

II. Simple Network Message Protocol

The basic idea of any network management system is that there are two types of systems in a networked configuration: *agents* and *managers*. SNMP is the protocol that allows an SNMP manager to communicate with an SNMP agent by exchanging messages. An SNMP message is sent as a UDP-packet to port 161 [7].
A SNMP message consists of a sequence of fields with varying data types. On the basis of the *Basic Encoding Rules (BER)* a recipient knows where one field ends and another begins.
SNMP encodes protocol fields using type-length-value (TLV) encoding. A *type*-field specifies the data type using a single byte identifier. A *length*-field specifies the length in bytes of the following data section, and *value* is the actual data communicated (such as numbers, string, etc). TLV encoding facilitates flexible protocol design, but requires the recipient to carefully check message validity.

A. The SNMPv3 Message Format

Figure 1 illustrates the message structure which is defined in RFC 3412 [8]. The first five fields are generated by the message processing model on outgoing messages and processed by the message processing model on incoming messages. The next six fields show security parameters used by `USM`. Finally, the PDU, together with the `contextEngineID` and `contextName` constitute a scoped PDU used for PDU processing.
The fields are described by [7] as follows:

- `msgVersion`: Set to 3 (SNMP Version 3)

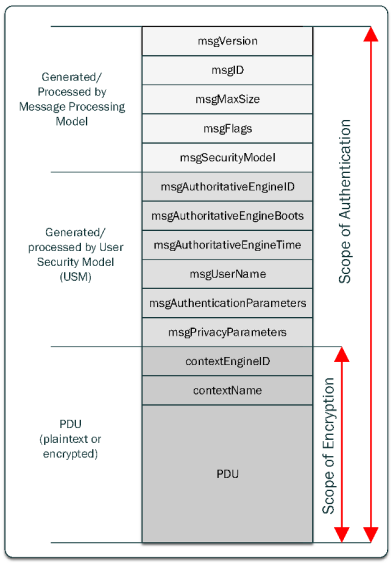


Fig. 1. SNMPv3 Message Format with USM

- `msgID`: An unique identifier used between two SNMP entities to coordinate request and response messages. The range of this ID is 0 through $2^{31} - 1$
- `msgMaxSize`: Maximum size of message with a range of 484 through $2^{31} - 1$
- `msgFlags`: An octet string containing three flags in the least significant three bits: `reportableFlag`, `privFlag` and `authFlag`
- `msgSecurityModel`: An identifier in the range of 0 through $2^{31} - 1$ that indicates which security model was used by the sender and therefore which security model should be used by the receiver. Value 3 is reserved for USM.
- `msgAuthoritativeEngineID`: This value refers to the source for a Trap Response or Report; and to the destination for a Get, GetNext, GetBulk, Set or Inform.
- `msgAuthoritativeEngineBoots`: An integer in the range 0 through $2^{31} - 1$ that represents the number of times that this SNMP engine has initialized since its configuration.
- `msgAuthoritativeEngineTime`: An integer in the range 0 through $2^{31} - 1$ that represents the number of seconds since the authoritative SNMP engine last incremented the `snmpEngineBoots` object.
- `msgUserName`: The user on whose behalf the message is being exchanged.
- `msgAuthenticationParameters`: Null if authentication is not being used. For the current definition of USM this parameter is an HMAC message code.
- `msgPrivacyParameters`: Null if privacy is not being used. For the current definition of USM, this parameter value forms the initialisation vector (IV) of the encryption algorithm.

B. Vulnerabilities

While a number of features associated with SNMP have changed since its initial standardization in 1988, the most important revisions of SNMPv3-Protocol focus on security [7]. According to RFC 3414, SNMPv3 does not provide any protection against DoS-Attacks [9]. For instance SNMPv3 is vulnerable to multiple stack-based buffer overflow issues. These vulnerabilities can be exploited by unauthenticated remote attackers. Strictly speaking, attackers can overflow the `msgAuthoritativeEngineID` and `msgUserName` [10]. SNMPv3 is also vulnerable to Man-in-the-middle-attacks. Since the discovery mechanism is itself vulnerable, it can be manipulated to allow an attacker to select the encryption and authentication keys used by the protocol. Successfully executed, such attacks could potentially allow an adversary to reveal information about devices within the network, as well as to potentially modify device behavior [11].

III. Netfilter

Netfilter is the packet filtering framework inside the Linux 2.4.x and later kernel series. Software inside this framework enables packet filtering, network address translation (NAT) and other packet mangling.
According to [12] the actual implementation is broken into two parts:

1) **Netfilter**: Netfilter is a set of hooks inside the Linux kernel that allows kernel modules to register callback functions with the network stack. A registered callback function is then called for every packet that traverses the respective hook within the network stack.
2) **Iptables**[1]: Iptables is a generic table structure for the definition of rulesets. Each rule within an IP table consists of a number of classifiers (*iptables matches*) and one connected action (*iptables target*).

As shown in figure 2 incoming packets that arrive at the network interface have to pass the `PREROUTING`-chain, before the local routing table is consulted, this is primarily used for NAT. If the packets are destined for the local system, they have go through the `INPUT`-chain. Here a decision is made whether the packets should be accepted or dropped. Packets that are destined for the protected network go through the `FORWARD`-chain, where a decision is made to forward them to their destination or drop them. Packets created by the firewall system itself, go through the `OUTPUT`-chain. After a routing decision has been made, outgoing packets have to pass through the `POSTROUTING`-chain [12]–[14].

Listing 1 shows the source code to initialize a hook. We see that the rules are applicable only to IPv6 (line 2). `Priority` specifies where in the order of execution this hook gets executed. In line 4 we see that this module is written for filter-table `FORWARD`.

[1] *iptables* is used for IPv4 only, for IPv6 *ip6tables* has to be used

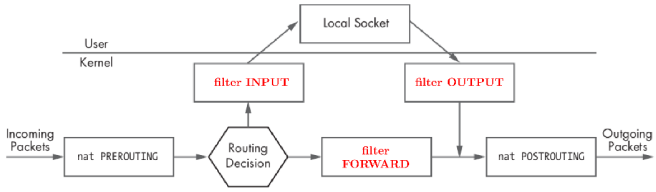


Fig. 2. Packetflow in Netfilter

Listing 1. example to initialize a hook

```
1 static struct nf_hook_ops myhook_ops __read_mostly = {
2     .pf         = NFPROTO_IPV6,     //IPv6-packets only
3     .priority   = 1,
4     .hooknum    = NF_INET_FORWARD,  //filter-table:FORWARD
5     .hookfn     = myhook_fn         //function to call
6 };
```

IV. Design of Framework

The aim of this research work is to design a framework to mitigate SNMP-DoS-attacks against a protected network. We focus on the prevention of flooding attacks through rate-limiting and the the mitigation of buffer-overflow attack through protocol checking.

A. Implementation Details

Our *rate-limiting module* aims to mitigate flooding attacks, where a high number of requests are sent to SNMP-Port 161. All incoming packets are allowed to pass, until a limit for the time-period is reached.
The values for limit and time can be set manually. For this project we have decided to accept 15 SNMP packets per second. Algorithm 1 shows the basic structure of the rate-limiting module:

Algorithm 1: Rate-Limiting

```
1  Set the value for limit;
2  Set the value for wait-time;
3  for each arriving packet do
4      if packet = UDP-packet then
5          if dest.-port = SNMP-Port then
               // SNMP-Port = 161
6              increment no. of packets;
7              if No. of packets > limit then
8                  drop packets;
9                  wait until the wait-time has elapsed;
10                 no. of packets are set to zero;
11             else
12                 accept packet;
13             end
14         else
15             accept packet;
16         end
17     else
18         accept packet;
19     end
20 end
```

The *packet-filtering module* tries to prevent buffer-overflow attacks on the smart object. The module filters malformed SNMP-packets by checking the syntax of the protocol, in particular the length of each element of the protocol. If a anomaly is detected, the packet will be dropped.
The basic structure for this module is shown in Algorithm 2:

Algorithm 2: Packet Filtering

```
1  for each arriving packet do
2      if packet = UDP-packet then
3          if source- or dest.-port = SNMP-Port then
               // SNMP-Port = 161
4              for each Element of SNMPv3 do
                   // check the length of each element
5                  if length allowed by definition then
6                      accept packet;
7                  else
8                      drop packet;
9                  end
10             end
11         else
12             accept packet;
13         end
14     else
15         accept packet;
16     end
17 end
```

B. Test-setup

We built a small test network (cf. Figure 3) in order to test the firewall modules. Computer 1 represents the attacker(s). The generated traffic has to pass the firewall, i.e. our implemented modules, to perform the attacks. The firewall rules have been deployed in the `FORWARD`-table. Both firewall and victim are represented by Computer 2.

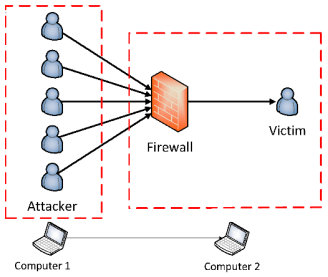


Fig. 3. Testbed for the evaluation of DoS-mitigation techniques

We chose to filter SNMP-packets on the `FORWARD`-Hook for two reasons: Firstly, to create a real-life-scenario, and secondly it would have been difficult to monitor the filtering process, if the packets were filtered on the `INPUT`-filter-table. Wireshark (or any other network protocol analyzer) would have captured the traffic before any packets could be inspected.

We used the Router-Advertisement Daemon *ratvd* in Computer 2 to create the aforementioned real-life scenario. Computer 2 is sending out ICMPv6 Router-Advertisements, just like a normal router would do. Computer 1 generates its IPv6-address from the received RAs, and registers Computer 2 as the default-route.
Computer 1 is sending packets to a specific IPv6-address using spoofed source addresses taken from the whole IPv6-address space. The packets are able to reach the host via the router/firewall. We do not need to implement the actual victim host, e.g. the smart object, because we are currently more interested in the correct filtering behaviour of the firewall,

than the behaviour of the smart object. In order to monitor the filtered traffic we simply start Wireshark on Computer 2.

C. Test-results

We used the tool *udp6*, which is a part of the *SI6-Networks-Toolkit*[2] to generate SNMP-flooding attacks. It generates more than 10^5 packets per second with spoofed IPv6-addresses. *udp6* was set-up to send packets to the SNMP-port.
In Table I we see the number of packets and their corresponding bit-rate that were detected with and without the rate-limiting-module. Figure 4 shows the graphical representation of these results.

TABLE I. RESULTS OF THE RATE-LIMITING MODULE

Time	Rate-Limited Packets		Flooding Attacks	
	no. of packets	bits	no. of packets	bits
1	15	7400	$1,58 \cdot 10^5$	$7,86 \cdot 10^7$
2	15	7400	$1,8 \cdot 10^5$	$8,94 \cdot 10^7$
3	15	7400	$2,58 \cdot 10^5$	$1,28 \cdot 10^8$
4	15	7400	$2,58 \cdot 10^5$	$1,28 \cdot 10^8$
5	15	7400	$2,59 \cdot 10^5$	$1,28 \cdot 10^8$
6	15	7400	$1,14 \cdot 10^5$	$5,65 \cdot 10^7$
7	15	7400	$2,58 \cdot 10^5$	$1,28 \cdot 10^8$
8	15	7400	$2,58 \cdot 10^5$	$1,28 \cdot 10^8$
9	15	7400	$2,49 \cdot 10^5$	$1,23 \cdot 10^8$
10	15	7400	$2,59 \cdot 10^5$	$1,26 \cdot 10^8$

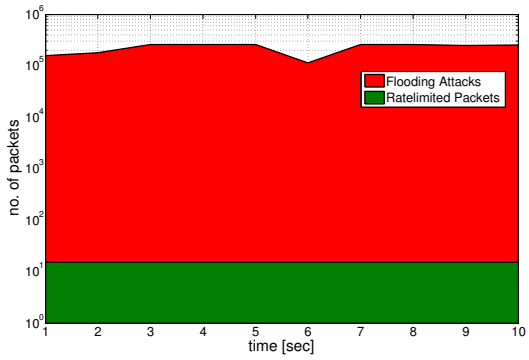


Fig. 4. Number of packets per second during the attack and after filtering

To generate malformed SNMPv3-packets we used the software *scapy*[3]. *Scapy* is a packet manipulation program written in Python. It is able to construct packets for a wide number of protocols including SNMP. We used this tool to generate values for SNMP-elements that where larger than specified in the standard and could potentially overflow the receive buffer of a smart object. We modified elements of the type *octet string* to contain a 10^5 Byte sequence of the value "A B C". The other elements were set to contain a 1000 Byte value. Our implementation correctly identified and dropped the malformed traffic (cf. Table II).

V. CONCLUSION

Mitigation of DoS/DDoS attacks should be part of an overall risk management strategy for the Internet of Things. In this paper, we propose a solution to mitigate two types of DoS-Attacks concerning the SNMPv3-protocol, namely Flooding and Buffer-overflow attacks by malformed packets. We have shown that our implementation can successfully protect smart objects from damage that these attacks may otherwise cause.

[2]for more information: www.si6networks.com/tools/ipv6toolkit
[3]for more information: www.secdev.org/projects/scapy/

TABLE II. RESULTS OF THE PACKET-FILT.-MODULE

Element	Test Conditions		Test Results
	Value	Pkt-Size (Byte)	
`Version`	$1000\ Byte$	1165	dropped
`msgID`	$1000\ Byte$	1165	dropped
`msgMaXSize`	$1000\ Byte$	1165	dropped
`msgFlags`	$"A\ B\ C\ " \cdot 10^5$	60150	dropped
`Sec.Model`	Value could not be changed		
`EngineID`	$"A\ B\ C\ " \cdot 10^5$	60150	dropped
`Boots`	$1000\ Byte$	1165	dropped
`Time`	$1000\ Byte$	1165	dropped
`UserName`	$"A\ B\ C\ " \cdot 10^5$	60150	dropped
`Auth.Param`	Value could not be changed		
`Priv.Param`	Value could not be changed		

REFERENCES

[1] J.-P. Vasseur and A. Dunkels, *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann, 2010.

