Prof. Dr. Thomas Schmidt
HAW Hamburg, Dept. Informatik,
Raum 480a, Tel.: 42875 - 8452
Email: t.schmidt@haw-hamburg.de
Web: http://inet.haw-hamburg.de/teaching

Verteilte Systeme

Aufgabe 4: Autonome Sensorik im Internet der Dinge (IoT)

Ziele:

1. IoT-Systeme kennen und programmieren lernen
2. Autonome Abstimmungsprozesse gestalten
3. Fehlertoleranz und Robustheit herstellen

Vorbemerkungen:

In dieser Aufgabe programmieren Sie ‚echte' IoT-Knoten unter dem Betriebssystem RIOT. Sie sprechen die Systeme über einen Hilfsrechner (Rasberry PI unter Linux) an, der mit dem RIOT Knoten per USB verbunden ist. Auf dem RasPI ist die Tool-Chain zum ‚Flashen' der embedded Hardware vorinstalliert.

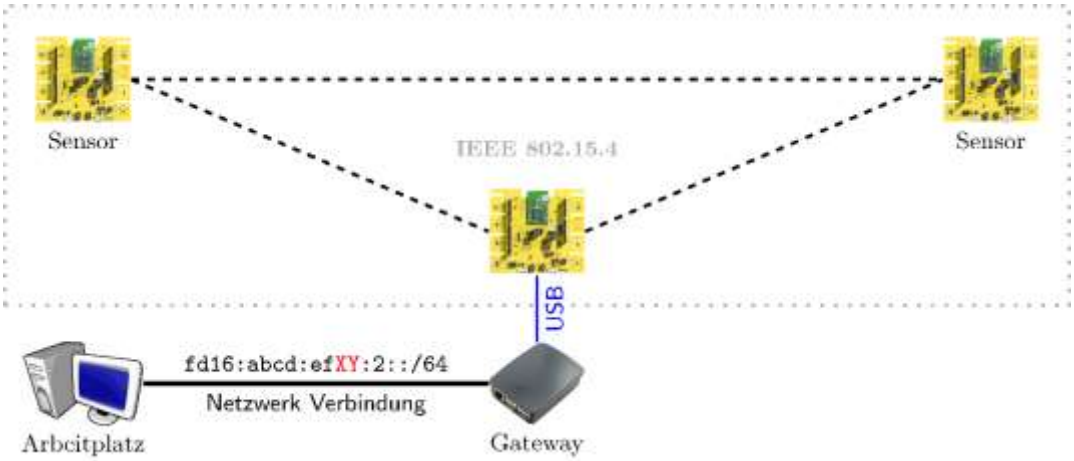

Figure 1: Experimenteller Aufbau mit Adresskonfigurationen

Die RIOT-Knoten messen die Temperatur und unterhalten sich über ein IEEE 802.15.4 LowPAN Funknetz miteinander. Dabei benutzen Sie das Constrained Application Protocol (CoAP) – eine leichtgewichtige http-Variante. Ein Knoten wird als Koordinator gewählt. Dieser Koordinator sammelt alle Temperaturmesswerte, errechnet die mittlere Temperatur und gibt sie regelmäßig bekannt.

Aufgabenstellung:

Implementieren Sie ein Temperatur-Sensornetzwerk, in welchem jeder RIOT-Knoten sowohl einfacher Sensor wie Koordinator sein kann. Im Detail implementieren Sie die folgenden Phasen:

Initialisierungsphase 0: Nach dem Start initialisiert sich das System, konfiguriert seine Hardware, Interfaces und Netzwerkkommunikation (Multicast Gruppenkommunikation, CoAP Stack). Diese Funktion wird Ihnen zur Verfügung gestellt. Hiernach hören die Knoten ständig auf der Multicast-Adresse `ff02::1`.

Koordinierungsphase 1 (Entdeckung): Ein Knoten, der (noch) keinen Koordinator kennt, broadcasted periodisch seine eigene link-lokale IP an die ‚all-nodes-multicast' Adresse `ff02::1` im Zeitabstand von `ELECT_MSG_INTERVAL`. Sobald eine Nachricht mit größerer IP empfangen wird, stellt der Knoten das Broadcasten ein.
Am Ende sollte nur noch der Knoten mit der höchsten IP zu hören sein.

Koordinierungsphase 2 (Koordinator-Wahl): Wurde nur noch eine (die höchste) IP für das Zeitintervall `ELECT_LEADER_THRESHOLD` empfangen, ist dieser Knoten der Koordinator, alle anderen werden Clients. Alle Clients melden sich mit CoAP (URI: "`coap://<Koordinator-IP>/nodes`") beim Koordinator an.

Arbeitsphase 3 (Temperaturmessung): Der Koordinator fragt reihum den eigenen und die entfernten Sensorwerte per CoAP ab (URI: "`coap://<client-IP>/sensor`") und errechnet dabei den exponentiell-gleitenden Mittelwert $\mathbf{X_i}$ nach der Formel ($\mathbf{X_0}$ **= <lokaler Sensorwert>, x** ist der aktuelle Meßwert):

$$\mathbf{X_{i+1} = (u-1)/u * X_i + 1/u * x}$$ - dabei ist **u = 16**

Am Ende jeder Runde schickt der Koordinator diesen Mittelwert per Multicast an alle Clients – Adresse `ff02::2017` – und pausiert für `ELECT_MSG_INTERVAL`.

Reparaturphase 4 (Fehlerkorrektur): Hört ein Client für einen Zeitraum `ELECT_LEADER_TIMEOUT` keine Abfragen vom Koordinator (oder kommt neu in das Sensornetz), startet er per Broadcast die Koordinierungsphase 1. Alle anderen Knoten gehen dann ebenfalls in die Koordinierungsphase 1 über und bestimmen einen neuen Koordinator.

Vorgehen:

1. Sie finden einen Implementierungsrahmen einschließlich vieler Hilfsfunktionen hier: https://github.com/inetrg/vslab-riot
2. Entwickeln und testen (!) Sie zunächst unter RIOT Native. Dabei lassen Sie RIOT als Prozess unter Linux oder MacOS ablaufen. Eine Anleitung finden Sie hier: https://github.com/inetrg/vslab-riot/wiki/Lab-Part-1
3. Wechseln Sie dann auf die im Praktikum bereitgestellte IoT Hardware. Eine Anleitung finden Sie hier: https://github.com/inetrg/vslab-riot/wiki/Lab-Part-2
4. Allgemeine Anleitungen und Tutorials zur RIOT-Entwicklung finden Sie hier: https://github.com/RIOT-OS/Tutorials