[2] W. Jackson, "*How hackers can turn the Internet of Things into a weapon*," Web Blog, May 2013. [Online]. Available: http://gcn.com/blogs/cybereye/2013/05/how-hackers-turn-internet-of-things-into-weapon.aspx

[3] C. Wueest and M. B. Barcena, "*Insecurity in the Internet of Things*," Whitepaper, Mar 2015. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/insecurity-in-the-internet-of-things.pdf

[4] S. Zehl and T. Scheffler, "Secure access and management of Smart Objects with SNMPv3," in *Proceedings of the Third IEEE International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, P. Friedrich, H. L. Cycon, B. Wolf, and J. Clauss, Eds. IEEE, 2013, pp. 366–370.

[5] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, *Internet Denial of Service - Attack and Defense Mechanisms*. Prentice Hall, 2004.

[6] P. Vixie, "Rate-limiting state," *ACM Queue*, vol. 12, no. 2, pp. 10:10–10:15, Feb. 2014. [Online]. Available: http://doi.acm.org/10.1145/2578508.2578510

[7] W. Stallings, *SNMP, SNMPv2, SNMPv3 and RMON 1 & 2*, 3rd ed. Reading, MA 01867, USA: Addison-Wesley Verlag, 1999, ISBN : 0-201-48534-6.

[8] J. Case, D. Harrington, R. Preshun, and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)," Internet Engineering Task Force, RFC 3412, December 2002. [Online]. Available: https://tools.ietf.org/html/rfc3412

[9] U. Blumenthal and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)," Internet Engineering Task Force, RFC 3414, Dec. 2002, updated by RFC 5590. [Online]. Available: http://tools.ietf.org/html/rfc3414

[10] R. Paleari, "*Multiple buffer overflows on Huawei SNMPv3 service*," Blog, Jun. 2013. [Online]. Available: http://blog.emaze.net/2013/05/multiple-buffer-overflows-on-huawei.html

[11] N. Lawrence and P. Traynor, "*Under New Management: Practical Attacks on SNMPv3*," Georgia Institute of Technology, Tech. Rep., August 2012. [Online]. Available: https://www.usenix.org/system/files/conference/woot12/woot12-final14.pdf

[12] R. Spenneberg, *Linux-Firewalls mit iptables & Co.*, 1st ed. 81829 München, Deutschland: Addison-Wesley Verlag, 2006.

[13] A. G. Lessig, *Linux Firewalls - Ein praktischer Einstieg*, 2nd ed. 50670 Kö, Deutschland: O'Reilly Verlag, 2006, ISBN-10: 10 3-89721-446-6.

[14] R. L. Ziegler, *Linux-Firewalls, Konzeption und Implementierung für Netzwerke und PCs* , 2nd ed. 81829 München, Deutschland: Markt+Technik Verlag, 2002, ISBN: 3-8272-6257-7.

Ausfalltolerante Datenhaltung durch Multicast Growth Codes

Isabel Madeleine Grimm*, Max van gen Hassend und Reiner Kolla‡
Lehrstuhl für Technische Informatik, Julius-Maximilians-Universität Würzburg
* Email: grimm@informatik.uni-wuerzburg.de
‡ Email: kolla@informatik.uni-wuerzburg.de

***Zusammenfassung*—Bei dem hier behandelten Multicast Growth Codes (MCGC) Verfahren handelt es sich um eine Form der Netzwerkkodierung, welche auf dem Prinzip der Growth Codes basiert. Ziel ist es, die Persistenz in Sensornetzen zu erhöhen und zu jedem Zeitpunkt möglichst viele Daten, trotz Knoten- oder Linkausfällen, rekonstruieren zu können. Hierzu werden von einem Sensorknoten erhobene Sensordaten mit bereits empfangenen Daten kombiniert, an Nachbarknoten gesendet und somit vielfach im Netz verteilt und gespeichert. Mögliche Realisierungen von MCGC sollen hier vorgestellt werden.**

I. Einleitung

Spätestens seit der Idee des Internet of Things (IoT) sind drahtlose Sensornetze mit einer Vielzahl kleinster autonomer Sensorknoten omnipräsent. Durch die große Anzahl an Knoten und den häufigen Einsatz unter rauen Bedingungen ist bei diesen Systemen eine hohe Ausfallwahrscheinlichkeit und Fehleranfälligkeit inherent gegeben. Aufgrund dieser Häufigkeit von Fehlern und temporären oder permanenten Knotenausfällen sowie der Risiken drahtloser Kommunikation mit sehr vielen Teilnehmern, ist Datenpersistenz ein entscheidendes Thema. Unser Ziel hierbei ist es, zu jedem Zeitpunkt einen möglichst großen Anteil der erhobenen Sensordaten rekonstruieren zu können und Daten zuverlässig im System zu halten. Um dies zu ermöglichen, werden Daten redundant verteilt und im Netz verteilt. Die hier verwendete Methode zur ausfalltoleranten Datenhaltung in Sensornetzen ist eine Form der Netzwerkkodierung, welche auf dem Prinzip der Growth Codes [1] basiert, und sei als Multicast Growth Codes (MCGC) bezeichnet.

Dieser Artikel beginnt mit einem Abschnitt zu Related Work, in welchem bestehende Alternativen zu Multicast Growth Codes kurz angesprochen werden. Danach folgt eine detaillierte Einführung in die Growth Codes. Anschließend wird das Prinzip der Multicast Growth Codes beschrieben sowie mögliche Realisierungen und Erweiterungen des Verfahrens vorgestellt.

II. Related Work

Redundante Datenhaltung wurde in [2] mit Hilfe einer Form der Netzwerkkodierung, welche speziell für statische, symmetrische Gitternetzwerke entwickelt wurde, realisiert. Hierbei wurden Codeworte unter Verwendung aller in vorherigen Iterationen empfangenen Codeworte kodiert. Eine andere Art der Netzwerkkodierung sind die in [1] verwendeten Growth Codes, welche als Basis für die Erweiterungen in unserem Artikel dienen und speziell auf hochdynamische Sensornetze zugeschnitten sind. Munaretto et al. haben in [3] verschiedene Veränderungen der Growth Codes, wie Multi-Codewort Encoding, aggressive Gradverteilung und die Verwendung der Gauß-Elimination zur Dekodierung, für weniger dynamische Systeme untersucht. In [4] haben sie anschließend unterschiedliche Gradverteilungen, sowie gezieltes Buffer-Management zur Steigerung der Persistenz analysiert. Die Autoren in [5] haben dagegen eine Kombination aus Fountain Codes und Netzwerkkodierung vorgestellt. Eine allgemeine Abgrenzung der Netzwerkkodierung gegen Routing speziell für drahtlose Netzwerke findet sich in [6]. Sathiamoorthy et al. haben in [7] die Vorteile verteilter Datenspeicherung mittels Erasure Codes insbesondere für Fahrzeugnetzwerke untersucht.

III. Growth Codes

Im Folgenden werden die Growth Codes entsprechend [1] beschrieben, um das Prinzip selbst sowie das Potential möglicher Erweiterungen zu verdeutlichen. Sensordaten, die von einem Knoten erhoben werden, werden bis zur nächsten Datenerhebung als das aktuelle Symbol des Knotens bezeichnet. Dieses Symbol wird, allein oder in Kombination mit anderen, als sogenanntes Codewort vielfach im Netz verteilt.

Hierzu wird das eigene Symbol zu Beginn an einer separaten Stelle sowie in jeder für den Datenaustausch verwendeten Speicherstelle als Codewort gespeichert. Die Zeit ist dabei in Runden unterteilt. Zu jeder Runde findet nun ein Datenaustausch mit einem zufällig gewählten Nachbarknoten statt. Hierfür greift der Knoten ein Codewort aus einer zufälligen Speicherstelle heraus und überprüft vor der Übertragung dessen Grad. Der Grad eines Codeworts entspricht hierbei gerade der Anzahl an Symbolen, die in diesem enthalten sind, d.h. die mittels xor zu diesem verknüpft wurden. Im Header eines Codewort-Pakets kann indirekt ausgelesen werden, welche Codeworte dieses beinhaltet. Durch die sogenannte Gradverteilungsfunktion wird der Grad deg_{max} vorgegeben, welchen ein Sensorkonten vor der Übertragung eines Codeworts für dieses Codewort dann anstrebt. Sollte der Grad des Codeworts kleiner diesem vorgegebenen deg_{max} sein, fügt der Knoten anschließend das eigene Symbol aus dem separaten Speicher via xor zu dem Codewort hinzu. Letzteres wird jedoch nur dann ausgeführt, wenn das eigene Symbol noch nicht bereits im Codewort enthalten ist. Anschließend wird das Codewort gegen ein Codewort ausgetauscht, welches von dem zufällig

gewählten Nachbarknoten gegebenenfalls erweitert und dann angeboten wurde. D.h. das Codewort des Nachbarknotens wird nun an die Stelle des eigenen Codeworts gespeichert und umgekehrt. Es findet folglich ein bidirektionaler Datenaustausch statt.

Die Senke ist in diesem Szenario direkt an einen der Knoten angeschlossen und erhält jede Runde das von dem Knoten als letztes durch seien Nachbarn erhaltene Codewort. Es wird also jede Runde genau ein Codewort empfangen und versucht zu dekodieren. Als Gradverteilungsfunktion wird in [1] die *Growth Codes Gradverteilungsfunktion* verwendet. Diese ist definiert als

$$\pi(k) = max\left(0, min\left(\frac{K_j - K_{j-1}}{k}, \frac{k - K_{j-1}}{k}\right)\right),$$

wobei

$$K_j = K_{j-1} + \sum_{i=R_{j-1}}^{R_j - 1} \frac{\binom{N}{j}}{\binom{i}{j-1}\binom{N-i}{1}},$$

$$K_1 = \sum_{i=0}^{R_1 - 1} \frac{N}{N-i}$$

mit $R_1 = \frac{N-1}{2}, ..., R_j = \frac{jN-1}{j+1}$ für $j \in [1, N-1]$ und $k(t)$ zum Zeitpunkt t von allen Knoten erzeugte Codeworte sowie N ursprüngliche Symbole, also N erhobene Sensordatensätze, d.h. für ein System aus anfangs N Sensorknoten. Diese Gradverteilungsfunktion wird umgesetzt, indem man die Transitionspunkte K_i, d.h. die Zeitpunkte, zu denen der vorgegebene Grad deg_{max} für Codeworte um 1 inkrementiert wird, entsprechend $K_1, K_2, ...$ wählt. D.h. bis K_1 Runden nach der letzten Sensordatenerhebung gilt für den anzustrebenden Grad eines Codewortes, welches für die Übertragung ausgewählt wurde, $deg_{max} = 1$. Anschließend sollte bis K_2 Runden $deg_{max} = 2$ gelten, und so weiter. Hat man ein System mit mehreren Senken, so muss K_i entsprechend skaliert werden, da nun mehrere Codeworte pro Runde von Senken empfangen werden. Diese Transitionspunkte werden in [1] als „hart“ in die Knoten implementiert angegeben.

Übertragene Codeworte sind zu Beginn unkodiert, d.h. sie besitzen Grad 1, später haben sie Grad 2 und mit fortschreitender Zeit einen regelmäßig anwachsenden, immer höheren Grad. Aus diesem Grund wird das Verfahren als „Growth Codes“ bezeichnet.

In [1] werden verschiedene Arten von Decodern verwendet. Interessant ist dabei der Decoder D, welcher ein empfangenes Codewort dekodiert, falls es Grad 1 besitzt oder sich nur um ein mittels xor verknüpftes Symbol von bereits Dekodiertem unterschiedet. Codeworte, die aktuell noch nicht dekodierbar sind, werden zwischengespeichert und in späteren Iterationen erneut versucht zu dekodieren.

Bei obiger Gradverteilung mit nur einer Senke im System und dem Decoder D gilt, dass, um bis zu $R_1 = \frac{N-1}{2}$ Symbole zu rekonstruieren, erwartungsgemäß K_1 in der Senke empfangene Grad-1 Codeworte, also K_1 Runden nötig sind. Um die nächsten $R_2 - R_1$ Symbole zu rekonstruieren, sind anschließend erwartungsgemäß höchstens $K_2 - K_1$ Grad-2 Codeworte nötig. Allgemein sind erwartungsgemäß höchstens K_j in der Senke empfangene Codeworte nötig, um genau R_j Symbole rekonstruieren zu können. Des Weiteren gilt, dass, um bis zu R_1 Symbole in der Senke zu rekonstruieren, im Optimalfall nur Codeworte von Grad 1 dort empfangen werden. Um die nächsten $R_2 - R_1$ Symbole rekonstruieren zu können, sollten anschließend idealerweise nur Codeworte von Grad 2 bei der Senke ankommen, bzw. allgemein um bis zu R_j Symbole zu rekonstruieren nur Codeworte vom Grad $\leq j$.

Die Senke erhält pro Runde genau ein Codewort von ihrem angeschlossenen Nachbarknoten. Dieses entspricht exakt dem vom Nachbarknoten als letztes empfangenen Codewort. Durch die Überprüfung und gegebenenfalls Anpassung des Grades vor der Übertragung erfüllt dieses Codewort die, durch Transitionszeitpunkte realisierte, Gradverteilung und besitzt den für die Dekodierung zu diesem Zeitpunkt optimalen Grad. Würde der Grad zu früh zu hoch gewählt, könnte die Senke zu Beginn gegebenenfalls nichts oder sehr wenig direkt dekodieren und müsste alle empfangenen Codeworte zwischenspeichern. Würde der Grad dagegen jeweils erst sehr spät erhöht werden, so erhält die Senke unter Umständen mehr doppelte Symbole und in einem Codewort würden weniger Symbole auf einmal pro Runde an die Senke kodiert übertragen werden als möglich. Das Ziel der Growth Codes ist jedoch zu jedem Zeitpunkt so viele Symbole wie möglich rekonstruieren zu können. Aus diesem Grund sollten die Transitionszeitpunkte und damit die optimale Gradverteilung eingehalten werden. Das Verfahren der Growth Codes wurde speziell für hochdynamische, verteilte Sensornetze entwickelt und ist bei Knotenausfällen, hinzukommenden Knoten und insbesondere bei sich stark bewegenden Netzen besonders effizient. In statischen Fällen eignet sich das Verfahren der Growth Codes dagegen nicht.

Ein Vorteil der Growth Codes ist deren gute Anpassungsfähigkeit an die sich verändernde Topologie eines Sensornetzes. Durch das Hinzufügen von Redundanz und der Verteilung dieser im Netz wird außerdem die Wahrscheinlichkeit erhöht, dass erhobene Daten selbst bei Knotenausfällen in der Senke rekonstruiert werden können. Nachteil der Growth Codes ist jedoch, dass die vorgegebene optimale Gradverteilung in der Realität häufig nicht eingehalten werden kann. Dies tritt speziell bei wenig Bewegung im Netz ein. In diesem Fall kommen keine neuen Knoten, also keine neue Information in Form von neuen Codeworten, in der Nachbarschaft hinzu. Somit kann bald das Symbol des jeweiligen Knotens nicht mehr zu dem Codewort aus dem Speicher hinzugefügt werden, da es bereits aufgrund vorhergehendem wiederholtem Datenaustausch mit immer gleichen Nachbarknoten bereits enthalten ist. Folglich erhöht sich der Grad der Codeworte zu langsam, die Gradverteilung kann also nicht erfüllt werden und die Senke kann damit nicht optimal dekodieren.

IV. Multicast Growth Codes

Der grundsätzliche Unterschied zwischen Growth Codes und MCGC liegt darin, an wen ein für die Übertragung

ausgewähltes Codewort gesendet wird. Bei Growth Codes wird das Codewort in der Regel an ausschließlich einen zufällig ausgewählten Nachbarknoten übertragen. Bei MCGC sendet man das Codewort dagegen an alle direkten Nachbarknoten, d.h. alle Knoten in 1-Hop-Nachbarschaft empfangen das gesendete Codewort. Auf diese Weise kommen bei allen Knoten, insbesondere also auch bei der Senke, zu jeder Runde viele Codeworte auf einmal an. Die Wahrscheinlichkeit für das Ankommen neuer Codeworte ist bei mehr empfangenen Codeworten größer, also kann die Senke mit jeder Runde bereits deutlich mehr dekodieren als bei den Growth Codes. Da ein Knoten nur ein ausgewähltes Codewort mehrfach sendet, empfängt jeder Knoten in jeder Runde deutlich mehr verschiedene Codeworte als er selbst aussendet. Würde man den Growth Codes Algorithmus ohne weitere Anpassungen nur durch Mulitcast erweitern, würden, speziell bei kleinem Speicher, viele zuvor gespeicherte Codeworte durch neu empfangene Codeworte direkt überschrieben werden. Im Worst Case wären diese gelöschten Codeworte zuvor noch nie ins Netz weitergeleitet worden. Damit würde ein großer Teil der Information verloren gehen. Aus diesem Grund wurde bei MCGC eine Art Empfangspuffer für ankommende Pakete zur weiteren Verarbeitung eingeführt. Hierbei sei allgemein angekommen, dass Speicher sehr günstig ist und das Hauptziel zuverlässige Datenhaltung ist.

A. Mögliche Erweiterungen

Im Folgenden sollen nun mögliche Implementierungsdetails und Realisierungen der Multicast Growth Codes vorgestellt werden. Für die Überlegungen sei angenommen, jeder Knoten könne selbst dekodieren und speichere bereits dekodierte Symbole ab. Dies bietet den Vorteil, dass jeder beliebige Knoten bei Bedarf unmittelbar als Senke fungieren kann und die Daten sehr zuverlässig im System gehalten werden, da sie überall gesammelt sind.

In oben beschriebenem Eingangspuffer können empfangene Pakete beispielsweise auf Dubletten überprüft werden. D.h. bei Codeworten von Grad 1 wird direkt verglichen, ob dieses Symbol bereits im Speicher enthalten ist. Codeworte höheren Grades werden, sofern möglich, im Knoten selbst dekodiert und alle dabei rekonstruierten, enthaltenen Symbole werden der Prüfung auf Dubletten unterzogen. Ist ein empfangenes Codewort nicht sofort dekodierbar, wird es, wie auch bei dem Decoder D der Growth Codes, zwischengespeichert und bei neu rekonstruierten Symbolen erneut versucht zu dekodieren.

Pro abgespeichertem Symbol kann ein Zähler verwendet werden, welcher angibt, wie häufig dieses Symbol nach erstmaligem Erhalt erneut empfangen wurde. Dies dient, neben dem Verhindern Dubletten abzuspeichern und dadurch unnötig Speicherplatz zu belegen, unter anderem der Bestimmung von Sendeprioritäten. Zusätzlich kann ein weiterer Zähler eingefügt werden, welcher angibt, wie häufig ein Paket bereits versendet wurde, um seltener gesendeten Codeworten anschließend eine höhere Priorität zuzuteilen. Insgesamt bietet es sich an beide Zähler für die Bestimmung von Sendeprioritäten zu kombinieren. Insbesondere bei statischen Systemen liefern die Zähler eine gute Möglichkeit abzuschätzen, wie häufig ein Paket in der Umgebung des Knotens bereits vorkommt. Für den Fall über einen längeren Zeitraum gleichbleibender Nachbarn sollten bevorzugt Codeworte gesendet werden, welche ein Großteil der Nachbarn mit hoher Wahrscheinlichkeit noch nicht erhalten hatten. Dies lässt sich gerade dadurch feststellen, indem man zählt wie häufig ein Symbol empfangen, also durch direkte Nachbarn zuvor an den betrachteten Knoten gesendet wurde. Auf diese Weise soll vermieden werden immer wieder die gleiche Information zu senden. Speziell bei nicht hochdynamischen, also stabileren Szenarien, ist diese Tatsache entscheidend für Ausfalltoleranz und Effizienz. Des Weiteren könnte man, speziell für ein Szenario mit regelmäßiger Datenerhebung, auch einen Zähler für das Alter und damit der Aktualität oder der Länge des bereits zurückgelegten Weges eines Codeworts in die Sendepriorität integrieren. Bei MCGC wird das zu sendende Codewort also nicht zufällig, sondern gezielt entsprechend verschiedener Kriterien ausgewählt. Zusätzlich zu der geschickten Auswahl des zu sendenden Basiscodeworts, wird auch das Symbol, welches mittels xor zu dem Codewort hinzugefügt werden soll, um den durch die jeweilige Gradverteilung vorgegebenen Grad deg_{max} zu erfüllen, zielgerichtet mit Hilfe obiger Zähler ausgewählt. Sollte der Grad des Basiscodeworts um mehr als 1 kleiner als der zu diesem Zeitpunkt optimale Grad deg_{max} sein, so wird bei MCGC, anderes als bei den Growth Codes, mehr als ein weiteres Symbol via xor hinzugefügt. Damit lässt sich ein Codewort mit deutlich abweichendem Grad in nur einem Schritt auf den vorgegeben Grad bringen, statt durch mehrere Knoten wandern zu müssen und in jeder Runde den Grad dabei um höchstens 1 erhöhen zu können. Im Gegensatz zu den Growth Codes, wird für das Hinzufügen außerdem nicht das eigene Symbol des Knotens aus dem separaten Speicher verwendet, sondern die jeweiligen Symbole werden aus dem Speicher der bereits im Knoten dekodierten Symbole gewählt. Da dadurch in der Regel viel mehr Symbole für die Verknüpfung via xor zur Auswahl stehen als nur das eigene Symbol, ist es möglich die Gradverteilung deutlich zuverlässiger zu erfüllen. Ist bei den Growth Codes das eigene Symbol bereits im Basiscodewort enthalten, kann es nicht mehr mittels xor hinzugefügt werden, da es auf diese Weise nur wieder aus dem Codewort gelöscht werden würde.

Neben der zielgerichteten Auswahl und Anpassung des Codeworts für die Übertragung, ist auch die Gradverteilungsfunktion eine wichtige Stellschraube für die Dekodierbarkeit der MCGC und muss ausführlich untersucht werden. Hierbei besteht beispielsweise die Möglichkeit die Transitionszeitpunkte mit Hilfe statistischer Überlegungen zuvor festzulegen und, wie bei [1], in die Knoten fest zu implementieren. Alternativ könnte jeder Knoten, anhand der bei ihm bereits dekodierten Symbole, einen für sich selbst zum jeweiligen Zeitpunkt optimalen Grad bestimmen und vor der Übertragung eines Codeworts zum Abgleich verwenden. Eine weitere Möglichkeit wäre, diesen, für den Knoten selbst optimalen, Grad mit jedem gesendeten Codewort an die direkten Nachbarn weiterzuleiten. Bei späteren Sendevorgängen der Nachbarknoten selbst

könnten diese dann den Grad des zu sendenden Codeworts an den, für ihre jeweiligen direkten Nachbarn, im Durchschnitt oder für den Worst Case besten Grad anpassen.

Da jeder Knoten potentielle Senke ist, ist es interessant zu untersuchen, zu welchem Zeitpunkt, in den unterschiedlichen Erweiterungen und bei verschiedenen Netzdurchmessern, ein zufälliger Knoten wie viele ursprüngliche Symbole rekonstruiert hat. Außerdem könnte ermittelt werden, zu welchem Zeitpunkt eine vorgegebene Anzahl von zufällig gewählten Knoten erwartungsgemäß alle Symbole rekonstruieren kann. Des Weiteren soll analysiert werden, wie viele Knoten nötig sind, um jeweils mit einer vorgegebene Wahrscheinlichkeit nach einer bestimmten Anzahl an Runden einen gewissen Anteil der ursprünglichen Symbole rekonstruieren zu können.

Zusätzlich sollen bei Analysen gewisse Störraten eingebaut, d.h. Fehlertoleranz hinsichtlich gestörter Links untersucht werden. Hierbei wird simuliert werden, dass nur ein Teil der 1-Hop-Nachbarn das von einem Knoten gesendete Paket auch tatsächlich empfängt bzw. allgemein nur ein gewisser Prozentsatz der Pakete ankommt. Wann und ob mit einer vorgegebenen Wahrscheinlichkeit dann alle Pakete rekonstruiert werden können, also wie tolerant das System gegenüber dieser Art von Störung ist, soll in Zukunft genauer analysiert werden. Außerdem soll intensiv untersucht werden, wie das System auf Ausfälle einzelner zufälliger Knoten, zentraler Knoten oder ganzer Nachbarschaften reagiert. Insgesamt soll das Hauptziel der ausfalltoleranten Datenhaltung bis zu einem gewissen Grad mit vorgegebener Wahrscheinlichkeit gewährleistet sein.

B. Vorteile der Multicast Growth Codes

Die wichtigsten Vorteile der Growth Codes werden auch bei Multicast Growth Codes beibehalten. So bleibt das Verfahren weiterhin für hochdynamische, verteilte Sensornetze effizient, soll nun jedoch so erweitert werden, dass es auch für weniger dynamische, realistischere Szenarien anwendbar ist. MCGC soll also weiterhin die Fähigkeit besitzen, sich gut an die, sich möglicherweise verändernde, Topologie eines Sensornetzes anzupassen. Insbesondere sollen mögliche Knotenausfälle durch die hinzugefügten und in jeder Runde vielfach verteilten Redundanzen gut kompensiert werden, d.h. ausfalltolerante Datenhaltung soll gewährleistet sein. Wie schon bei den Growth Codes, sollen nicht alle Symbole möglichst schnell, sondern zu jedem Zeitpunkt möglichst viele Symbole rekonstruiert werden können. Hierzu werden, durch die Verwendung von Multicast, in jeder Runde deutlich mehr Codeworte empfangen und damit auch mehr Symbole dekodierbar als bei den Growth Codes. Außerdem kann, durch zielgerichtete Auswahl und Anpassung des zu übertragenden Codeworts, die vorgegebene Gradverteilung schneller und zuverlässiger erreicht werden. Auf diese Weise haben die im Netz versendeten Codeworte zum Sendezeitpunkt immer den für die Dekodierung optimalen Grad. Zusätzlich ist, wie bei den Growth Codes, kein Wissen über das System als Ganzes nötig. Ausschließlich die direkte Nachbarschaft muss bekannt sein. Selbst die Position oder Richtung der Senke muss nicht bekannt sein. Sensordaten können also ohne aufwändige Initialisierung sofort erhoben und weiterverarbeitet bzw. weitergeleitet werden. Analog zu den Growth Codes wird auch hier der sogenannte „Flaschenhalseffekt“ durch das Verteilen der Codeworte im Netz, statt dem direkten Routen zur Senke, vermieden. Außerdem ist es nicht nötig, den besten Weg zur Senke vorab zu speichern. Die Daten erreichen die Senke durch zufällige, redundante Verteilung im Netz zuverlässig.

V. Zusammenfassung und Ausblick

Redundante Datenhaltung ist aufgrund steigender Fehler- und Ausfallraten in Sensornetzen von zentraler Bedeutung. Growth Codes sind ein Verfahren, welches Daten redundant über eine zufällige Folge von Nachbarknoten verteilt und somit vielfach im Netz speichert. In diesem Artikel wurden Growth Codes zu Multicast Growth Codes erweitert. Diese sollen sowohl für hochdynamische, verteilte Sensornetze, als auch für statischere Fälle anwendbar sein und den Ausfall einzelner Knoten oder sogar ganzer Teilnetze kompensieren können. Um dies zu erreichen, wurde jeder Knoten potentielle Senke und soll letztendlich alle durch das Netz erhobenen Sensordaten rekonstruieren und speichern, wodurch erheblich zur Datenpersistenz beigetragen wird. Außerdem werden die im Knoten dekodierten Symbole zur zielgerichteten Auswahl und Anpassung des zu übertragenden Codeworts verwendet. Somit kann die vorgegebene Gradverteilung schneller und zuverlässiger erfüllt werden. Auf diese Weise wird zu jedem Zeitpunkt der für die Dekodierung jeweils optimale Grad eines gesendeten Codeworts eingehalten. Die in diesem Artikel vorgestellten Ideen sollen durch konkrete Realisierungen mit verschiedenen Gradverteilungen, Erweiterungen um zusätzliche Kodierungen und systematische Analysen zu Fehler- und Ausfalltoleranz in Zukunft ergänzt werden.

Literatur

[1] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, “Growth codes: Maximizing sensor network data persistence,” in *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4. ACM, 2006, pp. 255–266.

[2] J. Widmer, C. Fragouli, and J.-Y. Le Boudec, “Low-complexity energy-efficient broadcasting in wireless ad-hoc networks using network coding,” in *In Proceedings*, no. LCA-CONF-2005-016, 2005.

[3] D. Munaretto, J. Widmer, M. Rossi, and M. Zorzi, “Network coding strategies for data persistence in static and mobile sensor networks,” in *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks and Workshops, 2007. WiOpt 2007. 5th International Symposium on*. IEEE, 2007, pp. 1–8.

[4] ——, “Resilient coding algorithms for sensor network data persistence,” in *Wireless Sensor Networks*. Springer, 2008, pp. 156–170.

[5] Q. Huang, K. Sun, X. Li, and D. O. Wu, “Just fun: A joint fountain coding and network coding approach to loss-tolerant information spreading,” in *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 2014, pp. 83–92.

[6] N. Alon, M. Ghaffari, B. Haeupler, and M. Khabbazian, “Broadcast throughput in radio networks: routing vs. network coding,” in *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2014, pp. 1831–1843.

[7] M. Sathiamoorthy, A. G. Dimakis, B. Krishnamachari, and F. Bai, “Distributed storage codes reduce latency in vehicular networks,” *Mobile Computing, IEEE Transactions on*, vol. 13, no. 9, pp. 2016–2027, 2014.

Beobachtbarkeit und Nachrichtenverteilung für ein drahtloses Sensornetz

† Dr.-Ing. Florian Evers, ‡ Patrick Kalka
Fraunhofer-Institut für Integrierte Schaltungen IIS, Nordostpark 93, D-90411 Nürnberg
† florian.evers@iis.fraunhofer.de ‡ patrick.kalka@iis.fraunhofer.de

***Zusammenfassung*—Bei der Entwicklung von Kommunikationsprotokollen für drahtlose Sensornetze stellt sich das Problem der Beobachtbarkeit. Oftmals lassen sich Fehler nur durch eine ganzheitliche Sicht auf das Netz eingrenzen, was bei steigender Anzahl an Sensorknoten zu einer ernsten Herausforderung wird. Die Lösung stellt ein als „Overlay-Netz“ bezeichnetes Werkzeug dar. Es erlaubt die Beobachtung verteilter Sensorknoten, bietet die zentralisierte Konfiguration ganzer Netze, unterstützt bei der Verteilung von Firmware-Updates und entlastet große Sensornetze durch die Ausleitung von Nutzdaten.**

I. Motivation

Selbstorganisiert arbeitende drahtlose Multi-Hop-fähige Sensornetztechnologien sind komplex und erfordern für die Entwicklung und Fehlersuche eine umfassende Beobachtbarkeit der internen Zustände aller Sensorknoten. Die verteilte Dynamik größerer Netze kann nicht mehr durch punktuell eingesetzte Debugger beobachtet werden, und es muss während der Entwicklung möglich sein, alle Sensorknoten ohne Zeitaufwand mit aktualisierter Firmware zu bespielen.

Das hier als Overlay-Netz vorgestellte Werkzeug erfüllt diese Aufgaben für die s-net®-Sensornetztechnologie des Fraunhofer-Instituts für Integrierte Schaltungen IIS und erlaubt die direkte Beobachtung aller Sensorknoten ohne eine Beeinflussung der Luftschnittstelle durch zusätzliche Nachrichten [1].

II. Grundlagen s-net®

Die Sensornetztechnologie s-net® ermöglicht langlebige batteriebetriebene Installationen und eignet sich insbesondere für die Anwendungen „Smart Metering“ [2], verteilte Messdatenerfassung, „Asset Tracking“ und „Smart Objects“. Lange Tiefschlafphasen des Mikrocontrollers und des Transceivers erlauben Batterielaufzeiten bis hin zu mehreren Jahren. Durch Multi-Hop-Kommunikation mit adaptiver Kanalwahl können große Areale abgedeckt werden [3].

Das Fraunhofer IIS hat unterschiedliche Hardware-Referenzplattformen entwickelt. Das S6TAG besitzt einen MSP430-Mikrocontroller, einen CC1121-Transceiver mit SMD-Antennen für das ISM-Band um 868 MHz, eine JTAG-Schnittstelle zur Entwicklung sowie eine USB-Schnittstelle zum Datenaustausch und zur Speisung. Alternativ gibt es die vergleichbaren S7TAGs für 2400 MHz und die S5DIVs für einen Multi-Band-Betrieb auf 433 MHz und 868 MHz. Als Software kommt jeweils der s-net®-Protokollstapel zum Einsatz, welcher um kundenspezifische Anwendungen erweitert wird.

A. Baumtopologie

Die Topologie eines s-net®-Netzes ist ein Baum. Dank der Multi-Hop-Fähigkeit von s-net® lassen sich weiträumige Areale abdecken. Ein beispielhaftes Netz wird in Abbildung 1 gezeigt, wobei sich drei Knotentypen unterscheiden lassen:

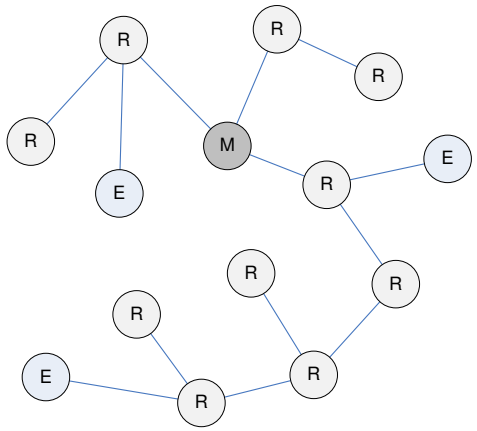


Abbildung 1. Der Masterknoten **M** bildet die Wurzel des aufgespannten Baumes. Routerknoten **R** in Funkreichweite verbinden sich mit dem Masterknoten. Das Netz vergrößert sich und weitere Knoten erhalten die Möglichkeit dem Netz beizutreten. Dieser Vorgang wiederholt sich bis alle Router- und Endknoten **E** Teil des Netzes geworden sind.

- Masterknoten: Jedes s-net®-Netz baut sich immer um einen Masterknoten herum auf, welcher als Wurzel der Baumtopologie den Kondensationskeim darstellt. Der Masterknoten bietet die zentrale Zeitbasis und ist für die Synchronität des TDMA-basierten Funkprotokolls verantwortlich. Er stellt zudem die zentrale Datensenke dar und benötigt daher immer eine Schnittstelle nach außen.
- Routerknoten: Routerknoten bilden das Rückgrat des Multi-Hop-fähigen Sensornetzes und erlauben die Abdeckung großer Areale. Sie assoziieren sich jeweils mit einem Elternknoten und erlauben dann ihrerseits die Assoziation von Kindknoten. Routerknoten weisen wegen des synchron arbeitenden Funkprotokolls von s-net® eine sehr geringe Stromaufnahme auf und können daher mit Batterien betrieben werden.
- Endknoten: Die Aufgabe von Endknoten besteht in der Generierung von Messdaten. Endknoten können keine eigenen Kindknoten annehmen und weisen eine geringere Stromaufnahme als Routerknoten auf. Werden in einem Szenario mobile Knoten benötigt, sollten diese vorzugsweise als Endknoten realisiert werden.

Weiterhin kann ein jeder Knoten per USB mit einem Gateway verbunden werden. Ein solches „Gateway“ ist ein Computer mit USB- und Netzwerkfähigkeit, beispielsweise ein „Raspberry Pi“-Miniaturrechner, mit installierter s-net®-Gateway-Software. Über solche Gateways findet der Nachrichtenaustausch zwischen dem Sensornetz und der Außenwelt statt, wobei wenigstens der Masterknoten an ein Gateway angeschlossen werden sollte. Allerdings ist der Einsatz von Gateways nicht auf den Masterknoten beschränkt; vielmehr lassen sich auch weitere oder gar alle Knoten eines Netzes an Gateways anschließen. Der Vorteil solcher „Entlastungsknoten“ wird in den Abschnitten IV-A2 und IV-A3 diskutiert.

B. Wegewahl im s-net®-Sensornetz

Für die Übertragung von Nutzdaten werden bei s-net® der Uplink und der Downlink unterschieden. Beide Richtungen werden durch Abbildung 2 illustriert und im Folgenden beleuchtet.

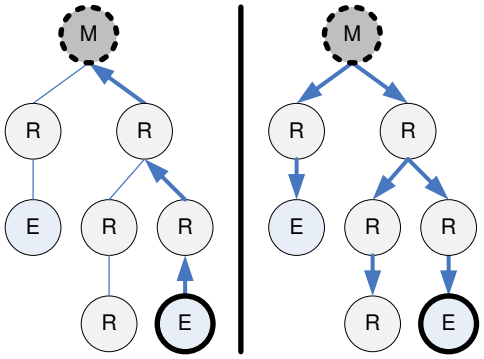


Abbildung 2. Für die Übertragung von Nutzdaten unterscheidet s-net® zwei Richtungen: im Uplink (links) werden Nachrichten des Endknotens **E** (fett) immer über den Elternknoten in Richtung der Datensenke am Masterknoten **M** übermittelt, wo sie das Sensornetz über ein Gateway (fett, gestrichelt) verlassen. In das Sensornetz eingespeiste Nachrichten (Downlink, rechts dargestellt) fluten die Baumtopologie und erreichen schließlich auch den adressierten Zielknoten **E**.

1) Datenübertragung im Uplink: Die von den Sensorknoten erfassten Daten müssen zu einem Gateway-Knoten übermittelt und von diesem über dessen Gateway an das Backend übertragen werden. Auch Monitoring-Nachrichten zur Anzeige interner Knotenzustände werden immer in dieser Richtung übertragen. Der Uplink entspricht somit der natürlichen Datenflussrichtung in einem Sensornetz zur weiträumigen Messdatenerfassung. Da am Masterknoten immer ein Gateway verfügbar ist, gestaltet sich die Wegewahl einfach: jeder Knoten verschickt seine Nachrichten immer an den jeweiligen Elternknoten, wodurch sie den direkten Weg die Baumtopologie entlang zum Masterknoten nehmen.

2) Datenübertragung im Downlink: Als Downlink wird die Kommunikationsrichtung ausgehend von Nutzer und Backend in Richtung des Sensornetzes bezeichnet. Nachrichten weisen hierbei immer eine Zieladresse aus dem Sensornetz auf, wobei zwischen Unicast- und Multicast-Adressierung unterschieden wird. Nachrichten im Downlink werden vom Einspeisepunkt aus in das Sensornetz geflutet, da s-net®-Knoten keine Routingtabelle besitzen. Nur wenn eine Nachricht über den Masterknoten eingespeist wird, kann sie alle Knoten des Sensornetzes unabhängig von der aktuellen Netztopologie erreichen. Typische Nachrichten betreffen die Konfiguration von Knoten oder dienen der Statusabfrage mittels Monitoring.

III. Problemstellung

Das bisherige Einsatzszenario von s-net® sieht lediglich am Masterknoten ein Gateway vor. Ein solches Gateway arbeitet als Vermittler von s-net®-Nachrichten zwischen dem Sensornetz und dem Internet. Anwender können die herausgeleiteten Nutzdaten mittels der IP-Adresse des Gateways abrufen und Nachrichten in das Sensornetz einspeisen. Für ein Update der Firmware müssen jedoch alle Sensorknoten eingesammelt, manuell mit einem aktualisierten Image bespielt und anschließend wieder korrekt ausgebracht werden.

Dieser Aufwand ist zwar für kleine Netze mit langer Laufzeit noch vertretbar, erschwert aber zunehmend die Arbeit der s-net®-Entwicklung, insbesondere da die Entwicklung eines verteilten Systems ein häufiges und möglichst automatisch ablaufendes Ausbringen neuer Firmware auf eine Vielzahl an verteilten Sensorknoten erforderlich macht. Neben dieser Herausforderung gilt es zudem das Problem der mangelhaften Beobachtbarkeit zu lösen:

Der Protokollstapel von s-net® bietet eine Vielzahl von Monitoring-Nachrichten an, welche dem Entwickler wichtige Informationen über den internen Zustand eines jeden Sensorknotens mitteilen können. Dieses Werkzeug ist jedoch nur einsetzbar wenn der zu untersuchende Sensorknoten erreichbar ist. Wenn einzig der Masterknoten mit einem Gateway verbunden ist, können auch nur diejenigen Knoten beobachtet werden, welche noch am Funknetz teilnehmen. Selbst dann steht man aber vor dem Problem, dass die zusätzlichen Monitoring-Nachrichten das Datenaufkommen im Sensornetz erhöhen es dadurch negativ beeinflussen. Die Datenrate von s-net® ist sehr gering mit einer Übertragungslatenz im höheren Sekundenbereich, wodurch die Erkennung und Deutung von Fehlverhalten mittels „Monitoring über die Luft“ erschwert wird.

Stünde für jeden Sensorknoten ein eigenes Gateway zur Verfügung, und könnten die Datenströme aller Gateways aggregiert an einem zentralen Sammelpunkt bereit gestellt werden, ließen sich alle genannten Probleme nachhaltig lösen. Mit dieser Erkenntnis war die Idee des als „Overlay-Netz“ bezeichneten Werkzeuges geboren.

IV. Das Overlay-Netz

Das Overlay-Netz besteht aus einer Vielzahl an Overlay-Rechnern, wobei Overlay-Rechner erweiterte Versionen der in Abschnitt II-A vorgestellten Gateways darstellen. An jedem dieser Overlay-Rechner können mehrere Sensorknoten über USB angeschlossen werden. Für jeden angeschlossenen Sensorknoten stellt der Overlay-Rechner die erwähnte Gateway-Funktionalität bereit. Einer der Overlay-Rechner übernimmt zudem die Aufgabe des Master-Gateways, welches dem Benutzer eine Weboberfläche zur Verwaltung sowie eine zentrale Anlaufstelle zur Interaktion mit seinen Sensornetz zur Verfügung stellt.

In jedem Büro und jedem Aufenthaltsraum der Abteilung Kommunikationsnetze wurde ein solcher Overlay-Rechner installiert, und es steht jedem s-net®-Entwickler frei, eigene Sensorknoten am abteilungsweiten Overlay-Netz zu betreiben.

A. Anwendungsfälle

Bei der Konzipierung des Overlay-Netzes standen drei Einsatzszenarien im Fokus:

1) Zentral gesteuertes Firmwareupdate: Die s-net®-Entwickler sollen in die Lage versetzt werden, neue Firmware-Images auf Knopfdruck in einer Weboberfläche auf alle ihre Sensorknoten ausbringen zu können. Da die Aktualisierung der Firmware von s-net®-Sensorknoten über USB bereits möglich war, sollen hierfür auch gleich die Overlay-Rechner mit herangezogen werden. Dabei lassen sich allerdings nur diejenigen Knoten aktualisieren, welche auch direkt über USB an einem Overlay-Rechner angeschlossen sind, aber hierbei wird eine beträchtliche Vereinfachung und Verringerung der Dauer von Firmwareupdates erwartet.

2) Beobachtbarkeit sicherstellen: Monitoring-Nachrichten eines jeden Knotens sollen nicht mehr über die Luftschnittstelle versendet werden, sondern sofort über USB an das jeweils angeschlossene Gateway übergeben werden. Die dabei von den verstreuten Gateways entgegen genommenen Nachrichten gilt es zu aggregieren und dem Nutzer an einem zentralen Sammelpunkt bereit zu stellen. Dabei ist zu beachten, dass mehrere Nutzer das Overlay-Netz benutzen werden und es nicht zu einer Vermischung von Nachrichten unterschiedlicher Gruppen von Sensorknoten kommt.

3) Entlastungsknoten: Für die Installation großer Sensornetze bietet sich der gezielte Einsatz dedizierter Entlastungsknoten an, welche alle im Uplink über sie laufenden Nachrichten abfangen und über ein angeschlossenes Gateway ausleiten. Auch hierbei müssen die Nachrichten aller Gateways aggregiert und an einem zentralen Sammelpunkt bereit gestellt werden. Dabei wird allerdings die Wegewahl im Downlink eine Herausforderung, da nicht mehr jeder Knoten über USB erreicht werden kann, sondern ein geeigneter Entlastungsknoten als Einspeisepunkt identifiziert werden muss.

V. Wegewahl im Overlay-Netz

Die Wegewahl innerhalb des Overlay-Netzes hat die Aufgabe die Luftschnittstelle des Sensornetzes weitestgehend zu entlasten. Hierbei müssen Uplink und Downlink getrennt voneinander betrachtet werden, aber auch zwischen Unicast- und Multicast-Adressierung unterschieden werden.

A. Uplink im Overlay-Netz

Die Wegewahl für das Herausleiten von Nachrichten aus dem Sensornetz ist trivial: jeder Knoten muss Nachrichten nur solange „nach oben“ in Richtung der Wurzel leiten, bis diese auf einen Knoten mit angeschlossenem Gateway stößt. Solche Gateway-Knoten leiten nach außen gerichtete Nachrichten über USB aus dem Sensornetz heraus. Dies ist spätestens bei Masterknoten der Fall, welcher zur Ausleitung ein angeschlossenes Gateway benötigt.

In großen Netzen mit hohem Datenaufkommen bietet es sich an, zusätzliche Gateway-Knoten als „Entlastungsknoten“ einzusetzen, welche durch das Herausleiten aller Nachrichten das Datenaufkommen in Richtung des Masterknotens verringern. Es ist wichtig zu unterstreichen, dass in der Praxis nicht jeder Knoten mit einem Gateway betrieben wird. Soll ein solcher Einsatzfall mit Hilfe des Overlay-Netzes beobachtet werden, ist jedoch für eine sorgfältige Konfiguration zu sorgen: Jeder Knoten verfügt dann zwar über ein angeschlossenes Gateway zwecks Beobachtung, aber nicht jeder dieser Knoten soll zum Ausleiten und Einspeisen von Nutzdaten herangezogen werden.

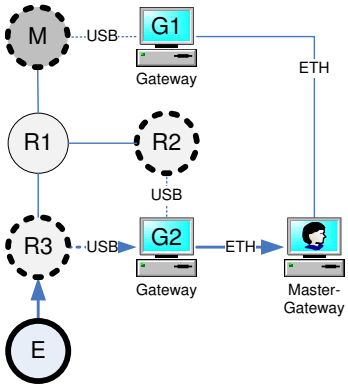


Abbildung 3. Dieses Sensornetz besteht aus einem Masterknoten (**M**), drei Routerknoten (**R1**, **R2** und **R3**) und einem Endknoten (**E**). Die Knoten **M**, **R2** und **R3** verfügen über ein Gateway und sind daher fett und gestrichelt dargestellt. Nachrichten des Endknotens **E** (fett) werden von **R3** aus dem Sensornetz herausgeleitet und dem Gateway **G2** übergeben. **G2** leitet die Nachrichten an das Master-Gateway weiter, welches sie schlussendlich für den Benutzer zum Abruf zur Verfügung stellt.

Eine solches Szenario wird in Abbildung 3 dargestellt. Der Masterknoten **M** ist per USB an Gateway **G1** angeschlossen, und die Routerknoten **R2** und **R3** hängen beide an Gateway **G2**. **G1** und **G2** sind mittels Ethernet mit dem Master-Gateway verbunden und bilden das Overlay-Netz. Sowohl der Routerknoten **R1** als auch der möglicherweise mobile Endknoten **E** sind mit keinem Overlay-Rechner verbunden. Da **R3** die Nachrichten von **E** abfängt wird das Datenaufkommen am Masterknoten **M** verringert.

B. Downlink im Overlay-Netz

Nutzdaten seitens des Benutzers, welche in das Sensornetz eingespeist werden sollen und einen konkreten Zielknoten adressieren, können prinzipiell immer dem Masterknoten übergeben werden. Der Masterknoten hat jedoch keine andere Wahl als das komplette Sensornetz zu fluten, sodass es ebenso wie im Uplink am Masterknoten zu einem Kapazitätsproblem kommen kann. Mittels zusätzlicher Entlastungsknoten lässt sich das Sensornetz jedoch auch in Downlink-Richtung entlasten, wobei jedoch zwischen Unicast- und Multicast-Nachrichten unterschieden werden muss.

1) Unicast-Adressierung: Abbildung 4 zeigt dasselbe Sensornetz wie aus Abbildung 3, jedoch mit umgekehrter Übertragungsrichtung: der Benutzer möchte eine Nachricht an den Endknoten **E** schicken, und zwar über den geeignetsten Weg im Sinne einer Entlastung des Sensornetzes.

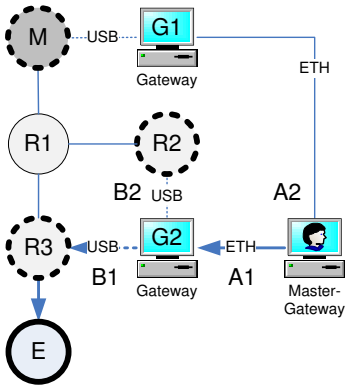


Abbildung 4. Zur Einspeisung von Nachrichten in das Sensornetz (Downlink) werden diese vom Nutzer dem Master-Gateway übergeben. Damit der Ziel-Endknoten **E** auf dem kürzesten Wege erreicht werden kann, müssen alle Gateways eine Wegewahlentscheidung fällen. Das Master-Gateway schickt die Nachrichten über den Weg **A1** in Richtung des Gateways **G2**, welches sie mittels USB **B1** an den Routerknoten **R3** übergibt. Ab hier wird der Zweig des Sensornetzes geflutet und der Zielknoten **E** erreicht.

In dieser Richtung ist die Wegewahl jedoch komplexer, da das Overlay-Netz Informationen über die Topologie des Sensornetzes benötigt. So sollte das Master-Gateway den Weg **A1** in Richtung des Gateways **G2**, und das Gateway **G2** den USB-Port **B1** zwecks Übergabe an den Routerknoten **R3** wählen. **R3** braucht dann nur noch den ihm unterstellten Zweig des Sensornetzes zu fluten um den Zielknoten **E** zu erreichen. Unklar ist an dieser Stelle jedoch noch, woher das Master-Gateway und **G2** ihr Wissen bezüglich der Netztopologie erhalten haben. Dies wird in Abschnitt V-C beleuchtet.

2) Multicast-Adressierung: s-net® unterstützt mehrere Multicast-Adressen, mit welchen bestimmte Typen von Knoten oder sogar alle Knoten eines Netzes adressiert werden können. Die bisherige Verwendung bestand darin, solche Multicast-Nachrichten dem Masterknoten zu übergeben und somit das gesamte Sensornetz zu fluten. Dies ist in großen Sensornetzen mit wenigen Einspeisepunkten sinnvoll, benötigt allerdings Kapazität und Zeit bis die Nachricht durch das gesamte Sensornetz propagiert worden ist. Sind hingegen viele oder sogar alle Sensorknoten am Overlay-Netz angeschlossen, kann es sich lohnen, Multicast-Nachrichten auf eine Vielzahl an Unicast-Nachrichten abzubilden, denn das Overlay-Netz kann dann jedem Sensorknoten „seine“ Nachricht sogar direkt über das Gateway zustellen. Da hierbei die Luftschnittstelle nicht belastet wird und sogar Sensorknoten, welche momentan nicht im Sensornetz eingebucht sind, erreicht werden können, bietet sich dieser Mechanismus für die Übertragung von Monitoring-Nachrichten an.

C. Aufbau der Weiterleitungstabellen

Für die Weiterleitung eingespeister Nachrichten (Downlink) benötigen alle Gateways einschließlich des Master-Gateways eine Weiterleitungstabelle, welche in Einklang mit der aktuellen Topologie des Sensornetzes steht. Statt jedoch die sich stetig ändernde Topologie des Sensornetzes nachzubilden reicht es aus, den herausgeleiteten Uplink-Datenverkehr zu beobachten. Hierbei ist bereits der kürzeste Weg über den korrekten Entlastungsknoten gefunden worden, daher gilt dieser Weg auch als beste Wahl für eine Zustellung im Downlink. Im Szenario aus Abbildung 3 beobachtet das Master-Gateway, dass ihm eine Nachricht von Knoten **E** über Gateway **G2** zugestellt wird. Es folgert daraus, dass Nachrichten an Knoten **E** idealerweise über das Gateway **G2** zugestellt werden sollten. Diese Information, welche auf Beobachtung der Absenderadressen ausgeleiteter Nachrichten basiert, wird in Form einer „Source Address Table“ (SAT) vermerkt.

Jedes Gateway pflegt eine solche SAT, jedoch sieht diese für die Gateways **G1** und **G2** geringfügig anders aus als die des Master-Gateways: die Wegewahl an **G2** bezieht sich auf die Auswahl der korrekten USB-Schnittstelle zwecks Übergabe der Nachricht an denjenigen Routerknoten, welcher dem Zielknoten **E** topologisch am nächsten liegt.

Die SATs werden laufend durch Beobachtungen aktualisiert und erlauben eine schnelle Anpassung der Wegewahl an sich ändernde Netztopologien. Es sind jedoch ein paar Fallstricke zu beachten: solange ein Knoten keine Nachrichten gesendet hat, existieren auch keine Einträge über ihn in den SATs. Das Overlay-Netz hat dann keine andere Wahl, als Nachrichten an unbekannte Adressen an den Masterknoten zwecks Flutung zu übergeben. Weiterhin sollten Knoten, welche nicht direkt am Overlay-Netz angeschlossenen sind, nach einem Wechsel des Elternknotens sofort eine Nachricht verschicken, damit zeitnah in den Gateways die SATs aktualisiert werden und somit das Risiko falscher Weiterleitungen im Downlink verringert wird.

VI. Zusammenfassung

Alle drei der folgenden Einsatzszenarien motivierten die Entwicklung des vorgestellten Overlay-Netzes und gelten mittlerweile als erfolgreich umgesetzt:

- Die zentralisierte Verwaltung von Knoten eines dauerhaft installierten Sensornetzes über eine Weboberfläche, insbesondere zur benutzerfreundlichen Konfiguration und zur Aktualisierung der Firmware aller Knoten,
- die Verbesserung der Beobachtbarkeit durch eine direkte Übertragung von Monitoring-Nachrichten über USB ohne eine Beeinträchtigung der Luftschnittstelle des Sensornetzes, und
- die Möglichkeit der Entlastung großer Sensornetze an neuralgischen Punkten bei kundenspezifischen Einsatzszenarien.

Das vorgestellte Overlay-Netz stellt eine Schlüsseltechnologie dar, welche die Erforschung und Entwicklung komplexer selbstorganisierter Vernetzungstechnologien in der Abteilung Kommunikationsnetze am Fraunhofer-Institut IIS erheblich vereinfacht.

Literatur

[1] P. Kalka, “Nachrichtenverteilung in einem verteilten Testsystem für ein drahtloses Sensornetz,” Bachelorarbeit, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Deutschland, Jul. 2015.

[2] M. Wenzel, “Asset Tracking mit drahtlosen Sensornetzen,” in *13. Ilmenauer TK-Manager Workshop*. Technische Universität Ilmenau: Telekommunikations-Manager (TKM) e.V., Sep. 2014, pp. 11 – 17.

[3] F. Evers, “Adaptive Störervermeidung in einem drahtlosen Multi-Hop-Sensornetz für Asset Tracking,” in *Scientific Reports – Journal of the University of Applied Sciences Mittweida*, ser. Kommunikationstechnik, no. 2. Hochschule Mittweida, Nov. 2014, pp. 23 – 25, 23. Internationale Wissenschaftliche Konferenz Mittweida IWKM 2014.

Exploiting Already Deployed Sensor Networks for Opportunistic Emergency Communication Services

Silvia Krug, and Jochen Seitz
Communication Networks Group, Technische Universität Ilmenau, Germany
{silvia.krug, jochen.seitz}@tu-ilmenau.de

***Abstract*—First responder networks face highly intermittent connectivity during early stages of a mission. Delay tolerant approaches are able to handle that at the cost of high additional delays. To leverage this effect alternatives such as mobile data ferries or the placement of additional nodes are discussed. However, these approaches are suboptimal because they create additional communication maintenance overhead for the first responders. Already deployed sensor networks are likely to survive the disaster, at least partially. Therefore, we exploit in this paper whether it is possible to use these nodes as additional relays and what prerequisites have to be fulfilled for that.**

***Index Terms*—Wireless Sensor Networks; Opportunistic Communication; Emergency Scenarios.**

I. Introduction

Natural or man-made disasters usually affect deployed communication infrastructure as well. Therefore, the first responders have to set up their own ad hoc based network, at least during the initial phase of the relief work. The coverage range of these networks is however limited, resulting in frequent disruptions. Since all first responders should return to the main incident coordination point eventually, asynchronous delay tolerant communication is possible in order to handle the intermittent connectivity.

For mission-critical information the experienced delay in DTN networks is however too large, if only the first responder devices are used [1]. Possible alternatives include the positioning of additional nodes or the deployment of mobile data ferries. Mobile data ferries collect messages from one part of the network and deliver them to the other parts. However, their number and movement range is most likely limited as well. Additional nodes have to be guarded against theft, potentially relocated during the mission and collected after the mission. Both options are suboptimal, as they require additional organizational overhead.

In developed countries, wireless sensor networks (WSNs) are currently deployed as part of smart buildings and intelligent transportation systems [2]. Besides that, sensor networks are employed to monitor wildlife [3], wildfires [4] or other potentially interesting regions (e.g. [5]) in remote areas as well. While the main purpose of the sensor network might be gone due to the disaster, the battery powered sensor nodes will stay operational for some time because they are designed to run for years without requiring any maintenance. Even if some sensor nodes are also affected by the outcome of the disaster, it is likely that some of them remain operational. Therefore, a set of already deployed sensor nodes is available which might help to bridge the partitioned first responder networks.

To do that, several requirements have to be fulfilled. In this paper, we will discuss the requirements and possible realization options for such a system.

The paper is organized as follows. In the next section we will briefly introduce the mentioned requirements and discuss related work in the context of disaster communication. Then, in Section III, core components for such a system and their realization options are explained. Afterwards, we show the impact of our concept for an example scenario in Section IV. Finally, in Section V, we draw a conclusion and discuss potential future work.

II. Related Work

Wireless sensor networks are already deployed for several applications like disaster preparedness or emergency management.

Yang et al. identified a number of opportunities where WSNs might assist on-site first responder safety in [6]. The options were derived from interviews with first responders in the UK. Based on these options, the authors propose systems to provide data related to the fire itself such as smoke, temperatures, and to track first responders. However, they only discuss the potential solutions but do not give any details. These systems focus on sensing and forward the information as efficiently as possible. Networking between the sensor and actuator nodes is required but is usually not considered as a means to support user communication.

Building evacuation systems is one example where sensor networks are used specifically for a disaster scenario. In [7] the authors review several related systems and show how sensors can be used to guide victims and first responders. The sensor networks are usually fixed but are able to detect mobile nodes as well as hazards.

The authors of [8] present a similar study as in [6] based on interviews with German fire fighters. They focus on front line communication and show how essential communication is for the overall mission goal. Then they focus on a 802.15.4-based system to support autonomous information gathering for incident commanders that involves the deployment of WSN nodes by the first responders. A similar approach is presented in [9]. There, the WSN is used to monitor vital signs and positions of fire fighters as well as the surrounding environmental conditions.

Besides these application-based WSN solutions related to disaster scenarios, approaches exist that discuss the usage of delay tolerant networks (DTNs) or opportunistic communication for WSN communication.

DTN capabilities for WSN nodes are introduced in [10] and [11]. In both approaches, the goal is to prevent data loss due to partially mobile nodes. To achieve this, the authors present independent DTN implementations for Contiki. Another option to handle unreliable links in WSNs is to use opportunistic communication. Such approaches aim to exploit the broadcast nature of the wireless medium to deliver messages to more than one relay at the same time [12], [13]. This allows to enhance the energy efficiency of the WSN. However, these approaches only allow the sensor nodes themselves to benefit from opportunistic or delay tolerant communication.

On the other hand, there are several approaches discussing the integration of additional nodes and exploiting their capabilities to enhance the overall performance. Nagy et al. [14] propose a system with access points that allow non-DTN nodes to benefit from DTN or opportunistic communication. While this integration itself is interesting, the authors do not cover the aspect of non-DTN nodes as additional relays. Another approach considers DTN-enabled mobile nodes to interconnect isolated WSNs [15]. They focus on efficient buffer management and data collection. This example shows how WSNs can benefit from DTN-enabled nodes, but not vice versa.

While these solutions are not directly related to the proposed scheme in this paper, they prove that opportunistic communication via sensor nodes is possible in general.

III. Conceptual Design

In this section, we present our concept of an emergency mode for WSN that will allow authorized first responders to opportunistically communicate via deployed WSN infrastructure.

A. Prerequisites

In order to allow opportunistic communication via 802.15.4 based sensor networks, the first responders have to be equipped with a device that is able to communicate with the sensors. This would ideally be a multi-interface node with at least one 802.15.4 interface. Without such an interface, an integration of both concepts is out of question. Figure 1 shows this schematically and presents one possible integration option of the required protocol stacks. Since sensor networks are also an option to monitor different parameters of the first responders or their equipment [16], [9], such a device can be assumed as present.

The proposed system should be able to transfer different types of information as required by the mission coordinators [8]. Among others, this could include sensor readings from the body area network of the first responders as well as recorded voice messages, or text messages. The nominal 802.15.4 data rate (250 kbit/s) should be sufficient for this.

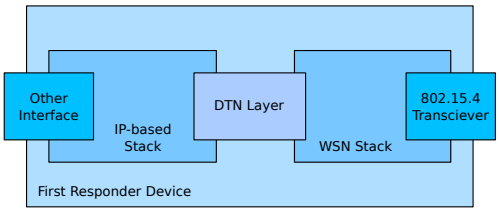


Fig. 1. Schematic concept of a first responder device with multiple interfaces.

However, the authors of [17] show that a lower effective data rate has to be expected. This would however still be sufficient for high priority messages with low data rate requirements, such as voice recordings, text messages or single images.

B. General Realization Options

Based on the previous reviews, there are two general options to integrate the desired communication mechanisms:

1) Employ all nodes with DTN capabilities
2) Allow first responder devices to opportunistically exploit existing nodes as relay

The first option would require all nodes to employ a DTN-enabled protocol stack as described in [10], [11]. This might be interesting for WSN applications with mobile nodes, such as the wildlife detection systems. But it would add additional overhead to any other deployment under normal operations, especially if the nodes were manually positioned to guarantee a certain coverage.

The second option would basically invert some of the mechanisms described in the previous section. Instead of using DTN-enabled nodes to collect and transmit sensor data, the mobile node would opportunistically offload its data to another DTN-enabled node via the underlying sensor network. The potentially multi-hop communication via the remaining underlying WSN nodes would therefore appear as a 1-hop DTN link. Figure 2 shows an example of such a network topology. In this case, no additional changes to the protocol stack of the sensor nodes are required, and thus no additional overhead is required. Besides that, such a system would still be able to in-cooperate DTN-enabled sensor nodes as well if they are present. Therefore, we focus on details of this option in this work.

C. Emergency Mode

When realizing the second option discussed above, we propose the integration of an emergency mode or application into the sensor network. If this mode is activated, the sensor nodes will relay the messages between the first responder devices. To that efficiently, they should stop from duty-cycling for energy efficiency, and eventually omit their original sensing tasks to release bandwidth.

While the cancellation of any energy-saving mechanisms is suboptimal for traditional sensor networks, it is feasible in case of disasters because the safety of first responders and victims has the highest priority and the original purpose of the

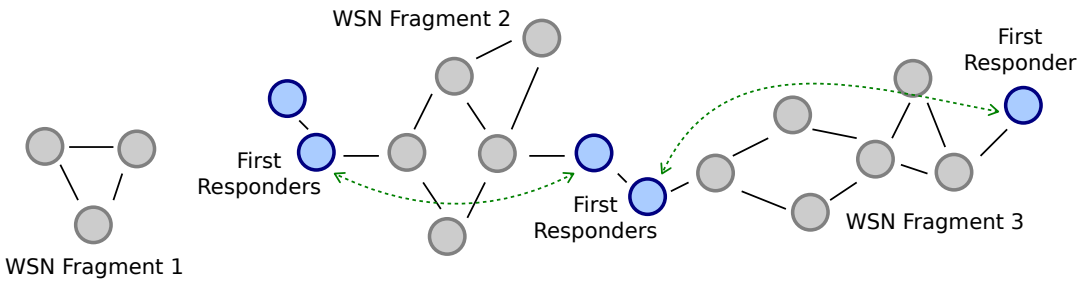


Fig. 2. Example topology of proposed solution showing several WSN fragments (gray nodes) of the original network and several first responder groups (blue nodes) communicating via the fragments in an opportunistic scheme (dashed arrows).

sensors might be gone as well. Whether the nodes should omit their initial task should be selected carefully, depending on the purpose of the sensor network. Especially, if the network was used to monitor the environment with respect to potential disasters (e.g. wildfires or landslides) the information gathered by the sensors might be useful for the first responders as well. In such a case, the overall mission will benefit from a combination of opportunistic first responder communication on one hand and the opportunistic offloading of the gathered data from the sensor nodes to the first responder devices and eventually to the mission coordinators on the other hand.

The activation and deactivation of the emergency mode should be based on a special message transfered by the first responder nodes via the 802.15.4 interface to the closest sensor node. Once the emergency mode is activated on the node, it broadcasts this information to all its neighbors and thus the mode is subsequently activated in all remaining sensor nodes. This procedure allows a transition from possibly broken normal operation modes to the emergency communication scheme. Even if currently no further first responder node is present, this initial activation phase can reduce the required setup time for future encounters. Once enabled, the senor nodes are able to forward messages between different first responders by effectively creating a local contact opportunity for the DTN nodes within reach of the sensor network, that would not be connected otherwise.

Ideally, the transition should require no additional reconfiguration on the sensor nodes. Therefore, the required discovery and configuration options have to be performed at the first responder device. This also includes the discovery of potential contacts.

D. Open Challenges

In order to enable our proposed scheme, several challenges have to be addressed at the sensor nodes.

Security Since sensor nodes are usually trimmed on energy efficiency, the emergency mode should only be available to authorized DTN nodes of first responders. Any implementation should ensure that the opportunistic usage is limited to those nodes only. Authentication codes could be one option for this.

Node Discovery Both the devices of the first responders and the sensor nodes themselves have to be able to discover and integrate new nodes into their network, in order to identify potential bridging zones. If only one first responder device is connected to the sensor network it should not offload data because it is not clear when the next communication opportunity arises and messages would have to be buffered within the sensor nodes in this case.

Routing If only some nodes of the original sensor network survive the disaster, the initial routing towards a base station does not apply anymore. Instead, new DTN nodes act as source and/or sink of communication, that were not part of the original deployment and thus changing the objective of the data flow. Therefore, the deployed routing mechanism has to be able to adapt to the changed requirements. While flooding is usually discussed as a valid option for opportunistic approaches, a careful evaluation of these schemes is required for this application due to the scarce resources of available storage, energy and bandwidth.

Addressing In order to find a route to another node this node has to be addressed properly. This is especially true for the proposed scenario, where foreign first responder devices are dynamically added to an existing network. A possible solution to uniquely address the nodes is to use IPv6 addresses that are already used in 6LoWPAN-based WSNs. The assigned addresses have to be unique and could be derived from the MAC-address of the 802.15.4 interface. In addition to that, name resolution might be required to map DTN names to the corresponding IPv6 address.

These points show that further work is needed in oder to integrate the WSN and DTN communication.

IV. Example Scenario

In this section we present an example scenario to illustrate the impact of our concept.

In the aftermath of disasters, fast and efficient search and rescue (SAR) for missing people is required in order to minimize the number of deaths. This is true for natural disasters such as earthquakes or floods but also for man-made

disasters such as terrorist attacks or large scale accidents. Panic among the affected people can lead to a large search area of several square kilometers that should be covered by the first responders.

To do that, the first responders will setup a coordination point, responsible for dispatching SAR units efficiently. Every team should stay in contact with this entity if possible. However, pure ad-hoc communication is not able to cover the complete area.

This results in disruptions as teams move to their respective search location. DTNs can mitigate this, but that means a message is carried as long as it might take the rescuer to return to an area with connectivity. Our previous results [1] showed that this can take up to an hour, depending on the distances that have to be covered and the speed of the movement. This is not acceptable, especially if crucial information regarding found victims or hazards has to be transmitted. Ideally, such information should be transferred within 3 minutes, according to emergency services radio regulations[18], before giving up a connection as broken. Therefore, any solution that is able to reduce the delay, even if it is within a relatively large range of several seconds to a few minutes, would improve the situation. Already deployed nodes that are still functioning are suitable candidates for this.

The total data volume is rather low, if we assume voice messages that resembles short command-like or communication which is recorded. Besides that, these messages are not generated with a high, periodic frequency. One message per minute representing 5 to 10 seconds of voice would be sufficient. Assuming 64 kbps for the voice recording and no further compression, this results in small messages.

However, several messages could be buffered if no further contacts were available. Once the node discovers a contact via the WSN, it should start to offload these messages. Whether all messages can be transferred is depending on the duration of this contact. It should be noted, that even partial delivery of the buffered messages is already an enhancement to the current situation.

V. Conclusion and Future Work

In this paper, we discussed existing options to integrate WSNs into emergency response. While the existing approaches mainly focus on sensing tasks and data dissemination of the WSN nodes, we present a conceptual scheme to aid the first responder communication. We then discussed potential realization options for such a scheme and identified open challenges. Based on an example scenario, we showed some possible benefits of our scheme.

WSN assisted DTN communication is one promising approach to further enhance first responder communication. Already deployed sensor networks have a high possibility to survive disastrous events at least partially. These nodes might be efficient relays for otherwise disconnected first responder groups.

Future investigations will focus on the evaluation of our scheme and possible solutions for the identified challenges.

References

[1] S. Krug, S. Schellenberg, and J. Seitz, "Impact of Traffic and Mobility Patterns on Network Performance in Disaster Scenarios," in *10th ACM MobiCom workshop on Challenged networks (CHANTS)*. Paris, France: ACM, Sep. 2015, (accepted).

[2] G. S. Tewolde, "Sensor and Network Technology for Intelligent Transportation Systems," in *International Conference on Electro/Information Technology (EIT)*. Indianapolis, IN, USA: IEEE, May 2012, pp. 1–7.

[3] S. Ehsan, K. Bradford, M. Brugger, B. Hamdaoui, Y. Kovchegov, D. Johnson, and M. Louhaichi, "Design and Analysis of Delay-Tolerant Sensor Networks for Monitoring and Tracking Free-Roaming Animals," *Transactions on Wireless Communications*, vol. 11, no. 3, pp. 1220–1227, 2012.

[4] A. Ko, N. Lee, R. Sham, C. So, and S. Kwok, "Intelligent wireless sensor network for wildfire detection," *WIT Transactions on Ecology and The Environment*, vol. 158, pp. 137–148, 2012.

[5] T. Naumowicz, R. Freeman, H. Kirk, B. Dean, M. Calsyn, A. Liers, A. Braendle, T. Guilford, and J. Schiller, "Wireless Sensor Network for Habitat Monitoring on Skomer Island," in *35th Conference on Local Computer Networks (LCN)*. Denver, CO, USA: IEEE, Oct. 2010, pp. 882–889.

[6] Y. Yang, U. Prasanna, L. Yang, and A. May, "Opportunities for WSN for Facilitating Fire Emergency Response," in *5th International Conference on Information and Automation for Sustainability (ICIAFs)*. Colombo, Sri Lanka: IEEE, Dec. 2010.

[7] E. Gelenbe and F.-J. Wu, "Large scale simulation for human evacuation and rescue," *Computers & Mathematics with Applications*, vol. 64, no. 12, pp. 3869–3880, 2012.

[8] M. Scholz, D. Gordon, L. Ramirez, S. Sigg, T. Dyrks, and M. Beigl, "A Concept for Support of Firefighter Frontline Communication," *Future Internet*, vol. 5, no. 2, pp. 113–127, 2013.

[9] H. Will, T. Hillebrandt, and M. Kyas, "Wireless Sensor Networks in Emergency Scenarios: The FeuerWhere Deployment," in *1st ACM international workshop on Sensor-Enhanced Safety and Security in Public Spaces*. ACM, 2012, pp. 9–14.

[10] L. Bruno, M. Franceschinis, C. Pastrone, R. Tomasi, and M. Spirito, "6LoWDTN: IPv6-Enabled Delay-Tolerant WSNs for Contiki," in *7th International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*. Barcelona, Spain: IEEE, Jun. 2011, pp. 1–6.

[11] G. von Zengen, F. Büsching, W. B. Pöttner, and L. Wolf, "An Overview of μDTN: Unifying DTNs and WSNs," in *11th GI/ITG KuVS Fachgespräch Drahtlose Sensornetze (FGSN)*, Darmstadt, Germany, Sep. 2012.

[12] G. Schaefer, F. Ingelrest, and M. Vetterli, "Potentials of Opportunistic Routing in Energy-Constrained Wireless Sensor Networks," in *Wireless Sensor Networks*. Springer, 2009, pp. 118–133.

[13] S. Guo, L. He, Y. Gu, B. Jiang, and T. He, "Opportunistic Flooding in Low-Duty-Cycle Wireless Sensor Networks with Unreliable Links," *Transactions on Computers*, vol. 63, no. 11, pp. 2787–2802, 2014.

[14] M. Nagy, T. Kärkkäinen, and J. Ott, "Enhancing Opportunistic Networks with Legacy Nodes," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 18, no. 3, pp. 10–18, 2015.

[15] Y.-C. Tseng, F.-J. Wu, and W.-T. Lai, "Opportunistic data collection for disconnected wireless sensor networks by mobile mules," *Ad Hoc Networks*, vol. 11, no. 3, pp. 1150–1164, 2013.

[16] S. Denef, D. Keyson, and R. Oppermann, "Rigid Structures, Independent Units, Monitoring: Organizing Patterns in Frontline Firefighting," in *CHI Conference on Human Factors in Computing Systems*. Vancouver, BC, Canada: ACM, May 2011, pp. 1949–1958.

[17] F. Büsching and L. Wolf, "Chancen für den Einsatz von unterbrechungstoleranten Kommunikationsprotokollen in drahtlosen körpernahen Netzwerken im medizinischen Umfeld," in *42th Annual Conference of the Gesellschaft für Informatik e.V. (INFORMATIK 2012)*, vol. 208, Braunschweig, Germany, Sep 2012.

[18] N. Eulig, "BOS-Sprechfunk," THW OV Northeim, Tech. Rep., Jan. 2010.

Towards an OS-independent Code-Update Function for WSN

Thomas Basmer Ievgen Kabin Mario Schölzel
IHP
Frankfurt (Oder), Germany
Email: [basmer|kabin|schoelzel]@ihp-microelectronics.com

***Abstract*—This paper presents the concept of an operating system independent code update function for wireless sensor networks. The code update is separated from the application and the operating system in order to provide a basic input/output functionality that can be used to update the whole operating system. Moreover, we show that such an I/O-functionality can be used to handle particular faults in sensor nodes making them more reliable. In order to achieve this goal it is important that the code-update functionality has a very small memory foot-print. For that reason we present preliminary results of a prototype implementation demonstrating the feasibility of the concept.**

I. Introduction

Wireless sensor networks (WSN) will be a major part of the IT-infrastructure for industry 4.0, smart cities, smart agriculture, smart homes, etc., resulting in heterogeneous large and long living WSNs. The maintenance of these networks will became a key issue, because hundreds or even thousands of nodes are distributed for sensing and controlling. Furthermore, these networks will grow and evolve over time, meaning that heterogeneous motes and heterogeneous operating systems (OS) may be used to run the same or similar application and protocols. However, changing an OS, or even an OS-version may require significant changes to an application and may incorporates incompatibility at protocol layer. In order to overcome these problems we propose a cross-platform (CP) concept that provides a unified interface to different OS and hardware motes as discussed in [1] and [2]. The CP can be considered as an abstraction layer for the OS-specific interface (see fig. 1) in order to allow for a simple porting of applications even from one OS to another. The protocol stack will be implemented based on the CP-interface. Hence established and well tested implementations can be used on different sensor node platforms and within different OS. This reduces the development costs and offers compatibility across various platforms and for evolving WSN. A mandatory part of such evolving and long living systems will be a code update functionality, which becomes difficult, when using different OS and/or hardware motes are used, because code-update functionalities provided by different OS varies significantly. Therefore we propose an OS independent wireless code update (WCU) function, which enables the reprogramming of dedicated nodes in large WSN. The basic concept of integrating the code-update functionality into an OS and CP-platform concept is shown in figure 1. The WCU is a separated function that does not use OS-specific functionality. Based on a trigger (e.g. provided from the application), the WCU functionality is activated and over-takes the control of the system. This

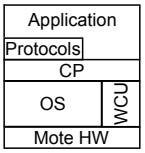


Fig. 1: Block diagram of a cross platform approach

allows for the replacement of the full OS-, CP-, application-, and WCU-image. Such a WCU functionality may be used for patching software bugs in the application as well as in the OS. Moreover it allows for handling permanent faults in the flash-memory of the mote. In microcontroller designs, as used for embedded systems, memory occupies up to 95 % of the chip area, and in particular flash memory suffers from aging effects in long living systems causing a permanent damage of some memory cells. Appropriate techniques to handle such faults were described in [3], [4], [5]. Periodically the flash-memory can be checked for such faults by a self-test program. Detected faults can be handled by installing a new image that avoids the usage of defective flash-memory cells. However, the test-program as well as the WCU functionality are subject to such faults, too. In order to reduce the probability that these functionalities will be affected by memory faults, their memory foot print must be very small. For that reason the WCU provides a very small kernel that is moved into the RAM of the mote and executed there. This kernel has full access to the flash-memory, such that the old image can be completely replaced, if necessary.

The rest of the paper is structured as follows. In II the related work is presented. After this our approach is presented in III. The paper closes with a description of our prototype implementation in IV and the conclusion in V.

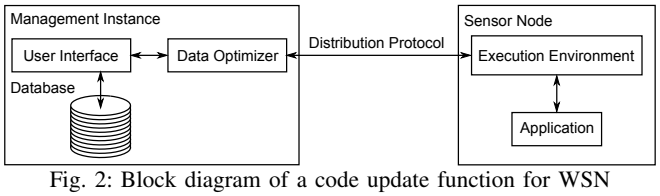


Fig. 2: Block diagram of a code update function for WSN

II. Related Work

WCU approaches are already available for WSN. In general they are part of the OS. As postulated in [6] a code update function can be divided into several parts: user interface, data optimizer, database, distribution protocol (DP) and execution environment (EXE). The structure of such a scheme is shown

in figure 2. Most available literature is related to DP and EXE because both topics have much space for improvements. Good summaries of these topics are [6] and [7].
DPs are subdivided into code consistency maintenance protocols (CCMP) and update delivery protocols. CCMPs ensure that all nodes have the same software version installed. It differentiates between user or node initiated updates. Sensor network management system (SNMS) [8] is an example of an user initiated update and Trickle [9] for node initiated updates. In SNMS the user has to gather information about the software in the network. Trickle uses information provided by neighboring nodes to decide if an update is necessary. DIP [10], DHV [11] and multicast-based code redistribution protocol (MCP) [12] are also node initiated CCMPs.
There is also a lot of work available for update delivery protocols. Here nodes have different software or software versions installed. The multi-hop, over-the-air code distribution protocol (MOAP) [13] uses a store and forward approach. Each node provides its software to other nodes in its one hop neighborhood. Other approaches use a three-way handshake for the update process. The first step is an advertizing phase where nodes provide information about available updates. In the second phase a node requests an update at a dedicated node and during the third phase the data is transmitted. It is a very popular approach used by protocols like sensor protocols for information via negotiation (SPIN) [14], Trickle, Deluge [15], Synapse [16] and Stream. The main differences of these protocols are the process of the advertisement phase and the handling of lost segments. The advertisement phase differs in:

- the interval between advertisements
- the trigger activating the sending of advertisements
- the provided information (one node or neighbors too)
- the structure of the advertisement (e.g hash value)

Lost data can be handled using retransmissions or special codes (e.g. Fountain codes) as in Synapse. Such codes enable a recalculation of lost packets avoiding retransmission.
For EXE three different approaches are discussed in literature [6]. The first one is a monolithic environment where application, kernel and protocols are combined into one image at compile time. Modular environment is the second approach. Here the software is divided into a fixed kernel part and re-loadable modules. Virtual machine environment, as third approach, virtualizes the hardware of a system and provides operations to higher layers. These higher layers use re-loadable scripts to implement applications accessing virtual hardware.
A big challenge for EXEs is the memory footprint of the implementation. In [17] it is summarized for some implementations. The given memory footprints for the non-volatile memory are in the range between 17 kB for Tiny Manager and 39 kB for Bombilla (a TinyOS clone of Maté [18]). The used random access memory (RAM) is up to 1 kB. In comparison to the available memory of the popular sensor node tmote sky (a.k.a. telosb) [19], with a program memory of 48 kB and a RAM of 10 kB, large EXE implementations lead to two problems:

- not enough memory left for complex applications
- large memory footprints increase the probability of erroneous memory cells used by the code update itself

III. Operating System independent code update

To enable an OS independent WCU our concept implements its EXE as a basic input\output system (BIOS) on the nodes. Every time a node needs an update the application is shut down and the BIOS is started. To supervise the updates, a management instance is used in the network. The updates are transmitted using a dedicated DP. This section introduces all parts of our WCU. The used classification of the parts is derived from [6].

A. Management Instance

The management of the WCU covers four topics: user interface, database, target selection and data optimization.

1) User Interface: It enables a simple and intuitive access to the management instance and should be expandable to include additional functionality. A graphical user interface is the first choice for our approach.

2) Database: A database is used to hold information about available software and updates. It also monitors available software versions in the network.

3) Target selection: Our concept covers three scenarios how a target can be chosen for an update. The first one is initiated by the management instance. Based on its information an update is initiated on a specified node. In the second scenario a node requests a reallocated software image if a memory failure had been detected. If the normal communication protocol is not affected, it is used to send the request. If not the node has to use the communication protocol of the BIOS. Hence the management instance has to monitor the communication in the network. If a node is not communicating for a while its neighboring nodes get the order to check if the affected node is in BIOS mode. If this is true a neighbor of the node becomes a gateway node between the affected node and the management instance. The target selection can also chose the source of an update. If the update is already available in the neighborhood of a node it is not necessary to transfer it through the whole network again. Another node can provide it.

4) Data optimizer: The data optimizer has to reduce the amount of data to send. It is important, because wireless data transmission consumes a lot of energy. The general approach forms the difference between the current software image and the new one. Only these difference are transmitted. This task can be very complex. Often the code is identical in many parts but aligned to other memory addresses. Our concept has no primary focus on this task. We will start with a full image transmission and a simple optimization by forming the difference between two images.

B. Distribution Protocol

As mentioned above, during updates four roles are active in the network: the affected node, the gateway node, the management instance and the other nodes of the network. To get the data from the database to the gateway node the normal communication protocol of the network is used. After this the gateway node also switches to BIOS mode using the DP of the BIOS to transmit the new software image. This DP has to ensure the whole update has been transmitted and received correctly and complete. It has to coexist with other

protocols to prevent disturbances between the code update and the normal functionality of the network. There are three different approaches to enable coexistence:

- WCU uses protocol of the application
- WCU implements protocol of the application
- WCU implements an own coexisting protocol

The first variant cannot be used in cases where the application is not executable anymore. Implementing the protocol of the application as part of the BIOS (second variant) increases its memory footprint. Due to this an adaptation of the BIOS to new applications is necessary. Implementing a simple protocol that can coexist with many other protocols, presented as variant three, seems to be the best solution. Therefore a minimal community is needed. A listen-before-talk (LBT) scheme could be a good compromise. The nodes involved in the code update have to run this LBT protocol neither they run an own MAC protocol. Due to this the risk for packet collisions between the code update communication protocol and the normal protocol can be minimized and the overhead for the normal communication protocol is minimal. But LBT does not guarantee fairness between the protocols. One solution can be the usage of adaptable sending timers inside the BIOS.

C. Execution Environment

The EXE is responsible to install the received code update correctly and to restart the application. Some parts of the EXE can be provided as part of the update itself. They are is send in the first packets of the update. Due to this they do not permanently allocate program memory. Another way is to include the EXE memory image installed on the node. By this the full EXE is available. As a first solution we have chosen this option. The EXE can be enabled externally or internally. An external

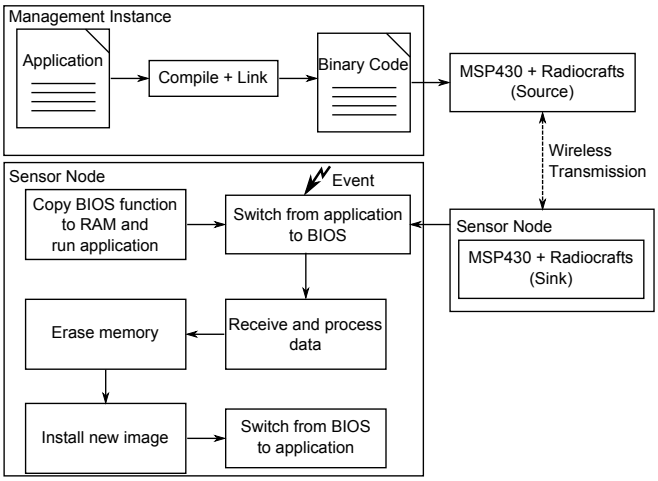


Fig. 3: Work flow of the code update prototype

trigger is send by the management instance, commanding a node to execute the BIOS functionality. Internally the BIOS can be triggered on a fault. There are two options: First, periodically the application is performing a memory test. If this test detects a fault the BIOS is activated. Second, a watchdog timer can be used to catch execution errors of the application, which may be caused by memory faults. On such an event the BIOS is activated and the memory is checked. If no errors are found the application is restarted. This approach ensures that on any system failure the node starts in a defined state, being able to communicate with the management instance. Please note that it is possible that the BIOS itself is affected. This problem can be mitigated by keeping the footprint of the BIOS small and having a second copy of the BIOS in the memory, whereby both BIOS instances are activated in alternation order on a watchdog event. On activation the BIOS functionality is copied to and executed from the RAM. By this, the BIOS gets full write access to the flash memory during update. A dummy interrupt routine is used by the BIOS to catch all interrupts that were enabled by the application, like running timers. This is done, because the BIOS itself might need some interrupts. Therefore it cannot globally disable all interrupts. After this the flash memory is erased and the new image is received segment-wise from the gateway node. If data for one segment has been received, it is stored in the flash-memory and the next segment is received from the gateway node. By this, the burden of managing the routing of a large memory image is shifted away from the node that is executing the code update, which helps to keep the BIOS small. After receiving the full image, it is run. Please note, that this is a restart of the system. Hence, we do not care about saving previous states. Moreover, it is possible to place a new image in the flash only by updating the modified portions of the old image, which significantly reduces the time for performing the code update. It is also possible to store the entire new memory image locally in an external memory before switching into BIOS mode. In that case the BIOS does not need to include a radio driver for communication with the gateway node. However, this requires that large portions of the application and the OS are not affected by memory faults.

IV. Implementation

We already have implemented a first prototype in C-language and have tested it on an MSP430 microcontroller based sensor node. For wireless communication a 169 MHz radio module from Radiocrafts has been used. As first step the management instance compiles and links the new application. Afterward, the resulting binary file can be send to the affected sensor node. If the update binary is very large it is not possible to buffer it in the RAM. Hence we only record data for one segment at once. A segment is the smallest erasable memory area of 512 B on an MSP430. There are two ways to relieve the RAM. First is to write the data into an external memory and copy it into the program memory after the complete image has been received. We have chosen the second way where data is written segmental into the program memory. If data for one segement has been received, it is erased the write data is written into it. During writing the communication is halted. Table I presents the duration of code updates for different sized binaries. The maximum available memory size on the used MSP430 is 256 kB. For our example application the update takes 53 seconds at 4.8 kbit/s. The size of our application is 2567 B, including BIOS functions. The actual transmitted data is three times the size of the binary. To shorten the duration it is necessary to reduce the image size. That could be done by only sending differences between the current image and the update. Figure 3 presents the update flow used in our

TABLE I: Code update duration at 4.8 kbit/s data rate

Binary size	2.6 kB	16 kB	32 kB	48 kB	256 kB
duration	53 s	5.4 min	10.7 min	16.3 min	86.9 min

prototype. If the senor node is started, the BIOS functions are copied into RAM. Subsequently the main function of the application is executed. On an event the BIOS is started and the node starts receiving the new binary image. A correct received packet is confirmed by sending an acknowledgment. Until the reception of a confirmation, the sending of new data is halted. The address information of the received data is extracted immediately after reception. If the last packet has been received the flash is erased and the new image is written into it. Finally the device reboots to execute the new application. Table II shows the code size values of the different parts of our prototype. The value for our BIOS is representing the erase, write and data processing functions. The size of this code depends strongly on the implemented DP and radio driver.

TABLE II: Code size of the different parts of the prototype implementation

Component	App 1	App 2	BIOS	Radio Driver
Code size	90 B	90 B	1700 B	800 B

V. Conclusion

This paper has presented a concept for an operating system independent wireless code update function for wireless sensor networks. Our concept extends the general application replacement function by a memory fault handling. Due to this a sensor node is able to check for memory faults and to request realigned code to emit erroneous memory cells. This increases the lifetime of a sensor node. A first prototype had been implemented and tested. the basic replacement functions have a memory footprint of 1700 bytes. The first results are promising. We further develop our code update function and integrate it into a cross platform.

Acknowledgment

The work presented in this paper is performed within the DIAMANT project funded by the BMBF under contract number 03IPT601X ("Zuverlässigkeit für hochsensible langlebige komplexe verteilte Anwendungen")

References

[1] M. Brzozowski, H. Salomon, K. Piotrowski, and P. Langendoerfer, "Cross-platform protocol development for sensor networks: Lessons learned," in *Proceedings of the 2Nd Workshop on Software Engineering for Sensor Network Applications*, ser. SESENA '11. New York, NY, USA: ACM, 2011, pp. 7–12. [Online]. Available: http://doi.acm.org/10.1145/1988051.1988054

[2] M. Brzozowski and P. Langendoerfer, "Is cross-platform protocol stack suitable for sensor networks? empirical evaluation," in *Wireless and Mobile Networking Conference (WMNC), 2013 6th Joint IFIP*, April 2013, pp. 1–8.

[3] P. Skoncej, F. Mühlbauer, and M. Schölzel, "Softwarebasierte selbstreparatur von flash-speichern für fehlertolerante mikrocontroller-basierte systeme," in *Workshop fr Fehlertolerante und energieeffiziente eingebettete Systeme: Methoden und Anwendungen*, 2015.

[4] M. Schölzel, P. Skoncej, and F. Vater, "On the feasibility of handling manufacturing faults in embedded memories by software means," in *IEEE International Workshop of Electronics, Control, Measurement, Signals and their application to Mechatronics*, 2015, submitted paper.

[5] M. Schölzel and P. Skoncej, "Software-based repair for memories in tiny embedded systems," in *Proc. European Test Symposium (ETS)*, 2015.

[6] C.-C. Han, R. Kumar, R. Shea, and M. Srivastava, "Sensor network software update management: A survey," *Int. J. Netw. Manag.*, vol. 15, no. 4, pp. 283–294, Jul. 2005. [Online]. Available: http://dx.doi.org/10.1002/nem.574

[7] H. C. Leligou, C. Massouros, E. Tsampasis, T. Zahariadis, D. Bargiotas, K. Papadopoulos, and S. Voliotis, "Reprogramming wireless sensor nodes," *International Journal of Computer Trends and Technology, May to June issue 2011*, vol. 7.

[8] G. Tolle and D. E. Culler, "Design of an application-cooperative management system for wireless sensor networks." in *EWSN*, vol. 5, 2005, pp. 121–132.

[9] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, ser. NSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251175.1251177

[10] K. Lin and P. Levis, "Data discovery and dissemination with dip," in *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, April 2008, pp. 433–444.

[11] T. Dang, N. Bulusu, W.-C. Feng, and S. Park, "Dhv: A code consistency maintenance protocol for multi-hop wireless sensor networks," in *Proceedings of the 6th European Conference on Wireless Sensor Networks*, ser. EWSN '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 327–342.

[12] W. Li, Y. Zhang, and B. Childers, "Mcp: An energy-efficient code distribution protocol for multi-application wsns," in *Distributed Computing in Sensor Systems*, ser. Lecture Notes in Computer Science, B. Krishnamachari, S. Suri, W. Heinzelman, and U. Mitra, Eds. Springer Berlin Heidelberg, 2009, vol. 5516, pp. 259–272.

[13] D. E. Thanos Stathopoulos, John Heidemann, "A remote code update mechanism for wireless sensor networks," Tech. Rep., November 26 2003. [Online]. Available: F17210EC8E10A05BE0306180528D7B54

[14] W. R. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive protocols for information dissemination in wireless sensor networks," in *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, ser. MobiCom '99. New York, NY, USA: ACM, 1999, pp. 174–185. [Online]. Available: http://doi.acm.org/10.1145/313451.313529

[15] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 81–94. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031506

[16] M. Rossi, G. Zanca, L. Stabellini, R. Crepaldi, A. Harris, and M. Zorzi, "Synapse: A network reprogramming protocol for wireless sensor networks using fountain codes," in *Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON '08. 5th Annual IEEE Communications Society Conference on*, June 2008, pp. 188–196.

[17] W. Munawar, M. Alizai, O. Landsiedel, and K. Wehrle, "Dynamic tinyos: Modular and transparent incremental code-updates for sensor networks," in *Communications (ICC), 2010 IEEE International Conference on*, May 2010, pp. 1–6.

[18] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS X. New York, NY, USA: ACM, 2002, pp. 85–95. [Online]. Available: http://doi.acm.org/10.1145/605397.605407

[19] *Telos - Ultra low power IEEE 802.15.4 compliant wireless sensor module*, moteiv, May 2004, rev B, datasheet.